

Prediction of Protein-Protein Interactions using Deep Learning Techniques

A thesis presented for the degree of
Doctor of Philosophy

by: **Farzan Soleymani**

Supervisors: **Dr. Davide Spinello, Dr. Eric Paquet**

Keywords: Protein-Protein Interaction, Temporal Convolutional Network, Convolutional Autoencoder,
Long-Short Term Memory, Sequence-based

Mechanical Engineering
University of Ottawa



Abstract

Proteins are considered the primary actors in living organisms. Proteins mainly perform their functions by interacting with other proteins. Protein-protein interactions underpin various biological activities such as metabolic cycles, signal transduction, and immune response. PPI identification has been addressed by various experimental methods such as the yeast two-hybrid, mass spectrometry, and protein microarrays, to mention a few. However, due to the sheer number of proteins, experimental methods for finding interacting and non-interacting protein pairs are time-consuming and costly. Therefore a sequence-based framework called ProtInteract is developed to predict protein-protein interaction. ProtInteract comprises two components: first, a novel autoencoder architecture that encodes each protein's primary structure to a lower-dimensional vector while preserving its underlying sequential pattern by extracting uncorrelated attributes and more expressive descriptors. This leads to faster training of the second network, a deep convolutional neural network (CNN) that receives encoded proteins and predicts their interaction. Three different scenarios formulate the prediction task. In each scenario, the deep CNN predicts the class of a given encoded protein pair. Each class indicates different ranges of confidence scores corresponding to the probability of whether a predicted interaction occurs or not. The proposed framework features significantly low computational complexity and relatively fast response. The present study makes two significant contributions to the field of protein-protein interaction (PPI) prediction. Firstly, it addresses the computational challenges posed by the high dimensionality of protein datasets through the use of dimensionality reduction techniques, which extract highly informative sequence attributes. Secondly, the proposed framework, ProtInteract, utilises this information to identify the interaction characteristics of a protein based on its amino acid configuration. ProtInteract encodes the protein's primary structure into a lower-dimensional vector space, thereby reducing the computational complexity of PPI prediction. Our results provide evidence of the proposed framework's accuracy and efficiency in predicting protein-protein interactions.

Contents

1	Introduction	1
2	Background and literature review	3
3	Protein–Protein Interaction Prediction	30
4	Conclusion	56
4.1	Completed Work	56
4.2	Future Work	57
5	Appendix-A	58
5.1	Financial portfolio optimization with online deep reinforcement learning and restricted stacked au- toencoder—DeepBreath	58
5.2	Deep Graph Convolutional Reinforcement Learning for Financial Portfolio Management - DeepPocket	75
5.3	Long-term financial predictions based on Feynman–Dirac path integrals, deep Bayesian networks and temporal generative adversarial networks	91
6	Appendix-B	102
6.1	Non-autonomous State–Feedback to Stabilise the Error Dynamics in Time-Varying Area Coverage Control Problems	102
6.2	Temporal Difference Learning of Area Coverage Control with Multi-Agent Systems	108
6.3	Optimal Non-Autonomous Area Coverage Control with Adaptive Reinforcement Learning	117

Table 1: Acronyms

AE	Autoencoder
GRU	Gated Recurrent Unit
LSTM	Long-short Term Memory
RNN	Recurrent Neural Networks
PPI	Protein-Protein Interaction
CNN	Convolutional Neural Network
TCN	Temporal Convolutional Network
GCN	Graph Convolutional Neural Networks
Conv-TCN-AE	Convolutional Autoencoder with TCN layers
Conv-LSTM-AE	Convolutional Autoencoder with LSTM layers

List of Tables

1	Acronyms	ii
---	--------------------	----

List of Figures

1.1	Proteins may be represented as pseudo-time series such as financial instruments in the stock market.	2
3.1	Convolutional Autoencoder architecture that extracts low-dimensional informative features.	30

Chapter 1

Introduction

Proteins are macromolecules constituted by chains of amino acids, with side-chain groups extended from a connected backbone [1]. Proteins are abundant in living systems and conduct a wide range of unique functions such as transport, storage, membrane composition, enzymatic action, communication between cells, and tissue regeneration [2], among others. Proteins may interact with DNA, RNA, ligands, and other proteins to carry out cellular and biological functions [3]. The latter occurs by physical interaction between two or more proteins [4, 5]. Protein-protein interactions (PPI) create complexes that conduct numerous biological processes and activities such as metabolic cycles, signal transduction, DNA transcription and replication, catalysis, and immune response [6, 7, 8, 9, 10, 11, 12]. These interactions result from a specific biomolecular event to serve a non-generic function [4, 5, 13]. Thus, one may obtain biological insights into protein functions, disease prevalence, understanding cell physiology, and therapy development by identifying interaction amongst protein pairs [14, 15, 16, 17]. Comprehensive identification of PPIs can help to decode the molecular mechanisms of the specific biological functions involved [18]. It has been demonstrated by Jansen et al. [19] that understanding the interaction between known and unknown functional proteins can significantly contribute to deciphering protein functions.

We now understand the close relationship between abnormal PPIs and various pathologies such as neurodegenerative conditions, cancer, and infectious diseases [20, 21, 22, 18]. Identifying functional PPIs has proven to be a turning point in developing technology for targeted and individualised therapies [23, 8, 16, 18]. Recent advances in deep learning frameworks, fuelled by a large amount of data available for training, have revolutionised scientific studies [24]. In particular, it has become possible, to some extent, to predict PPI [25, 18]. The PPI and protein-ligand binding problems have also gained attention in bioinformatics, and computer-aided drug discovery [15, 26, 27]. Deep learning architectures for PPI prediction may operate on different inputs, including primary structure [28, 29, 30, 31], domain composition [32, 33], gene expression [34, 35, 36], and the 3-D protein structure [37], to mention a few.

It is often believed that 3-D structure delivers a relatively more complete set of information for PPI prediction [38]. However, all the vital information required to identify PPIs is encoded in the amino acid sequence, i.e., the primary structure [39], thus sequence-based methods have gained particular attention [40]. Moreover, obtaining the 3-D structure of proteins is experimentally expensive and time-consuming, whereas the amino acid sequence (primary structure) may be obtained inexpensively using high-throughput methods [41]. Many real-world time-series datasets (sequential) are multivariate and comprise abundant features representing the underlying system, such as financial market volatility analysis, health monitoring, and automotive control systems, to name a few [42, 43]. Accurately modelling these high-dimensional datasets has become a major interest in the deep learning community.

The underlying structure of financial datasets, such as the stock market and proteins, are represented as sequential data [44], the former being time-series data, while the latter is generated as a sequence of amino acids forming a chain [45, 46]. In the stock market, a financial instrument may be characterised using financial features, including stock prices, average true range, and exponential moving average, which continuously change over the interval that stock markets are open, creating time-series datasets [47] (see Chapter 5). Similarly, proteins can be modelled using the physicochemical properties of their amino acid chain.

Additionally, the financial instruments are dependent on past and current events, and thus, their trajectory follows a causal directionality of time, i.e., time flows from the past to the future. Such behaviour can be observed in the composition of proteins, where the underlying sequential structure and orientation of proteins are dictated by the well-defined order of amino acids in the peptide chain: the amino end of each amino acid (its N-terminus) links to the carboxyl end (C-terminus) of the growing peptide chain [48, 49], and the position of each amino acid

along a specific chain determines the chain’s final three-dimensional configuration and its binding ability [50].

The arrangement of amino acids along a protein determines a causal directionality, which is contrary to an assumption on which traditional neural networks are based, namely the independence of all the input vector units [51, 52]. Therefore, proteins may be considered pseudo-time series, as illustrated in Fig. 1.1, where the time may be assimilated to the amino acid positions in the polypeptide chain, which determines the functionality and properties of that protein [53]. Traditional neural networks cannot capture sequential information. Therefore, certain methods can be employed to capture sequential patterns, such as long short-term memory (LSTM) and temporal convolutional network (TCN) [54, 42, 55].

This thesis is dedicated to developing a generic framework that may be transferred to a wide range of time- and pseudo-time-series problems, such as area coverage control in an evolving environment, financial markets, and PPIs. The common factor among these problems is their underlying structure, that is, sequential evolution [56, 57]. For instance, one may formulate an adaptive control scheme for the area coverage control in an environment with time-varying risk densities. The adaptive control scheme is developed using temporal difference learning, aiming to obtain optimal trajectories concerning a given performance index later in time, relying on time-dependent measurements along the trajectories.

In another example, the price of financial instruments in the stock market may be assimilated into time series; thus, the underlying distribution associated with financial instruments may vary over time. Similarly, the primary structure of proteins is constructed of a sequence of amino acids that follow a causal directionality posed by N-terminus to C-terminus translation. Therefore, proteins may be modelled and analysed as a pseudo-time series [48, 49, 58]. Accordingly, the knowledge may be transferred from prior designed frameworks regarding the prediction of stock market behaviour with financial time-series data [59, 60] to a novel framework to predict PPIs [61, 62].

In this thesis, I propose a sequence-based method called ProtInteract that predicts PPIs through two main tasks: encoding and prediction. A novel autoencoder architecture encodes each protein to remedy the computational complexity posed by the high dimensionality of protein representation while extracting sequential attributes of the protein’s primary structure. ProtInteract is trained using the STRING database, which contains PPI datasets for various organisms. The prediction tasks are formulated as three different multi-class classification scenarios, comprising two, three, and five classes [63].

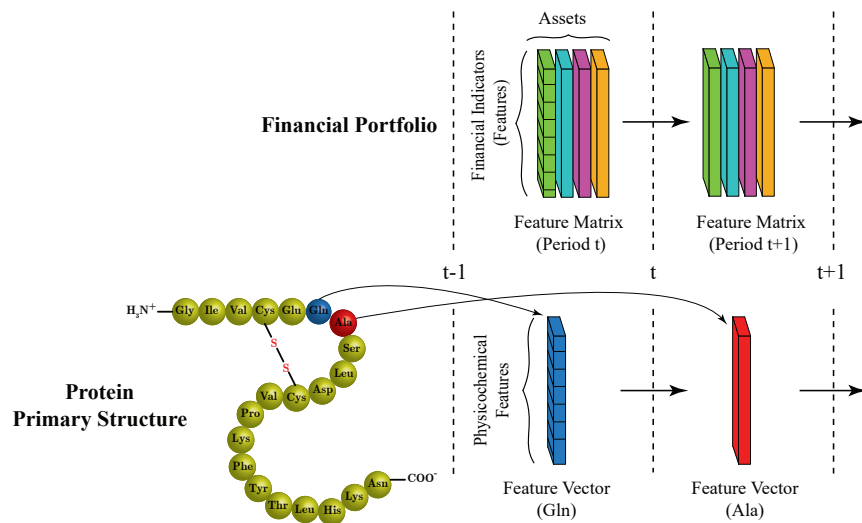


Figure 1.1: Proteins may be represented as pseudo-time series such as financial instruments in the stock market.

This thesis is organised by scientific articles as follows: In Chapter 2, some of the most recent sequence-based and structure-based PPI prediction methods, along with protein datasets and data processing methods, are discussed. Chapter 3 outlines the proposed framework, and the experimental results are presented for the examples of *Homo sapiens* and *Mus musculus*. Finally, Chapter 4 concludes this thesis. The future work is addressed in Section 4.2. The prior works on financial time series and the developed deep learning methods for portfolio management and stock market prediction are presented in Chapter 5. The area coverage control problem with multi-agent systems is discussed in Chapter 6.

Chapter 2

Background and literature review

A comprehensive literature review on the state of the art methods, which includes references to commonly used protein-protein interaction databases and a review of several recently developed computational methods addressing sequence and structure-based PPI prediction, is presented in the following paper, published in the Computational and Structural Biotechnology journal on August 30, 2022. In addition, a brief discussion of protein structures related to the class of computational methods is presented.

journal homepage: www.elsevier.com/locate/csbj

Review

Protein–protein interaction prediction with deep learning: A comprehensive review

Farzan Soleymani^a, Eric Paquet^{b,*}, Herna Viktor^c, Wojtek Michalowski^d, Davide Spinello^a^a Department of Mechanical Engineering, University of Ottawa, Ottawa, ON, Canada^b National Research Council, 1200 Montreal Road, Ottawa, ON K1A 0R6, Canada^c School of Electrical Engineering and Computer Science, University of Ottawa, ON, Canada^d Telfer School of Management, University of Ottawa, ON, K1N 6N5, Canada

ARTICLE INFO

Article history:

Received 22 June 2022

Received in revised form 29 August 2022

Accepted 30 August 2022

Available online 19 September 2022

Keywords:

Protein–protein interaction

Deep learning

Protein design

Sequence-based

Structure-based

ABSTRACT

Most proteins perform their biological function by interacting with themselves or other molecules. Thus, one may obtain biological insights into protein functions, disease prevalence, and therapy development by identifying protein–protein interactions (PPI). However, finding the interacting and non-interacting protein pairs through experimental approaches is labour-intensive and time-consuming, owing to the variety of proteins. Hence, protein–protein interaction and protein–ligand binding problems have drawn attention in the fields of bioinformatics and computer-aided drug discovery. Deep learning methods paved the way for scientists to predict the 3-D structure of proteins from genomes, predict the functions and attributes of a protein, and modify and design new proteins to provide desired functions. This review focuses on recent deep learning methods applied to problems including predicting protein functions, protein–protein interaction and their sites, protein–ligand binding, and protein design.

Crown Copyright © 2022 Published by Elsevier B.V. on behalf of Research Network of Computational and Structural Biotechnology. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

Contents

1.	Introduction	5317
2.	Protein Structure	5317
2.1.	Primary structure	5317
2.2.	Secondary structure	5320
2.3.	Tertiary structure	5320
2.4.	Quaternary structure	5320
3.	Protein Shapes	5321
3.1.	Protein folding	5322
4.	PPI Databases	5323
5.	Deep Learning Models	5323
6.	PPI Prediction Methods	5326
6.1.	PPI Site Prediction	5326
6.2.	Structure-based PPI Prediction	5326
6.3.	Structure-based PPI Prediction Using Computational Methods	5327
6.4.	Sequence-based PPI Prediction	5328
6.5.	Sequence-based PPI Prediction Using Computational Methods	5329
7.	Protein Design	5332
7.1.	Protein Function	5332
7.2.	Structure Design	5334
7.3.	Sequence Design	5334

* Corresponding author.

8. Conclusion	5335
Declaration of Competing Interest	5335
Acknowledgement	5335
References	5335

1. Introduction

Proteins are organic molecules abundant in living systems and conduct a wide range of unique functions such as transport, storage, membrane composition, and enzymatic action [1], among others. Proteins may interact with DNA, RNA, ligands, and other proteins to carry out cellular and biological functions [2]. The latter occurs by physical interaction between two or more proteins [3,4]. These interactions ought to comply with two conditions: first, the interaction must be by design, i.e. the result of a specific biomolecular event; second, the interaction has evolved to serve a certain non-generic function [3–5]. Thus, one may obtain biological insights into protein functions, disease prevalence, and therapy development by identifying interaction amongst protein pairs [6–8]. Hence, protein–protein interaction (PPI) and protein–ligand binding problems have drawn attention in bioinformatics and computer-aided drug discovery [7,9,10]. Computational methods paved the way for scientists to predict the 3-D structures of proteins from genomes and, hence, to predict their functions and attributes, allowing them to modify proteins and design new ones to target desired functions. However, experimental validation benchmarking remains challenging [11].

Protein–protein interactions compose complexes to conduct numerous biological processes and functions such as metabolic cycles, signal transduction, DNA transcription and replication, catalysis, and immune response [12–18]. The activities of cells and their functions are affected by abnormalities in protein interactions, leading to numerous diseases such as cancer and chronic degenerative diseases [19]. Comprehensive identification of PPIs can help to decode the molecular mechanisms of the specific biological functions involved [19]. The proximity of proteins in PPI is of paramount importance for specific functionality. Despite significant efforts in molecular biology and genomics, the functions of most proteins are not yet established [20–22]. It has been demonstrated by Jansen et al. [23] that the interaction between known and unknown functional proteins can significantly contribute toward deciphering many protein functions. Therefore, predicting PPIs has become a crucial challenge in the field of bioinformatics [19].

PPIs may help in decoding the functionality of unannotated proteins [19,24]. Therefore, many experimental studies have been conducted to identify PPI, among which the yeast two-hybrid [25–27], mass spectrometry [28–32], protein microarrays [33–36] are often used [37,38]. However, these approaches are laborious and time-consuming, which makes them difficult to employ for all protein pairs [9,39,40]. Moreover, the validity of the experimental techniques is highly dependent on how well one implements the assay protocols in target organisms [41]. Therefore, one may use computational methods as pre-treatment in advance of the experimental methods, aiming to reduce false-positive and false-negative results [24,41,42].

A protein comprises a unique linear sequence of amino acids called its primary structure, which determines the folded shape or conformation. The local secondary structure elements, such as strands, helices, and random coils, are created as a result of interactions between the protein backbone, the side-chains, and the environment and extended to the ultimate 3-D structure of the protein [43]. The large number of possible configurations of the peptide backbone, and the desirable chemical bonding geometry

and interactions, make the problem of modelling protein structures challenging [17]. This paper reviews recent advances in deep learning methods developed and/or applied to problems, including predicting protein functions, protein–protein interactions and their sites, protein–ligand binding, and protein design.

This review is structured as follows. We first outline protein structure architectures in Section 2. Next, we describe protein shapes in Section 3. Then, we present some of the main resources for protein structures and sequences in Section 4. Section 5 briefly explains some of the most commonly used deep learning methods, and Section 6 provides an overview of PPI prediction. Section 6.2 and Section 6.3 discuss structure-based PPI prediction methods and their computational solutions, respectively. Sequence-based PPI prediction methods and their associated computational solutions are described in Section 6.4 and Section 6.5. Section 7 reviews deep learning methods addressing protein design problem and Section 8 concludes the review.

2. Protein Structure

Proteins are a broad class of biomolecules forming more than 50% of the dry weight of cells [44]. Their diverse functionality and abundance determine the function and structure of cells, with each protein being an agent performing a specific biological role [44]. Genes are the basic physical and functional units of inheritance and act as instructions to create the proteins that are the agents of biological function. In fact, a unique protein structure is encoded by each gene in cellular DNA, which leads to numerous possible structures [1]. The uniqueness of proteins originates in the amino acid sequences and the bonds that hold them together.

The interaction between proteins is mainly non-covalent [43] except for covalent disulfide bonds (formed by the coupling of two thiol (–SH) groups), between the cysteine amino acid residues of the interacting partner proteins. Hydrogen bonding between proteins in a specific PPI is the most important type of non-covalent interaction.

The main and side-chain atoms of the different amino acid residues are involved in the hydrogen bonding between interacting protein partners. The ion pairs, which form mainly between an acidic and a basic amino acid in the proteins, form the second most important non-covalent interaction between protein partners [45]. The stability of protein structures is also affected by long-range interactions. The impact of short, medium and long-range interactions on various structural classes of proteins are discussed in [46–48].

As stated in [47], the all- α protein class, i.e., the proteins whose secondary structure is completely formed by α -helices apart from a few β -sheets on the edges [49], are governed by medium-range interactions. In contrast, long-range interactions dominate in all- β proteins, in which the secondary structure is mainly composed of β -sheets aside from some α -helices on the edges [49].

The primary to quaternary protein structures are examined in more detail in the following sections.

2.1. Primary structure

The primary structure, as shown in Fig. 1, is a unique, linear, amino acid sequence that forms the backbone of protein. Intramolecular bonding and folding of the linear amino acid chain

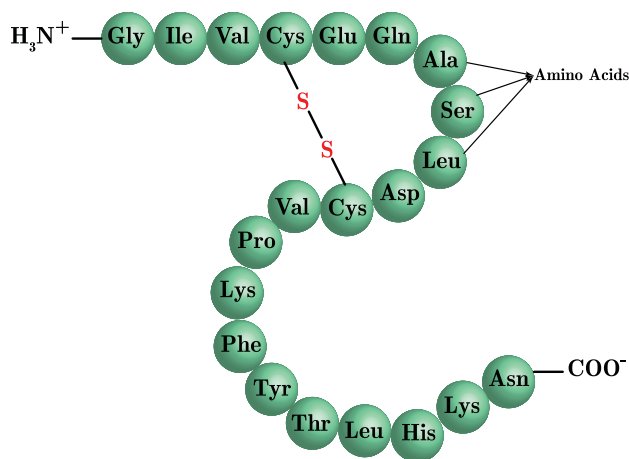


Fig. 1. primary structure.

eventually establish the protein's three-dimensional shape. The sequence of protein is determined by the gene encoding it, so changing the gene's DNA sequence may alter the protein's amino acid sequence and, thus, the protein's overall structure and function.

Amino acids, the building blocks of proteins, are small organic molecules composed of a central carbon atom, called the α -carbon, attached to an amino group ($-\text{NH}_2$), a carboxyl group ($-\text{COOH}$), and a hydrogen atom [50]. The carboxyl group is typically deprotonated and carries a negative charge at physiological pH (7.2–7.4) [51], whereas the amino group is typically protonated and shows a positive charge. The identity of each amino acid depends on its R group, which is an atom or group attached to the central atom. For example, the R group of glycine, as shown in Fig. 2 is a hydrogen atom, while the R group of alanine is a methyl group ($-\text{CH}_3$). Fig. 2 illustrates the twenty common amino acids, each of which has a unique side chain. The side chains govern each acid's chemical behaviour, e.g. whether it is acidic, basic, polar, or nonpolar. Nonpolar amino acids contain aliphatic (hydrocarbon) chains, while polar neutral amino acids contain a hydroxyl ($-\text{OH}$), sulfur, or amide in the R group. Polar acidic amino acids have a carboxylic acid group in the side chain, in addition to the

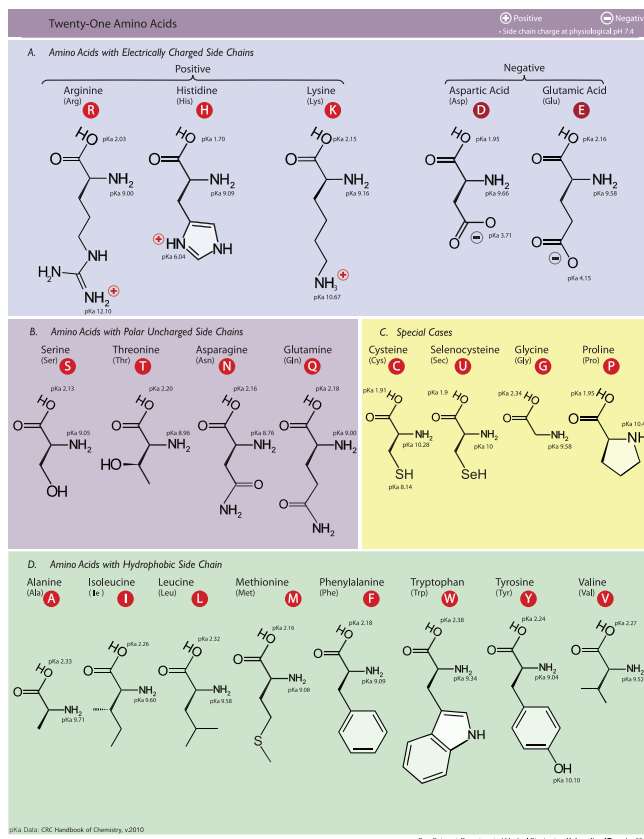


Fig. 2. Amino acids.

one in the backbone. Polar basic amino acids contain an amine group (which may be neutral or charged) in the side chain, in addition to that in the backbone.

The physicochemical properties of 20 common amino acids are reported in Table 1. SASA represents the solvent-accessible surface area, and the side-chain net charge number is given by NCN [52]. These properties help determine the feasibility of protein's interac-

Table 1

Physicochemical properties of 20 amino acids. Column (a) steric parameters (graph shape index) [54,57]; (b) volume; (c) isoelectric point; (d) helix probability [58]; (e) sheet probability [58]; (f) hydrophobicity [59]; (g) hydrophilicity [59]; (h) side-chain residue size [52,54,60]; (i) polarity [52]; (j) polarizability [52]); (SASA) solvent-accessible surface area; (NCN) net charge number [52].

Amino acid	Symbol	a	b	c	d	e	f	g	h	i	j	SASA	NCN
Alanine	A	1.28	1.00	6.11	0.42	0.23	0.62	−0.50	27.50	8.10	0.046	1.181	0.007187
Cysteine	C	1.77	2.43	6.35	0.17	0.41	0.29	−1.00	44.60	5.50	0.128	1.461	−0.03661
Aspartate	D	1.60	2.78	2.95	0.25	0.20	−0.90	3.00	40.00	13.00	0.105	1.587	−0.02382
Glutamate	E	1.56	3.78	3.09	0.42	0.21	−0.74	3.00	62.00	12.30	0.151	1.862	0.006802
Phenylalanine	F	2.94	5.89	5.67	0.30	0.38	1.19	−2.50	115.50	5.20	0.29	2.228	0.037552
Glycine	G	0.00	0.00	6.07	0.13	0.15	0.48	0.00	0.00	9.00	0.00	0.881	0.179052
Histidine	H	2.99	4.66	7.69	0.27	0.30	−0.40	−0.50	79.00	10.40	0.23	2.025	−0.01069
Isoleucine	I	4.19	4.00	6.04	0.30	0.45	1.38	−1.80	93.50	5.20	0.186	1.81	0.021631
Lysine	K	1.89	4.77	9.99	0.32	0.27	−1.50	3.00	100.00	11.30	0.219	2.258	0.017708
Leucine	L	2.59	4.00	6.04	0.39	0.31	1.06	−1.80	93.50	4.90	0.186	1.931	0.051672
Methionine	M	2.35	4.43	5.71	0.38	0.32	0.64	−1.30	94.10	5.70	0.221	2.034	0.002683
Asparagine	N	1.60	2.95	6.52	0.21	0.22	−0.78	2.00	58.70	11.60	0.134	1.655	0.005392
Proline	P	2.67	2.72	6.80	0.13	0.34	0.12	0.00	41.90	8.00	0.131	1.468	0.23953
Glutamine	Q	1.56	3.95	5.65	0.36	0.25	−0.85	0.20	80.70	10.50	0.18	1.932	0.049211
Arginine	R	2.34	6.13	10.74	0.36	0.25	−2.53	3.00	105.00	10.50	0.291	2.56	0.043587
Serine	S	1.31	1.60	5.70	0.20	0.28	−0.18	0.30	29.30	9.20	0.062	1.298	0.004627
Threonine	T	3.03	2.60	5.60	0.21	0.36	−0.05	−0.40	51.30	8.60	0.108	1.525	0.003352
Valine	V	3.67	3.00	6.02	0.27	0.49	1.08	−1.50	71.50	5.90	0.14	1.645	0.057004
Tryptophan	W	3.21	8.08	5.94	0.32	0.42	0.81	−3.40	145.50	5.40	0.409	2.663	0.037977
Tyrosine	Y	2.94	6.47	5.66	0.25	0.41	0.26	−2.30	117.30	6.20	0.298	2.368	0.023599

Table 2

The nonstandard amino acids [65–67].

Name	Symbol	Abbr
Aspartic acid or Asparagine	B	Asx
Leucine or Isoleucine	J	Xle
Pyrrolysine	O	Pyl
Selenocysteine	U	Sec
Glutamic acid or Glutamine	Z	Glx
unknown amino acid	X	Unk

Table 3

amino acids classified by side chain properties.

Charge	Positive	H, K, R
	Negative	D, E
	Neutral	A, C, N, P, Q, S, F, G, I, L, M, T, V, W
Polarity	Polar	Y
	Nonpolar	C, D, E, H, K, N, Q, R, S, T
Aromaticity	Aliphatic	A, F, G, I, L, M, P, V, W
	Aromatic	I, L, V
	Neutral	F, H, W, Y
Size	Small	A, C, D, E, Q, R, S, G, K, M, N, P, T
	Medium	A, G, P, S
	Large	D, N, T

tion [52]. The physicochemical attributes of amino acids, such as hydrophathy [53], isoelectric, the pH at which the molecule carries

no net charge [54,55], and charge, play crucial roles in identifying the interaction between protein sequences [56].

Beyond the common amino acids shown in Table 1, there are also nonstandard amino acids [61]. These are also known as biosynthetic amino acids, and require complex synthetic and translational mechanisms that differ from the canonical enzymatic system used for the 20 standard amino acids [62]) namely, pyrrolysine [63] and selenocysteine [64]. These nonstandard amino acids are presented in Table 2. Sometimes it is not possible to differentiate two closely related amino acids. Therefore, we have the indeterminate residues in protein sequences as represented by symbols B, J, Z and X.

One way to classify amino acids is based on the side chains, as shown in Table 3 [68,69], in which case.

Multiple amino acids are linked together by peptide bonds, forming a long chain called the polypeptide. The order of the amino acids determines the polypeptide's functionality. Polypeptides are classified by the number of amino acid units in the chain. Each amino acid is linked covalently to its neighbours by peptide bonds, in a dehydration synthesis (condensation) reaction. Each protein is composed of one or more polypeptide chains. During protein synthesis, the carboxyl group (–COOH) of the amino acid at the end of the growing polypeptide chain reacts with the amino group of an incoming amino acid, forging a peptide bond and releasing a water molecule. Peptide bonds connect the carbon of the carboxyl group of one amino acid to the nitrogen of the amino group of the next, as shown in Fig. 3.

Table 4

Databases for PPI prediction

Type	Database	Description	Last update	URL
Protein–Protein Interactions	STRING [109]	Functional associations between protein pairs, which contains 67,592,464 proteins from 14094 organisms; 20,052,394,042 interactions.	2021	https://string-db.org/
	IntAct [116]	Contains manually curated datasets (topical), interactomes (for 16 different species) and annotations of experimental evidence.	2021	https://www.ebi.ac.uk/intact/home
	Biogrid [111]	Contains 2,467,140 protein and genetic interactions, 29,417 chemical interactions and 1,128,339 PTMs from major model organism species.	2020	http://www.thebiogrid.org/
	DIP [112]	Experimentally determined PPI database including biological information of proteins, PPIs and experimental techniques for identifying interactions.	2020	https://dip.doe-mbi.ucla.edu/dip/Main.cgi
	Negatome 2.0 [108]	Contains 21,795 interactions, with scores of zero and one, using text mining from literature and analysing protein complexes from PDB.	2014	http://mips.helmholtz-muenchen.de/proj/ppi/negotome/
	MINT [114]	Experimentally curated PPI database that includes approximately 117001 PPIs from 607 different species.	2012	https://mint.bio.uniroma2.it/
	HPRD [115]	Consists of 41,327 PPIs, 93,710 PTMs, 22,490 Subcellular Localizations and 112,158 Protein Expressions.	2010	http://www.hprd.org
	BIND [113]	PPIs collected from of humans, yeasts, nematodes, etc.	2005	http://download.baderlab.org/BINDTranslation
Protein sequences	UniProt [106]	A collection of protein sequence and functional information, including UniProtKB, UniParc, UniRef and Proteomes. UniProtKB contains 567,483 reviewed (Swiss-Prot)—manually annotated, and 231,354,261 unreviewed (TrEMBL)—computationally analysed, protein sequences.	2020	http://www.uniprot.org
	SWISS-MODEL [107]	A web-based integrated service providing information for protein structure homology modelling. The repository contains 2,217,470 models from SWISS-MODEL for UniProtKB targets, as well as 180,107 structures from PDB with mapping to UniProtKB.	2020	https://swissmodel.expasy.org/
	PIR [119]	Integrated protein resources, including protein sequences and high-quality annotations by integrating more than 90 biological databases.	2022	http://pir.georgetown.edu/
Higher-level structures	RCSB PDB [110]	Information about the 3-D structure of proteins, nucleic acids, and complex assemblies. 191144 structures, 57349 human sequence structures, and 14406 nucleic acid-containing Structures	2021	https://www.rcsb.org/
	SCOP [122]	Classification of known proteins and a comprehensive description of the structural and evolutionary relationships between them. As of 2022-05-30, this dataset contains 72,448 non-redundant domains, representing 858,316 protein structures.	2022	http://scop.mrc-lmb.cam.ac.uk/scop
Genomic information	CGD [124]	A resource for genomic sequence data, genes and protein information for <i>Candida albicans</i> and related species.	2022	http://www.candidagenome.org/

Polypeptide chains are directional, i.e. its ends are chemically distinct from one another. The end with a free amino group is called the amino terminus or N-terminus, while the other end has a free carboxyl group, and is known as the carboxyl terminus or C-terminus (see Fig. 3).

Most of the side chains are nonpolar, several are positively or negatively charged, some are polar but not charged. These features, and their consequent bonds, are responsible for protein structure and functionality by maintaining the protein in a specific shape or conformation. The polar side chains can form hydrogen bonds, while the charged side chains can form ionic bonds. Hydrophobic side chains interact via van der Waals interactions [1]. Consequently, protein folding is directed by the side-chain interactions, the sequence and the location of amino acids in that protein. The order of the acids, i.e. the primary structure, determines which bond types can form at each location along the polypeptide, and thus governs the protein's tertiary structures [70].

2.2. Secondary structure

Secondary structures result from interactions between parts of the polypeptide chain. The most common folding patterns are α -helices and β -pleated sheets [44].

In an α -helix, the hydrogen bonding occurs between the carbonyl group ($C=O$) of one amino acid and the hydrogen atom of the amino acid four places further along the chain. This bonding pattern draws the polypeptide chain into a helix, with each turn of containing 3.6 amino acids. The R groups stick outwards from the α -helix, and are free to interact. In a β -pleated sheet, segments of a polypeptide chain align next to each other, making a sheet structure coupled by hydrogen bonds between carbonyl and amino groups of backbone, while the R groups extend above and below the plane of the sheet.

The strands of a β -pleated sheet may be parallel (i.e. their N- and C-termini match up), or anti-parallel (i.e. the N-terminus of one strand alongside the C-terminus of the next). In certain cases, the amino acids are not found in α -helices or β -pleated sheets. For instance, proline is known as a "helix breaker" owing to its unusual R group, which bonds to the amino group to form a ring creating a bend in the chain that prevents helix formation. Proline is generally found in bends, unstructured regions between secondary structures. Proteins can contain α -helices, β -pleated sheets or both, or may form neither type.

2.3. Tertiary structure

The tertiary structure, as shown in Fig. 5, is formed as the polypeptide chains of protein molecules fold into a more compact shape with a low surface-to-volume ratio. The tertiary structure results mainly from electrostatic forces between the R groups. For instance, oppositely charged R groups bond ionically, while similarly charged R groups repel one another. Similarly, polar R groups may form hydrogen bonds and other dipole-dipole interactions.

A cluster of amino acids with nonpolar, hydrophobic R groups on the inside of the protein leaves the hydrophilic amino acids on the outside to interact with nearby water molecules.

The tertiary structure can also be produced by disulfide bonds. Disulfide bonds are covalent and hence keep parts of the polypeptide firmly attached to each other [44]. A synthesis of a tertiary structure is portrayed in Fig. 6.

2.4. Quaternary structure

Many proteins comprise two or more polypeptide chains that interact to form a stable folded structure, known as a subunit of

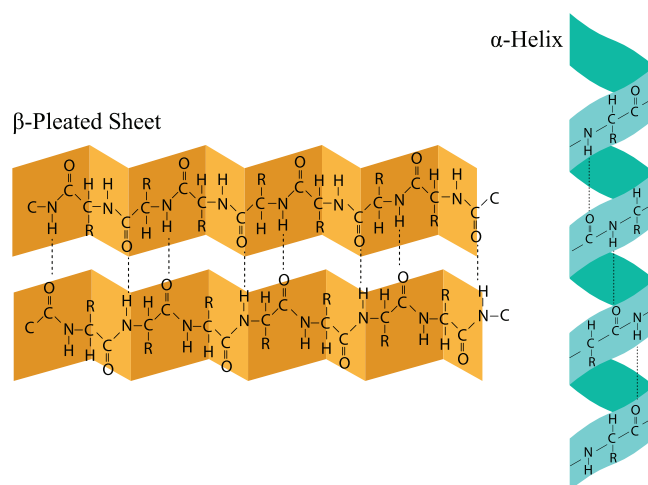


Fig. 4. secondary structure.

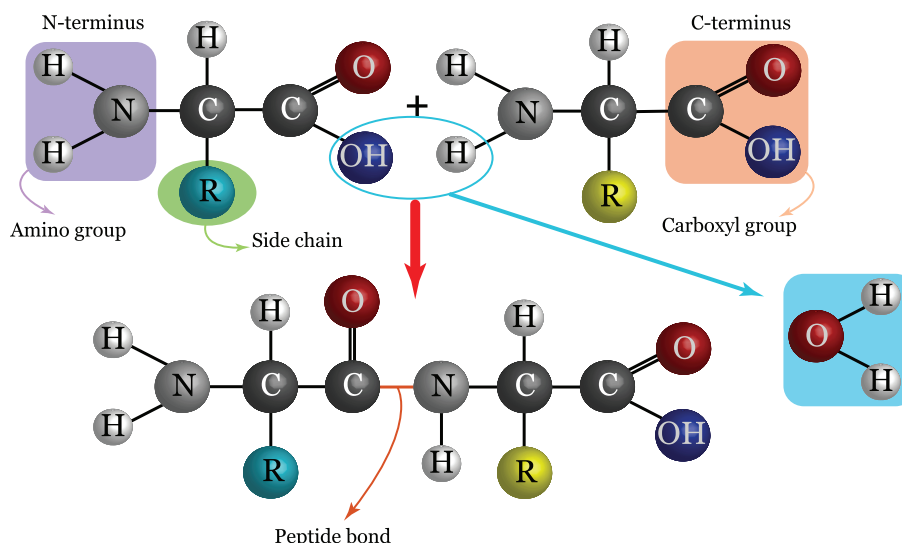


Fig. 3. peptide bond formation. The N-terminus is on the left, and the C-terminus is on the right.

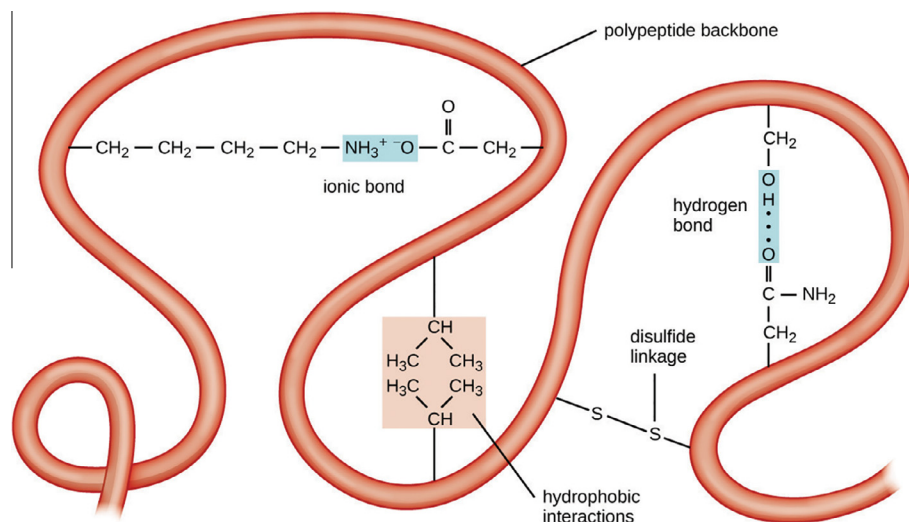


Fig. 5. tertiary structure [70].

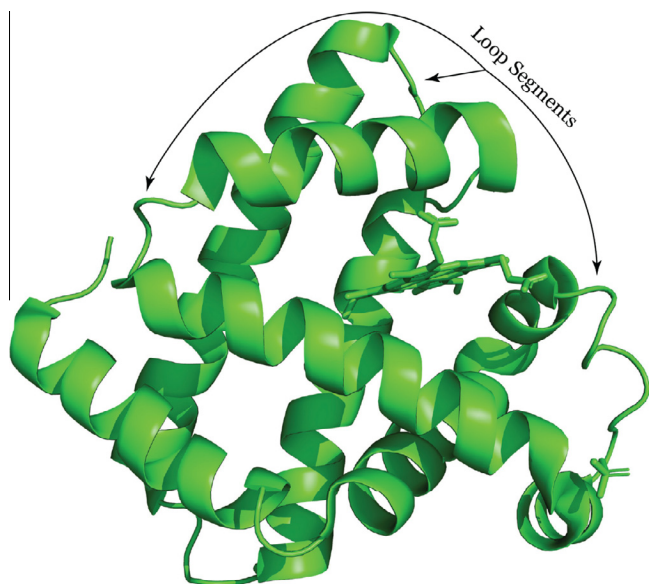
Fig. 6. myoglobin illustrates a type of tertiary structure consisting of α helices connected by loop segments.

Fig. 7. aspartate transcarbamoylase is an enzyme at the beginning of the pathway for pyrimidine synthesis, presents a remarkable example of quaternary structure.

the protein. The amino acid sequences of each subunit can either be identical (as in tobacco mosaic virus protein), similar (as in the α and β chains of hemoglobin), or entirely different (as in aspartate transcarbamoylase see Fig. 7). Subunit arrangement establishes the protein's quaternary structure.

In the following section, protein shapes are discussed.

3. Protein Shapes

Proteins may be classified on shape and solubility into three global classes: fibrous (Fig. 8), globular (Fig. 9), or membrane (Fig. 10).

In general, fibrous proteins have relatively simple, regular linear structures, and often provide cells with structural functions.

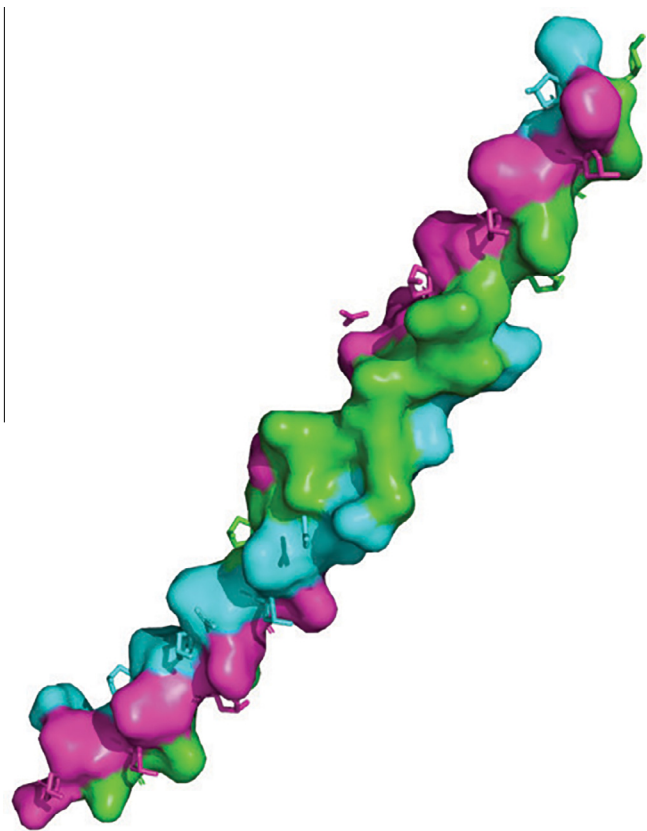


Fig. 8. a small part of collagen separated by chains.

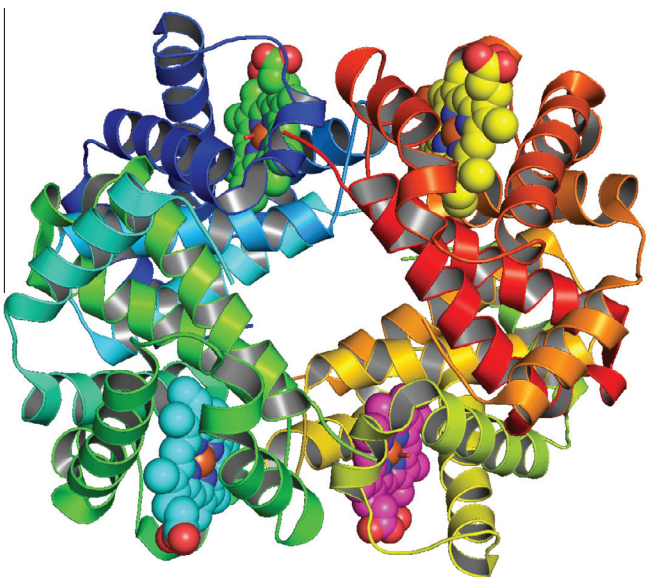


Fig. 9. haemoglobin, a globular protein.

Fibrous proteins are usually insoluble in water and dilute salt solutions. A well-known example of such proteins is collagen, abundant in all animals [71]. As illustrated in Fig. 8, collagen is composed of three chains, each containing 1400 amino acids, twisted together into a triple helix. Glycine appears in every third position along each chain, and, due to its small size, it perfectly fits inside the helix. Proline and hydroxyproline [72] fill numerous

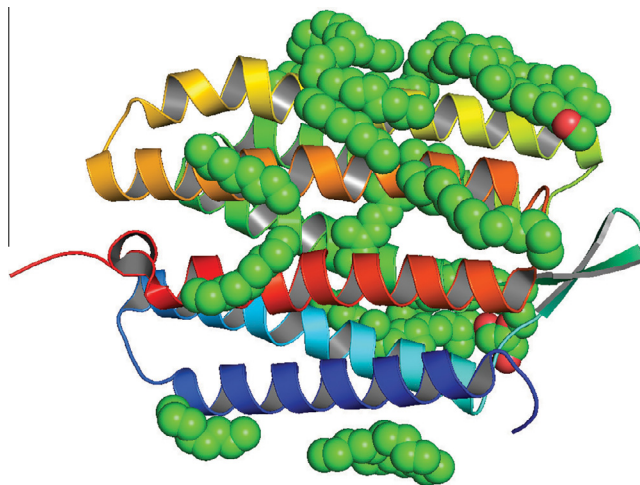


Fig. 10. bacteriorhodopsin, a membrane protein.

positions on a chain. There are numerous types of collagen, all comprising a long stretch of triple helix attached to different ends.

On the other hand, globular proteins are nearly spherical, as shown in Fig. 9 and very soluble in aqueous solutions. Examples include haemoglobin, in the red blood cells, that binds to oxygen.

Membrane proteins have hydrophobic side chains directed outwards, and interact with the nonpolar phase within membranes. Therefore, membrane proteins are insoluble in aqueous solutions but can be solubilised in solutions of detergents. Bacteriorhodopsin (Fig. 10) represents an example of such proteins which is made by halophilic (salt-loving) bacteria. This protein pumps protons across cell membranes, powered by sunlight [44].

PPI prediction and protein design may benefit from classifying deformable protein shapes. A novel classification method for protein shapes, based on their macromolecular surfaces, is introduced in [73]. They proposed a novel description, based on bifractional Fokker–Planck and Dirac–Kähler equations for deformable shapes.

3.1. Protein folding

Over the past two decades, considerable efforts have been made in the protein design field, which has further expanded due to the evolution of computational methods and machine learning algorithms. Some of the successful examples include novel folds in protein design [74,75], enzymes [76,77], antibodies [78–80], vaccines [78,81], ligand-binding proteins [82,83], protein assemblies [84–88], and membrane proteins [89–91]. Some of the most recent comprehensive reviews in this field are presented in [92–95]. Generally, the backbone structure of a target protein forms the input for computational protein design. An optimal sequence can be generated using computational sampling methods, seeking potential folding into the desired structure for experimental validation.

A vital component of the solution process involves the scoring function, which can distinguish folds that are or are not physically compatible with a given amino acid sequence [96]. One approach to defining the scoring function considers van der Waals and electrostatic energy along with knowledge-based terms such as backbone dihedral preference statistics about protein structures [97,98], and side-chain rotamers [99]. There is a gap between automated protein design and current approaches, which mostly depend on human experience. This is due to restrictions on artificially created sequences which must comply with various factors such as *in silico* folding free-energy landscape [100,101] and shape complementarity [87].

Despite the rapidly growing number of known protein structures, the number of unique protein folds is converging, suggesting

that statistical learning based on existing structures leads to progress in design methods [102–104]. This statistical potential enables machine learning, especially deep-learning neural networks, to be used for accurate prediction and feature extraction [105].

Some of the commonly used resources for the structure of proteins and sequence are discussed in the following section.

4. PPI Databases

There are several known PPI databases, such as Uniprot [106], SWISS-MODEL [107], Negatome 2.0 [108], STRING [109], RCSB PDB [110], BioGRID [111], DIP [112], BIND [113], MINT [114], HPRD [115] and IntAct [116]. However, among these databases, some are not currently being maintained, such as BIND and HPRD, and are thus rarely used [117]. STRING, IntAct and MINT provide interaction scores from different sources to indicate their reliability. The Negatome 2.0 dataset comprises the manually curated interacting protein pairs from literature and analysed protein complexes from PDB, with scores of zero and one to indicate non-interacting and interacting pairs [108].

Computational methods often use the proteins' biological information, including protein sequences and protein structures. The biological characteristics and high-level structure of proteins are affected significantly by their primary structure. Therefore, one

may use the knowledge extracted from protein sequences to estimate the interaction likelihood between protein pairs [118]. Protein sequences can be obtained from the STRING [109], PDB [110], UniProt [106], PIR [119], SWISS-MODEL [107], and TrEMBL [120] databases. Information on higher-level protein structures can be acquired from PDB [121] and SCOP [122]. Dandekar et al. have asserted that proteins encoded by conserved gene pairs physically interact [123]. That basis is used, in genomic-based computational methods, for prediction. Genomic information can be found in The Candida Genome Database (CGD) [124].

In the following section, some of deep learning methods are briefly explained.

5. Deep Learning Models

Autoencoders, as illustrated in Fig. 11, are a type of unsupervised feedforward neural network reconstructing the output from the high-dimensional and possibly correlated input feature space. [125]. It consists of two parts, the encoder and the decoder. The encoder maps the input data into a low-dimensional and uncorrelated features space, called the latent layer, while the decoder reconstructs the input data from the latent layer. Autoencoders remove redundancies and correlations while extracting highly informative features [126,127].

Recurrent neural networks (RNNs) can capture contextual information when mapping input to output sequences. However, RNNs often suffer from vanishing gradients, limiting the context range they can access [128,129]. To address this problem, long-short term memory (LSTM) architecture was introduced [130], as illustrated in Fig. 12.

The LSTM architecture consists of recurrently connected memory blocks and corresponding control gates, the forget gate f_t , the input gate i_t , and the output gate o_t , which update and control the cell states [131]. The input and forget gates control current network memory and the flow of new information. Specifically, as new information flows into the network, the forget gate manages the information that needs to be removed from cell states, while the input gate controls the information that needs to be stored in cell states. Finally, the output gate determines the encoded information that needs to be forwarded as the input for the next step.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (1a)$$

$$h_t = o_t \otimes \tanh(S_t) \quad (1b)$$

where σ is the sigmoid activation function, W is the weight matrix, b is the bias vector, and $\otimes$ is the point-wise product. The initial operation is performed by the forget gate f_t , Eq. 1a, which determines whether the information should be kept or removed. The LSTM

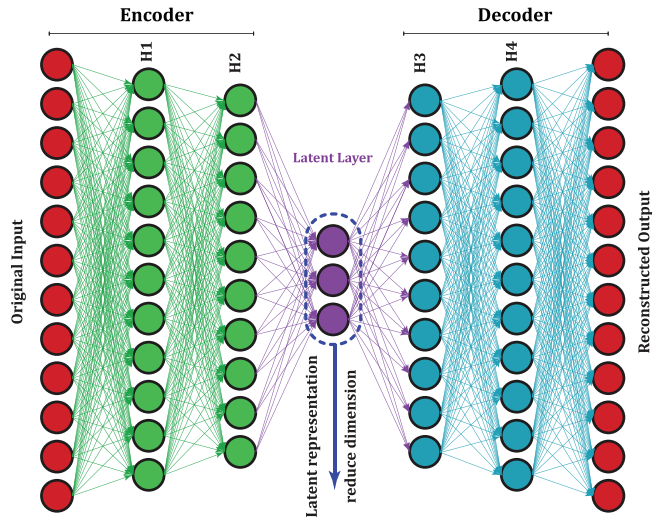


Fig. 11. Autoencoder architecture.

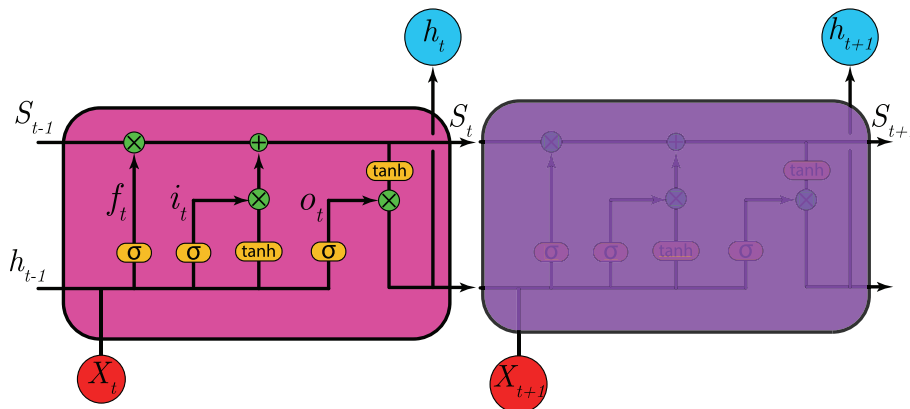


Fig. 12. LSTM architecture.

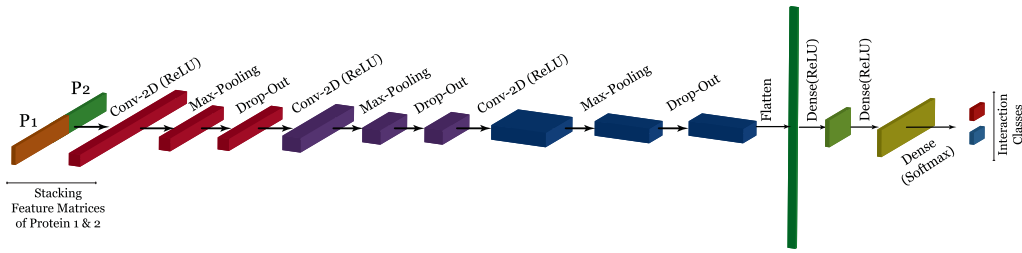


Fig. 13. convolutional neural network architecture. The input is the feature matrices of two proteins. The output predicts the interaction score between two proteins.

architecture contains a hidden state h_t , Eq. 1b that is formed by sequential information.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2a)$$

$$\hat{S}_t = \sigma(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (2b)$$

The next step involves storing the new input information in the cell state via the input gate i_t , Eq. 2a. Therefore, the cell state can be modified through candidate values $\hat{S}_t$, Eq. 2b, and Eq. 3a. Finally, the LSTM determines the output of each unit as Eq. 3b.

$$S_t = f_t \otimes S_{t-1} + i_t \otimes \hat{S}_t \quad (3a)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3b)$$

Despite the many advantages of LSTM, it is a computationally demanding architecture and slow to train.

The convolutional neural network (CNN) architecture is illustrated in Fig. 13. Its input is a matrix of the encoded representation of two proteins stacked in two columns. In this example, the CNN architecture comprises three 2-D convolutional layers and three dense layers. The convolutional and max-pooling layers reduce the size of the input tensor. The dropout layers are used to reduce overfitting and improve generalisation error [132]. The flatten layer reduces the dimensionality of the input. In addition, three densely connected layers reduce the features to the desired size. Finally, the output is obtained from a densely connected layer with the softmax activation function, classifying interactions into interacting and noninteracting pairs.

Most data used in deep learning can be readily represented in Euclidean space [133], where the convolution operation is properly defined [134]. However, when data cannot be represented on a reg-

ular grid due to the complex nature of their correlations [135,136], standard convolution cannot be directly applied to non-Euclidean geometries, limiting the applicability of CNNs [137,138].

However, the convolution theorem states that [137] convolution may be evaluated using Fourier transform. The Fourier transform is first performed for both the input and the filter. Then, both transformations are multiplied by the Hadamard product. Finally, the inverse Fourier transform of the Hadamard product is evaluated. If the Fourier transform is defined correctly, the convolution theorem remains valid under non-Euclidean geometry [137,134], allowing the application of CNNs to non-Euclidean geometries [138]. The spectral graph convolution in the non-Euclidean domain can be obtained by applying the Fourier transform graph and convolution theorem to both the input signal and the convolving filter [139].

Graph convolution extracts underlying local information by collecting node information in the local neighbourhood. Localisation can be achieved by expressing the filters in terms of Chebyshev polynomials of the first kind [140,141]. Fig. 14 illustrates a graph convolutional network (GCN) with stacked layers to extract multi-scale substructure features [138]. The propagation rule for the multi-layer GCN is given by:

$$\mathbf{f}_i^{l+1} = \sigma \left(\sum_{j=1}^p \Phi \mathbf{G}_{ij} \Phi^T \mathbf{f}_j^l \right), \quad i = 1, \dots, q, \quad j = 1, \dots, p \quad (4)$$

Where σ is the nonlinear activation function (l) and $\hat{\mathbf{G}} = \text{diag}(g_o(\lambda))$. The number of features is denoted by (q), and the number of assets is denoted by (p).

Locality is assumed for all nodes in GCN. As the size of the neighbourhood increases, algorithmic time and space complexity also increase [142]. This issue violates the purpose of using deep

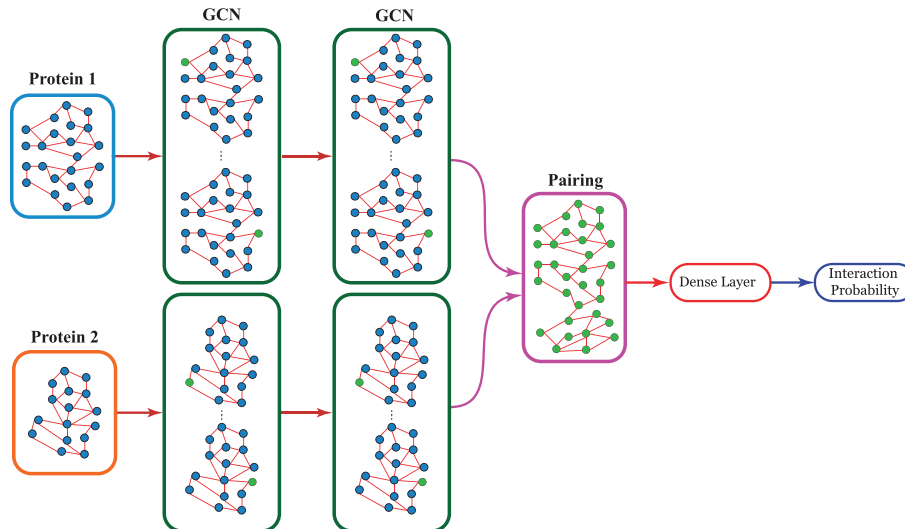


Fig. 14. graph convolutional network architecture for PPI prediction.

models. While few studies have addressed this issue (e.g., skip connection-based models), how to construct a deep architecture that can better adaptively exploit deeper structural graph patterns is still an open challenge [136].

Generative models aim to model the underlying distribution of the data, enabling the generation of new samples with comparable properties to those on which the model was trained [143,144]. Numerous generative models have been developed on the basis of deep neural networks, such as Variational Autoencoder (VAE) [145–147], Generative adversarial Network (GAN) [148], and deep autoregressive models [149–151].

In their original form, GAN algorithms are composed of two components, namely, generator and discriminator, with the generator producing synthetic data while the discriminator evaluates the discrepancy between the generated data and the real data. Each network attempts to improve its performance until an equilibrium is reached, where the discriminator is unable to detect the fake samples and the generator fails to produce better samples [148,152–154].

As illustrated in Fig. 15, given a data distribution, $\mathbf{x} \sim p_x$, $\mathbf{x} \in \mathcal{X}$, the generator learns the distribution p_G which maps the latent

variable drawn from a prior distribution $\mathbf{z} \sim p_z$, $\mathbf{z} \in \mathcal{Z}$ to the sample space as $G: \mathcal{Z} \rightarrow \mathcal{X}$, while the discriminator is trained to distinguish between fake and real samples via a score $D: \mathcal{X} \rightarrow \mathbb{R}$ [148].

The VAE architecture, as shown in Fig. 16, is a class of generative models based on variational Bayesian inference with multivariate prior distribution [155–157], initially introduced in [145]. The VAEs comprise two linked models that are individually parameterised, namely the encoder or recognition model and the decoder or generative model. Unlike autoencoders, in which the encoder compresses the input features into real-valued latent features, the encoder in a VAE stochastically maps the observed variables' x -space to a probabilistic latent z -space (latent variable) [158].

Fractionally strided convolutions, also known as transposed convolutions, perform a reverse spatial transformation by switching the forward and backward pass [159]. Fractionally strided convolutions may allow for recovering the shape of the initial feature map but do not guarantee retrieving the input itself [159]. This allows the network to learn its own spatial downsampling and upsampling. An extension of the 2-D GAN framework, called conditional GAN, has been proposed in [160] that applies conditions on class labels for both the generator and the discriminator networks. Multimodal data generation is better represented using conditional GANs.

Both generator and discriminator are trained based on an additional information placed as condition $\mathbf{y}$ in the input layer, as depicted in Fig. 17. The adversarial training framework allows for flexible-joint hidden representations composed from input noise $p_z(\mathbf{z})$ and $\mathbf{y}$ in the generator [160]. The fake samples are generated as $G(\mathbf{x}, \mathbf{z}) = \mathbf{x}^* | \mathbf{y}$ ($\mathbf{x}^*$ is synthetic sample given $\mathbf{y}$ as a condition) aiming to resemble real samples as well as possible. The discriminator receives real samples with labels $(\mathbf{x}, \mathbf{y})$ and fake samples from generator $(\mathbf{x}^* | \mathbf{y}, \mathbf{y})$. The discriminator outputs a single probability through a sigmoid activation function (σ) indicating its decision on fake and real inputs.

The following section represents PPI prediction methods.

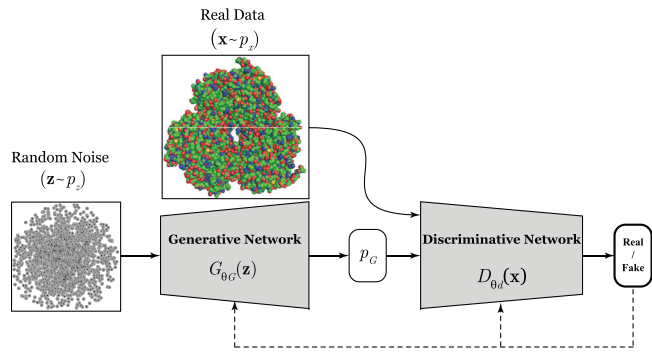


Fig. 15. GAN architecture.

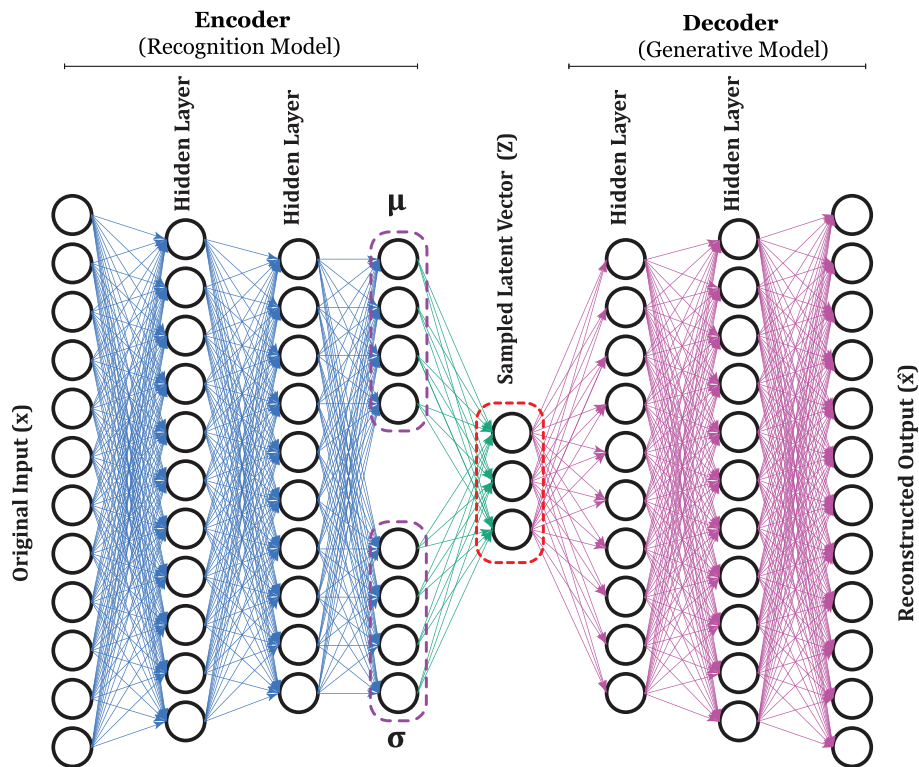


Fig. 16. variational autoencoder architecture.

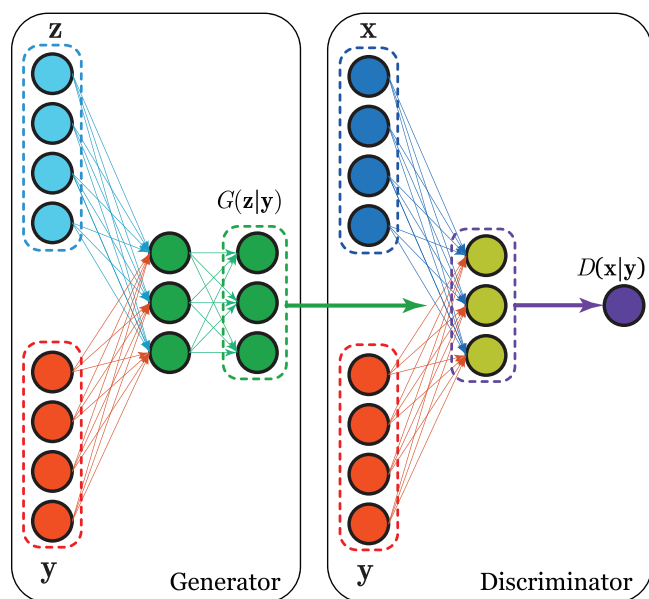


Fig. 17. the conditional GAN architecture.

6. PPI Prediction Methods

High-throughput experimental methods have produced PPIs at an ever greater rate, but these acquired data are noisy with both false positives and false negatives. For instance, mass spectrometry methods may not be able to detect transient or weak interactions [161–165]. The noise levels of different PPI-identifying technologies are studied in [161], showing that high-throughput methods such as two-hybrid system, mass spectrometry, protein chip and phage display have relatively high noise levels.

From a practical perspective, studying PPIs provides the foundation for diagnostic and therapeutic medical applications, thus facilitating the design of novel drugs [117,166–168]. The development of computational methods for the PPI prediction problem is motivated by such shortcomings.

Recent advances in computational modelling methods have brought about exceptional findings in protein design, including enzymes [76,77,169], the development of new therapies [170,171], biosensors [172], and small-molecule binders [82]. However, these methods are mainly suited to modifying naturally found proteins [173]. On the other hand, creating proteins *de novo* provides full control over their structure and function [92,174]. Hence, a new objective is to discover new, non-native folds or structural elements as building blocks for novel proteins [173].

Computational protein design mainly aims to automate the fabrication of proteins with specific structural and functional properties [9,73]. This field has gained traction in the past two decades, such as in the design of novel 3-D folds [74], protein complexes [87], and enzymes [169]. Even though these methods have shown great achievements, current approaches are unreliable as initial designs frequently fail, entailing multiple trial-and-error cycles [175,176]. Since these approaches are highly dependent on the accuracy of complex energy functions for protein physics and the performance of sampling algorithms for jointly exploring the protein sequence, it is difficult to determine the source of the poor reliability [177–180]. Nevertheless, computational methods have facilitated the generation of synthetic protein domains which mimic natural folds using sequences unlike those in nature [181–183].

Quick computational testing of many possible outcomes, potentially narrowing the set of necessary experiments, would ultimately

save time. Among the computational methods addressing the PPI prediction problem, some use extracted features as inputs to learn the model [184], while others extract new protein information [185–187]. These methods are further explained in Section 6.5. The information extracted from a tertiary structure of proteins may be used to predict PPI. There exist several experimental techniques for determining a tertiary structure of proteins, including X-ray crystallography and NMR spectroscopy [188]. It is suggested in [189] that locations of protein–protein binding sites are engraved in the proteins' structures. Although experimentally determined 3-D protein structures may facilitate the detection of interaction sites and the understanding of protein functions, experimental biological methods are laborious and time-consuming, and consequently, the geometries of only a small fraction of known proteins have been determined [189–194]. To address this shortcoming, various studies use deep learning to predict, from protein structure and other protein features, potential PPI [185,186,195–197]. Some of these methods are discussed further in Section 6.3.

6.1. PPI Site Prediction

Identifying PPI sites is crucial for understanding the mechanisms of disease and for novel drug design. PPI binding sites consist of amino acid residues forming chemical bonds with a part of another molecule [40]. Identifying interaction domains in sequences helps in understanding cell regulatory mechanisms, locating drug targets and predicting protein functions [198]. Yuan et al. have addressed PPI site prediction as a graph node classification problem, modelling proteins as undirected graphs. They developed GraphPPIs [199] to predict PPI sites.

PPI site predictions are roughly categorised into three categories: protein–protein docking, structure-based, and sequence-based methods. Docking methods aim to generate structures of the resulting protein complex [200], as proposed in [201], by defining a scoring function for novel shape complementarity at the initial docking stage. Some of the recent sequence-based methods for predicting protein–protein interaction sites include: attention-based convolutional neural networks [197], simplified LSTM [191], the DeepPPIISP method which uses a combination of local contextual and global sequence features [196], CNN with a residue binding propensity to address data imbalance [202], and the DELPHI method which comprises an ensemble structure as a combination of CNN and recurrent neural network (RNN) [203]. CNN and LSTM architectures are illustrated in Fig. 13 and 12 respectively.

Structure-based methods for PPI prediction are addressed in the next subsection.

6.2. Structure-based PPI Prediction

Proteins adopt complex 3-D structures to perform biological functions via physical contact between effectors and regulators. The effectors may be characterised as the molecules that activate or suppress the regulator's function and alter gene expression as a result [204,205]. Therefore, predicting which residues are involved in PPIs may help structure-based drug discovery, improve the accuracy of protein–protein docking, and obtain richer annotation of protein function [206,207]. A protein may interact with multiple partners over different or overlapping sections of its surface. These interactions may occur at different times or, when the interaction site is large, simultaneously.

Structure-based methods exploit information such as similarity in protein structure to predict PPIs [208]. For instance, two proteins, A' and B', structured similarly to two interacting proteins A and B, respectively, can be assumed also interact with each other [209]. Structure-based techniques often employ empirical scoring functions, physics-based methods, knowledge-based approaches,

or quantitative structure–activity relationship methods to determine both the binding affinities and structural orientations of PPIs [210]. Protein–protein docking techniques can model the orientation of two interacting proteins and their binding affinity and identify key residues in PPIs [210].

Docking-based methods use the structures of individual proteins to predict the structure of the complex. Generally, the only information available is the structure of these individual proteins. The docking method includes two steps. Firstly, the binding orientations of two interacting proteins are identified. Secondly, the binding free energy between the interacting proteins are estimated [211,212]. A global search is conducted by holding the target protein (receptor) stationary while moving the ligand around it. After modelling all possible orientations, the interactions between the two proteins are determined [210]. The global search method demands an unlimited number of translations and rotations, making it a computationally expensive approach. To address this issue, a fast Fourier transform (FFT) approximation has been used in [213].

The local docking technique may improve solutions found by the global docking approach. In the global docking scenario, the sampling starts from a random point, whereas the local techniques assume a known starting point (binding mode) and restrict the sampling search around it [214,215]. The ZDOCK server is among the commonly adopted docking resources which employ FFT-based global search [216]. RosettaDock is a local protein–protein docking algorithm based on a Monte Carlo search. It allows for user-defined initial poses or random orientation of the two proteins. RosettaDock aims to find the system with the lowest energy, initially through a low-resolution optimisation, followed by a high-resolution refinement [217]. Finally, the docking score is estimated by an all-atom energy function [218–220]. The structural features and physicochemical properties are used for showing the models of unknown PPIs. MEGADOCK is a template-free docking methods [221] identify the most promising interactions from a large set of potential interaction sites by assessing the unbound protein components. This method investigates a protein docking approach based on the tertiary structures of the target proteins and physicochemical properties. The docking calculation is accelerated using a novel scoring function called the real Pairwise Shape Complementarity (rPSC) score. Although docking methods have proven successful for some proteins, they fail to deliver the same performance for proteins that sustain conformational changes during interaction [222].

Homologous proteins, i.e. proteins exhibiting similarity through common ancestors sequences [223], are apt to adopt the same binding interfaces [224]. However, the PPI interfaces may be structurally similar, even though their global structures differ [225]. Template-based docking techniques predict PPIs by comparing a protein–protein complex under examination against templates, i.e. other, experimentally determined, protein–protein complex structures [168,211,226–229]. In general, these techniques operate in five steps i) developing the template library, (ii) selecting the target set, (iii) searching for the similarities between target and template, (iv) refinement and (v) scoring. Developing the template library is the most crucial step. In the last decade, the number of experimentally determined structures has grown exponentially, thus improving the performance of template-based techniques [210].

The search for similarity between target and template is performed globally and locally, and can be conducted through sequence alignment [230], structural alignment, and threading [231–234]. Alignments can be obtained from sequences, structures, or feature information from both sequences and structures. The structure framework in the aligned regions of the template

with the highest alignment score is selected as the basis of the target protein structure [222,235] (See Fig. 18).

The next subsection will review computational methods predicting PPI using protein 3-D structure.

6.3. Structure-based PPI Prediction Using Computational Methods

In order to address the high-dimensionality problem of protein structure, several dimensional reduction techniques have been applied, such as random forests [237] and the support vector machine (SVM) and its derivatives [238,239]. Northey et al. introduced a multi-layer perception network (MLP)-based method called IntPred [240] to predict interaction by splitting proteins into a group of patches that integrates 3-D structural information into a feature set.

In recent years, many graph convolutional network (GCN) variants (see Fig. 14) [134,241] have been successfully employed in a variety of tasks with graph-structured data [242], such as protein solubility prediction [243], genomic analysis [244] and drug discovery [245]. A GCN-based approach is proposed in [246] to acquire positional information in PPIs. Their representation method combines the information from the amino acid sequence and the protein positions. In order to determine the amino acids of an interacting protein interface, Fout et al. integrated 3-D structures into a GCN [247]. To accurately predict interactions between query proteins entirely from 3-D structural data, Baranwal et al. proposed a GCN-based mutual attention classifier called Struc-t2Graph [248]. The generative model proposed in [177] is a graph-based model which captures the joint distribution of the full protein sequence, which is founded on long-range interactions resulting from the protein structure. A multimodal approach based on LSTM is proposed in [187], which predicts PPI by integrating structural and sequential information about proteins into the input feature set.

The advantages and disadvantages of these methods are listed in Table 5. Moreover, Table 6 lists the datasets used by each method.

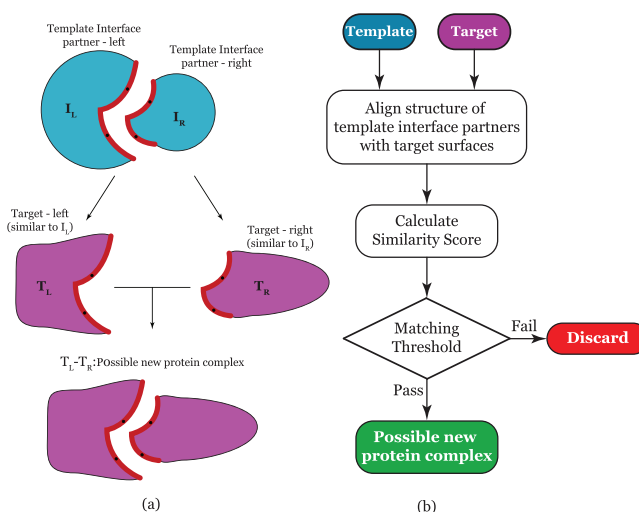


Fig. 18. schematic illustration of PRISM algorithm as an example of template-based docking method for PPI prediction [236] (a) If the template interface on complementary partners (I_L and I_R) are similar to any two targets surfaces (T_L and T_R), these two targets may interact and create a protein complex. The black points illustrate hot spot residues. (b) The algorithm flowchart includes the template data set and the target data set. The surface of each partner of the template interface is aligned with the target surfaces. If the matching threshold for hot spot residues passes, the target proteins may form an interacting pair [236,224].

Table 5

Summary of advantages and disadvantages of structure-based deep learning methods for PPI prediction.

Framework	Description	Advantage	Disadvantage
GCN-based [2017] [247]	This study proposed a pairwise classification architecture in which one or more graph convolution layers process the neighbourhood of a residue in each protein. Then, the representation of two residues is paired and passed through a dense layer for classification. This study analysed several GCN-based methods, concluding that neighbourhood-based convolution methods outperform diffusion-based convolution and SVM-based methods.	The proposed convolution operators and obtained features may be helpful for other applications, including protein function, catalytic and other functional residues, and protein interactions with DNA and RNA.	The accuracy of this approach is examined based on a limited number of labelled training examples.
IntPred [2018] [240]	This method uses a random forest to predict protein–protein interface sites at both the surface patch and residue levels.	The performance of a binary classifier can be evaluated using different measurements, such as the Matthews' correlation coefficient (MCC), sensitivity, precision, and specificity [240]. IntPred outperformed the methods ProMate [189], PIER [249], PINUP [250], and meta-PPISP [251], but not SPPIDER [252], based on MCC.	The performance of this method depends on the application. For instance, IntPred was better suited to cases in which false positives are less well tolerated than false negatives.
Graph-based generative model [2019] [177]	This method uses a graph transformer model for designing protein sequences given graph representations of 3-D protein structures, leveraging the spatial locality of dependencies in molecular structures.	This method uses a self-attention mechanism to capture higher-order, interaction-based dependencies between sequence and structure. The graph-based model offers computational efficiency due to the representation of long-range sequence dependencies by short-range sequence dependencies in 3-D space [253–255]. Additionally, they achieved linear computational scaling concerning the sequence length and representational flexibility for coarse and fine-grained structure descriptions.	The evaluation dataset only contained chains up to a length of 500, limiting the applicability of this approach.
Struct2Graph [2020] [248]	In this method, graph embeddings of each protein are obtained using an assigned GCN. Next, relevant geometric features associated with query protein pairs are extracted using a mutual attention network. Finally, a feedforward neural network performs a binary classification between interacting and noninteracting pairs.	Struct2Graph only uses 3-D structural information to predict the PPI. They have reported state-of-the-art performance on both balanced and unbalanced datasets.	Limited availability of 3-D structural information may restrict the applicability of this method.
LSTM-based [2020] [187]	The proposed method integrates the 3-D structure and sequence-based information of proteins to predict PPIs. The 3-D coordinate information, hydropathy index, isoelectric point, and amino acid charges of each protein are fed into a pre-trained ResNet50 model to extract features from these attributes. A stacked autoencoder obtains the compact form of encoded proteins using autocovariance and conjoint triad. The structural features from ResNet50 are passed through LSTM and concatenated with features from the stacked autoencoder. The merged features are then fed into the classifier to predict protein pair labels.	This method performs well despite being trained on a low number of instances.	Limited availability of 3-D structural information may restrict the applicability of this method. Additionally, LSTM models are computationally demanding and slow.

The sequence-based methods for PPI prediction are reviewed in the next section.

6.4. Sequence-based PPI Prediction

Traditional methods often analyse protein sequences based on multiple sequence alignments. This leads to a simple inference of functional and structural constraints from sequence data [265]. While protein design and engineering have benefited from evolutionary information of alignments [266–268], adding distant proteins will induce large and unreliable alignments [269], restricting the diversity of sequences.

Unlike docking and structure-based methods, sequence-based methods do not require structural data and instead leverage the abundance of existing protein sequence data from sequencing technology, especially since the introduction of metagenomics [106,270,271]. One may predict PPIs from amino acid sequence similarity in the known interactions, depending on interactions already identified in one species to infer interaction in different species [24,272]. The sequence-based method hence focuses on primary structure, disregarding the protein's 3-D shape [273].

In domain-based technique, specific sequences in the protein structure are represented by conserved domains. Conserved Domains may be defined by local multiple sequence alignments, including a wide range of organisms to display sequence regions

Table 6

Datasets of structure-based methods.

Framework	Data Processing
GCN-based [2017] [247]	Version 5 of the docking benchmark dataset was used by this study [256], comprising a selected subset of structures generated from X-ray crystallography or nuclear magnetic resonance experiments and containing the atomic coordinates of each amino acid residue in the protein from the Protein Data Bank (PDB). Proteins with 29 to 1,979 residues are included. Since proteins may change their shape upon binding, the features are computed from the unbound form of the protein in the complex. The labels are acquired from the structure of the proteins in the complex.
IntPred [2018] [240]	The training dataset comprised 58,397 biological units from protein, interfaces, structures and assemblies (PISA), including transient and obligate interfaces [257]. Structures with a resolution below 3 Å or Rfactor above 30%, Viral capsids, NMR entries and proteins with fewer than 30 amino acids are removed. Any structure with more than one chain is retained, resulting in 25,876 structures constructed from 87,738 chains. The chains were clustered at 25% sequence similarity using PISCES [258] to remove redundancy. The final training set contained 4,345 chains. For the test set, no clustering was performed, resulting in 4,204 chains. The NOXclass [259] is used to construct a dataset of obligate and transient interfaces. This method predicts protein interactions as either obligate or non-obligate (transient) with or without crystal packing contacts. The dataset was obtained from the CATH (version.4.2) [260]. The training, validation, and testing sets were divided into 80/10/10 sets by randomly assigning their CATH topology classifications (CAT code). The resulting dataset included 18,024 chains in the training set, 608 in the validation set, and 1,120 in the test set, with zero CAT overlap.
Graph-based generative model [2019] [177]	The database was generated based only on direct/physical protein interactions. Therefore, IntAct [116] and STRING [261] were selected, and only concordant matches between these two databases were chosen as true interactions. The organisms included in this dataset were <i>S.cerevisiae</i> , <i>H.sapiens</i> , <i>E.coli</i> , <i>C.elegans</i> , and <i>S. aureus</i> , resulting in 427,503 pairs from IntAct and 852,327 pairs from STRING. Only "direct association/interactions" from IntAct and "binding" from STRING were regarded as physical interactions. Only extracting concordant, physical interaction data reduced the interactions to 12,676 pairs for IntAct and 446,548 pairs for STRING. Negative PPI was retrieved from [262]. Structure information for this method was acquired from PDB files, which reduced the total number of pairs to 117,933 (5,580 positive and 112,353 negative). All proteins were matched with PDB files using UniProt accession numbers (UniProt Acc) and mapped PDB files [263]. Finally, PDB files were curated based on the length of their chain ID and the highest resolution within each PDB file, resulting in 5,580 negative pairs for a balanced dataset.
Struct2Graph [2020] [248]	This study used two PPI datasets, Pan's PPI dataset [264] and <i>S. cerevisiae</i> PPI data obtained from the Database of Interacting Proteins (DIP; version 20160731; see Stacked Autoencoder (SAE) and DeepPPI for further details). Structure information was only available for 10,359 protein sequences in the Pan's PPI dataset and 1,308 proteins in the <i>S. cerevisiae</i> PPI dataset. Therefore, Pan's PPI dataset contained 25,493 pairs (18,025 positive and 7468 negative), while the <i>S. cerevisiae</i> PPI dataset contained 4,314 positive and 6,265 negative pairs.
LSTM-based [2020] [187]	

containing the same, or comparable, patterns of amino acids [274]. This idea can be used to predict the subcellular location of the protein and the class and subclass of the enzyme, to find functional interactions, and to identify the membrane protein [275]. Computational approaches have been developed to predict PPIs based on the information this technique provides [275]. A domain-based method to estimate the interaction map of *E.coli*, for example, is developed in [276]. Another domain-based method is introduced by Kim et al. [277] which estimates the probability of the interaction between interacted domains. Using the relevant vector machine (RVR) and domain features with support vector regression (SVR), Kamada et al. identified PPIs [278]. DomainGA is a multi-parameter optimisation approach which is developed to predict the score of PPI [279].

The ortholog-based techniques also use similarities between amino acid sequences [210]. The annotations to a functionally determined protein sequence are transferred to similar parts of a target sequence. This work relies on databases of annotated proteins to construct the homologous model of the studied protein [280]. Significant sequence similarities may be shared among multiple proteins from an organism in systems in diverse organisms. Thus, if a significant similarity is found between an input protein and an annotated protein (with known functions), the input protein may be hypothesised to possess similar properties or functions. In order to identify these functions, paralog and ortholog approaches are employed. Orthologs are homologous genes that evolved by vertical descent from a single ancestral gene; In contrast, paralogs evolved by duplication [223,281]. For instance, the orthologs of two interacting proteins, A and B, can interact similarly in different species [210].

Computational methods predicting PPI using protein sequence are addressed in the next subsection.

6.5. Sequence-based PPI Prediction Using Computational Methods

The Interface Weighted RAPtor (iWRAP) integrates a boosting classifier with a novel linear programming formulation for interface alignment to predict interacting proteins encoded by the entire yeast genome. The interface profiles are constructed using SCOPPI [282], based on the sequence and structural similarity of the interface [283]. The Universal In Silico Predictor of Protein-protein Interactions (UNISPPi) uses the primary structure to classify protein pairs as interacting or non-interacting [284]. A matrix-based representation of protein sequence coupled with the SVM algorithm is proposed in [285], using the order of primary structure and its dipeptide information. The sequence-based methods may be split into two distinctive techniques: domain-based and ortholog-based [273]. PPI prediction is conducted based on sequences in [286] by defining units of three adjacent amino acids and measuring the frequency of those units in a protein sequence. Other methods such as amino acid index distribution [287], conjoint triad method (CT) [286] and autocovariance (AC) [288] are developed to extract features such as locations of amino acids, frequencies and physicochemical properties, with the aim of representing a protein sequence.

An SVM based approach, ACT-SVM, is developed in [289] to extract features from protein sequences as the input vector for the classifier. A sequence-based human PPI prediction is developed in [18] based on a Stacked autoencoder (SAE). Another sequence-based PPI prediction approach is introduced in [165] called D-

Table 7

Summary of advantages and disadvantages of sequence-based Deep Learning methods regarding PPI prediction.

Framework	Description	Advantage	Disadvantage
SVM-conjoint triad [2007] [286]	Each protein sequence was represented in this study by a vector of amino acid features. The model was developed based on a support vector machine (SVM) integrated with a kernel function and a conjoint triad feature for describing amino acids. This method mapped different types of PPI networks using only sequence information, which could be applied to explore networks for any newly discovered protein with unknown biological relationships. They suggested that methods without local environments for amino acids are often unreliable, so a conjoint triad method was used.	The 20 standard amino acids were clustered into several classes based on their dipoles and side chain volumes to achieve dimensionality reduction of the vector space. This method might predict PPI networks created by pairwise PPIs.	The limited available information on protein pairs restricts the applicability of this method. Additionally, it mainly considers the properties of two nearby amino acids, overlooking long-range interactions.
SVM-autocovariance [2008] [288]	This method combined a new feature representation using autocovariance (AC) and a support vector machine (SVM). AC considers the interactions between more distant amino acids in the protein sequence, specifically long-range interactions. This is an improvement over the method proposed in [286]. This model was evaluated using an independent dataset of 11,474 yeast PPIs.	The conjoint triad (CT) method only considered the attributes of an amino acid and its two neighbouring amino acids [286], while long-range interactions are accounted for by the AC method. In this study, AC variables represented information on interactions between one amino acid and its 30 neighbouring amino acids in the protein sequence.	The model achieved a low prediction accuracy of 58.42% in a negative dataset created using the Prcp method [286].
UNISPPi [2013] [284]	This method used a decision tree model, predicting PPIs using only 20 amino acid frequency combinations from interacting and noninteracting proteins as learning features. This study indicated that asparagine, cysteine, and isoleucine frequencies are important features for discerning between interacting and noninteracting protein pairs.	This method was scalable due to using a limited number of attributes. Moreover, this method was based on experimentally validated instances from various species, covering many species.	Instances with a classification score of 0.50 were classified as neither PPIs nor non-PPIs, limiting the applicability of this method. Additionally, the obtained accuracy of 79.4% for interacting and 72.6% for noninteracting pairs are relatively low.
SVM-based method [2015] [285]	PPI prediction was addressed by integrating a support vector machine (SVM) and a novel matrix-based representation of the sequence order and dipeptide information of the primary protein sequence, extracting more information than amino acid dipeptide composition. The SVM classified the interaction between protein pairs using these feature vectors.	This method extracts more information hidden in protein primary sequences than amino acid dipeptide composition.	SVM algorithms performed relatively poorly with noisy data and are unsuitable for large datasets since training time may increase significantly [303]. Moreover, finding a proper kernel function was difficult.
DeepPPI [2017] [292]	The DeepPPI method used a deep neural network architecture network for each protein to extract high-level discriminative features from common protein descriptors. The interaction between two proteins was determined using the one-hot encoding label. This method comprises two different architectures: DeepPPI-Sep, which uses two separate networks as input for each protein, and DeepPPI-Con, which directly links two proteins in a single network.	This method can capture informative features of protein pairs by a layer-wise abstraction. In addition, DeepPPI can automatically learn an internal distributed feature representation from the data.	The accuracy of DeepPPI for All Human/Yeast dataset are relatively low, and the accuracy of methods proposed in [304] exceeds that of DeepPPI.
Stacked Autoencoder (SAE) [2017] [18]	This method used a stacked autoencoder to predict PPI. The feature extraction from protein sequences was performed using autocovariance (AC) and the conjoint triad (CT).	SAE can learn hidden interaction features of protein sequences.	They used a synthetic negative interaction dataset, and the accuracy of this model for negative interactions is relatively low.
DPPI [2018] [298]	This method performed sequence-based PPI prediction using a deep, Siamese-like convolutional neural network combined with random projection and data augmentation. This method captured the composition information, sequential order of amino acids, and co-occurrence of interacting sequence motifs in a protein pair. Each protein was characterised as a probabilistic sequence profile generated by PASI-BLAST. The patterns in each sequence were identified using the convolutional module, comprising multiple layers. The representations learned by the convolutional module were projected to two different spaces using the random projection module, allowing DPPI to explore the combination of protein motifs.	DPPI addresses interactions for both homodimeric and heterodimeric proteins. Moreover, this method could model binding affinities.	This method yields lower PPI prediction accuracy on the <i>S.cerevisiae</i> core dataset from PIPR based on 5-fold cross-validation compared to PIPR [299] and DeepTrio [302].

Table 7 (continued)

Framework	Description	Advantage	Disadvantage
PIPR [2019] [299]	This study proposed an end-to-end framework for PPI prediction based on amino acid sequences using a deep residual recurrent convolutional neural network in the Siamese architecture. This method leveraged an automatic multi-granular feature selection to capture local significant and sequential features from protein sequences.	The Siamese-based learning architecture captured the mutual influence of protein pairs and allowed for generalising to address different PPI prediction tasks without needing predefined features.	RCNN was built using bidirectional gated recurrent units (bidirectional-GRU). However, GRUs suffer from slow convergence and low learning efficiency [305].
S-VGAE [2020] [300]	This model proposed a signed variational graph autoencoder (S-VGAE) that combined sequence information and graph structure. In this method, the PPI network was regarded as an undirected graph. This framework comprised three parts. First, coding the raw protein sequences. Second, the S-VGAE model extracted vector embedding for each protein with sequence information and graph structure. Finally, a simple three-layer softmax classifier. This model was inspired by the variational graph autoencoder (VGAE) [306] that uses latent variables to learn interpretable representations for undirected graphs.	In this method, the cost function was modified only to consider highly confident interactions, making it more robust to noise.	This model used the conjoint triad (CT) method [286] to encode amino acids. However, CT does not account for long-range interactions in the protein sequence.
ACT-SVM [2020] [289]	This method performed feature extraction on the protein sequence to obtain a vector, composition, and transition descriptor and integrated them into a vector. Then, the feature vector was fed into the SVM classifier. The performance of this method was evaluated using 5-fold, 8-fold, and 10-fold cross-validation on H. pylori and human datasets.	They have observed that SVM method outperforms K-Nearest Neighbour (KNN), ANN, RFM, Naive Bayes, Logistic Regression, s for the H. pylori protein pairs.	Finding the proper kernel and hyperparameters was challenging, and training time for SVM classifiers increases with dataset size [307].
D-SCRIPT [2021] [165]	Deep sequence contact residue interaction prediction transfer (D-script) is an interpretable deep learning method generating structurally informative features given protein sequences using a pre-trained language model from [290]. This method used projection modules to reduce the dimension of features, including the residue-contact map of the protein. Finally, the interaction probability was predicted based on the contact maps.	D-SCRIPT generalised to new species considering the sparsity of training data for most model organisms (i.e., it was relatively accurate for cross-species PPI prediction).	Despite its performance for cross-species PPI prediction, D-SCRIPT underperformed on within-species evaluations. The training dataset only included proteins with 50–800 amino acids, limiting the applicability of this method.
SPNet [2021] [9]	The Siamese pyramid network (SPNet) architecture used self-binding and folding amino acid sequences to predict the binding probability for two proteins based solely on their amino acid sequences. Subsequent screening through potential candidates was performed based on binding probabilities.	This architecture consisted of a multilevel pyramid feature structure encompassing various PPI mechanisms to reduce gradient explosion and disappearance, a multilevel Siamese neural network with an attention mechanism, and a multilevel, trainable binding probability prediction network.	
BiLSTM-RF [2021] [291]	The BiLSTM-RF model extracted features of protein pairs in the human database. BiLSTM comprises forward and backwards LSTMs and is capable of bidirectional encoding (i.e., encoding front-to-back and back-to-front information). A random forest classifier (RF) was built with 100 trees and used a voting strategy to integrate these results to predict the interaction.	BiLSTM extracted the sequence and position of the biological information in the protein sequence.	LSTM models are computationally demanding and slow. Moreover, a large number of trees in the random forest leads to a longer training time.
Heterogeneous Network [2021] [296]	PPI prediction was performed using a computational sequence and network representation learning-based model. Local features were extracted from the protein sequence using the <i>k</i> -mer method (<i>k</i> = 3), while global features were extracted from the heterogeneous network. The latter captured network structure and obtained potential linked information. This method integrated local features with global features to represent protein nodes.	The protein node contained protein attribute and network structure information by integrating local and global features.	Model accuracy is relatively low compared to other deep learning methods such as DPPI [298].

(continued on next page)

Table 7 (continued)

Framework	Description	Advantage	Disadvantage
OR-RCNN [2021] [301]	This method was called ordinal regression and recurrent convolutional neural network (OR-RCNN), which predicted PPIs based on their confidence score. The architecture comprised two recurrent convolutional neural networks (RCNNs) encoders, which shared the same parameters, to extract robust local features and sequential information from protein pairs. Then, one novel embedding vector was obtained by element-wise multiplication of the two embedding vectors from RCNNs. The second part of the architecture performed an ordinal regression model via multiple sub-classifiers that use the ordinal information behind the confidence score. Finally, the confidence score determined the existence of PPI with a threshold.	This method offered better accuracy compared to some existing models, such as autocovariance [288] and composition transition distribution (CTD) descriptor [308] for feature description, and random forest (RF) [309], extreme gradient boosting (XGBoost) [310], and support vector machine (SVM) [311] for the prediction.	The RCNN was built using bidirectional gated recurrent units (bidirectional-GRU). However, GRUs suffer from slow convergence and low learning efficiency [305].
DeepTrio [2022] [302]	The DeepTrio method used a deep-learning framework based on a mask multiscale CNN architecture that performed binary PPI prediction by capturing multiscale contextual information of protein sequences using multiple parallel filters. This method used a single-protein class, allowing it to distinguish relative and intrinsic properties. This method was also made available as an online tool to address cross-platform usage and dependency-related issues.	DeepTrio is available both online and offline.	DeepTrio yields lower PPI prediction accuracy on the <i>S.cerevisiae</i> core dataset from PIPR based on 5-fold cross-validation compared to PIPR [299]. DeepTrio achieves lower PPI prediction accuracy on the <i>S.cerevisiae</i> core dataset from DeepFE-PPI based on 5-fold cross-validation compared to DeepFE-PPI [312].

SCRIPT (Deep Sequence Contact Residue Interaction Prediction Transfer), which models protein structure using a pre-trained language model from [290]. A novel deep learning approach called Siamese Pyramid Network (SPNet) architecture is proposed in [9], which predicts the binding probability of two proteins based on their amino acid sequences. This method is employed to discover the proteins that potentially bind with the 2019-nCov spike, in order to find future vaccines.

Learning protein pair representation is tackled by deep learning methods such as BiLSTM-RF, which uses LSTM to extract the features of protein pair sequences and a random forest classifier [291], and DeepPPI [292] that uses a separate network for each protein and learns high-level features from raw protein features. The EnsDNN approach extracts the interaction information of proteins from amino acid sequences, using AC descriptor [293], local descriptor (LD) [294,295] and multi-scale continuous and discontinuous local descriptor (MCD) [15,237]. A heterogeneous network for PPI prediction is presented in [296] which uses the concatenation of local and global features to present protein node. The local features are extracted from protein sequence by the k -mer method ($k = 3$)¹, while the global features are extracted from the heterogeneous network, and heterogeneous networks by LINE (Large-scale Information Network Embedding), respectively, and random forests to classify and predict potential protein pairs.

DPPI [298] performs sequence-based PPI prediction using a deep, Siamese-like convolutional neural network combined with random projection and data augmentation. This method captures the composition information, sequential order of amino acids, and co-occurrence of interacting sequence motifs in a protein pair. The PIPR method [299] predicts PPI by integrating a deep residual recurrent convolutional neural network (residual-RCNN) in the Siamese architecture, leveraging both robust local features and contextualised information on the protein sequence. The signed variational graph auto-encoder (S-VGAE) graph-based method [300] considers the PPI network as an undirected graph, combining sequence informa-

tion and graph structure to predict PPI. A deep learning method called OR-RCNN is developed to predict PPI [301], which is composed of two recurrent convolutional neural networks (RCNNs) to extract local features and sequential information from the protein pairs and ordinal regression to construct multiple sub-classifiers. The sequence-based PPI prediction approach DeepTrio [302] uses mask multiple parallel convolutional neural networks.

Table 7 represents further analysis of the sequence-based methods. Additionally, the datasets of each of these methods are reported in Table 8.

Some of the methods addressing protein design problems, including protein function, sequence, and structure, are discussed in the following section.

7. Protein Design

Most protein design problems require profound knowledge and subjective expertise to analyse obstacles and obtain optimal design strategy. However, with the emergence of deep neural networks, computational capacity and the available historical data, new computational methods have shown advantageous in many cases, such as RNNs' successful application in generating SMILES (Simplified Molecular Input Line Entry System) sequences for *de novo drug discovery* [328,329] and *optimising the RNN output to obtain specific properties through transfer learning and fine-tuning on desired sequences* [330]. Recently, numerous studies have been conducted on *predicting protein properties and generating new molecules and DNA sequences*. These include, for example, *graph neural networks for molecule representation* [331–334], *prediction of amino acid sequence for a particular structure using deep neural networks* [104], *using GAN to generate DNA sequence* [335], and *structure prediction methods using neural networks* [180,336,337].

7.1. Protein Function

In several instances, functional folded proteins have been acquired from random-sequence libraries; however, this process

¹ k -mers are the substrings with length k within a biological sequence [297]

Table 8

Datasets of sequence-based methods.

Framework	Data Processing
SVM-conjoint triad [2007] [286]	A dataset comprising 16,443 nonredundant entries of experimentally verified PPI was extracted from the Human Protein Reference Database (HPRD; version 2005–0913; www.hprd.org). These interactions are primarily based on individual in vivo (e.g., coimmunoprecipitation) or in vitro (e.g., GST pull-down) experiments [286]. The negative dataset was created by excluding pairs that appeared in the positive dataset. For example, if AB and IJ are interacting pairs, AI, AJ, BI, and BJ may be noninteracting pairs. Additional conditions were applied, including an equal number of negative and positive pairs (16,443 in this study) and harmonious contributions of proteins forming the negative set. Therefore, the training set is equally distributed, comprising 32,486 protein pairs. The test set contained another 400 protein pairs. Both positive and negative pairs were randomly selected.
SVM-autocovariance [2008] [288]	The PPI data was extracted from the <i>S. cerevisiae</i> core subset of the Database of Interacting Proteins (DIP; version.20070219) [112], containing 5,966 interaction pairs. The expression profile reliability (EPR) and paralogous verification method (PVM) were used to test the reliability of this core subset [313]. By removing proteins with fewer than 50 amino acids, 5,943 protein pairs formed the final positive data set. The CD-HIT program was used to obtain a nonredundant subset with a sequence identity of 40%. The negative dataset was created using the Psub method, assuming that proteins located in different subcellular localisations do not interact. The subcellular location information was extracted from Swiss-Prot (http://www.expasy.org/sprot/). This method excluded proteins without subcellular localisation information and those marked as 'putative' or 'hypothetical', while proteins localised to the cytoplasm, nucleus, endoplasmic reticulum, Golgi apparatus, lysosome, and mitochondrion remained. The noninteracting pairs were generated by pairing proteins from one subset with those from the other. This strategy must satisfy the following conditions: (1) The DIP yeast interacting pairs do not include any noninteracting pairs, (2) there is an equal number of negative and positive pairs, and (3) the negative set should have a harmonious contribution.
SVM-based [2015] [285]	This method was evaluated using <i>S. cerevisiae</i> and <i>H. pylori</i> PPI datasets. The former was obtained from the <i>S. cerevisiae</i> core subset of the Database of Interacting Proteins (DIP). The non-redundant and negative pairs were obtained according to Guo et al. [288]. Therefore, the PPI dataset included 11,188 interacting and noninteracting pairs. The <i>H. pylori</i> dataset contained 2,916 protein pairs (1,458 interacting and 1,458 noninteracting) following [314].
DeepPPI [2017] [292]	The dataset evaluating the DeepPPI comprised 11,188 negative and positive protein pairs from <i>S. cerevisiae</i> obtained from the Database of Interacting Proteins (DIP; version 20160731), 1,458 interacting and 1,458 noninteracting pairs from <i>H. pylori</i> , 3,899 interacting and 4,262 noninteracting pairs from humans, 4,013 interacting pairs from <i>C.elegans</i> , 6,954 interacting pairs from <i>E. coli</i> , 1,412 interacting pairs from <i>H. sapiens</i> , 313 interacting pairs from <i>M. musculus</i> , and one additional <i>H. pylori</i> data set of 1,420 interacting pairs used in [315]. The negative dataset was created by pairing proteins from one subcellular location information extracted from Swiss-Prot (http://www.expasy.org/sprot/) with proteins from other locations.
Stacked Autoencoder (SAE) [2017] [18]	Pan's PPI dataset was acquired from [264], comprising 36,630 positive PPIs from the human protein reference database (HPRD, version 2007). Negative PPIs were generated by pairing proteins discovered in different subcellular locations from the Swiss-Prot database (version 57.3). After removing proteins with fewer than 50 amino acid residues, 2,184 unique proteins from six subcellular locations (cytoplasm, nucleus, endoplasmic reticulum, Golgi apparatus, lysosome, and mitochondrion) remained. The addition of negative pairs from the Negatome dataset [108] provided 36,480 total negative pairs. Protein pairs with nonstandard amino acids such as U and X were removed, resulting in a benchmark dataset of 36,545 positive and 36,323 negative pairs. The pre-training set contained 33,052 positive and 32,816 negative pairs, while 7,000 randomly selected pairs (3,493 positive and 3,507 negative) formed the test set. Pre-training and testing used 10-fold cross-validation. The external test sets used in this study included the 2010 version of the HPRD dataset, the 2010 HPRD NR dataset, the DIP dataset, and the HIPPIE dataset.
DPPI [2018] [298]	This study used human and yeast datasets from [316]. The human PPI dataset was created by taking the 10% top-scoring interactions from the Hippie database v1.2 [317]. The yeast PPI dataset was retrieved from DIP database [318]. Negative pairs were generated by randomly sampling from all proteins, where a 10:1 negative-to-positive ratio was considered [316]. Additionally, data redundancy regarding sequence similarity in PPI (>40%) was removed following the strategy of [316]. Finally, a 10-fold cross-validation was performed.
PIPR [2019] [299]	Guo's dataset [288] comprised 2,497 proteins forming 11,188 PPI pairs, half representing positive pairs and half representing negative pairs. Interaction pairs for <i>H. sapiens</i> were obtained from the STRING database (version 10.5) [319]. Three thousand randomly selected proteins and 8,000 proteins that shared < 40% sequence identity formed two subsets. Finally, protein binding affinity data were obtained from the SKEMPI dataset [320], comprising 3,047 binding affinity changes after mutation of protein subunits within a protein complex for use in the affinity estimation task.
ACT-SVM [2020] [289]	Following [321] a nonredundant dataset including <i>H. pylori</i> and human PPI was created. The <i>H. pylori</i> dataset comprised 1,458 interacting and 1,457 noninteracting protein pairs, while the human dataset comprised 3,899 interacting and 4,262 noninteracting protein pairs.
S-VGAE [2020] [300]	This study used data from the human protein reference database (HPRD) and the Database of Interacting Protein (DIP) for humans, <i>Drosophila</i> , <i>E.coli</i> , and <i>C.elegans</i> .

(continued on next page)

Table 8 (continued)

Framework	Data Processing
D-SCRIPT [2021] [165]	This study used a dataset from the STRING database (version 11) [261]. The positive pairs were limited to interactions associated with a positive experimental-evidence score. Only proteins containing 50–800 amino acids were retained. Additionally, proteins meeting the 40% similarity threshold were clustered using CD-HIT [322,323], removing redundant PPI from the dataset and preventing the model from memorising interactions based only on sequence similarity. Negative pairs were generated by randomly pairing proteins from the nonredundant set with a 10:1 negative-to-positive ratio [316]. The human PPI dataset comprised 47,932 positive and 479,320 negative protein interactions. Training and validation sets comprised 80% (38,345) and 20% (9,587) of pairs, respectively.
SPNet [2021] [9]	This study used the dataset of [324] comprising all the amino acids retrieved from the UniProt repository on 18 June 2019 and all proteins from the H. sapiens. The dataset comprised 16,210 unique proteins with a maximum length of 1,166 amino acids creating 104,262 total pairs. The training and validation sets contained 91,036 and 12,506 pairs, respectively, of which 33,318 and 6,094 pairs belonged to binding proteins (i.e., proteins with the potential to construct either transient or long-lived complexes). Two test sets were used in this study. Test-460 was a balanced strict set with 230 true positive and 230 true negative instances. Test-720 contained 260 true positive and 460 true negative instances. A 24-bin one-hot indicator represented each of the 20 standard amino acids and two stop codons, with the last two bins representing unknown or ambiguous amino acids.
BiLSTM-RF [2021] [291]	A nonredundant human dataset was retrieved from the DIP database. Sequences were clustered using the CD-HIT tool based on sequence similarity to remove redundancy and establish a nonredundant human PPI dataset [325,321]. This dataset included 4,262 interacting protein pairs and 3,899 noninteracting protein pairs.
Heterogeneous Network [2021] [296]	The 20 amino acids are divided into four groups based on their side chain polarity [286]: Ala, Val, Leu, Ile, Met, Phe, Trp, and Pro; Gly, Ser, Thr, Cys, Asn, Gln, and Tyr; Arg, Lys, and His; Asp and Glu. The protein sequences were simplified to a $4 \times 4 \times 4$ dimensional vector using the 3-mer method. Each vector dimension indicated the frequency of the amino acid sequence in the original protein sequence. Each dimension was initialised at zero. With a sliding window of length three, the whole protein sequence was scanned in steps of one. The amino acid sequence in the window was attached to the corresponding vector position in each step. Then, the vector was normalised. Finally, the vector obtained using 3-mers was an attribute feature.
OR-RCNN [2021] [301]	This study used datasets derived from the STRING database [319] for S. cerevisiae and H. sapiens. Each interaction was associated with a confidence score between zero (for noninteracting) and one (for interacting with the highest confidence). The confidence score interval was separated into K sub-intervals of equal length, where $K = 20$, (0, 0.05), [0.05, 0.1), ..., [0.95, 1). The retrieved data was limited to protein sequences of length 50–2000. For S. cerevisiae, they randomly selected 5400 data points from each sub-interval, while H. sapiens included 5000 randomly selected data points for each sub-interval.
DeepTrio [2022] [302]	The training dataset contained 90% of the data in each sub-interval, and the testing dataset contained the remaining 10%. The training and testing datasets were obtained from the Biological General Repository for Interaction Datasets (BioGRID) [326] and the Database of Interacting Proteins (DIP) [318,112]. The BioGRID database contains PPIs derived from multiple major species based on the criteria that interacting pairs must be validated by at least two different experimental systems or published sources. The S. cerevisiae and H. sapiens benchmark datasets from BioGRID were used for training. Protein sequences were obtained from the UniProt [327] and restricted to lengths of 150–1,500 amino acids. The S. cerevisiae dataset contained 255 pairs after removing proteins >1,500 amino acids. The PIPR's dataset [299] comprised 231 pairs after proteins longer than 2,000 amino acids were removed.

is often laborious and limited in the types of protein they can model [338–340,92].

Machine learning algorithms offer an alternative and possibly complementary approach capable of using the information available in protein sequence and structure databases. The information about the structural and biophysical constraints on the amino acid sequence within functional proteins can be found in natural sequence variation. However, these data are not labelled, which presents a challenge for straightforward supervised learning techniques. This is where generative modelling methods show promise due to their capability to exploit these data unsupervised.

A GAN-based data augmentation approach, FFPred-GAN, has been proposed in [341] to tackle the protein function prediction problem by learning the distribution of protein amino acid sequence-based biophysical features and producing high-quality artificial protein feature samples. In the presence of auxiliary information, generative models can conduct the generative process by modelling the data distribution conditioned on the auxiliary variables. In particular, designing a protein may entail preserving a special function while modifying a property such as stability or solubility. An example of these models is conditional GAN [160]. Such a generative framework is proposed in [177], which learns a conditional generative model for protein sequences by considering a certain target structure represented by a graph over the R-group of

amino acids. A VAE-based method is developed in [342] to generate novel variants of bacterial luciferase, an enzyme that emits light through the oxidation of flavin mononucleotide (FMN₂). A combination of reinforcement learning and RNNs is proposed in [343] to generate optimal molecules for biological activity.

7.2. Structure Design

Generative models for protein structures and modelling have been studied in [177,344], among which [104,345,346] have employed neural network-based models for sequences given their 3-D structure, modelling the amino acids independently from each other. Deep generative models have enabled new and viable protein structures [347]. Predicting the missing segments of corrupted protein structures can also be achieved using GAN, as represented by [173], in which the training data is restricted to structural information about the distances between adjacent α -carbons on the protein backbone.

7.3. Sequence Design

In addition to developing structure-based models, deep generative models have gained considerable attention for analysing protein sequences in individual protein families [348–351]. Even

though these approaches have proven effective, they require that a large number of sequences from a particular family are already available. This assumption cannot be met when designing novel proteins that diverge significantly from natural sequences, owing to an unbalanced dataset, non-interacting proteins [177]. Protein-GAN [352] is a GAN-based method with a customized temporal convolutional network [353] and self-attention mechanism [354] that aims to learn vital long-range inter-residue interactions and sequence motifs, as well as focusing on functional areas [355].

A conditional variational autoencoder (CVAE) model was developed in [356] to design protein sequences conditioned on a 1-D, context-free, grammar-based specification for folding topology. In [357,358], the conditional distributions of single amino acids are modelled, considering the encompassing structure and sequence context of the given protein, using convolutional neural networks. The generative model proposed in [177] is a graph-based model that captures the joint distribution of the full protein sequence, established on long-range interactions resulting from the protein 3-D structure.

Building on several recent successful studies using deep learning methods in modelling protein sequences such as contact prediction [359], prediction of secondary structure [360,361], and prediction of the fitness effects of mutations [348], generative modelling methods have begun to show potential for designing new sequences [177,349,362,356,363,350,364,365]. A LSTM is used in [366] as a generative approach in terms of amino acid sequences and peptide *de novo* design.

8. Conclusion

In the last decade, advancements in deep learning algorithms and GPUs as accelerators for high-performance computing have facilitated resolving intricate problems [367] concerning protein–protein and protein–ligand interaction, and drug discovery. This review offers an outline of protein structures and how they interact with other proteins, towards understanding their wide range of functionalities. Additionally, we outlined several deep learning methods and their applications to predicting protein–protein interaction, new drug delivery methods, and the improvement of existing solutions.

The available datasets for deep learning methods can be divided into structure-based and sequence-based. However, there is more sequential information available for proteins than there is 3-D structural information, thus driving progress in the development of sequence-based methods [368]. In fact, all the vital information required to identify PPIs is encoded in the proteins' amino acid sequence [369]. Several studies have been conducted by combining structural and sequential information.

Nevertheless, the viability of these techniques is yet to be verified experimentally. Continuing work is needed, such as analysing the strengths and limitations of different methods and the possibilities for incorporation into existing engineering operations. First and foremost, representation of the proteins to the network is a matter of importance [342]. This issue has been tackled using graph-based representations to model protein sequences and their 3-D structures [177,365]. Additionally, we may bolster our limited knowledge of protein folding mechanisms using deep reinforcement learning methods, aiming to find possible trajectories from extended protein chains to well-folded protein structures [355].

Following the Covid-19 pandemic and the dire need for rapid and reliable methods to create vaccines, one may see the potential of deep learning methods for solving such problems [370]. Additionally, a range of neurodegenerative diseases, infectious diseases, and cancers are closely related to abnormal protein–protein interactions [371–373]. Therefore, identifying protein–protein interactions

using deep learning methods helps pave the way towards developing new drugs and targeted therapeutic approaches [8,14,374].

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

All persons who have made substantial contributions to the work reported in the manuscript (e.g., technical help, writing and editing assistance, general support), but who do not meet the criteria for authorship, are named in the Acknowledgements and have given us their written permission to be named. If we have not included an Acknowledgements, then that indicates that we have not received substantial contributions from non-authors.

References

- [1] M.A. Clark, J. Choi, *Biology* (2018).
- [2] Zhang F, Shi W, Zhang J, Zeng M, Li M, Kurgan L. Probselect: accurate prediction of protein-binding residues from proteins sequences via dynamic predictor selection. *Bioinformatics* 2020;36:i735–44.
- [3] Chatr-Aryamontri A, Ceol A, Licata L, Cesareni G. Protein interactions: integration leads to belief. *Trends in Biochemical Sciences* 2008;33(6): 241–241.
- [4] De Las Rivas J, Fontanillo C. Protein–protein interactions essentials: key concepts to building and analyzing interactome networks. *PLoS Computational Biology* 2010;6(6):e1000807.
- [5] Mackay JP, Sunde M, Lowry JA, Crossley M, Matthews JM. Protein interactions: is seeing believing? *Trends in Biochemical Sciences* 2007;32(12):530–1.
- [6] Zhao Z, Gong X. Protein–protein interaction interface residue pair prediction based on deep learning architecture. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2017;16:1753–9.
- [7] Zhao J, Cao Y, Zhang L. Exploring the computational methods for protein–ligand binding site prediction. *Computational and structural biotechnology journal* 2020;18:417–26.
- [8] Lu H, Zhou Q, He J, Jiang Z, Peng C, Tong R, Shi J. Recent advances in the development of protein–protein interactions modulators: mechanisms and clinical trials. *Signal transduction and targeted therapy* 2020;5(1):1–23.
- [9] Wu J, Paquet E, Viktor HL, Michalowski W. Paying attention: Using a siamese pyramid network for the prediction of protein–protein interactions with folding and self-binding primary sequences. *International Joint Conference on Neural Networks (IJCNN)* 2021;2021:1–8. <https://doi.org/10.1109/IJCNN52387.2021.9534212>.
- [10] Jiang P, Huang S, Fu Z, Sun Z, Lakowski TM, Hu P. Deep graph embedding for prioritizing synergistic anticancer drug combinations. *Computational and Structural. Biotechnology Journal* 2020;18:427–38.
- [11] Gao W, Mahajan SP, Sulam J, Gray JJ. Deep learning in protein structural modeling and design. *Patterns* 2020;100142.
- [12] Jones S, Thornton JM. Principles of protein–protein interactions. *Proceedings of the National Academy of Sciences* 1996;93(1):13–20.
- [13] Zhou H-X, Shan Y. Prediction of protein interaction sites from sequence profile and residue neighbor list. *Proteins: Structure, Function, and Bioinformatics* 2001;44(3):336–43.
- [14] Skrabanek L, Saini HK, Bader GD, Enright AJ. Computational prediction of protein–protein interactions. *Molecular biotechnology* 2008;38(1):1–17.
- [15] Z.-H. You, L. Zhu, C.-H. Zheng, H.-J. Yu, S.-P. Deng, Z. Ji. Prediction of protein–protein interactions from amino acid sequences using a novel multi-scale continuous and discontinuous feature set, in: *BMC bioinformatics*, Vol. 15, Springer, 2014, pp. 1–9.
- [16] Sandhya S, Mudgal R, Kumar G, Sowdhamini R, Srinivasan N. Protein sequence design and its applications. *Current Opinion in Structural Biology* 2016;37:71–80.
- [17] S. Nivedha, S. Bhavani, A survey on prediction of protein–protein interactions, in: *Journal of Physics: Conference Series*, Vol. 1937, IOP Publishing, 2021, p. 012011.
- [18] Sun T, Zhou B, Lai L, Pei J. Sequence-based prediction of protein protein interaction using a deep-learning algorithm. *BMC bioinformatics* 2017;18(1):1–8.
- [19] Wang Y, You Z, Li L, Chen Z. A survey of current trends in computational predictions of protein–protein interactions. *Frontiers of Computer Science* 2020;14(4):1–12.
- [20] Phizicky EM, Fields S. Protein–protein interactions: methods for detection and analysis. *Microbiological reviews* 1995;59(1):94–123.

- [21] Zhang C, Zheng W, Cheng M, Omenn GS, Freddolino PL, Zhang Y. Functions of essential genes and a scale-free protein interaction network revealed by structure-based function and interaction prediction for a minimal genome. *Journal of proteome research* 2021;20(2):1178–89.
- [22] Chirgadze Y, Boshkova E, Kargatov A, Chirgadze N. Functional identification of 'hypothetical protein-structures with unknown function. *Journal of Biomolecular Structure and Dynamics* 2022:1–5.
- [23] Jansen R, Yu H, Greenbaum D, Kluger Y, Krogan NJ, Chung S, Emili A, Snyder M, Greenblatt JF, Gerstein M. bayesian networks approach for predicting protein-protein interactions from genomic data. *Science* 2003;302:449–53.
- [24] Wang T, Li L, Huang Y-A, Zhang H, Ma Y, Zhou X. Prediction of protein-protein interactions from amino acid sequences based on continuous and discrete wavelet transform features. *Molecules* 2018;23(4):823.
- [25] Schwikowski B, Uetz P, Fields S. A network of protein–protein interactions in yeast. *Nature biotechnology* 2000;18(12):1257–61.
- [26] Ito T, Tashiro K, Muta S, Ozawa R, Chiba T, Nishizawa M, Yamamoto K, Kuhara S, Sakaki Y. Toward a protein–protein interaction map of the budding yeast: a comprehensive system to examine two-hybrid interactions in all possible combinations between the yeast proteins. *Proceedings of the National Academy of Sciences* 2000;97(3):1143–7.
- [27] Gavin A-C, Bösch M, Krause R, Grandi P, Marzioch M, Bauer A, Schultz J, Rick JM, Michon A-M, Cruciat C-M, et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature* 2002;415(6868):141–7.
- [28] Ho Y, Gruhler A, Heilbut A, Bader GD, Moore L, Adams S-L, Millar A, Taylor P, Bennett K, Boutilier K, et al. Systematic identification of protein complexes in *saccharomyces cerevisiae* by mass spectrometry. *Nature* 2002;415(6868):180–3.
- [29] Pandey A, Mann M. Proteomics to study genes and genomes. *Nature* 2000;405(6788):837–46.
- [30] Figgeys D. Novel approaches to map protein interactions. *Current opinion in biotechnology* 2003;14(1):119–25.
- [31] Noor Z, Ahn SB, Baker MS, Ranganathan S, Mohamedali A. Mass spectrometry-based protein identification in proteomics—a review. *Briefings in bioinformatics* 2021;22(2):1620–38.
- [32] Garza KY, Feider CL, Klein DR, Rosenberg JA, Brodbelt JS, Eberlin LS. Desorption electrospray ionization mass spectrometry imaging of proteins directly from biological tissue sections. *Analytical chemistry* 2018;90(13):7785–9.
- [33] MacBeath G, Schreiber SL. Printing proteins as microarrays for high-throughput function determination. *Science* 2000;289(5485):1760–3.
- [34] Büsow K, Nordhoff E, Lübbert C, Lehrach H, Walter G. A human cDNA library for high-throughput protein expression screening. *Genomics* 2000;65(1):1–8. <https://doi.org/10.1006/geno.2000.6141>. <https://www.sciencedirect.com/science/article/pii/S088875430096141X>.
- [35] Brizuela L, Braun P, LaBaer J. Flexgene repository: from sequenced genomes to gene repositories for high-throughput functional biology and proteomics. *Molecular and biochemical parasitology* 2001;118(2):155–65.
- [36] Brizuela L, Richardson A, Marsichky G, LaBaer J. The flexgene repository: exploiting the fruits of the genome projects by creating a needed resource to face the challenges of the post-genomic era. *Archives of medical research* 2002;33(4):318–24.
- [37] A. Droit, G.G. Poirier, J.M. Hunter, Experimental and bioinformatic approaches for interrogating protein–protein interactions to determine protein function, *Journal of Molecular Endocrinology* 34 (2) (01 Apr. 2005) 263–280.
- [38] Zhou M, Li Q, Wang R. Current experimental methods for characterizing protein–protein interactions. *ChemMedChem* 2016;11(8):738.
- [39] Piehler J. New methodologies for measuring protein interactions in vivo and in vitro. *Current Opinion in Structural Biology* 2005;15(1):4–14.
- [40] Rao VS, Srinivas K, Sujini G, Kumar G. Protein-protein interaction detection: methods and analysis. *International journal of proteomics* 2014.
- [41] Ding Z, Kihara D. Computational identification of protein-protein interactions in model plant proteomes. *Scientific reports* 2019;9(1):1–13.
- [42] S. Tsukiyama, M.M. Hasan, S. Fujii, H. Kurata, Lstm-phv: Prediction of human-virus protein-protein interactions by Lstm with word2vec, *bioRxiv*.
- [43] Lodish H, Zipursky SL. *Molecular cell biology*. Biochem Mol Biol Educ 2001;29:126–33.
- [44] R.H. Garrett, *Biochemistry*, Cengage Learning Canada Inc, 2015.
- [45] A. Bagchi, Protein-protein interactions: Basics, characteristics, and predictions, in: *Soft Computing for Biological Systems*, Springer, 2018, pp. 111–120.
- [46] Go N, Taketomi H. Respective roles of short-and long-range interactions in protein folding. *Proceedings of the National Academy of Sciences* 1978;75(2):559–63.
- [47] Gromiha MM, Selvaraj S. Importance of long-range interactions in protein folding. *Biophysical chemistry* 1999;77(1):49–68.
- [48] Melikh AV, Meijer DK. On a generalized levinthal's paradox: The role of long-and short range interactions in complex bio-molecular reactions, including protein and dna folding. *Progress in Biophysics and Molecular Biology* 2018;132:57–79.
- [49] Gromiha MM, Selvaraj S. Influence of medium and long range interactions in different structural classes of globular proteins. *Journal of Biological Physics* 1997;23(3):151–62.
- [50] Buxbaum E. *Fundamentals of protein structure and function*, Vol. 31. Springer; 2007.
- [51] Maloy S. Amino acids. In: Maloy S, Hughes K, editors. *Brenner's Encyclopedia of Genetics* (Second Edition). Edition: Academic Press, San Diego; 2013. p. 108–10.
- [52] Yang L, Han Y, Zhang H, Li W, Dai Y. Prediction of protein-protein interactions with local weight-sharing mechanism in deep learning. *BioMed Research International* 2020.
- [53] Kyte J, Doolittle RF. A simple method for displaying the hydropathic character of a protein. *Journal of molecular biology* 1982;157(1):105–32.
- [54] Meiler J, Müller M, Zeidler A, Schmäschke F. Generation and evaluation of dimension-reduced amino acid parameter representations by artificial neural networks. *Molecular modeling annual* 2001;7(9):360–9.
- [55] Pogliani L. Molecular connectivity model for determination of isoelectric point of amino acids. *Journal of pharmaceutical sciences* 1992;81(4):334–6.
- [56] Biro J. Amino acid size, charge, hydrophathy indices and matrices for protein structure analysis. *Theoretical Biology and Medical Modelling* 2006;3(1):1–12.
- [57] Fauchère J-L, Charton M, Kier LB, Verloop A, Pliska V. Amino acid side chain parameters for correlation studies in biology and pharmacology. *International journal of peptide and protein research* 1988;32(4):269–78.
- [58] Holm L, Sander C. Mapping the protein universe. *Science* 1996;273(5275):595–602.
- [59] K-Y. Law, Definitions for hydrophilicity, hydrophobicity, and superhydrophobicity: getting the basics right (2014).
- [60] Jha K, Saha S, Tanveer M. Prediction of protein-protein interactions using stacked auto-encoder. *Transactions on Emerging Telecommunications Technologies* 2021:e4256.
- [61] Lu Y, Freeland S. On the evolution of the standard amino-acid alphabet. *Genome biology* 2006;7(1):1–6.
- [62] Schmidt RL, Simonović M. Synthesis and decoding of selenocysteine and human health. *Croatian medical journal* 2012;53(6):535–50.
- [63] Zhang Y, Baranov PV, Atkins JF, Gladyshev VN. Pyrrolysine and selenocysteine use dissimilar decoding strategies. *Journal of Biological Chemistry* 2005;280(21):20740–51.
- [64] Hatfield DL, Gladyshev VN. How selenium has altered our understanding of the genetic code. *Molecular and cellular biology* 2002;22(11):3565–76.
- [65] Turanov AA, Xu X-M, Carlson BA, Yoo M-H, Gladyshev VN, Hatfield DL. Biosynthesis of selenocysteine, the 21st amino acid in the genetic code, and a novel pathway for cysteine biosynthesis. *Advances in nutrition* 2011;2(2):122–8.
- [66] Gdr HB, Sharon N, Australia EW. Nomenclature and symbolism for amino acids and peptides. *Pure and Applied Chemistry* 1984;56:595–624.
- [67] *Biochemistry Human*. Elsevier 2018. <https://doi.org/10.1016/c2009-0-63992-1>.
- [68] Alhazmi HAM. Mobility shift-affinity capillary electrophoresis for investigation of protein-metal ion interactions: aspects of method development, validation and high throughput screening. Ph.D. thesis 2015. <https://doi.org/10.24355/dbbs.084-201506241157-0>.
- [69] Tahir M, Khan F, Hayat M, Alshehri MD. An effective machine learning-based model for the prediction of protein–protein interaction sites in health systems. *Neural Computing and Applications* 2022:1–11.
- [70] Keenleyside W. *Microbiology: Canadian Edition*, Pressbooks 2019.
- [71] Shoulders MD, Raines RT. Collagen structure and stability. *Annual review of biochemistry* 2009;78:929–58.
- [72] Stegemann H, Stalder K. Determination of hydroxyproline. *Clinica chimica acta* 1967;18(2):267–73.
- [73] E. Paquet, H. Viktor, K. Madi, J. Wu, Deformable protein shape classification based on deep learning, and the fractional fokkerplanck and khlerdirac equations., *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- [74] Kuhlman B, Dantas G, Ireton GC, Varani G, Stoddard BL, Baker D. Design of a novel globular protein fold with atomic-level accuracy. *science* 2003;302(5649):1364–8.
- [75] Fernandez-Fuentes N, Dybas JM, Fiser A. Structural characteristics of novel protein folds. *PLoS Computational Biology* 2010;6(4):e1000750.
- [76] Jiang L, Althoff EA, Clemente FR, Doyle L, Röthlisberger D, Zanghellini A, Gallaher JL, Betker JL, Tanaka F, Barbas CF, et al. De novo computational design of retro-aldol enzymes. *science* 2008;319(5868):1387–91.
- [77] Röthlisberger D, Khersonsky O, Wollacott AM, Jiang L, DeChancie J, Betker J, Gallaher JL, Althoff EA, Zanghellini A, Dym O, et al. Kemp elimination catalysts by computational enzyme design. *Nature* 2008;453:190–5.
- [78] Correia BE, Ban Y-EA, Holmes MA, Xu H, Ellingson K, Kraft Z, Carrico C, Boni E, Sather DN, Zenobia C, et al. Computational design of epitope-scaffolds allows induction of antibodies specific for a poorly immunogenic hiv vaccine epitope. *Structure* 2010;18(9):1116–26.
- [79] Leaver-Fay A, Froning KJ, Atwell S, Aldaz H, Pustilnik A, Lu F, Huang F, Yuan R, Hassanali S, Chamberlain AK, et al. Computationally designed bispecific antibodies using negative state repertoires. *Structure* 2016;24(4):641–51.
- [80] Lewis SM, Wu X, Pustilnik A, Sereno A, Huang F, Rick HL, Guntas G, Leaver-Fay A, Smith EM, Ho C, et al. Generation of bispecific igh antibodies by structure-based design of an orthogonal fab interface. *Nature biotechnology* 2014;32(2):191–8.
- [81] Correia BE, Bates JT, Loomis RJ, Baneyx G, Carrico C, Jardine JG, Rupert P, Correnti C, Kalyuzhnyi O, Vittal V, et al. Proof of principle for epitope-focused vaccine design. *Nature* 2014;507(7491):201–6.
- [82] Tinberg CE, Khare SD, Dou J, Doyle L, Nelson JW, Schena A, Jankowski W, Kalodimos CG, Johnsson K, Stoddard BL, et al. Computational design of ligand-

- binding proteins with high affinity and selectivity. *Nature* 2013;501(7466):212–6.
- [83] Zhou L, Bosscher M, Zhang C, Özçubukçu S, Zhang L, Zhang W, Li CJ, Liu J, Jensen MP, Lai L, et al. A protein engineered to bind uranyl selectively and with femtomolar affinity. *Nature chemistry* 2014;6(3):236–41.
- [84] King NP, Sheffler W, Sawaya MR, Vollmar BS, Sumida JP, André I, Gonen T, Yeates TO, Baker D. Computational design of self-assembling protein nanomaterials with atomic level accuracy. *Science* 2012;336(6085):1171–4.
- [85] King NP, Bale JB, Sheffler W, McNamara DE, Gonen S, Gonen T, Yeates TO, Baker D. Accurate design of co-assembling multi-component protein nanomaterials. *Nature* 2014;510(7503):103–8.
- [86] Gonen S, DiMaio F, Gonen T, Baker D. Design of ordered two-dimensional arrays mediated by noncovalent protein-protein interfaces. *Science* 2015;348(6241):1365–8.
- [87] Bale JB, Gonen S, Liu Y, Sheffler W, Ellis D, Thomas C, Cascio D, Yeates TO, Gonen T, King NP, et al. Accurate design of megadalton-scale two-component icosahedral protein complexes. *Science* 2016;353(6297):389–94.
- [88] Samish I. Computational protein design. Springer; 2017.
- [89] Korendovych IV, Senes A, Kim YH, Lear JD, Fry HC, Therien MJ, Blasie JK, Walker FA, DeGrado WF. De novo design and molecular assembly of a transmembrane diporphyrin-binding protein complex. *Journal of the American Chemical Society* 2010;132(44):15516–8.
- [90] Joh NH, Wang T, Bhate MP, Acharya R, Wu Y, Grabe M, Hong M, Grigoryan G, DeGrado WF. De novo design of a transmembrane zn^{2+} -transporting four-helix bundle. *Science* 2014;346(6216):1520–4.
- [91] Zhang Y, Bartz R, Grigoryan G, Bryant M, Aaronson J, Beck S, Innocent N, Klein L, Procopio W, Tucker T, et al. Computational design and experimental characterization of peptides intended for ph-dependent membrane insertion and pore formation. *ACS chemical biology* 2015;10(4):1082–93.
- [92] Huang P-S, Boyken SE, Baker D. The coming of age of de novo protein design. *Nature* 2016;537(7620):320–7.
- [93] Norn CH, André I. Computational design of protein self-assembly. *Current Opinion in Structural Biology* 2016;39:39–45.
- [94] Liu H, Chen Q. Computational protein design for given backbone: recent progresses in general method-related aspects. *Current Opinion in Structural Biology* 2016;39:89–95.
- [95] Yang W, Lai L. Computational design of ligand-binding proteins. *Current Opinion in Structural Biology* 2017;45:67–73.
- [96] Dima RI, Banavar JR, Maritan A. Scoring functions in protein folding and design. *Protein Science* 2000;9(4):812–9.
- [97] Li Z, Yang Y, Zhan J, Dai L, Zhou Y. Energy functions in de novo protein design: current challenges and future prospects. *Annual review of biophysics* 2013;42:315–35.
- [98] Boas FE, Harbury PB. Potential energy functions for protein design. *Current Opinion in Structural Biology* 2007;17(2):199–204.
- [99] Shapovalov MV, Dunbrack Jr RL. A smoothed backbone-dependent rotamer library for proteins derived from adaptive kernel density estimates and regressions. *Structure* 2011;19(6):844–58.
- [100] Doyle L, Hallinan J, Bolduc J, Parmeggiani F, Baker D, Stoddard BL, Bradley P. Rational design of α -helical tandem repeat proteins with closed architectures. *Nature* 2015;528(7583):585–8.
- [101] Bhardwaj G, Mulligan VK, Bahl CD, Gilmore JM, Harvey PJ, Cheneval O, Buchko GW, Pulavarti SV, Kaas Q, Eletsky A, et al. Accurate de novo design of hyperstable constrained peptides. *Nature* 2016;538(7625):329–35.
- [102] Broom A, Trainor K, MacKenzie DW, Meiering EM. Using natural sequences and modularity to design common and novel protein topologies. *Current Opinion in Structural Biology* 2016;38:26–36.
- [103] Khersonsky O, Fleishman SJ. Why reinvent the wheel? building new proteins based on ready-made parts. *Protein Science* 2016;25(7):1179–87.
- [104] Wang J, Cao H, Zhang JZ, Qi Y. Computational protein design with deep learning neural networks. *Scientific reports* 2018;8(1):1–9.
- [105] LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature* 2015;521(7553):436–44.
- [106] T.U. Consortium, UniProt: the universal protein knowledgebase in 2021. *Nucleic Acids Research* 49 (D1) D480–D489. doi:10.1093/nar/gkaa1100.
- [107] Waterhouse A, Bertoni M, Bienert S, Studer G, Tauriello G, Gumienny R, Heer FT, de Beer TAP, Rempfer C, Bordoli L, et al. Swiss-model: homology modelling of protein structures and complexes. *Nucleic acids research* 2018;46(W1):W296–303.
- [108] Blohm P, Frishman G, Smialowski P, Goebels F, Wachinger B, Ruepp A, Frishman D. Negatome 2.0: a database of non-interacting proteins derived by literature mining, manual annotation and protein structure analysis. *Nucleic acids research* 2014;42(D1):D396–400.
- [109] Szklarczyk D, Gable AL, Nastou KC, Lyon D, Kirsch R, Pyysalo S, Doncheva NT, Legeay M, Fang T, Bork P, et al. The string database in 2021: customizable protein-protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic acids research* 2021;49(D1):D605–12.
- [110] Bittrich S, Rose Y, Segura J, Lowe R, Westbrook JD, Duarte JM, Burley SK. Rcsb protein data bank: improved annotation, search and visualization of membrane protein structures archived in the pdb. *Bioinformatics* 2022;38(5):1452–4.
- [111] Oughtred R, Rust J, Chang C, Breitkreutz B-J, Stark C, Willems A, Boucher L, Leung G, Kolas N, Zhang F, et al. The biogrid database: A comprehensive biomedical resource of curated protein, genetic, and chemical interactions. *Protein Science* 2021;30(1):187–200.
- [112] Xenarios I, Salwinski L, Duan XJ, Higney P, Kim S-M, Eisenberg D. Dip, the database of interacting proteins: a research tool for studying cellular networks of protein interactions. *Nucleic acids research* 2002;30(1):303–5.
- [113] Bader GD, Betel D, Hogue CW. Bind: the biomolecular interaction network database. *Nucleic acids research* 2003;31(1):248–50.
- [114] Calderone A, Iannuccelli M, Peluso D, Licata L. Using the mint database to search protein interactions. *Current Protocols in Bioinformatics* 2020;69(1):e93.
- [115] Keshava Prasad T, Goel R, Kandasamy K, Keerthikumar S, Kumar S, Mathivanan S, Telikicherla D, Raju R, Shafreen B, Venugopal A, et al. Human protein reference database–2009 update. *Nucleic acids research* 2009;37(suppl_1):D767–72.
- [116] Orchard S, Ammari M, Aranda B, Breuza L, Briganti L, Broackes-Carter F, Campbell NH, Chavali G, Chen C, Del-Toro N, et al. The mintact project–intact as a common curation platform for 11 molecular interaction databases. *Nucleic acids research* 2014;42(D1):D358–63.
- [117] Hu L, Wang X, Huang Y-A, Hu P, You Z-H. A survey on computational models for predicting protein-protein interactions. *Briefings in Bioinformatics* 2021;22(5):bbab036.
- [118] Anfinsen CB. Principles that govern the folding of protein chains. *Science* 1973;181(4096):223–30.
- [119] Panchal GK, Das S, Sakure A, Singh BP, Hati S. Production and characterization of antioxidative peptides during lactic fermentation of goat milk. *Journal of Food Processing and Preservation* 2021;45(12):e15992.
- [120] V.K. Chaturvedi, D. Mishra, A. Tiwari, V. Snijesh, N.A. Shaik, M. Singh. Sequence databases, in: *Essentials of Bioinformatics*, Volume I, Springer, 2019, pp. 29–46.
- [121] Rose Y, Duarte JM, Lowe R, Segura J, Bi C, Bhikadiya C, Chen L, Rose AS, Bittrich S, Burley SK, et al. Rcsb protein data bank: architectural advances towards integrated searching and efficient access to macromolecular structure data from the pdb archive. *Journal of molecular biology* 2021;433(11):166704.
- [122] Andreeva A, Kulesha E, Gough J, Murzin AG. The scop database in 2020: expanded classification of representative family and superfamily domains of known protein structures. *Nucleic acids research* 2020;48(D1):D376–82.
- [123] Dandekar T, Snel B, Huynen M, Bork P. Conservation of gene order: a fingerprint of proteins that physically interact. *Trends in Biochemical Sciences* 1998;23(9):324–8.
- [124] Nishimoto AT, Sharma C, Rogers PD. Molecular and genetic basis of azole antifungal resistance in the opportunistic pathogenic fungus *candida albicans*. *Journal of Antimicrobial Chemotherapy* 2020;75(2):257–70.
- [125] Y. Bengio, L. Yao, G. Alain, P. Vincent, Generalized denoising auto-encoders as generative models, arXiv preprint arXiv:1305.6663.
- [126] Bengio Y, Goodfellow IJ, Courville A. Deep learning. *Nature* 2015;521(7553):436–44.
- [127] Soleymani F, Paquet E. Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder-deepbreath. *Expert Systems with Applications* 2020;156:113456.
- [128] Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 1994;5(2):157–66.
- [129] Hochreiter S, Bengio Y, Frasconi P, Schmidhuber J, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.
- [130] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural computation* 1997;9(8):1735–80.
- [131] James J, Lam AY, Hill DJ, Li VO. Delay aware intelligent transient stability assessment system. *IEEE Access* 2017;5:17230–9.
- [132] Srivastava N, Mansimov E, Salakhudinov R. Unsupervised learning of video representations using lstms. In: *International conference on machine learning PMLR*. p. 843–52.
- [133] Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 2017;34(4):18–42.
- [134] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv preprint arXiv:1506.05163.
- [135] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, arXiv preprint arXiv:1901.00596.
- [136] Zhang S, Tong H, Xu J, Maciejewski R. Graph convolutional networks: a comprehensive review. *Computational Social Networks* 2019;6(1):11.
- [137] Shuman DI, Narang SK, Frossard P, Ortega A, Vandergheynst P. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine* 2013;30(3):83–98.
- [138] Soleymani F, Paquet E. Deep graph convolutional reinforcement learning for financial portfolio management–deep-pocket. *Expert Systems with Applications* 2021;182:115127.
- [139] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, arXiv preprint arXiv:1312.6203.
- [140] Shuman DI, Vandergheynst P, Frossard P. Chebyshev polynomial approximation for distributed signal processing. In: *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS) IEEE*. p. 1–8.
- [141] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: *Advances in neural information processing systems*, 2016, pp. 3844–3852.

- [142] Ullah I, Manzo M, Shah M, Madden MG. Graph convolutional networks: analysis, improvements and results. *Applied Intelligence* 2022;52(8):9033–44.
- [143] I. Goodfellow, Nips 2016 tutorial: Generative adversarial networks, arXiv preprint arXiv:1701.00160.
- [144] Soleymani F, Paquet E. Long-term financial predictions based on feynman-dirac path integrals, deep bayesian networks and temporal generative adversarial networks. *Machine Learning with Applications* 2022;7:100255.
- [145] D.P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114.
- [146] Rezende DJ, Mohamed S, Wierstra D. Stochastic backpropagation and approximate inference in deep generative models. In: *International conference on machine learning PMLR*. p. 1278–86.
- [147] C. Doersch, Tutorial on variational autoencoders, arXiv preprint arXiv:1606.05908.
- [148] I.J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, arXiv preprint arXiv:1406.2661.
- [149] Bengio Y, Ducharme R, Vincent P, Janvin C. A neural probabilistic language model. *The journal of machine learning research* 2003;3:1137–55.
- [150] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, arXiv preprint arXiv:1609.03499.
- [151] A. Van Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, in: *International Conference on Machine Learning, PMLR*, 2016, pp. 1747–1756.
- [152] Choi Y, Choi M, Kim M, Ha J-W, Kim S, Choo J. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. p. 8789–97.
- [153] Yang G, Yu S, Dong H, Slabaugh G, Dragotti PL, Ye X, Liu F, Arridge S, Keegan J, Guo Y, et al. Dagan: Deep de-aliasing generative adversarial networks for fast compressed sensing mri reconstruction. *IEEE transactions on medical imaging* 2017;37(6):1310–21.
- [154] Seeliger K, Güçlü U, Ambrogioni L, Güçlütürk Y, van Gerven MA. Generative adversarial networks for reconstructing natural images from brain activity. *NeuroImage* 2018;181:775–85.
- [155] D.P. Kingma, M. Welling, An introduction to variational autoencoders, arXiv preprint arXiv:1906.02691.
- [156] S. Khobahi, M. Soltanalian, Model-aware deep architectures for one-bit compressive variational autoencoding, arXiv preprint arXiv:1911.12410.
- [157] An J, Cho S. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* 2015;2(1):1–18.
- [158] Kingma DP, Salimans T, Jozefowicz R, Chen X, Sutskever I, Welling M. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems* 2016;29:4743–51.
- [159] V. Dumoulin, F. Visin, A guide to convolution arithmetic for deep learning, arXiv preprint arXiv:1603.07285.
- [160] M. Mirza, S. Osindero, Conditional generative adversarial nets, arXiv preprint arXiv:1411.1784.
- [161] Chen Y, Xu D. Computational analyses of high-throughput protein-protein interaction data. *Current Protein and Peptide Science* 2003;4(3):159–80.
- [162] Yakubu RR, Nieves E, Weiss LM. The methods employed in mass spectrometric analysis of posttranslational modifications (ptms) and protein-protein interactions (ppis). *Advancements of mass spectrometry in biomedical research* 2019:169–98.
- [163] Lenz S, Sinn LR, O'Reilly FJ, Fischer L, Wegner F, Rappsilber J. Reliable identification of protein-protein interactions by crosslinking mass spectrometry. *Nature communications* 2021;12(1):1–11.
- [164] Yugandhar K, Gupta S, Yu H. Inferring protein-protein interaction networks from mass spectrometry-based proteomic approaches: a mini-review. *Computational and Structural. Biotechnology Journal* 2019;17:805–11.
- [165] S. Sledzieski, R. Singh, L. Cowen, B. Berger, Sequence-based prediction of protein-protein interactions: a structure-aware interpretable deep learning model, bioRxiv.
- [166] Bakail M, Ochsenbein F. Targeting protein-protein interactions, a wide open field for drug design. *Comptes Rendus Chimie* 2016;19(1–2):19–27.
- [167] Murakami Y, Tripathi LP, Prathipati P, Mizuguchi K. Network analysis and in silico prediction of protein-protein interactions with applications in drug discovery. *Current Opinion in Structural Biology* 2017;44:134–42.
- [168] Marchand A, Van Hall-Beauvais AK, Correia BE. Computational design of novel protein-protein interactions—an overview on methodological approaches and applications. *Current Opinion in Structural Biology* 2022;74:102370.
- [169] Siegel JB, Zanghellini A, Lovick HM, Kiss G, Lambert AR, Clair JLS, Gallaher JL, Hilvert D, Gelb MH, Stoddard BL, et al. Computational design of an enzyme catalyst for a stereoselective bimolecular diels-alder reaction. *Science* 2010;329(5989):309–13.
- [170] Whitehead TA, Chevalier A, Song Y, Dreyfus C, Fleishman SJ, De Mattos C, Myers CA, Kamisetty H, Blair P, Wilson IA, et al. Optimization of affinity, specificity and function of designed influenza inhibitors using deep sequencing. *Nature biotechnology* 2012;30:543–8.
- [171] Strauch E-M, Bernard SM, La D, Bohn AJ, Lee PS, Anderson CE, Nieusma T, Holstein CA, Garcia NK, Hooper KA, et al. Computational design of trimeric influenza-neutralizing proteins targeting the hemagglutinin receptor binding site. *Nature biotechnology* 2017;35:667–71.
- [172] Smart AD, Pache RA, Thomsen ND, Kortemme T, Davis GW, Wells JA. Engineering a light-activated caspase-3 for precise ablation of neurons in vivo. *Proceedings of the National Academy of Sciences* 2017;114(39):E8174–83.
- [173] Anand N, Huang P-S. Generative modeling for protein structures. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. p. 7505–16.
- [174] Sormani G, Harteveld Z, Rosset S, Correia B, Laio A. A rosetta-based protein design protocol converging to natural sequences. *The Journal of Chemical Physics* 2021;154(7):074114.
- [175] Koga N, Tatsumi-Koga R, Liu G, Xiao R, Acton TB, Montelione GT, Baker D. Principles for designing ideal protein structures. *Nature* 2012;491(7423):222–7.
- [176] Rocklin GJ, Chidyausiku TM, Goresnik I, Ford A, Houlston S, Lemak A, Carter L, Ravichandran R, Mulligan VK, Chevalier A, et al. Global analysis of protein folding using massively parallel design, synthesis, and testing. *Science* 2017;357(6347):168–75.
- [177] J. Ingraham, V. Garg, R. Barzilay, T. Jaakkola, Generative models for graph-based protein design, *Advances in Neural Information Processing Systems* 32.
- [178] Namrata A, Raphael E, Po-Ssu H. Fully differentiable full-atom protein backbone generation. In: *Proceedings of the International Conference on Learning Representations (ICLR) Workshops*.
- [179] J. Ingraham, A. Riesselman, C. Sander, D. Marks, Learning protein structure with a differentiable simulator, in: *International Conference on Learning Representations*, 2018.
- [180] AlQurashi M. End-to-end differentiable learning of protein structure. *Cell systems* 2019;8(4):292–301.
- [181] Dahiyat BI, Mayo SL. De novo protein design: fully automated sequence selection. *Science* 1997;278(5335):82–7.
- [182] Kraemer-Pecore CM, Lecomte JT, Desjarlais JR. A de novo redesign of the ww domain. *Protein Science* 2003;12(10):2194–205.
- [183] Russ WP, Lowery DM, Mishra P, Yaffe MB, Ranganathan R. Natural-like function in artificial ww domains. *Nature* 2005;437(7058):579–83.
- [184] Ding Z, Kihara D. Computational methods for predicting protein-protein interactions using various protein features. *Current protocols in protein science* 2018;93(1):e62.
- [185] Hosur R, Peng J, Vinayagam A, Stelzl U, Xu J, Perrimon N, Bienkowska J, Berger B. A computational framework for boosting confidence in high-throughput protein-protein interaction datasets. *Genome biology* 2012;13(8):1–14.
- [186] Mirabella C, Wallner B. Interpred: a pipeline to identify and model protein-protein interactions. *Proteins: Structure, Function, and Bioinformatics* 2017;85(6):1159–70.
- [187] Jha K, Saha S. Amalgamation of 3d structure and sequence information for protein-protein interaction prediction. *Scientific Reports* 2020;10(1):1–14.
- [188] Scarff CA, Thalassinou K, Hilton GR, Scrivens JH. Travelling wave ion mobility mass spectrometry studies of protein structure: biological significance and comparison with x-ray crystallography and nuclear magnetic resonance spectroscopy measurements. *Rapid Communications in Mass Spectrometry: An International Journal Devoted to the Rapid Dissemination of Up-to-the-Minute Research in Mass Spectrometry* 2008;22(20):3297–304.
- [189] Neuvirth H, Raz R, Schreiber G. Promate: a structure based prediction program to identify the location of protein-protein binding sites. *Journal of molecular biology* 2004;338(1):181–99.
- [190] Drewes G, Bouwmeester T. Global approaches to protein-protein interactions. *Current opinion in cell biology* 2003;15(2):199–205.
- [191] Zhang B, Li J, Quan L, Chen Y, Lü Q. Sequence-based prediction of protein-protein interaction sites by simplified long short-term memory network. *Neurocomputing* 2019;357:86–100.
- [192] Terentiev A, Moldogazieva N, Shaitan K. Dynamic proteomics in modeling of the living cell. protein-protein interactions. *Biochemistry (Moscow)* 2009;74(13):1586–607.
- [193] Brettner LM, Masel J. Protein stickiness, rather than number of functional protein-protein interactions, predicts expression noise and plasticity in yeast. *BMC systems biology* 2012;6(1):1–10.
- [194] Wodak SJ, Vlasblom J, Turinsky AL, Pu S. Protein-protein interaction networks: the puzzling riches. *Current Opinion in Structural Biology* 2013;23(6):941–53.
- [195] Hou Q, De Geest PF, Vranken WF, Heringa J, Feenstra KA. Seeing the trees through the forest: sequence-based homo-and heteromeric protein-protein interaction sites prediction using random forest. *Bioinformatics* 2017;33(10):1479–87.
- [196] Zeng M, Zhang F, Wu F-X, Li Y, Wang J, Li M. Protein-protein interaction site prediction through combining local and global features with deep neural networks. *Bioinformatics* 2020;36(4):1114–20.
- [197] Lu S, Li Y, Nan X, Zhang S. Attention-based convolutional neural networks for protein-protein interaction site prediction. In: *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE; 2021. p. 141–4.
- [198] Bonetta L. Interactome under construction. *Nature* 2010;468(7325):851–2.
- [199] Yuan Q, Chen J, Zhao H, Zhou Y, Yang Y. Structure-aware protein-protein interaction site prediction using deep graph convolutional network. *Bioinformatics* 2022;38(1):125–32.
- [200] Hou Q, Lensink MF, Heringa J, Feenstra KA. Club-martini: selecting favourable interactions amongst available candidates, a coarse-grained simulation approach to scoring docking decoys. *PLoS one* 2016;11(5):e0155251.

- [201] B. Huang, M. Schroeder, Using protein binding site prediction to improve protein docking, *Gene* 422 (1) (2008) 14–21, physical and Chemical Foundations of Bioinformatics Methods. doi: 10.1016/j.gene.2008.06.014.
- [202] Xie Z, Deng X, Shu K. Prediction of protein–protein interaction sites using convolutional neural network and improved data sets. *International journal of molecular sciences* 2020;21(2):467.
- [203] Li Y, Golding GB, Ilie L. Delphi: accurate deep ensemble model for protein interaction sites prediction. *Bioinformatics* 2021;37(7):896–904.
- [204] Landete J. Effector molecules and regulatory proteins: Applications. *Trends in Biotechnology* 2016;34(10):777–80.
- [205] Jambas AR, Day B, Cangea C, Liò P, Blundell TL. Deep learning for protein–protein interaction site prediction. *Proteomics Data Analysis*, Springer 2021:263–88.
- [206] Zhang C, Freddolino PL, Zhang Y. Cofactor: improved protein function prediction by combining structure, sequence and protein–protein interaction information. *Nucleic acids research* 2017;45(W1):W291–9.
- [207] Jubb H, Higuero AP, Winter A, Blundell TL. Structural biology and drug discovery for protein–protein interactions. *Trends in pharmacological sciences* 2012;33(5):241–8.
- [208] Hoskins J, Lovell S, Blundell TL. An algorithm for predicting protein–protein interaction sites: abnormally exposed amino acid residues and secondary structure elements. *Protein Science* 2006;15(5):1017–29.
- [209] Shi T-L, Li Y-X, Cai Y-D, Chou K-C. Computational methods for protein–protein interaction and their application. *Current Protein and Peptide Science* 2005;6(5):443–9.
- [210] K. Al-Khafaji, T. Taskin-Tok, Computational techniques for studying protein–protein interactions, in: *Advances in Protein Molecular and Structural Biology Methods*, Elsevier, 2022, pp. 125–135.
- [211] Vakser IA. Protein–protein docking: From interaction to interactome. *Biophysical journal* 2014;107(8):1785–93.
- [212] Siebenmorgen T, Zacharias M. Computational prediction of protein–protein binding affinities, *Wiley Interdisciplinary Reviews: Computational Molecular Science* 2020;10(3):e1448.
- [213] H.J. Nussbaumer, The fast fourier transform, in: *Fast Fourier Transform and Convolution Algorithms*, Springer, 1981, pp. 80–111.
- [214] Chaudhury S, Lyskov S, Gray JJ. Pyrosetta: a script-based interface for implementing molecular modeling algorithms using rosetta. *Bioinformatics* 2010;26(5):689–91.
- [215] D. Varela, I. André, A memetic algorithm enables global all-atom protein–protein docking with sidechain flexibility, *bioRxiv*.
- [216] Pierce BG, Wiehe K, Hwang H, Kim B-H, Vreven T, Weng Z. Zdock server: interactive docking prediction of protein–protein complexes and symmetric multimers. *Bioinformatics* 2014;30(12):1771–3.
- [217] Mashiah E, Nussinov R, Wolfson HJ. Fiberdock: flexible induced-fit backbone refinement in molecular docking, *Proteins: Structure, Function, and Bioinformatics* 2010;78(6):1503–19.
- [218] Gray JJ, Moughon S, Wang C, Schueler-Furman O, Kuhlman B, Rohl CA, Baker D. Protein–protein docking with simultaneous optimization of rigid-body displacement and side-chain conformations. *Journal of molecular biology* 2003;331(1):281–99.
- [219] Wang C, Bradley P, Baker D. Protein–protein docking with backbone flexibility. *Journal of molecular biology* 2007;373(2):503–19.
- [220] Chaudhury S, Berrondo M, Weitzner BD, Muthu P, Bergman H, Gray JJ. Benchmarking and analysis of protein docking performance in rosetta v3.2. *PLoS one* 2011;6(8):e22477.
- [221] Ohue M, Matsuzaki Y, Uchikoga N, Ishida T, Akiyama Y. Megadock: an all-to-all protein–protein interaction prediction system using tertiary structure data. *Protein and peptide letters* 2014;21(8):766–78.
- [222] Szilagy A, Zhang Y. Template-based structure modeling of protein–protein interactions. *Current Opinion in Structural Biology* 2014;24:10–23.
- [223] Tatusov RL, Koonin EV, Lipman DJ. A genomic perspective on protein families. *Science* 1997;278(5338):631–7.
- [224] Tuncbag N, Gursay A, Nussinov R, Keskin O. Predicting protein–protein interactions on a proteome scale by matching evolutionary and structural similarities at interfaces using prism. *Nature protocols* 2011;6(9):1341–54.
- [225] Keskin O, Tsai C-J, Wolfson H, Nussinov R. A new, structurally nonredundant, diverse data set of protein–protein interfaces and its implications. *Protein Science* 2004;13(4):1043–55.
- [226] Tuncbag N, Gursay A, Keskin O. Prediction of protein–protein interactions: unifying evolution and structure at protein interfaces. *Physical biology* 2011;8(3):035006.
- [227] Kundrotas PJ, Zhu Z, Janin J, Vakser IA. Templates are available to model nearly all complexes of structurally characterized proteins. *Proceedings of the National Academy of Sciences* 2012;109(24):9438–41.
- [228] Fukuhara N, Kawabata T. Homcos: a server to predict interacting protein pairs and interacting sites by homology modeling of complex structures. *Nucleic acids research* 2008;36(suppl. 2):W185–9.
- [229] Ghoorah AW, Devignes M-D, Smal-Tabbone M, Ritchie DW. Spatial clustering of protein binding sites for template based protein docking. *Bioinformatics* 2011;27(20):2820–7.
- [230] Zhang C, Zheng W, Mortuza S, Li Y, Zhang Y. Deepmsa: constructing deep multiple sequence alignment to improve contact prediction and fold-recognition for distant-homology proteins. *Bioinformatics* 2020;36(7):2105–12.
- [231] Mulnaes D, Porta N, Clemens R, Apanasenko I, Reiners J, Gremer L, Neudecker P, Smits SH, Gohlke H. Topmodel: template-based protein structure prediction at low sequence identity using top-down consensus and deep neural networks. *Journal of chemical theory and computation* 2020;16(3):1953–67.
- [232] Yan Y, Wen Z, Wang X, Huang S-Y. Addressing recent docking challenges: A hybrid strategy to integrate template-based and free protein–protein docking, *Proteins: Structure, Function, and Bioinformatics* 2017;85(3):497–512.
- [233] Dapkūnas J, Venclovas Č. Template-based modeling of protein complexes using the ppi3d web server. *Protein Structure Prediction*, Springer 2020:139–55.
- [234] Lee H, Baek M, Lee GR, Park S, Seok C. Template-based modeling and ab initio refinement of protein oligomer structures using galaxy in capri round 30, *Proteins: Structure, Function, and Bioinformatics* 2017;85(3):399–407.
- [235] Kundrotas PJ, Lensink MF, Alexov E. Homology-based modeling of 3d structures of protein–protein complexes using alignments of modified sequence profiles. *International journal of biological macromolecules* 2008;43(2):198–208.
- [236] Ogmen U, Keskin O, Aytuna AS, Nussinov R, Gursay A. Prism: protein interactions by structural matching. *Nucleic acids research* 2005;33(suppl.2):W331–6.
- [237] You Z-H, Chan KC, Hu P. Predicting protein–protein interactions from primary protein sequences using a novel multi-scale local feature representation scheme and the random forest. *PLoS one* 2015;10(5):e0125811.
- [238] Huang Y-A, You Z-H, Gao X, Wong L, Wang L. Using weighted sparse representation model combined with discrete cosine transformation to predict protein–protein interactions from protein sequence. *BioMed research international* 2015.
- [239] You Z-H, Yu J-Z, Zhu L, Li S, Wen Z-K. A mapreduce based parallel svm for large-scale predicting protein–protein interactions. *Neurocomputing* 2014;145:37–43.
- [240] Northey TC, Barešić A, Martin AC. Intpred: a structure-based predictor of protein–protein interaction sites. *Bioinformatics* 2018;34(2):223–9.
- [241] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, *arXiv preprint arXiv:1609.02907*.
- [242] Madi K, Paquet E, Kheddouci H. New graph distance for deformable 3d objects recognition based on triangle-stars decomposition. *Pattern Recognition* 2019;90:297–307.
- [243] Chen J, Zheng S, Zhao H, Yang Y. Structure-aware protein solubility prediction from sequence through graph convolutional network and predicted contact map. *Journal of cheminformatics* 2021;13(1):1–10.
- [244] Rao J, Zhou X, Lu Y, Zhao H, Yang Y. Imputing single-cell rna-seq data by combining graph convolution and autoencoder neural networks. *Iscience* 2021;24(5):102393.
- [245] Y. Song, S. Zheng, Z. Niu, Z.-H. Fu, Y. Lu, Y. Yang, Communicative representation learning on attributed molecular graphs., in: *IJCAI*, Vol. 2020, 2020, pp. 2831–2838.
- [246] Liu L, Zhu X, Ma Y, Piao H, Yang Y, Hao X, Fu Y, Wang L, Peng J. Combining sequence and network information to enhance protein–protein interaction prediction. *BMC bioinformatics* 2020;21(16):1–13.
- [247] A. Fout, J. Byrd, B. Shariat, A. Ben-Hur, Protein interface prediction using graph convolutional networks, *Advances in neural information processing systems* 30.
- [248] M. Baranwal, A. Magner, J. Saldinger, E.S. Turali-Emre, S. Kozarekar, P. Elvati, J. S. VanEpps, N.A. Kotov, A. Violi, A.O. Hero, Struct2graph: A graph attention network for structure based predictions of protein–protein interactions, *bioRxiv*.
- [249] Kufareva I, Budagyan L, Raush E, Totrov M, Abagyan R. Pier: protein interface recognition for structural proteomics, *Proteins: Structure, Function, and Bioinformatics* 2007;67(2):400–17.
- [250] Liang S, Zhang C, Liu S, Zhou Y. Protein binding site prediction using an empirical scoring function. *Nucleic acids research* 2006;34(13):3698–707.
- [251] Qin S, Zhou H-X. meta-ppisp: a meta web server for protein–protein interaction site prediction. *Bioinformatics* 2007;23(24):3386–7.
- [252] Porollo A, Meller J. Prediction-based fingerprints of protein–protein interactions, *Proteins: Structure, Function, and Bioinformatics* 2007;66(3):630–45.
- [253] Marks DS, Colwell LJ, Sheridan R, Hopf TA, Pagnani A, Zecchina R, Sander C. Protein 3d structure computed from evolutionary sequence variation. *PLoS one* 2011;6(12):e28766.
- [254] Morcos F, Pagnani A, Lunt B, Bertolino A, Marks DS, Sander C, Zecchina R, Onuchic JN, Hwa T, Weigt M. Direct-coupling analysis of residue coevolution captures native contacts across many protein families. *Proceedings of the National Academy of Sciences* 2011;108(49):E1293–301.
- [255] Balakrishnan S, Kamisetty H, Carbonell JG, Lee S-I, Langmead CJ. Learning generative models for protein fold families, *Proteins: Structure, Function, and Bioinformatics* 2011;79(4):1061–78.
- [256] Vreven T, Moal IH, Vangone A, Pierce BG, Kastiris PL, Torchala M, Chaleil R, Jiménez-García B, Bates PA, Fernandez-Recio J, et al. Updates to the integrated protein–protein interaction benchmarks: docking benchmark version 5 and affinity benchmark version 2. *Journal of molecular biology* 2015;427(19):3031–41.
- [257] Krissinel E, Henrick K. Inference of macromolecular assemblies from crystalline state, *Journal of molecular biology* 2007;372(3):774–97.
- [258] Wang G, Dunbrack Jr RL. Pisces: a protein sequence culling server. *Bioinformatics* 2003;19(12):1589–91.

- [259] Zhu H, Domingues FS, Sommer I, Lengauer T. Noxclass: prediction of protein-protein interaction types. *BMC bioinformatics* 2006;7(1):1–15.
- [260] Orengo CA, Michie AD, Jones S, Jones DT, Swindells MB, Thornton JM. Cath—a hierarchic classification of protein domain structures. *Structure* 1997;5(8):1093–109.
- [261] Szklarczyk D, Gable AL, Lyon D, Junge A, Wyder S, Huerta-Cepas J, Simonovic M, Doncheva NT, Morris JH, Bork P, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research* 2019;47(D1):D607–13.
- [262] Trabuco LG, Betts MJ, Russell RB. Negative protein–protein interaction datasets derived from large-scale two-hybrid experiments. *Methods* 2012;58(4):343–8.
- [263] A. Bateman, Uniprot: a universal hub of protein knowledge, in: *Protein Science*, Vol. 28, WILEY 111 RIVER ST, HOBOKEN 07030–5774, NJ USA, 2019, pp. 32–32.
- [264] Pan X-Y, Zhang Y-N, Shen H-B. Large-scale prediction of human protein-protein interactions from amino acid sequence based on latent topic features. *Journal of proteome research* 2010;9(10):4992–5001.
- [265] Cocco S, Feinauer C, Figliuzzi M, Monasson R, Weigt M. Inverse statistical physics of protein sequences: a key issues review. *Reports on Progress in Physics* 2018;81(3):032601.
- [266] Lehmann M, Kostrewa D, Wyss M, Brugger R, D'Arcy A, Pasamontes L, van Loon AP. From dna sequence to improved functionality: using protein sequence comparisons to rapidly design a thermostable consensus phytase. *Protein engineering* 2000;13(1):49–57.
- [267] Socolich M, Lockless SW, Russ WP, Lee H, Gardner KH, Ranganathan R. Evolutionary information for specifying a protein fold. *Nature* 2005;437(7058):512–8.
- [268] Porebski BT, Buckle AM. Consensus protein design. *Protein Engineering, Design and Selection* 2016;29(7):245–51.
- [269] Sievers F, Higgins DG. Clustal omega for making accurate alignments of many protein sequences. *Protein Science* 2018;27(1):135–45.
- [270] Hugenoltz P, Tyson GV. Metagenomics. *Nature* 2008;455(7212):481–3.
- [271] Cumberworth A, Lamour G, Babu MM, Gsponer J. Promiscuity as a functional trait: intrinsically disordered regions as central players of interactomes. *Biochemical Journal* 2013;454(3):361–9.
- [272] Bock JR, Gough DA. Predicting protein–protein interactions from primary structure. *Bioinformatics* 2001;17(5):455–60.
- [273] Chou K-C, Maggiora GM. Domain structural class prediction. *Protein Engineering* 1998;11(7):523–38.
- [274] Lu S, Wang J, Chitsaz F, Derbyshire MK, Geer RC, Gonzales NR, Gwadz M, Hurwitz DI, Marchler GH, Song JS, et al. Cdd/sparcle: the conserved domain database in 2020. *Nucleic acids research* 2020;48(D1):D265–8.
- [275] X. Li, L. Yang, X. Zhang, X. Jiao, Prediction of protein-protein interactions based on domain, Computational and mathematical methods in medicine 2019.
- [276] Wojcik J, Schächter V. Protein-protein interaction map inference using interacting domain profile pairs. *Bioinformatics* 2001;17(suppl_1):S296–305.
- [277] Wan KK, Park J, Suh JK. Large scale statistical prediction of protein-protein interaction by potentially interacting domain (pid) pair. *Genome Informatics* 2002;13:42–50.
- [278] Kamada M, Sakuma Y, Hayashida M, Akutsu T. Prediction of protein-protein interaction strength using domain features with supervised regression. *The Scientific World Journal* 2014.
- [279] Singhal M, Resat H. A domain-based approach to predict protein-protein interactions. *Bmc Bioinformatics* 2007;8(1):1–19.
- [280] Lee S-A, Chan C-H, Tsai C-H, Lai J-M, Wang F-S, Kao C-Y, Huang C-YF. Ortholog-based protein-protein interaction prediction and its application to inter-species interactions. *BMC bioinformatics* 2008;9(12):1–9.
- [281] Koonin EV. Orthologs, paralogs, and evolutionary genomics. *Annu. Rev. Genet.* 2005;39:309–38.
- [282] Winter C, Henschel A, Kim WK, Schroeder M. Scoppi: a structural classification of protein–protein interfaces. *Nucleic acids research* 2006;34(suppl_1):D310–4.
- [283] Hosur R, Xu J, Bienkowska J, Berger B. iwrap: an interface threading approach with application to prediction of cancer-related protein–protein interactions. *Journal of molecular biology* 2011;405(5):1295–310.
- [284] Valente GT, Acencio ML, Martins C, Lemke N. The development of a universal in silico predictor of protein-protein interactions. *PloS one* 2013;8(5):e65587.
- [285] You Z-H, Li J, Gao X, He Z, Zhu L, Lei Y-K, Ji Z. Detecting protein-protein interactions with a novel matrix-based protein sequence representation and support vector machines. *BioMed research international* 2015.
- [286] Shen J, Zhang J, Luo X, Zhu W, Yu K, Chen K, Li Y, Jiang H. Predicting protein-protein interactions based only on sequences information. *Proceedings of the National Academy of Sciences* 2007;104(11):4337–41.
- [287] Zhang S-W, Hao L-Y, Zhang T-H. Prediction of protein–protein interaction with pairwise kernel support vector machine. *International journal of molecular sciences* 2014;15:3220–33.
- [288] Guo Y, Yu L, Wen Z, Li M. Using support vector machine combined with auto covariance to predict protein-protein interactions from protein sequences. *Nucleic acids research* 2008;36(9):3025–30.
- [289] Ma W, Cao Y, Bao W, Yang B, Chen Y. Act-svm: Prediction of protein-protein interactions based on support vector basis model. *Scientific Programming* 2020.
- [290] T. Bepler, B. Berger, Learning protein sequence embeddings using information from structure, arXiv preprint arXiv:1902.08661.
- [291] Ma W, Bao W, Cao Y, Yang B, Chen Y. Prediction of protein-protein interaction based on deep learning feature representation and random forest. *International Conference on Intelligent Computing*, Springer 2021:654–62.
- [292] Du X, Sun S, Hu C, Yao Y, Yan Y, Zhang Y. Deepppi: boosting prediction of protein–protein interactions with deep neural networks. *Journal of chemical information and modeling* 2017;57(6):1499–510.
- [293] Wold S, Jonsson J, Sjöström M, Sandberg M, Rännar S. Dna and peptide sequences and chemical processes multivariately modelled by principal component analysis and partial least-squares projections to latent structures. *Analytica Chimica Acta* 1993;277(2):239–53.
- [294] Davies MN, Secker A, Freitas AA, Clark E, Timmis J, Flower DR. Optimizing amino acid groupings for gpcr classification. *Bioinformatics* 2008;24(18):1980–6.
- [295] Tong JC, Tammi MT. Prediction of protein allergenicity using local description of amino acid sequence. *Frontiers in Bioscience* 2008;13(16):6072–8.
- [296] Su X-R, You Z-H, Chen Z-H, Yi H-C, Guo Z-H. Protein-protein interaction prediction by integrating sequence information and heterogeneous network representation. In: *International Conference on Intelligent Computing*. Springer; 2021. p. 617–26.
- [297] Manekar SC, Sathe SR. A benchmark study of k-mer counting methods for high-throughput sequencing. *GigaScience* 2018;7(12):giy125.
- [298] Hashemifar S, Neyshabur B, Khan AA, Xu J. Predicting protein–protein interactions through sequence-based deep learning. *Bioinformatics* 2018;34:i802–10.
- [299] Chen M, Ju CJ-T, Zhou G, Chen X, Zhang T, Chang K-W, Zaniolo C, Wang W. Multifaceted protein–protein interaction prediction based on siamese residual rcnn. *Bioinformatics* 2019;35(14):i305–14.
- [300] Yang F, Fan K, Song D, Lin H. Graph-based prediction of protein-protein interactions with attributed signed graph embedding. *BMC bioinformatics* 2020;21(1):1–16.
- [301] Xu W, Gao Y, Wang Y, Guan J. Protein–protein interaction prediction based on ordinal regression and recurrent convolutional neural networks. *BMC bioinformatics* 2021;22(6):1–21.
- [302] Hu X, Feng C, Zhou Y, Harrison A, Chen M. Deeptrio: a ternary prediction system for protein–protein interaction using mask multiple parallel convolutional neural networks. *Bioinformatics* 2022;38(3):694–702.
- [303] Xie Y, Zhang T. A fault diagnosis approach using svm with data dimension reduction by pca and lda method. In: *2015 Chinese Automation Congress (CAC)*. IEEE; 2015. p. 869–74.
- [304] Saha I, Zubek J, Klingström T, Forsberg S, Wikander J, Kierczak M, Maulik U, Plewczynski D. Ensemble learning prediction of protein–protein interactions using proteins functional annotations. *Molecular BioSystems* 2014;10(4):820–30.
- [305] X. Wang, J. Xu, W. Shi, J. Liu, Ogru: An optimized gated recurrent unit neural network, in: *Journal of Physics: Conference Series*, Vol. 1325, IOP Publishing, 2019, p. 012089.
- [306] T.N. Kipf, M. Welling, Variational graph auto-encoders, arXiv preprint arXiv:1611.07308.
- [307] Cervantes J, Garcia-Lamont F, Rodríguez-Mazahua L, Lopez A. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing* 2020;408:189–215.
- [308] Yang L, Xia J-F, Gui J. Prediction of protein-protein interactions from protein sequence using local descriptors. *Protein and Peptide Letters* 2010;17(9):1085–90.
- [309] Wong L, You Z-H, Li S, Huang Y-A, Liu G. Detection of protein-protein interactions from amino acid sequences using a rotation forest model with a novel pr-lpq descriptor. In: *International Conference on Intelligent Computing*. Springer; 2015. p. 713–20.
- [310] Chen T, Guestrin C. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. p. 785–94.
- [311] Chang C-C, Lin C-J. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)* 2011;2(3):1–27.
- [312] Yao Y, Du X, Xiao Y, Zhu H. An integration of deep learning with feature embedding for protein–protein interaction prediction. *PeerJ* 2019;7:e7126.
- [313] Deane CM, Salwinski Ł, Xenarios I, Eisenberg D. Protein interactions: two methods for assessment of the reliability of high throughput observations. *Molecular & Cellular Proteomics* 2002;1(5):349–56.
- [314] Martin S, Roe D, Faulon J-L. Predicting protein-protein interactions using signature products. *Bioinformatics* 2005;21(2):218–26.
- [315] Zhou YZ, Gao Y, Zheng YY. Prediction of protein-protein interactions using local description of amino acid sequence. In: *Advances in computer science and education applications*. Springer; 2011. p. 254–62.
- [316] Hamp T, Rost B. Evolutionary profiles improve protein–protein interaction prediction from sequence. *Bioinformatics* 2015;31(12):1945–50.
- [317] Schaefer MH, Fontaine J-F, Vinayagam A, Porras P, Wanker EE, Andrade-Navarro MA. Hippie: Integrating protein interaction networks with experiment based quality scores. *PloS one* 2012;7(2):e31826.
- [318] Salwinski L, Miller CS, Smith AJ, Pettit FK, Bowie JU, Eisenberg D. The database of interacting proteins: 2004 update. *Nucleic acids research* 2004;32(suppl_1):D449–51.
- [319] D. Szklarczyk, J.H. Morris, H. Cook, M. Kuhn, S. Wyder, M. Simonovic, A. Santos, N.T. Doncheva, A. Roth, P. Bork, et al., The string database in 2017:

- quality-controlled protein–protein association networks, made broadly accessible, *Nucleic acids research* (2016) gkw937.
- [320] Moal IH, Fernández-Recio J. Skempi: a structural kinetic and energetic database of mutant protein interactions and its use in empirical models. *Bioinformatics* 2012;28(20):2600–7.
- [321] Kong M, Zhang Y, Xu D, Chen W, Dehmer M. Fcnp-wsrc: protein–protein interactions prediction via weighted sparse representation based classification. *Frontiers in genetics* 2020;11:18.
- [322] Li W, Godzik A. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics* 2006;22(13):1658–9.
- [323] Fu L, Niu B, Zhu Z, Wu S, Li W. Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics* 2012;28(23):3150–2.
- [324] F. Richoux, C. Servantie, C. Borès, S. Téletchéa, Comparing two deep learning sequence-based models for protein–protein interaction prediction, *arXiv preprint arXiv:1901.06268*.
- [325] Li W, Jaroszewski L, Godzik A. Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics* 2001;17(3):282–3.
- [326] Oughtred R, Stark C, Breitkreutz B-J, Rust J, Boucher L, Chang C, Kolas N, O'Donnell L, Leung G, McAdam R, et al. The biogrid interaction database: 2019 update. *Nucleic acids research* 2019;47(D1):D529–41.
- [327] U. Consortium. Uniprot: a worldwide hub of protein knowledge. *Nucleic acids research* 2019;47(D1):D506–15.
- [328] Popova M, Isayev O, Tropsha A. Deep reinforcement learning for de novo drug design. *Science advances*, 4, 2018, p. eaap7885.
- [329] Segler MH, Kogej T, Tyrchan C, Waller MP. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science* 2018;4(1):120–31.
- [330] Gupta A, Müller AT, Huismann BJ, Fuchs JA, Schneider P, Schneider G. Generative recurrent networks for de novo drug design. *Molecular informatics* 2018;37(1–2):1700111.
- [331] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, *arXiv preprint arXiv:1509.09292*.
- [332] Wu Z, Ramsundar B, Feinberg EN, Gomes J, Geniesse C, Pappu AS, Leswing K, Pande V. Moleculenet: a benchmark for molecular machine learning. *Chemical science* 2018;9(2):513–30.
- [333] Gómez-Bombarelli R, Wei JN, Duvenaud D, Hernández-Lobato JM, Sánchez-Lengeling B, Sheberla D, Aguilera-Iparraguirre J, Hirzel TD, Adams RP, Aspuru-Guzik A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science* 2018;4(2):268–76.
- [334] Kusner MJ, Paige B, Hernández-Lobato JM. Grammar variational autoencoder. *International Conference on Machine Learning*. PMLR 2017:1945–54.
- [335] N. Killoran, L.J. Lee, A. DeLong, D. Duvenaud, B.J. Frey, Generating and designing dna with deep generative models, *arXiv preprint arXiv:1712.06148*.
- [336] Kamisetty H, Ovchinnikov S, Baker D. Assessing the utility of coevolution-based residue–residue contact predictions in a sequence- and structure-rich era. *Proceedings of the National Academy of Sciences* 2013;110(39):15674–9.
- [337] Hopf TA, Schärfe CP, Rodrigues JP, Green AG, Kohlbacher O, Sander C, Bonvin AM, Marks DS. Sequence co-evolution gives 3d contacts and structures of protein complexes. *Elife* 2014;3:e03430.
- [338] Keefe AD, Szostak JW. Functional proteins from a random-sequence library. *Nature* 2001;410(6829):715–8.
- [339] Fisher MA, McKinley KL, Bradley LH, Viola SR, Hecht MH. De novo designed proteins from a library of artificial sequences function in *Escherichia coli* and enable cell growth. *PLoS one* 2011;6(1):e15364.
- [340] Murphy GS, Greisman JB, Hecht MH. De novo proteins with life-sustaining functions are structurally dynamic. *Journal of molecular biology* 2016;428(2):399–411.
- [341] Wan C, Jones DT. Protein function prediction is improved by creating synthetic feature samples with generative adversarial networks. *Nature Machine Intelligence* 2020;2(9):540–50.
- [342] Hawkins-Hooker A, Depardieu F, Baur S, Couairon G, Chen A, Bikard D. Generating functional protein variants with variational autoencoders. *PLoS Computational Biology* 2021;17(2):e1008736.
- [343] Olivecrona M, Blaschke T, Engkvist O, Chen H. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics* 2017;9(1):1–14.
- [344] Yang KK, Wu Z, Arnold FH. Machine-learning-guided directed evolution for protein engineering. *Nature methods* 2019;16(8):687–94.
- [345] O'Connell J, Li Z, Hanson J, Heffernan R, Lyons J, Paliwal K, Dehzangi A, Yang Y, Zhou Y. Spin2: Predicting sequence profiles from protein structures using deep neural networks. *Proteins: Structure, Function, and Bioinformatics* 2018;86(6):629–33.
- [346] Chen S, Sun Z, Lin L, Liu Z, Liu X, Chong Y, Lu Y, Zhao H, Yang Y. To improve protein sequence profile prediction through image captioning on pairwise residue distance map. *Journal of chemical information and modeling* 2019;60(1):391–9.
- [347] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, *arXiv preprint arXiv:1511.06434*.
- [348] Riesselman AJ, Ingraham JB, Marks DS. Deep generative models of genetic variation capture the effects of mutations. *Nature methods* 2018;15(10):816–22.
- [349] Tubiana J, Cocco S, Monasson R. Learning protein constitutive motifs from sequence data. *Elife* 2019;8:e39397.
- [350] Riesselman A, Shin J-E, Kollasch A, McMahon C, Simon E, Sander C, Manglik A, Kruse A, Marks D. Accelerating protein design using autoregressive generative models. *BioRxiv* 2019:757252.
- [351] S. Sinai, E. Kelsic, G.M. Church, M.A. Nowak, Variational auto-encoding of protein sequences, *arXiv preprint arXiv:1712.03346*.
- [352] Repecka D, Jauniskis V, Karpus L, Rembeza E, Rokaitis I, Zrimec J, Poviloniene S, Laurynenas A, Viknander S, Abuajwa W, et al. Expanding functional protein sequence spaces using generative adversarial networks. *Nature Machine Intelligence* 2021;3(4):324–33.
- [353] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, *arXiv preprint arXiv:1803.01271*.
- [354] H. Zhang, I. Goodfellow, D. Metaxas, A. Odena, Self-attention generative adversarial networks, in: *International conference on machine learning*, PMLR, 2019, pp. 7354–7363.
- [355] W. Ding, K. Nakai, H. Gong, Protein design via deep learning, *Briefings in Bioinformatics*.
- [356] Greener JG, Moffat L, Jones DT. Design of metalloproteins and novel protein folds using variational autoencoders. *Scientific reports* 2018;8(1):1–12.
- [357] W. Boomsma, J. Frellsen, Spherical convolutions and their application in molecular modelling, in: *NIPS*, Vol. 2, 2017, p. 6.
- [358] M. Weiler, M. Geiger, M. Welling, W. Boomsma, T. Cohen, 3d steerable cnns: Learning rotationally equivariant features in volumetric data, *arXiv preprint arXiv:1807.02547*.
- [359] Wang S, Sun S, Li Z, Zhang R, Xu J. Accurate de novo prediction of protein contact map by ultra-deep learning model. *PLoS Computational Biology* 2017;13(1):e1005324.
- [360] Spencer M, Eickholt J, Cheng J. A deep learning network approach to ab initio protein secondary structure prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2014;12(1):103–12.
- [361] Wang S, Peng J, Ma J, Xu J. Protein secondary structure prediction using deep convolutional neural fields. *Scientific reports* 2016;6(1):1–11.
- [362] P. Das, K. Wadhawan, O. Chang, T. Sercu, C.D. Santos, M. Riemer, V. Chenthamarakshan, I. Padhi, A. Mojsilovic, Pepcvae: Semi-supervised targeted design of antimicrobial peptide sequences, *arXiv preprint arXiv:1810.07743*.
- [363] D. Repecka, V. Jauniskis, L. Karpus, E. Rembeza, J. Zrimec, S. Poviloniene, I. Rokaitis, A. Laurynenas, W. Abuajwa, O. Savolainen, et al., Expanding functional protein sequence space using generative adversarial networks, *BioRxiv* (2019) 789719.
- [364] A. Madani, B. McCann, N. Naik, N.S. Keskar, N. Anand, R.R. Eguchi, P.-S. Huang, R. Socher, Progen: Language modeling for protein generation, *arXiv preprint arXiv:2004.03497*.
- [365] Strokach A, Becerra D, Corbi-Verge C, Perez-Riba A, Kim PM. Fast and flexible protein design using deep graph neural networks. *Cell Systems* 2020;11(4):402–11.
- [366] Muller AT, Hiss JA, Schneider G. Recurrent neural network model for constructive peptide design. *Journal of chemical information and modeling* 2018;58(2):472–9.
- [367] R. Lim, Methods for accelerating machine learning in high performance computing, *University of Oregon–Area–2019–01*.
- [368] Pan J, Li L-P, Yu C-Q, You Z-H, Guan Y-J, Ren Z-H. Sequence-based prediction of plant protein–protein interactions by combining discrete sine transformation with rotation forest. *Evolutionary Bioinformatics* 2021;17:11769343211050067.
- [369] Jia J, Li X, Qiu W, Xiao X, Chou K-C. ippi-pseAAC (cgr): Identify protein–protein interactions by incorporating chaos game representation into pseAAC. *Journal of theoretical biology* 2019;460:195–203.
- [370] Chang C-K, Lin S-M, Satange R, Lin S-C, Sun S-C, Wu H-Y, Kehn-Hall K, Hou M-H. Targeting protein–protein interaction interfaces in covid-19 drug discovery. *Computational and Structural Biotechnology Journal* 2021;19:2246–55.
- [371] S. Ferrari, F. Pellati, M. Costi, Disruption of protein–protein interfaces (2013).
- [372] Rual J-F, Venkatesan K, Hao T, Hirozane-Kishikawa T, Dricot A, Li N, Berriz GF, Gibbons FD, Dreze M, Ayivi-Guedehoussou N, et al. Towards a proteome-scale map of the human protein–protein interaction network. *Nature* 2005;437(7062):1173–8.
- [373] Stelzl U, Worm U, Lalowski M, Haenic G, Brembeck FH, Goehler H, Stroedicke M, Zenkner M, Schoenherr A, Koeppen S, et al. A human protein–protein interaction network: a resource for annotating the proteome. *Cell* 2005;122(6):957–68.
- [374] F. Browne, H. Zheng, H. Wang, F. Azuaje, From experimental approaches to computational techniques: a review on the prediction of protein–protein interactions, *Advances in Artificial Intelligence* (16877470).

Chapter 3

Protein–Protein Interaction Prediction

This chapter presents a deep learning framework, ProtInteract, which is presented in the following paper, published on February 26, 2023, in the Computational and Structural Biotechnology journal. The ProtInteract is developed to predict PPIs using the configuration of their amino acid sequence. As established earlier, the primary structure of proteins may be conceptualised as pseudo-time series, which is analogous to the computational framework as the financial prediction, as illustrated in Fig. 1.1. Therefore, the ProtInteract framework is inspired by previously developed deep learning methods to predict the stock market and portfolio management [59, 60, 64]. A restricted stacked autoencoder has been developed in [59, 60] to extract highly informative sequence attributes and perform dimensionality reduction.

To reduce the high-dimensional protein presentation, initially, a convolutional autoencoder (Fig. 3.1) was designed containing, 2-D convolutional and max-pooling layers to reduce the size of the input tensor.

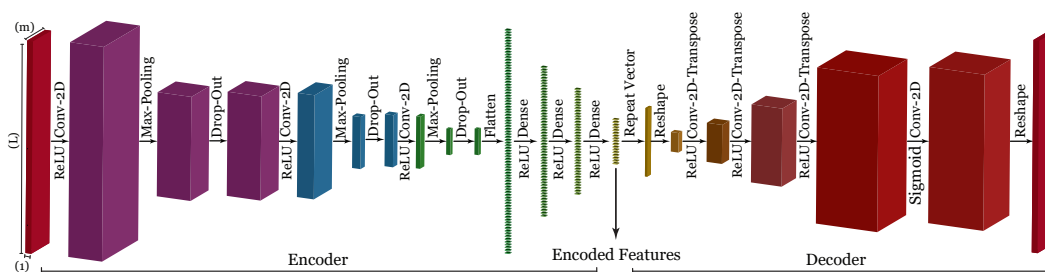


Figure 3.1: Convolutional Autoencoder architecture that extracts low-dimensional informative features.

Despite their ability to learn local representations, standard CNN does not comply with causal directionality; thereby, the future may influence the past [65]. Therefore, the convolutional Autoencoder does not support the underlying sequential structure that is encoded by each protein. This inspired the development of the novel autoencoder architecture that encodes protein to a lower dimension, addressing the computational challenges posed by the high dimension of protein datasets and extracting highly informative sequence attributes by integrating LSTM and TCN frameworks.

The ProtInteract framework predicts PPIs through two main tasks: encoding and prediction. A novel autoencoder architecture encodes each protein to remedy the computational complexity posed by the high dimensionality of protein representation while extracting sequential attributes of the protein’s primary structure. This framework can identify high-affinity PPI which may indicate ligands of a given receptor (target protein) [66].



ProtInteract: A deep learning framework for predicting protein–protein interactions



Farzan Soleymani^a, Eric Paquet^{b,*}, Herna Lydia Viktor^c, Wojtek Michalowski^d, Davide Spinello^a

^a Department of Mechanical Engineering, University of Ottawa, Ottawa, ON K1N 6N5, Canada

^b National Research Council, 1200 Montreal Road, Ottawa, ON K1A 0R6, Canada

^c School of Electrical Engineering and Computer Science, University of Ottawa, ON K1N 6N5, Canada

^d Telfer School of Management, University of Ottawa, ON K1N 6N5, Canada

ARTICLE INFO

Article history:

Received 5 December 2022

Received in revised form 20 January 2023

Accepted 20 January 2023

Available online 25 January 2023

Keywords:

Protein–Protein interaction

Autoencoder

Long short-term memory

Temporal convolutional, Network

Convolutional neural network

Sequential pattern

Feature selection

ABSTRACT

Proteins mainly perform their functions by interacting with other proteins. Protein–protein interactions underpin various biological activities such as metabolic cycles, signal transduction, and immune response. However, due to the sheer number of proteins, experimental methods for finding interacting and non-interacting protein pairs are time-consuming and costly. We therefore developed the ProtInteract framework to predict protein–protein interaction. ProtInteract comprises two components: first, a novel auto-encoder architecture that encodes each protein's primary structure to a lower-dimensional vector while preserving its underlying sequence attributes. This leads to faster training of the second network, a deep convolutional neural network (CNN) that receives encoded proteins and predicts their interaction under three different scenarios. In each scenario, the deep CNN predicts the class of a given encoded protein pair. Each class indicates different ranges of confidence scores corresponding to the probability of whether a predicted interaction occurs or not. The proposed framework features significantly low computational complexity and relatively fast response. The contributions of this work are twofold. First, ProtInteract assimilates the protein's primary structure into a pseudo-time series. Therefore, we leverage the nature of the time series of proteins and their physicochemical properties to encode a protein's amino acid sequence into a lower-dimensional vector space. This approach enables extracting highly informative sequence attributes while reducing computational complexity. Second, the ProtInteract framework utilises this information to identify protein interactions with other proteins based on its amino acid configuration. Our results suggest that the proposed framework performs with high accuracy and efficiency in predicting protein–protein interactions.

Crown Copyright © 2023 Published by Elsevier B.V. on behalf of Research Network of Computational and Structural Biotechnology. This is an open access article under the CC BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>).

1. Introduction

A *protein* is a macromolecule constituted by chains of amino acids, with side-chain groups extending from a connected backbone [1]. Protein–protein interactions (PPI) play a vital role in almost every intracellular biological and biochemical activity [2,3,4]. Most proteins execute their functions by interacting with other proteins. These functions include nutrient transport, enzymatic reactions,

immune response, communication between cells, and tissue regeneration. Thus, accurately identifying PPIs is essential for understanding cell physiology [5]. We now understand the close relationship between abnormal PPIs and a range of pathologies such as neurodegenerative conditions, cancer, and infectious diseases [6,7,8]: identifying functional PPIs has proved to be a turning point in how technology can instaurate the foundations for target, and individualised therapies [9,10,11].

Many have addressed the PPI identification problem via experimental methods such as yeast two-hybrid screening [12,13,14], mass spectrometry [15,16,17,18,19], protein microarrays [20,21,22,23]. These approaches are often costly and time-consuming [24,25,26]. Computational methods aim, therefore, to reduce false-positive and

* Corresponding author.

false-negative results by providing preliminary testing prior to the experiments [27,28,29].

Recent advances in deep learning frameworks, fuelled by the large amount of data available for training, have revolutionised scientific studies [30]. In particular, it has become possible, to some extent, to predict PPI [31,32].

Nevertheless, the performance of deep learning algorithms is affected by multiple factors such as network structure, which may involve multiple sub-networks and the feature selection mechanism along with the activation and pooling functions, noise, and imbalanced datasets [33,34]. To handle problems such as noise, wavelet transforms have been employed [35], while more informative representations and dimensionality reduction have been implemented using methods such as Boltzmann machines and autoencoders [36]. For problems with high feature dimensionality, learning can be optimised by extracting uncorrelated features while keeping highly informative features using convolutional and stacked autoencoders [33,34,35,37]. In biological contexts, modelling protein structures includes prediction from the amino acid sequence, designing proteins with specific functionalities, and predicting properties or behaviour of a protein, which require a broad knowledge at the molecular level [38].

Deep learning architectures for PPI prediction may operate on different inputs, including primary structure [39,40,41,42], domain composition [43,44], gene expression [45,46,47], and the proteins' 3-D structure [48], to mention a few. It is often believed that 3-D structure delivers a relatively more complete set of information for PPI prediction [49]. However, all the vital information required to identify PPIs is encoded in the amino acid sequence, i.e. the primary structure [50], so sequence-based methods have gained particular attention [51]. DeepPPI [52], for example, aims to improve PPI prediction using a variety of statistical descriptors to learn the protein characteristics. TransPPI [53] captures latent patterns in the protein sequence using a Siamese-like architecture of convolutional layers. This method predicts interaction probabilities by concatenating a pair of protein representations from two identical sub-networks. CAMP is another PPI prediction method based on convolutional layers; it receives an integration of multiple features such as primary and second structures, physicochemical properties, and protein evolutionary information as input [54]. This method includes a self-attention layer to learn the long-range dependencies between residues in protein sequences. The D-SCRIPT framework [55] comprises a pre-trained, bidirectional long short-term memory (BiLSTM) model [56] that generates structurally informative features for each protein and a projection module to reduce the dimensionality of embeddings. These low-dimensional embeddings include the protein's residue contact map; these maps are used to predict the interaction probabilities. Another convolution-based framework, called DeepTrio [57], performs binary PPI prediction by employing multiple parallel convolutional neural networks. PIPR [58] is a sequence-based framework which integrates a convolution-pooling layer and bidirectional residual gated recurrent units (GRU) [59], extracting the local features and global sequential information of proteins, and thus capturing the mutual effects of protein pairs in PPI prediction tasks. S-VGAE [60] uses a variational graph autoencoder [61] to learn the latent features of proteins. A PPI identification method using GCNs [5] integrates the one-hot encoding representation of protein sequences and the positional information of each protein in the PPI network.

In this paper, we propose a sequence-based method called ProtInteract that predicts protein–protein interactions through two main tasks: encoding and prediction. A novel autoencoder architecture encodes each protein to remedy the computational complexity posed by the high dimensionality of protein representation while extracting sequential attributes of the protein's primary

structure. This autoencoder is composed of 2-D convolutional and temporal convolutional network (TCN) layers (see Section 5.1).

ProtInteract is trained on the STRING database, which contains PPI datasets for various organisms. In each dataset, protein pairs are associated with an interaction score which shows the confidence of STRING assessment on an interaction being valid, given the available evidence [62,63]. Those interaction scores are the basis for ProtInteract's prediction process. These interaction scores are then normalised between zero (non-interacting pairs) and one (interacting pairs with the highest confidence). We formulate the prediction task as three different multiclass classification scenarios comprising, respectively two, three and five classes [64]. For the two-class scenario, interactions are divided into non-interacting ($0 \leq \text{interaction score} \leq 0.5$) and interacting pairs ($0.5 < \text{interaction score} \leq 1$). For the three-class scenario, a range for erroneous results ($0.4 < \text{interaction score} \leq 0.7$) is considered as well as non-interacting ($0 \leq \text{interaction score} \leq 0.4$), and interacting pairs ($0.7 < \text{interaction score} \leq 1$). In the STRING database, an interaction score of 0.5 may suggest that, on average, every second interaction is labelled incorrectly (a false positive) [63, 65, 66]; we consider a wider span for such interactions. The final scenario comprises five classes, where the non-interacting and interacting spans are divided into high- and low-confidence spans (see Table 4), to define five different classes. In each scenario, a deep convolutional neural network (deep CNN) predicts the probability of correctly classifying an interaction. Additionally, many other researchers employed an LSTM architecture as the baseline model when dealing with time series and sequential datasets [67,68,69,70], which inspired to leverage LSTM as a baseline model to demonstrate the comparative advantages of the TCN architecture. The rest of the paper is organised as follows: In Section 2, some of the most recent methods for addressing the PPI problem are briefly discussed. Section 3 outlines our proposed framework. The novel autoencoder architecture will be presented in Section 5.1, and the baseline model in section 10.2. The multi-class classification scenarios are discussed in Section 6. The architecture of the deep CNN is presented in Section 7. Section 8 will discuss the experimental results, through the examples of *Homo sapiens* and *Mus musculus*. Finally, Section 9 concludes this work.

2. PPI prediction

Among the computational methods developed for PPI prediction, some aim to extract new features from protein information while others use the extracted features to learn new models [71]. For instance, the approach proposed by Guo et al. [72] to obtain non-interacting protein pairs has been used to develop a SVM-based method in [41], and a PIPR method in [58]. A multimodal approach based on LSTM is proposed in [71], which predicts PPI by integrating structural and sequential information of proteins as additional features to the input feature set. The physicochemical attributes of amino acids, such as hydropathy [73], isoelectric point, and charge, play crucial roles in identifying interactions between protein sequences [74]. The PPI prediction is conducted based on sequences in [75] by defining units of three adjacent amino acids and measuring the frequencies of those units in a protein sequence. Other methods such as amino acid index distribution [76], the conjoint triad method [75], and autocovariance [72] extract features such as locations of amino acids, frequencies and physicochemical properties aiming to represent a protein sequence.

In order to reduce the high dimensionality of protein structures, several techniques have been developed, such as random forest [41] and support vector machine (SVM) and its derivatives [77,78]. Another SVM-based approach named ACT-SVM [79] extracts features from protein sequences for input vector to the classifier. A sequence-based human PPI prediction, proposed in [80], is based on a stacked autoencoder (SAE). Another sequence-based PPI prediction

approach, Deep Sequence Contact Residue Interaction Prediction Transfer (D-SCRIPT) [81], models protein structure using the pre-trained language model in [56]. A heterogeneous network is used for PPI prediction in [82]. They represent proteins by concatenating local and global features. They used a k -mer method¹ [83] to extract local features. To extract global features, LINE (Large-scale Information Network Embedding) is employed [84]. Finally, a random forest classifies the potential protein pairs.

A deep learning method for PPI prediction, called ordinal regression and recurrent convolutional neural network (OR-RCNN) [85], comprises two recurrent convolutional neural networks (RCNNs) to extract local features and sequential information from the protein pairs, and ordinal regression to construct multiple sub-classifiers. The viability of these techniques is yet to be verified experimentally. Therefore, to understand the potential of generative models in protein engineering, more needs to be done, such as analysing the strengths and limitations of different models and the possibilities for incorporation into existing engineering operations. First and foremost, representation of the input to the network is a matter of great importance [86]. Given the high dimensionality of protein sequence datasets—which include amino acid physicochemical features and large chains of amino acids—deep learning methods may suffer from extended training times. On the other hand, reducing the number of amino acid features may lead to underfitting [87,88].

We propose, here, a framework for learning a meaningful representation of protein sequence while reducing its dimensionality using a novel autoencoder. PPIs are characterised based on their associated score from the STRING database. This score, a value from zero and one, expresses the confidence of an interaction existence based on the available evidence. A score of zero indicates the highest confidence that a pair of proteins do not interact. In contrast, interaction with the highest confidence is characterised by a score of one. In this work, we composed the PPI prediction as three separate classification problems by defining various ranges of interaction scores. The first scenario is a classification with two classes, where the interaction score range is bisected, indicating only non-interacting and interacting pairs (Table 2). The second and third scenarios designate multiple levels of confidence: a score in (0.4,0.7] indicates medium confidence, which may include false-positive samples, with the extremes indicating one or two levels of confidence in the interacting and non-interacting pairs (Tables 3, 4). The concept of defining three scenarios for ProtInteract is rooted in the idea of identifying high-affinity protein-protein interactions (PPIs) with scores within the range of (0.9,1] (the probability of interactions falling within the most extreme confidence ranges). This expands the potential applications of ProtInteract to include the identification of protein-ligand pairs [89]. Furthermore, by establishing these scenarios, we are able to eliminate uncertainty present in the datasets, specifically regarding interactions falsely labelled as positive within the range of [0.4,0.7]. This allows for more accurate classification of interactions within higher confidence ranges. These scenarios will be discussed in more detail in Section 6.

3. ProtInteract outline

Most protein design problems require profound knowledge to analyse difficulties and obtain optimal design [90]. However, the emergence of deep neural networks has provided the computational capacity to exploit the rich reserves of existing historical data. The proposed framework comprises two deep neural networks, each responsible for one task. First, a deep learning architecture reduces the input data size by extracting meaningful sequential patterns

from the protein sequence. Second, a deep learning architecture performs the PPI prediction task using the extracted features.

For the first task, we propose two different architectures, namely a convolutional-TCN autoencoder and a convolutional-LSTM autoencoder, and compare their accuracies by using the latter as a baseline model. The main reason for this autoencoder framework is to reduce the dimensionality and to remove redundancies from the input tensor, which accelerates the training of the following network for PPI prediction. The intuition behind adopting TCN and LSTM in the framework is their ability to extract sequential patterns. As first introduced in [91], TCN allows a hierarchical, bi-level analysis of sequential data such as time series [92]. TCNs tend to outperform and have a longer long-term memory than recurrent neural networks (RNNs) such as LSTM and Gated Recurrent Units (GRU) when sequential data are involved [93,94]. Here, we show the advantage of using TCN over LSTM in our architecture.

In the following step, a deep convolutional neural network (CNN) architecture receives encoded proteins and predicts their interaction according to the three defined different scenarios: two, three and five classes, according to Tables 2, 3 and 4, respectively. For each scenario, the deep CNN predicts the class of each interaction given the encoded protein pairs. A schematic representation of these steps is shown in Fig. 1. The network structure is described in greater detail below.

The following section discusses our approach to encoding proteins using the physicochemical properties of their amino acids.

4. Protein representation

The first step toward developing a deep learning algorithm for PPI prediction is preprocessing the datasets to encode the categorical amino acids into a set of numerical values [95]. A common technique is one-hot encoding [95,96], which may not be well-suited to problems with high cardinality, such as encoding proteins of significant length. For example, for a dataset comprising the twenty standard amino acids and an unknown amino acid (X), Alanine (A) might be represented as [10000000000000000000] using one-hot encoding, which is a simple vector with twenty zeros. Here, one-hot encoding results in sparse vectors with many non-informative zeroes representing a protein of some length [97,98]. Such a high proportion of zeroes in the encodings results in a neural network that struggles to learn because it observes mostly zero-valued inputs. To eliminate this difficulty, each amino acid is first characterised by ten physicochemical features, as reported in Table 1. These features are highly informative and represent various aspects of the protein. For inter-protein bindings to form, amino acids should chemically complement each other, allowing hydrophobic, polar, or charged interactions [99]. The hydropathy properties of amino acids, i.e., hydrophobicity and hydrophilicity, are the dominant properties of protein-protein binding [100,101]. The helix and sheet probabilities are two statistically driven parameters [102]. The protein's isoelectric point pI (i.e., the pH at which its net charge is zero), its side-chain net charge number NCN [103,104], and its solvent-accessible surface area (SASA) are key factors in protein stability and folding studies [105]. The features of unknown amino acids (X) are calculated by the median of each known feature from twenty standard amino acids. With the varied ranges of the physicochemical features, min-max normalisation is performed for each feature reported in Table 1. These ten attributes have proven efficient in determining the potential for protein interactions.

Given a protein dataset, such as that of *Homo sapiens*, the number of amino acids in each protein may be substantially different. For instance, TRP-Cage protein [PDB code: 2M7D] has the shortest length in the *H. sapiens* dataset, comprising twenty amino acids, while Titin² is

¹ In an amino acid sequence of length N , any $k < N$ consecutive amino acids are called a k -mer. These are collected using a sliding window of length $1 \leq k \leq N$.

² <https://pdb101.rcsb.org/motm/185>

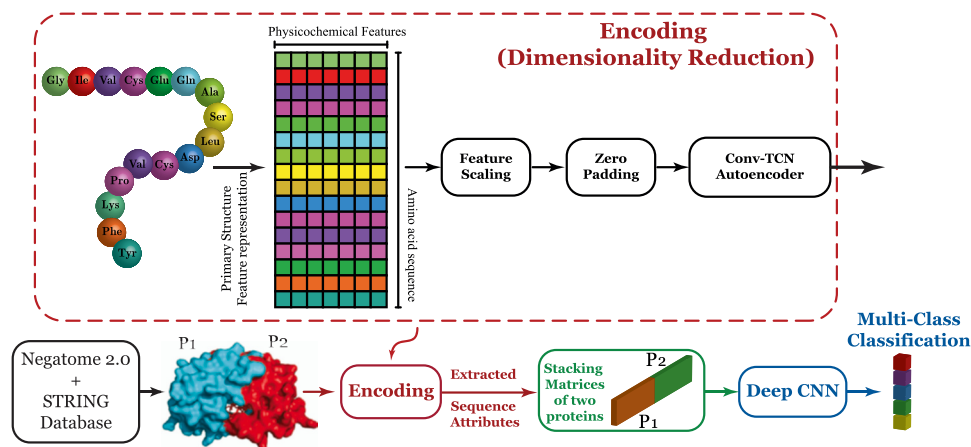


Fig. 1. The ProtInteract framework comprises a convolutional TCN autoencoder that extracts highly informative sequential patterns encoded by each protein and lowers its dimension. A deep CNN predicts the probability of an interaction falling within the range of a defined class. The two proteins form a complex, which is represented in this architecture by the stacking of the two proteins.

Table 1

Physicochemical properties of 20 amino acids: a) volume; b) isoelectric point; c) helix probability; d) sheet probability; e) hydrophobicity; f) hydrophilicity; g) polarity; h) polarisability) [102,104,106,107].

Amino acid	Symbol	a	b	c	d	e	f	g	h	SASA	NCN
Alanine	A	1.00	6.11	0.42	0.23	0.62	-0.50	8.10	0.046	1.181	0.007187
Cysteine	C	2.43	6.35	0.17	0.41	0.29	-1.00	5.50	0.128	1.461	-0.03661
Aspartate	D	2.78	2.95	0.25	0.20	-0.90	3.00	13.00	0.105	1.587	-0.02382
Glutamate	E	3.78	3.09	0.42	0.21	-0.74	3.00	12.30	0.151	1.862	0.006802
Phenylalanine	F	5.89	5.67	0.30	0.38	1.19	-2.50	5.20	0.290	2.228	0.037552
Glycine	G	0.00	6.07	0.13	0.15	0.48	0.00	9.00	0.000	0.881	0.179052
Histidine	H	4.66	7.69	0.27	0.30	-0.40	-0.50	10.40	0.230	2.025	-0.01069
Isoleucine	I	4.00	6.04	0.30	0.45	1.38	-1.80	5.20	0.186	1.810	0.021631
Lysine	K	4.77	9.99	0.32	0.27	-1.50	3.00	11.30	0.219	2.258	0.017708
Leucine	L	4.00	6.04	0.39	0.31	1.06	-1.80	4.90	0.186	1.931	0.051672
Methionine	M	4.43	5.71	0.38	0.32	0.64	-1.30	5.70	0.221	2.034	0.002683
Asparagine	N	2.95	6.52	0.21	0.22	-0.78	2.00	11.60	0.134	1.655	0.005392
Proline	P	2.72	6.80	0.13	0.34	0.12	0.00	8.00	0.131	1.468	0.23953
Glutamine	Q	3.95	5.65	0.36	0.25	-0.85	0.20	10.50	0.180	1.932	0.049211
Arginine	R	6.13	10.74	0.36	0.25	-2.53	3.00	10.50	0.291	2.560	0.043587
Serine	S	1.60	5.70	0.20	0.28	-0.18	0.30	9.20	0.062	1.298	0.004627
Threonine	T	2.60	5.60	0.21	0.36	-0.05	-0.40	8.60	0.108	1.525	0.003352
Valine	V	3.00	6.02	0.27	0.49	1.08	-1.50	5.90	0.140	1.645	0.057004
Tryptophan	W	8.08	5.94	0.32	0.42	0.81	-3.40	5.40	0.409	2.663	0.037977
Tyrosine	Y	6.47	5.66	0.25	0.41	0.26	-2.30	6.20	0.298	2.368	0.023599

composed of more than 34,000 amino acids, thus making it difficult to represent to the neural network. Since nearly 98% of *H. sapiens* and *M. musculus* sequence datasets from the STRING database are proteins with a length of 2048 amino acids or fewer; we ignore the longer proteins in this paper with only a small loss of content. Additionally, to equalise the variety of lengths, all protein encodings are padded, with zeroes, to 2048 units. Consequently, each protein may be encoded as a tensor with the unwieldy dimensionality of 2048×10 , requiring slow and computationally expensive training [33,36].

The underlying sequential structure and orientation of proteins are dictated by the well-defined order of amino acids in the peptide chain: the amino end of each amino acid (its N-terminus) links to the carboxyl end (C-terminus) of the growing peptide chain [108,109], and the position of each amino acid along a specific chain determines the chain's final three-dimensional configuration and its binding ability [110]. The arrangement of amino acids along a protein determines a causal directionality which is contrary to an assumption on which traditional neural networks are based, viz. the independence of all the input vector units [35,111]. Thus, traditional neural networks lack the ability to capture sequential information. For this reason, the recurrent neural network (RNN) and its variants, such as the gated recurrent unit (GRU) [59], and LSTM [112,113]

incorporate hidden states which are generated by sequential information, thus capturing the underlying distribution of such datasets [91,114,115]. However, these methods are often computationally expensive, and training is slow [93,116,117]. This led to the development of the temporal convolutional network (TCN) for sequence modelling based on dilated causal 1-D convolutional layers [93]. TCN outperforms LSTM in a diverse range of tasks and datasets [93]. Owing to those advantages, we employ TCN layers as the auto-encoder architecture's main framework, and we propose a baseline model using LSTM layers for comparison (see Section 10). The TCN architecture is explained in the following section.

4.1. Temporal convolutional network (TCN)

The many advantages of LSTM come at a high computational cost, consequently, and it is slow to train [93,116]. We overcome these difficulties using the temporal convolutional network.

The TCN architecture leverages two basic principles. Firstly, there is no information leakage from future to past, owing to causal directionality in the convolution, i.e. for an input sequence with length t as $\{0, \dots, t-1\}$ the i^{th} element of the output sequence depends only on the elements that precede it in the input sequence [118], i.e. the

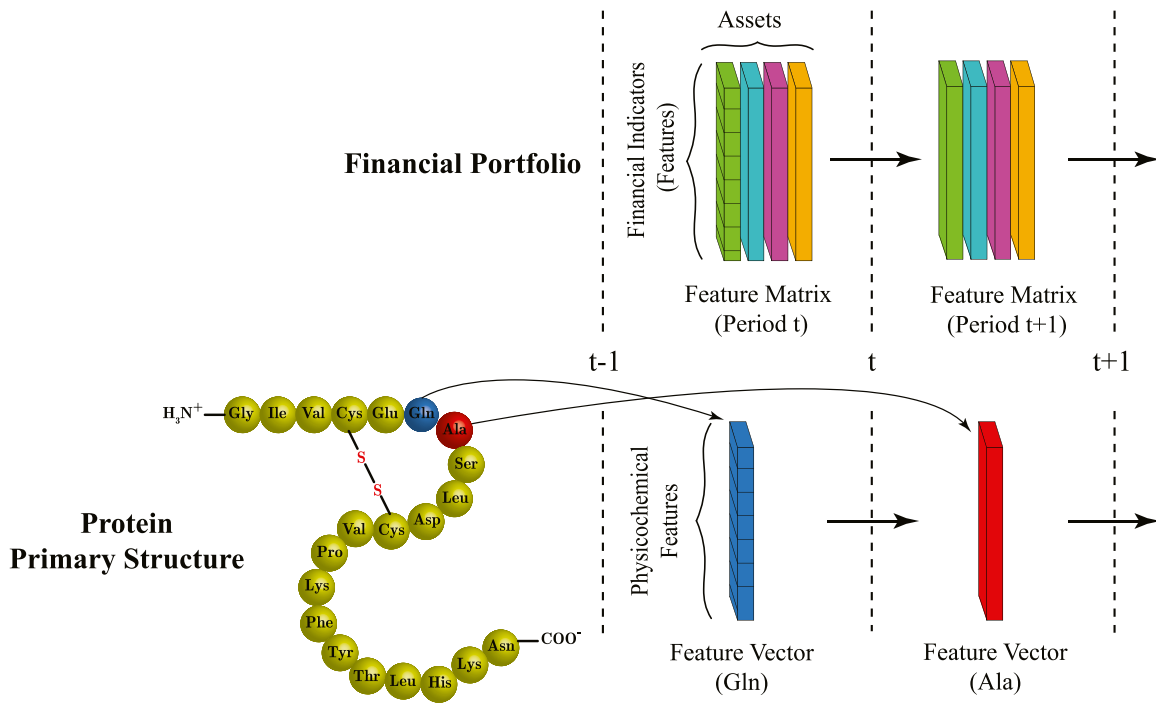


Fig. 2. Proteins may be represented as pseudo-time series such as financial instruments in stock market.

input sequence with indices $\{0, \dots, i\}$. The unique bond directionality (i.e. sequentiality) in the peptide backbone imposed by the N-terminus to-C-terminus translation of amino acids is analogous to that of time-series data [108,109,119]. Therefore, one may consider proteins as pseudo-time series as illustrated in Fig. 2, where the time is a proxy for the amino acid positions in the polypeptide chain, which determines the functionality and properties of that protein [38].

Second, the TCN architecture can map an input sequence of any length to an output sequence with the same length, just as RNNs can. To accomplish this, the TCN employs a 1-D fully convolutional architecture, in which the input and hidden layers have the same length.

In a simple causal convolution network, the size of the receptive field determines how far into the past the network can access. This size grows linearly with the network depth. This precludes causal convolution from being applied to sequential tasks requiring a long

history. To avoid this obstacle, the TCN architecture uses dilated convolutions that provide an exponentially large receptive field [120]. The dilated convolution operator $F(s)$ on element s of a 1-D input sequence $x \in \mathbb{R}^n$ and filter $f: \{0, \dots, k-1\} \rightarrow \mathbb{R}$ is defined as

$$F(s) = (x *_{d} f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-i \cdot d}$$

where d is the dilation factor, k is the filter size, and $s - i \cdot d$ indicates the direction of the past [93]. In addition to network depth, the receptive field depends on filter size k and the dilation factor d [118], so, to expand the receptive field, one may increase the dilation factor, which includes a wider range of inputs, i.e. it captures long range sequential patterns [118,121]. Therefore, the dilation factor adjusts the effective history of the TCN. Fig. 3 depicts a TCN with three dilated causal convolutional layers.

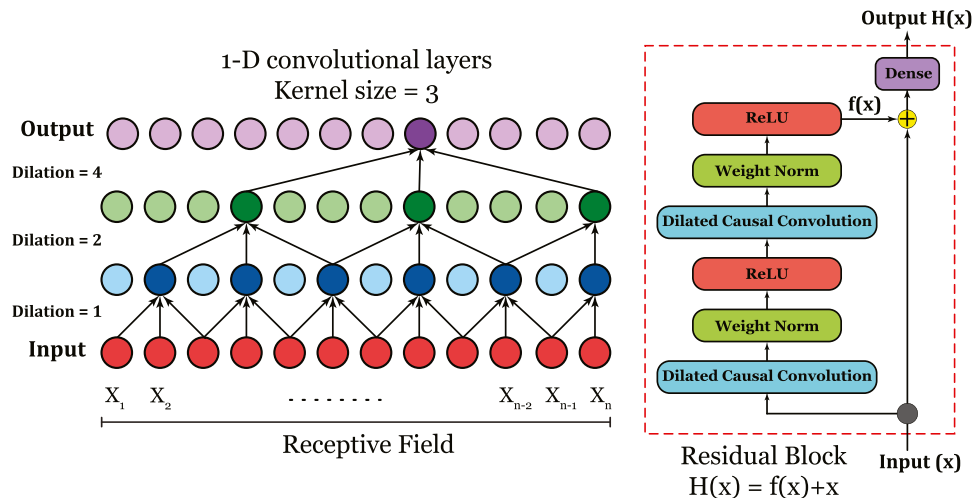


Fig. 3. a) A temporal convolution with dilation factors $d = 1, 2, 4$ and a filter size of $k = 3$; b) Architecture of the residual block associated with the TCN.

The network depth is essential for learning robust representations, but it leads to vanishing gradients. In order to circumvent this issue and build deep networks, the residual block has been developed [93,122]. Residual blocks are employed between TCN layers, allowing for training much deeper models and accelerating convergence [123]. Essentially, as the name suggests, the residual blocks learn residual mapping with reference to the input. The intuition behind such a design is that optimising the residual mapping is easier than optimising the desired underlying mapping (mapping function from input to output) [122]. Let $H(x)$ and $F(x)$ respectively denote the desired underlying mapping and residual mapping. We may represent the original mapping as $H(x) = F(x) + x$ for input x . If identity mapping is desired, as shown in Fig. 3(b), it would be easier to push the residual mapping $F(x)$ to zero rather than to fit an identity mapping by a stack of nonlinear layers [122,124]. The residual module for TCN consists of a dilated causal convolution, followed by weight normalisation [125] and a nonlinear activation function (a rectified linear unit, ReLU) [126].

In the RNN timeline, the later frames are computed once their predecessors are complete. In contrast, convolutional networks can perform the computations in parallel owing to the independent use of the same kernel (or similar kernel) in each layer repeatedly [117]. TCN offers a flexible receptive field that may be expanded to cover the entire sequence length and captures longer effective history by integrating dilated convolutions and residual layers [121]. TCNs tend to have a more extended long-term memory than RNNs, making them more suitable for capturing long-range dependencies as a common binding mechanism [93, 94, 127, 128].

Given the high-dimensional tensor representing each protein sequence and the size of protein datasets, training a neural network may be slow and computationally expensive [33, 36, 117]. To address this issue, a novel autoencoder framework is proposed, which extracts informative abstract features that characterise each protein in a lower dimension.

In the next section, the fundamentals of autoencoder structure are discussed.

5. Autoencoder

Autoencoders, as depicted in Fig. 4, are a type of unsupervised, feedforward neural network that reconstructs the output from the input [129]. They consist of two parts, namely, the encoder and the decoder. The encoder maps the input data into a lower-dimensional space in the so-called latent layer (latent representation), while the decoder reconstructs the input data from the latent layer; such a network is said to be under-complete because of the constriction that the latent layer imposes [130]. Autoencoders are capable of learning nonlinear dimension reduction functions, removing redundancies and correlations while extracting highly informative features [130]. The input layer represents a high-dimensional and possibly correlated feature space, and this network is used to extract low-dimensional and uncorrelated features via the latent layer (the encoder output).

The intention behind using autoencoders is to transform the high-dimensional input tensor into a low-dimensional, informative uncorrelated feature [33,34].

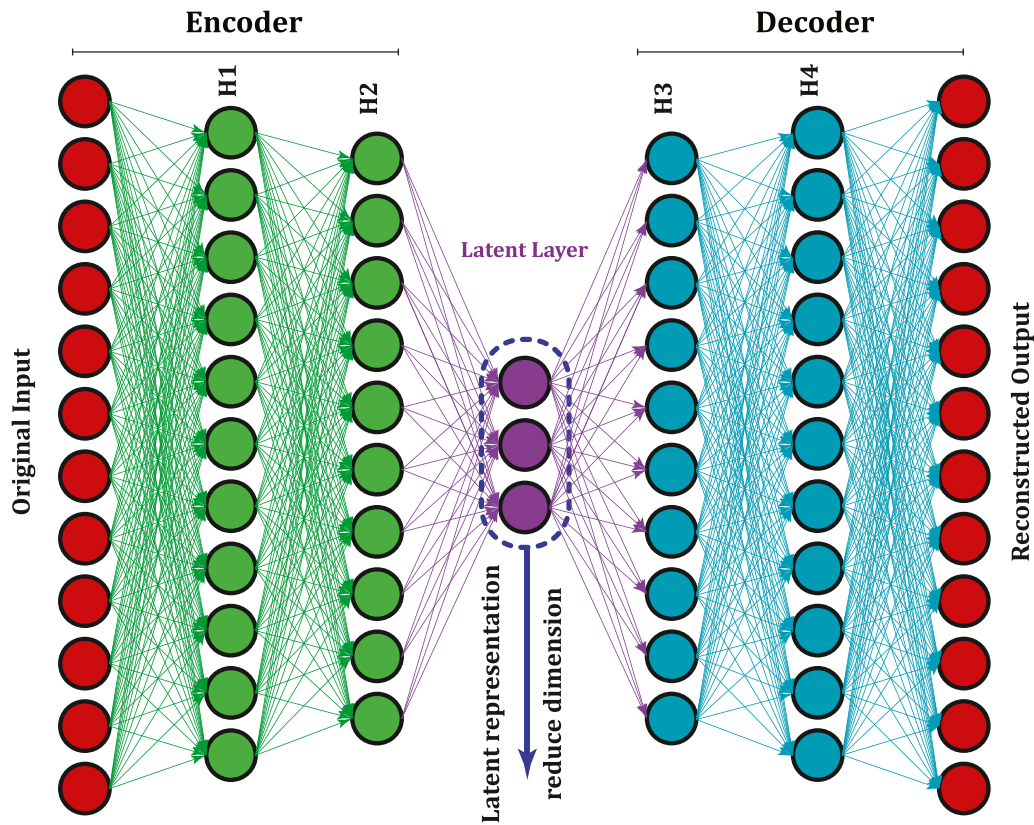


Fig. 4. Autoencoder architecture.

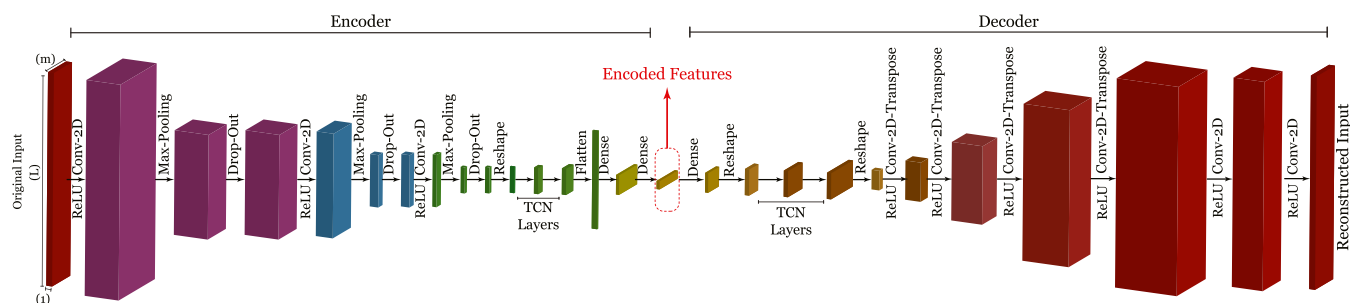


Fig. 5. Conv-TCN-AE architecture that reduces the dimension of input data and extracts informative and uncorrelated features.

The proposed autoencoder architecture is composed of 2-D convolutional layers that reduce the input tensor dimension, while TCN layers extract sequential patterns. These patterns are extracted from the latent layer and fed into the deep CNN to predict the probability of PPI (interacting vs non-interacting). Let us now examine the proposed autoencoder architecture more closely.

5.1. Convolutional TCN Autoencoder

The architecture of the convolutional TCN autoencoder (Conv-TCN-AE) is shown in Fig. 5, comprising an encoder and a decoder. This architecture is inspired by convolutional autoencoders, which learn the low-dimensional representation of the input tensor [131,132]. However, the standard convolutional layer fails to comply with causal directionality so to extract the underlying sequential attributes encoded by each protein and long-range dependencies, TCN layers are employed after the 2-D convolutional layers [94].

Initially, the encoder receives a tensor of 2048×10 dimensions, representing a protein sequence plus the physicochemical features of its amino acids. This tensor is passed through 2-D convolutional and max-pooling layers to reduce its dimensionality. The dropout

layers are employed to reduce over-fitting and to improve generalisation error [133]. Since the TCN layers are based on causal 1-D convolutional layers, a Reshape layer is placed between the 2-D convolutional layers and TCN layers. Next, the densely connected layer extracts these features with the desired encoding size. The encoder generates a fixed-sized vector in the latent layer, which captures the informative sequential attributes delivered by the input tensor. These latent features may provide information on the short- and long-range interactions that may occur after protein folding. These features will represent a protein in a lower-dimensional space, which reduces the computational complexity of predicting interactions.

Following encoding, the decoder reconstructs the input data from the latent features. The features are transformed into a 2-D tensor using a dense layer and a Reshape layer. Next, two TCN layers and then a Reshape layer transform the tensor to 3-D. Then, four 2-D transposed convolutional layers restore the original input tensor. Two 2-D convolution layers fully reconstruct the input with its original dimensions. The pseudocode of this architecture is presented in Alg. 1.

Algorithm 1. Convolutional-TCN Autoencoder [134].

Input: Encoded sequence tensor= $S_t [L \times m \times 1]_{t=1}^T$, encode-size, input-size, num-features

Output: Latent layer - Informative pattern of sequence of amino acids in a protein,

Initialization:

1. **Encoder:**

- **2-D-convolutional layer:**{act-fcn=ReLU, kernel-size=(2,2), num-filters=64}
- **Max Pooling layer:**{Pool size=(2,2), Padding="same"}
- **Dropout Layer:** [0.3]
- **2-D-convolutional layer:**{act-fcn=ReLU, kernel-size=(2,2), num-filters=32}
- **Max Pooling layer:**{Pool size=(2,2), Padding="same"}
- **Dropout Layer:** [0.3]
- **2-D-convolutional layer:**{act-fcn=ReLU, kernel-size=(2,2), num-filters=16}
- **Max Pooling layer:** {Pool size=(2,1), Padding="same"}
- **Reshape:** (to 2-D tensor)
- **TCN layer:**{num-filters=16, act-fcn=ReLU, kernel-size=3, dilations=[1,2,4]}
- **TCN layer:**{num-filters=8, act-fcn=ReLU, kernel-size=3, dilations=[1,2,4]}
- **Flatten**
- **Dense layer:**{units=2×encode-size, act-fcn=ReLU}
- **Dense layer:**{units=encode-size, act-fcn=ReLU, layer weight regularizers (kernel-bias-activity)}
- *Encoder Output:* Latent layer - low dimensional, highly informative patterns.

2. **Decoder:**

- **Dense layer:** {units=encode-size}
 - **Reshape** (to 2-D tensor)
 - **TCN layer:** {num-filters=4, act-fcn=ReLU, kernel-size=3, dilations=[1,2,4]}
 - **TCN layer:** {num-filters=num-features, act-fcn=ReLU, kernel-size=3, dilations=[1,2,4]}
 - **Reshape layer:** {Extend dimension to 3-D tensor}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=16, Strides=(2,1)}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=32, Strides=(4,1)}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=64, Strides=(4,1)}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=128, Strides=(4,1)}
 - **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=8, Padding="same"}
 - **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=1, Padding="same"}
 - *Decoder Output:* Reconstructed input vector
 - *Compiler:* {**Optimiser:** Adam [134], **Loss:** Mean Squared Error}
-

6. Classifying interactions

The datasets used in this work are retrieved from the publicly available databases STRING [63] and Negatome 2.0 [135]. The

Negatome 2.0 includes experimentally curated negative (non-interacting protein pairs, i.e., with the lowest confidence score of zero for the interaction) and positive interactions (pairs with the highest confidence score of one for the interaction) for various organisms,

such as *Homo sapiens*, *Mus musculus*, *Saccharomyces cerevisiae*, *Escherichia coli*, and *Drosophila melanogaster*. The STRING database includes a broader range of organisms compared to Negatome 2.0, such as *Caenorhabditis elegans*, *Danio rerio*, and *Arabidopsis thaliana*. To facilitate access to these datasets, we provided a [GitHub repository](#) with all data incorporated into this study. This repository includes the Negatome 2.0 datasets for *Homo sapiens* and *Mus musculus* (231 KB) and the protein sequence datasets with the amino acid sequence of each protein for *Homo sapiens* and *Mus musculus* (12.6 MB) used to train the autoencoders and PPI datasets (700 MB), which predict the PPIs.

The protein network dataset retrieved from the STRING database comprises pairs of proteins, and their respective interaction scores computed based on seven evidence channels [62,63], as follows:

1. Experiments channel: evidence originating from laboratory experiments (including biophysical, biochemical and genetic experiments). The primary interaction databases organised in the IMEx consortium [136,137], plus BioGRID [138], are the main source for this channel.
2. Database channel: evidence that has been confirmed by a human expert curator using information imported from pathway databases [139].
3. Text mining channel: evidence collected by searching through all PubMed abstracts, through an in-house collection of more than three million full-text articles, and through other text collections related to proteins [140,141]. An association score is defined based on how frequently a pair of proteins are mentioned in the same paper. When parsing one or more sentences via natural language processing, if a concept connecting the two proteins is discovered (such as ‘binding’ or ‘phosphorylation by’), the association score is increased.
4. Co-expression channel: for this channel, gene expression data resulting from various expression experiments are normalised, pruned and then correlated [62,63]. A high association score is assigned to protein pairs with consistent similarity in their expression patterns under different conditions.
5. Neighbourhood channel: this is a genome-based prediction channel where an association score is given to protein pairs consistently observed in each other’s genome neighbourhood. This and the next two channels are generally most relevant for Bacteria and Archaea.
6. Fusion channel: an association score is given to a protein pair when there are one or more organisms whose respective orthologs have fused into a single, protein-coding gene.
7. Co-occurrence channel: in a given organism, the phylogenetic distribution of orthologs of all proteins is evaluated. An association score is assigned if two proteins show high resemblance in this distribution, i.e. their orthologs tend to be observed as ‘present’ or ‘absent’ in the same subsets of organisms [142].

Table 2
Confidence score thresholds for two classes of protein–protein interactions.

Classes	Confidence Score	Description
class 1	$0 \geq - \leq 0.5$	The score threshold is assigned to non-interacting protein pairs. A score of zero suggests a non-interactive pair with the highest confidence
class 2	$> 0.5 - \leq 1$	This range indicates interacting protein pairs based on experimental and theoretical evidence. A score of one indicates full confidence in the protein pair’s interactivity.

Each technique has its own disadvantages leading to biases, false positives and false negatives. The STRING dataset quantifies these uncertainties by allocating scores to proposed protein interactions indicating the probability of the interaction existence according to the nature and quality of the supporting evidence [62,63]. All these scores are integrated into a final “combined score” that reflects the confidence on whether an interaction between two non-identical proteins is biologically meaningful, given all the contributing evidence in the STRING database [63]. Consequently, the interactions are associated with a score between zero and one (zero for non-interacting pairs and one for pairs whose interaction is indicated by strong evidence).

In this work, the prediction of protein–protein interactions is formulated as multi-class classification problems. Three separate scenarios are defined to evaluate the interactions from different perspectives. Firstly, we begin with a common scenario where interactions are divided into interacting and non-interacting pairs. As described in Table 2, interactions with the score within the [0,0.5] range are labelled non-interacting, and those belonging to (0.5,1] range are labelled interacting pairs. In the STRING database, interactions with a score of 0.5 are indicated as false positives, suggesting erroneous labelling for roughly every second interaction [65,66].

The next scenarios take a more nuanced approach: interactions scoring 0.4 in STRING are considered mediumconfidence, and interactions scoring above 0.7 are high-confidence interactions (i.e. more likely to interact, based on the available evidence) [62, 63, 65]. We hence propose a score range of (0.4,0.7], indicating medium confidence, for the second and third scenarios.

This mid-range may include falsely labelled (false-positive) pairs. The second scenario comprises three classes – non-interacting, interacting, and the aforementioned medium-confidence interactions, as shown in Table 3.

Finally, the third scenario comprises five classes. The non-interacting and interaction ranges from the second scenario are further subdivided into two, as reported in Table 4. The interactions scoring in [0, 0.15] are noninteracting with the highest confidence, while those in (0.15, 0.4] are non-interacting with relatively less confidence. At the upper end, (0.7, 0.9] and (0.9,1] represent interacting pairs with high and highest confidence, respectively [65, 66, 143]. One advantage of defining these thresholds is that one may find high-affinity PPIs among those protein pairs with interaction scores in (0.9,1] range, which extends the applications of ProtInteract to identifying protein–ligand pairs [89].

The next section illustrates the deep CNN architecture.

7. Predicting PPI using a Deep CNN

A deep CNN is designed, as shown in Fig. 6, to predict the probability of an interaction belonging to a class according to three defined scenarios. Initially, each protein is encoded using a Conv-

Table 3

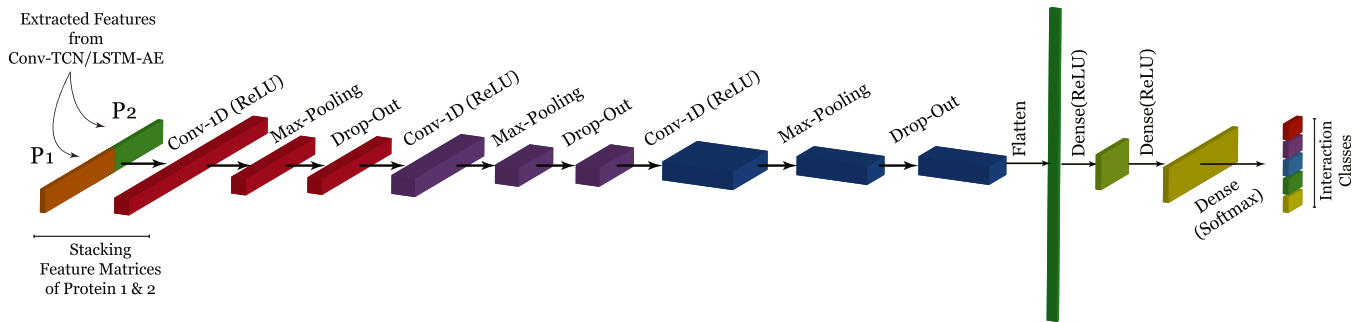
Confidence score thresholds for three classes of protein–protein interactions.

Classes	Confidence Score	Description
class 1	$0 \geq - \leq 0.4$	The interaction confidence threshold is low.
class 2	$> 0.4 - \leq 0.7$	A score in this range may indicate a falsely labelled pair.(for instance, a score of 0.5 suggests that nearly every second interaction may be incorrectly labelled as an interacting pair).
class 3	$> 0.7 - \leq 1$	This range shows highly interactive protein pairs based on experimental and theoretical evidence. A score of one indicates full confidence in the protein pair's interactivity.

Table 4

Confidence score thresholds for five classes of protein–protein interactions.

Classes	Confidence Score	Description
class 1	$0 \geq - \leq 0.15$	The confidence of interaction is very low.
class 2	$> 0.15 - \leq 0.4$	This score range shows low confidence in the occurrence of an interaction between a protein pair.
class 3	$> 0.4 - \leq 0.7$	A score in this range may indicate a falsely labelled pair.
class 4	$> 0.7 - \leq 0.9$	Interaction scores within this range indicate high confidence of interaction between two proteins.
class 5	$> 0.9 - \leq 1$	This range indicates highly interactive protein pairs, based on experimental and theoretical evidence.

**Fig. 6.** Deep CNN architecture outputs the interaction probabilities in the defined classes. The output dimension changes for each scenario, where we have two neurons (two classes) for the first scenario, three neurons (three classes) for the second scenario and five neurons (five classes) for the final scenario.

TCN-AE (in ProtInteract) or Conv-LSTM-AE (for the baseline model). Then, protein pairs are constructed by stacked encoded proteins in tandem, as depicted in Fig. 1, to create the input tensor for the deep CNN.

The CNN architecture is described in Alg. 2. The CNN comprises three 1-D-convolutional layers, max-pooling layers to downsample the tensor, dropout layers to avoid overfitting and data leakage, and dense layers to map the tensor from the preceding layer to a desirable size. Finally, the output is obtained from a densely connected

layer using a softmax activation function, which yields the probability of an interaction falling within the defined range of classes for each scenario according to Tables 2, 3 and 4. As described above, the first scenario entails a simple yes–no prediction of whether a protein pair is interacting or non-interacting. The second scenario includes a medium-confidence range in addition to interacting and non-interacting pairs. Finally, in the third scenario, the deep CNN outputs five values indicating the probability of given interaction belonging to each of the classes described in Table 4.

Algorithm 2. CNN – PPI prediction [134].

Input Protein interaction pairs, *num_cls* : Number of classes

Output PPI-prediction

Initialisation:

- **1-D-convolutional layer:** {num-filters=64, act-fcn=ReLU, kernel-size=3}
- **Max Pooling layer:** {Pool size=2, Padding="same"}
- **Dropout Layer:** [0.25]
- **1-D-convolutional layer:** {num-filters=32, act-fcn=ReLU, kernel-size=3}
- **Max Pooling layer:** {Pool size=2, Padding="same"}
- **Dropout Layer:** [0.25]
- **1-D-convolutional layer:** {num-filters=16, act-fcn=ReLU, kernel-size=3}
- **Max Pooling layer:** {Pool size=2, Padding="same"}
- **Dropout Layer:** [0.2]
- **Flatten layer**
- **Dense layer:** {units=16, act-fcn=ReLU}
- **Dense layer:** {units=8, act-fcn=ReLU}
- **Dense layer:** {units=5, act-fcn=ReLU}
- **Dense layer:** {units=*num_cls* :, act-fcn=Softmax, layer weight regularizers {kernel-bias-activity}}
- **Output**
A score shows the probability of an interaction belonging to each of the classes.
- **Compiler:** {**Optimiser:** Adam [134], **Loss:** sparse categorical cross entropy}

8. Experimental results

Training the proposed framework requires two types of datasets: protein sequences and protein–protein interactions. In the first stage, dimensionality is reduced on a protein sequence dataset, extracting an abstract representation while preserving the sequential structure information in each protein. This is followed by encoding the PPI dataset and performing PPI prediction based on the scenarios in Tables 2, 3 and 4. The data processing and the training of the neural networks were performed on an HP Apollo System with 40 Dual Intel Xeon Gold 6149 processors, one Nvidia Tesla V100 (16 GB) graphics processor, and 378 GB of memory. The following sections explain these stages further in detail.

8.1. Dimensionality Reduction

Training and evaluation datasets are retrieved from the STRING database, which contains multiple datasets, including protein sequences, protein network data, and association scores between orthologous groups, amongst others. Here, we use the protein sequence to train the networks at the encoding stage and protein network datasets to train the deep CNN. This work is conducted on two species, namely, *Homo sapiens* and *Mus musculus*. The protein sequence dataset comprises protein IDs and their amino acid sequence from multiple species, including those reported in Table 5. Here, we consider proteins with a length of 2048 amino acids or less in order to constrain the overall data dimensionality.

The training hyper-parameters for both Conv-TCN-AE and Conv-LSTM-AE are acquired by grid search for the TCN and LSTM

Table 5

The number of protein sequence instances retrieved from STRING database for *H. sapiens* and *M. musculus*.

Species	Samples
<i>H. sapiens</i>	19542
<i>M. musculus</i>	22024

architectures. Both architectures are optimised in terms of hyperparameters and can be fairly compared. Grid search is an exhaustive method for finding an optimal combination of hyperparameters. This method divides the domain of hyperparameters into a discrete grid and calculates the performance metrics of every combination of hyperparameters using k-fold cross-validation. The part of the grid that maximises the average value in cross-validation yields the optimal combination of hyperparameters [144], as shown in Table 6. The training is conducted based on five-fold cross-validation. Five-fold cross-validation is an evaluation method that helps discover a machine learning model configuration that can best predict the outcome of unseen data. With this approach, the data is randomly shuffled and divided into five subsets of equal size, called "folds". The process is

Table 6

Training and validation hyperparameters for Conv-TCN-AE and Conv-LSTM-AE.

Parameter	Value	Description
Padding Length	2048	Padding length of all amino acid sequences
Epochs	100	Training epochs per protein
Batch Size	16	Size of training batch

repeated five times, each using a different fold as the testing set (20%) and the combined other four folds (80%) as the training set. This allows for a robust evaluation of the model's performance, as each data point is used for testing exactly once. The model's performance is determined by averaging the results from the five iterations. First, this method supports reducing overfitting by training and evaluating the model on different subsets of the data. Second, data leakage is avoided by ensuring the model is not trained on or exposed to the test data, which provides a more realistic estimate of its ability to generalise to new data [145]. Finally, the validation set used during training is created by randomly selecting 20% of the data from the training set. The accuracy of each network corresponds to its ability to reconstruct encoded data after 100 epochs.

The Conv-TCN-AE and Conv-LSTM-AE reconstruction accuracy (similarity of the input tensor to reconstructed output tensor by reducing mean squared error) are compared for *H. sapiens* and *M. musculus* in Fig. 7 and 8 respectively.

It may be concluded that the TCN architecture is slightly more accurate. It is worth mentioning that ConvTCN-AE is noticeably faster and less computationally demanding to train compared to

Conv-LSTM-AE [111]. In our tests, it took 2 h 37 min to train Conv-TCN-AE through five-fold cross-validation, whereas ConvLSTM-AE took 72 h 41 min to train, with the same number of LSTM and TCN layers. Moreover, both Conv-TCN-AE and Conv-LSTM-AE obtain better accuracy for *H. sapiens* than for *M. musculus*.(Table 7).

The PPI prediction stage is discussed thoroughly in the next section.

8.2. PPI prediction

As mentioned earlier, ProtInteract's performance is assessed using two organisms, *H. sapiens* and *M. musculus*. Subsequently, a deep CNN predicts the probability of an interaction belonging to the

Table 7
The five-fold cross-validation training times for Conv-TCN-AE and Conv-LSTM-AE.

Architecture	Training Time
Conv-TCN-AE	2 h 37 min
Conv-LSTM-AE	72 h 41 min

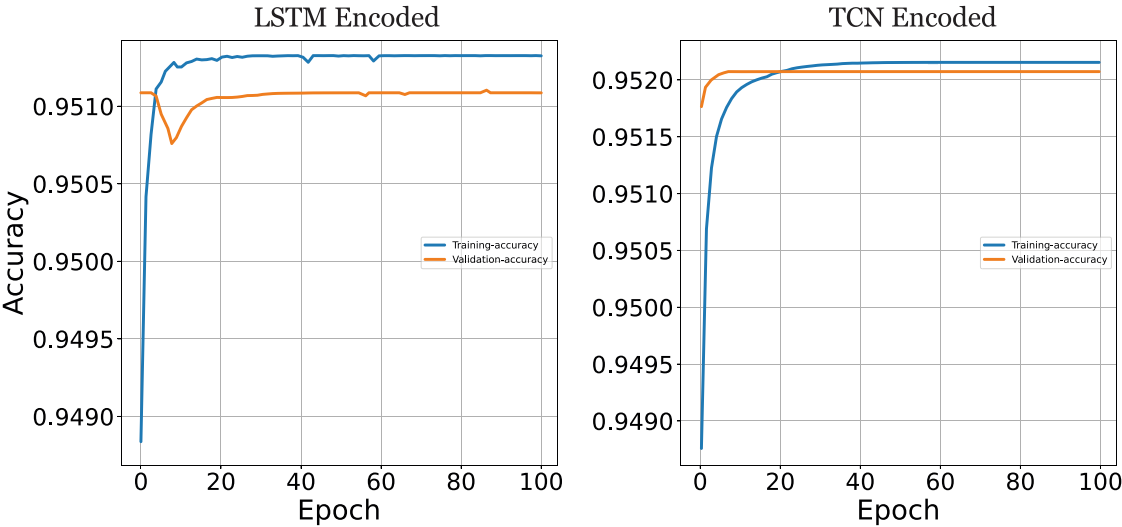


Fig. 7. The accuracy of Conv-TCN-AE and Conv-LSTM-AE for *H. sapiens*.

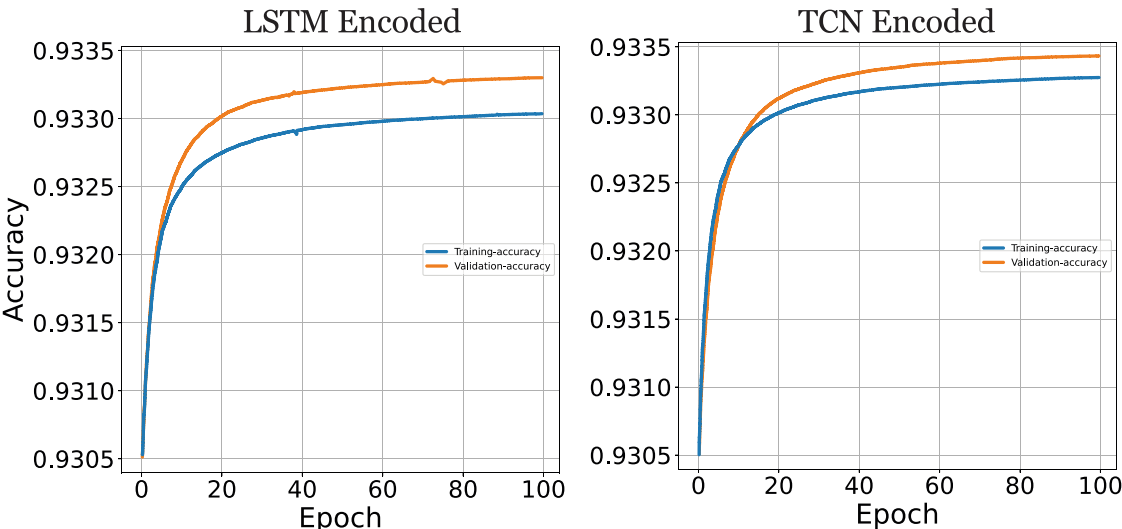


Fig. 8. The accuracy of Conv-TCN-AE and Conv-LSTM-AE for *M. musculus*.

defined classes according to three defined scenarios (see [Tables 2, 3 and 4](#)). The dataset for this task is created by merging the STRING and the Negatome 2.0 datasets and removing the duplicated samples. As mentioned earlier, Negatome 2.0 only comprises interactions with a score of zero non-interacting with the highest confidence) and one (interacting with the highest confidence) [135], whereas STRING includes protein pairs with interaction scores distributed on a continuum from zero and one [62, 63, 65]. Both STRING and Negatome 2.0 datasets are imbalanced in the sense of having uneven numbers of non-interacting, and interacting pairs [146].

For instance, let us consider the *H. sapiens* PPI dataset from STRING. Nearly 83% of the dataset belongs to interactions in class 1 from [Table 2](#) or class 1 from [Table 3](#) or classes 1 and 2 from [Table 4](#); showing considerable imbalance in the dataset, which has an adverse effect on training the deep CNN [147]. To tackle this issue, we randomly downsample the classes with a higher number to the class with the least number of instances. For instance, the *H. sapiens* dataset that is distributed into three classes contains 858726 samples in class 1, 112706 in class 2 and 74814 in class 3. Therefore, we randomly sample 74814 pairs from classes 1 and 2 to obtain three balanced classes. Thus, a balanced dataset is created for each scenario, and the number of samples is reported in [Table 8](#). Furthermore, the training, testing and validation are performed using five-fold cross-validation.

The hyperparameters for training the CNN are reported in [Table 9](#). These parameters were obtained by grid search [144].

Table 8

The balanced protein–protein interaction datasets for *H. sapiens* and *M. musculus* include a different number of protein pairs for each class, corresponding to each classification scenario.

Species/Scenario (samples)	Two classes	Three classes	Five classes
<i>H. sapiens</i>	160113	74813	27128
<i>M. musculus</i>	75501	42905	11650

Table 9

Hyperparameters of training and validation of CNN for PPI prediction.

Parameters	Value	Description
Epochs	500	Training epochs
Batch Size	64	Size of training batch
α	5×10^{-5}	Learning rate

8.2.1. *H. sapiens* PPI prediction results

The classification accuracies (correctly labelling an interaction) using the deep CNN are illustrated in [Fig. 9](#), [Figs. 10 and 11](#) for two, three and five classes respectively. Accordingly, the deep CNN achieves relatively higher accuracy in all three scenarios for those proteins encoded by Conv-TCN-AE than Conv-LSTM-AE. This may be due to the extended long-term memory of TCN layers, which allows better retention of information on long-range interactions compared to LSTM layers [93, 94, 127]. Therefore, it can be concluded that encoded features of Conv-TCN-AE encapsulate more informative attributes.

The Receiver Operator Characteristic (ROC) curve is a graphical plot visualising the ability of deep CNN to correctly predict the class of an interaction. A plot relates the true positive rate (TPR) on the y-axis versus the false positive rate (FPR) on the x-axis. Accordingly, the top left corner of the plot represents a false positive rate of zero and a true positive rate of one. Thus, a larger area under the curve (AUC) is usually better. Here, the deep CNN predicts the probability of an interaction falling within the range of the defined classes, reflecting the confidence level of an interaction between a pair of proteins.

[Fig. 12](#) shows the ROC curve for the two-class classification scenario. The deep CNN obtains higher AUC using TCN-encoded features than LSTM-encoded features.

[Fig. 13](#) illustrates the ROC curve for the three-class classification scenario. For both TCN-encoded and LSTM-encoded features, the AUC diminishes compared to the two-class classification.

The ROC curve for the five-class scenario is depicted in [Fig. 14](#), where AUC is the lowest among all three scenarios. Note that the AUC for all three scenarios is higher for TCN-encoded features than LSTM-encoded features. This may once again show the advantage of the TCN architecture compared to LSTM architecture for encoding proteins. From AUC for all scenarios, it may be inferred that Pro-Interact can accurately classify pairs with the lowest and highest interaction confidence.

8.2.2. *M. musculus* PPI prediction results

The classification accuracies for *M. musculus* are shown in [Fig. 15](#), [Figs. 16 and 17](#), for all three scenarios, respectively.

By comparing [Fig. 9](#) to [Fig. 15](#), classification accuracy for *M. musculus* is slightly lower than *H. sapiens*. This may be due to the smaller training set for *M. musculus* compared to *H. sapiens* (see [Table 8](#)).

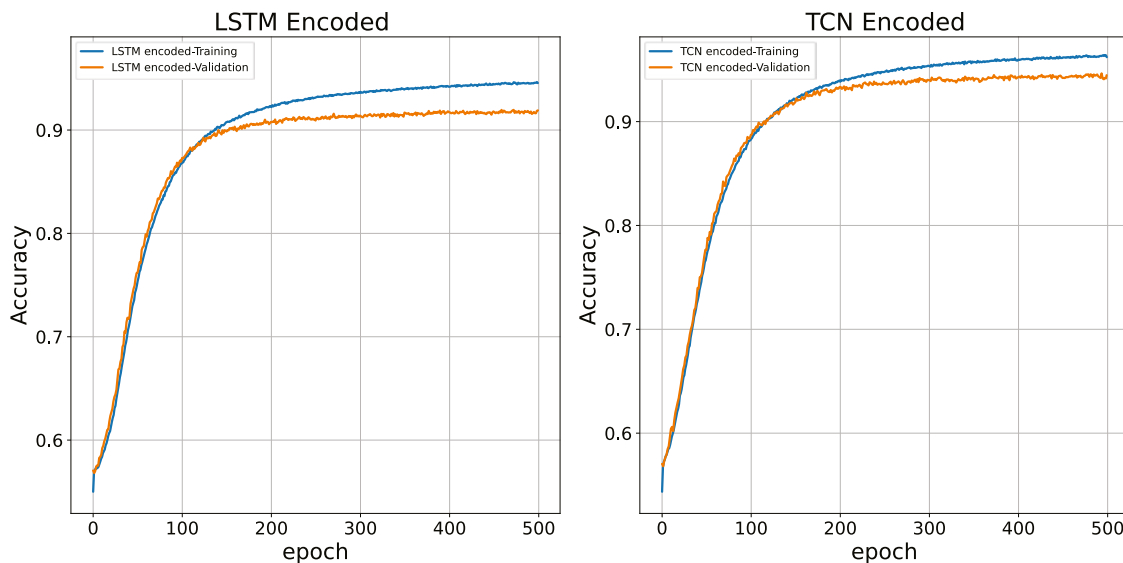


Fig. 9. Prediction accuracy of CNN with input features encoded by Conv-LSTM-AE and Conv-TCN-AE for two-class classification (*H. sapiens*).

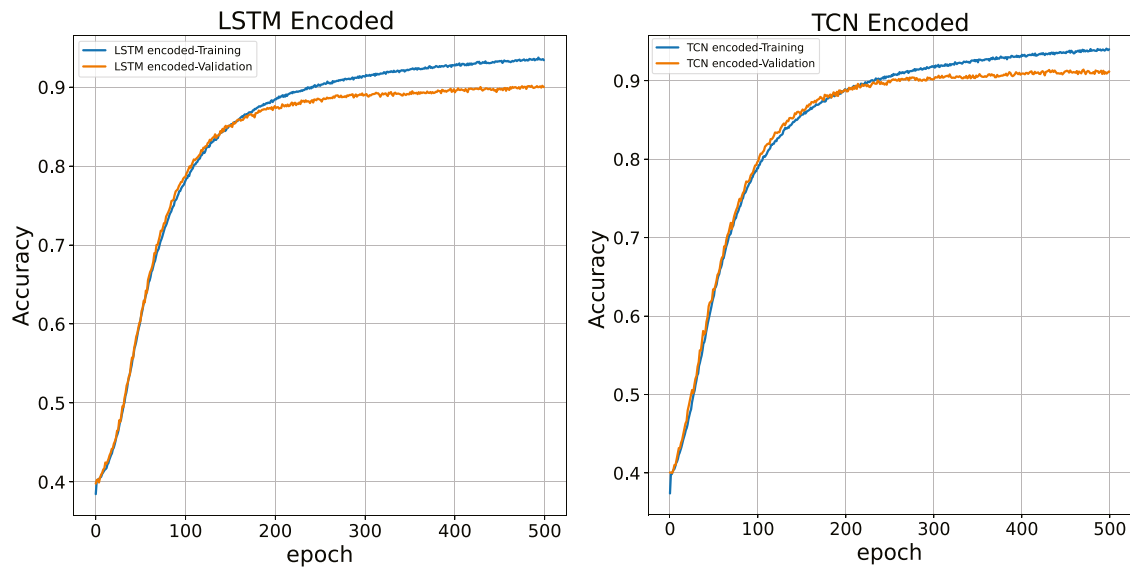


Fig. 10. Prediction accuracy of CNN with input features encoded by Conv-LSTM-AE and Conv-TCN-AE for three-class classification (*H. sapiens*).

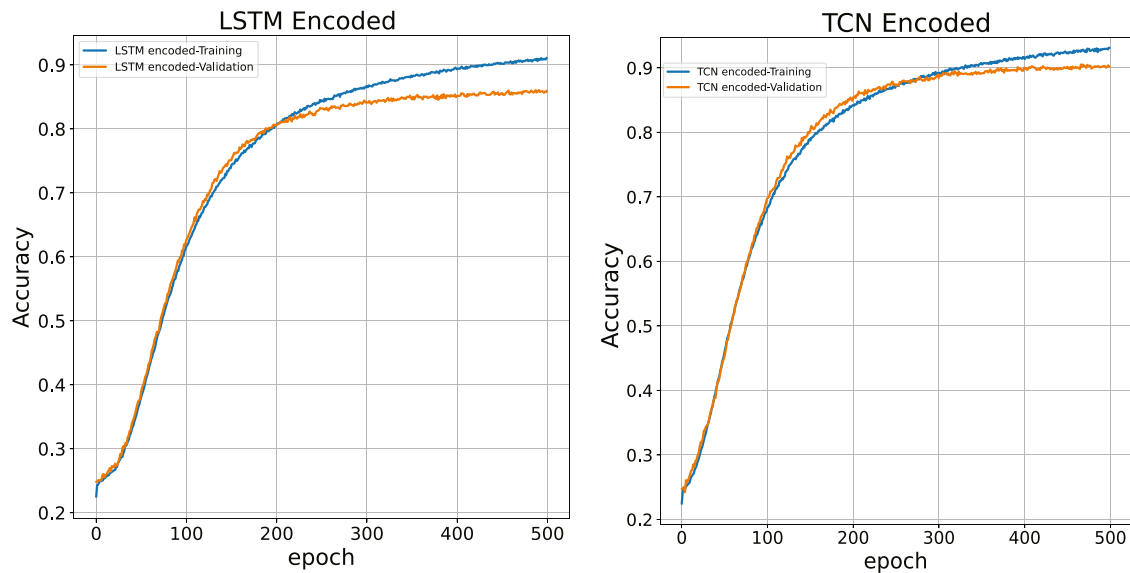


Fig. 11. Prediction accuracy of CNN with input features encoded by Conv-LSTM-AE and Conv-TCN-AE for five-class classification (*H. sapiens*).

On increasing the number of classes, the classification accuracy decreases.

The following figures, Figs. 12, 13, and Fig. 14, display the ROC curves for *M. musculus* for all three scenarios, respectively.

From Figs. 18, 19 and 20, the average AUC for both TCN-encoded and LSTM-encoded cases show slight decreases with the increase of classes. However, classes three from the second scenario and five from the third scenario have shown equal or

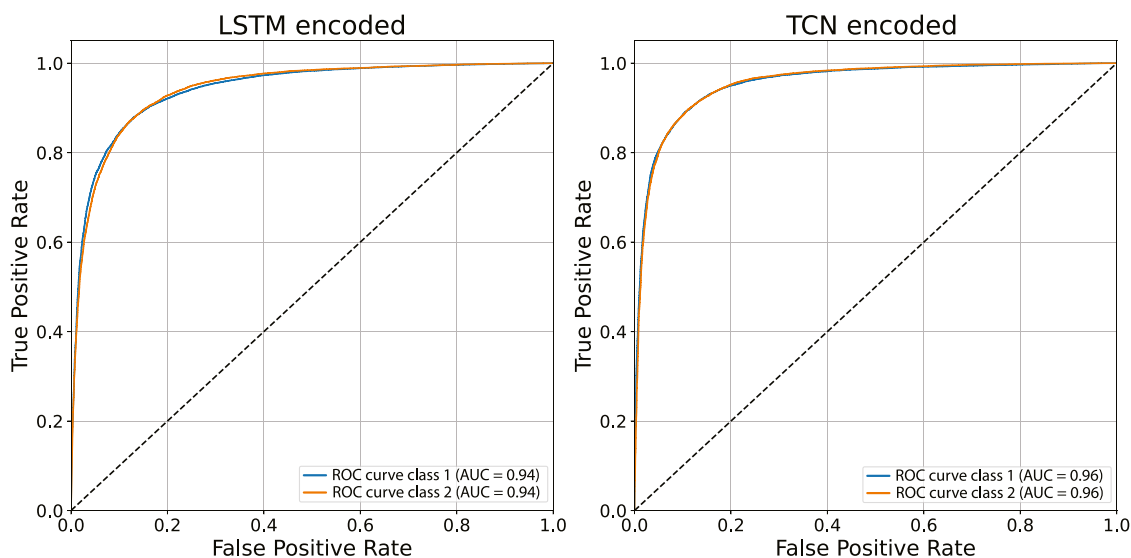


Fig. 12. ROC curve for two-class classification using LSTM- and TCN-encoded features (*H. sapiens*).

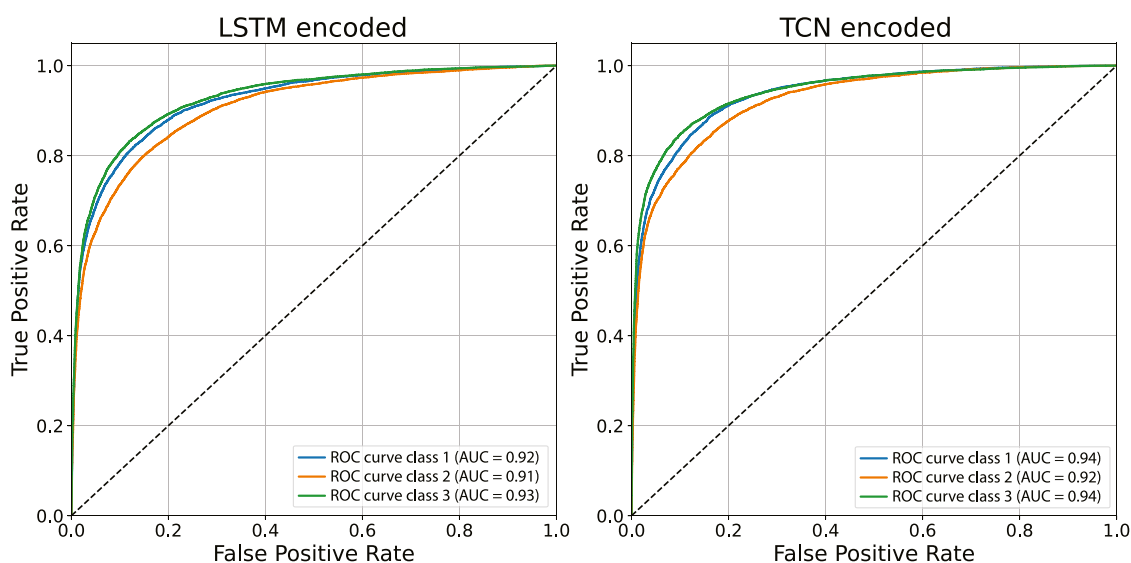


Fig. 13. ROC curve for three-class classification using LSTM- and TCN-encoded features (*H. sapiens*).

higher AUC compared to both classes from the two-class classification scenario. This means that ProtInteract can distinguish the classes with the highest confidence of interaction regardless of the scenario.

8.3. Discussion

By analysing the results obtained by ProtInteract, as reported in [Table 10](#), one may infer that ProtInteract classifies the interaction

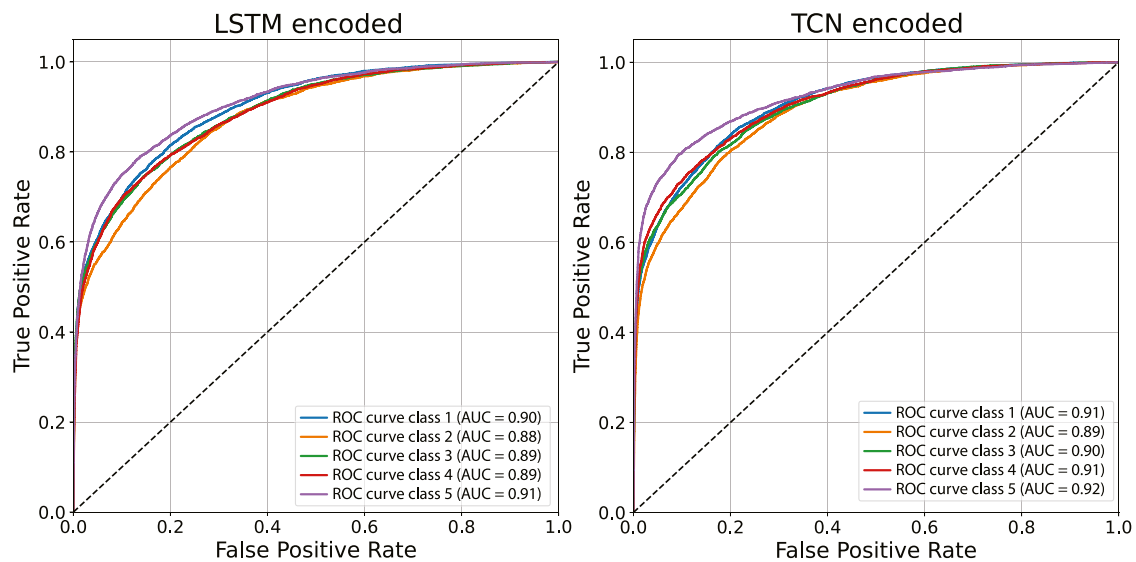


Fig. 14. ROC curve for five-class classification using LSTM- and TCN-encoded features (*H. sapiens*).

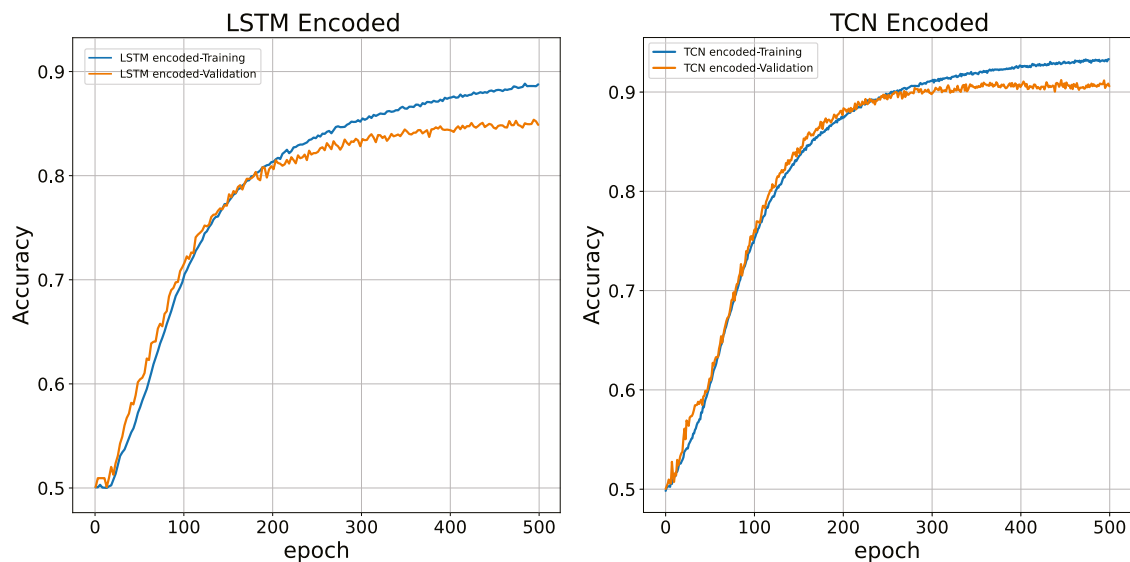


Fig. 15. Prediction accuracy of CNN with input features encoded by Conv-LSTM-AE and Conv-TCN-AE for two classes classification (*M. musculus*).

more accurately for *H. sapiens* than for *M. musculus*. This may be due to the fact that *H. sapiens* has 112%, 74%, and 132.50% more training samples in the balanced datasets, as reported in Table 8, compared to the *M. musculus* balanced dataset for two, three and five classes,

respectively. Additionally, the lower number of training samples among three scenarios results in a slight decline in classification accuracy (see Table 8).

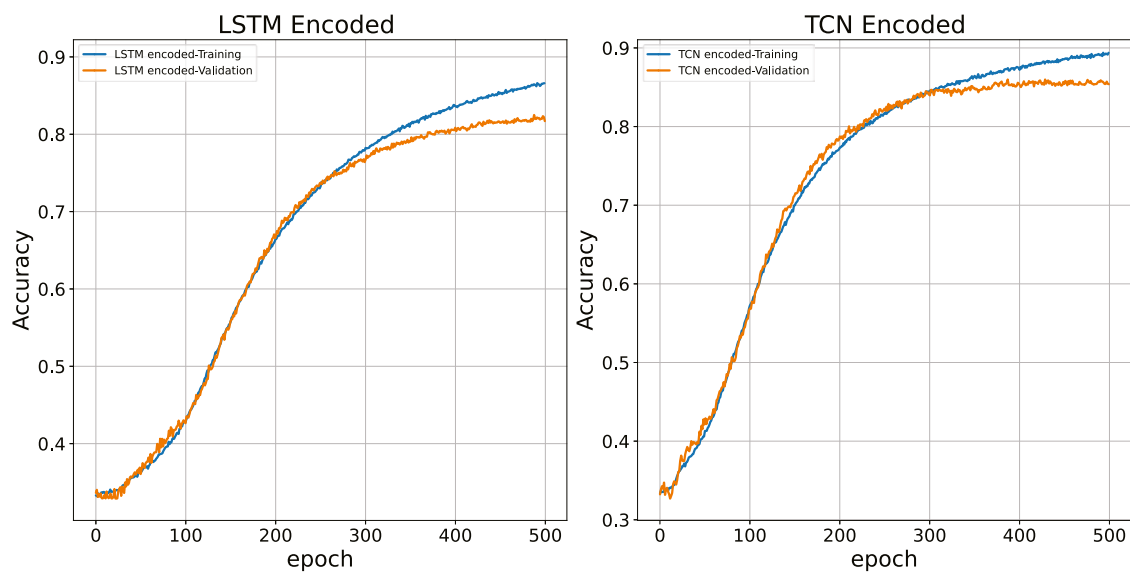


Fig. 16. Prediction accuracy of CNN with input features encoded by Conv-LSTM-AE and Conv-TCN-AE for three classes classification (*M. musculus*).

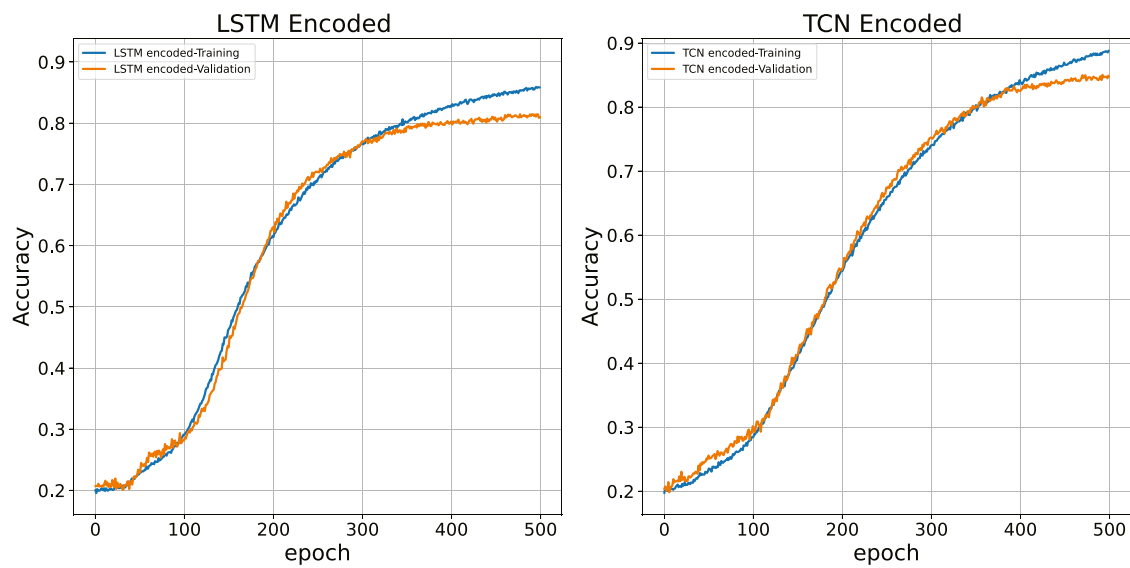


Fig. 17. Prediction accuracy of CNN with input features encoded by Conv-LSTM-AE and Conv-TCN-AE for five classes classification (*M. musculus*).

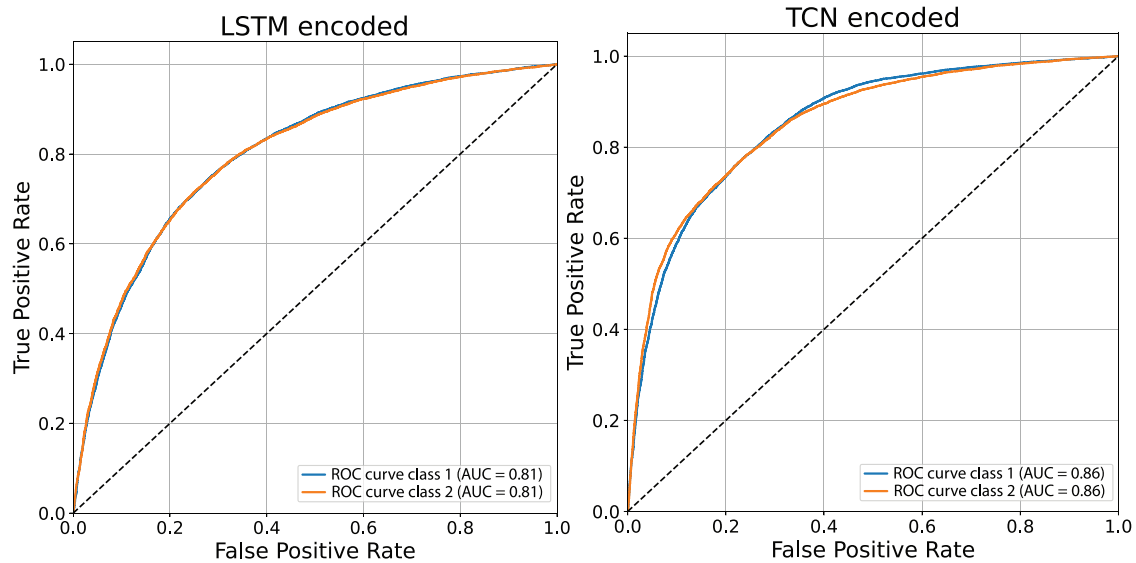


Fig. 18. ROC curve for two-class classification using LSTM- and TCN-encoded features (*M. musculus*).

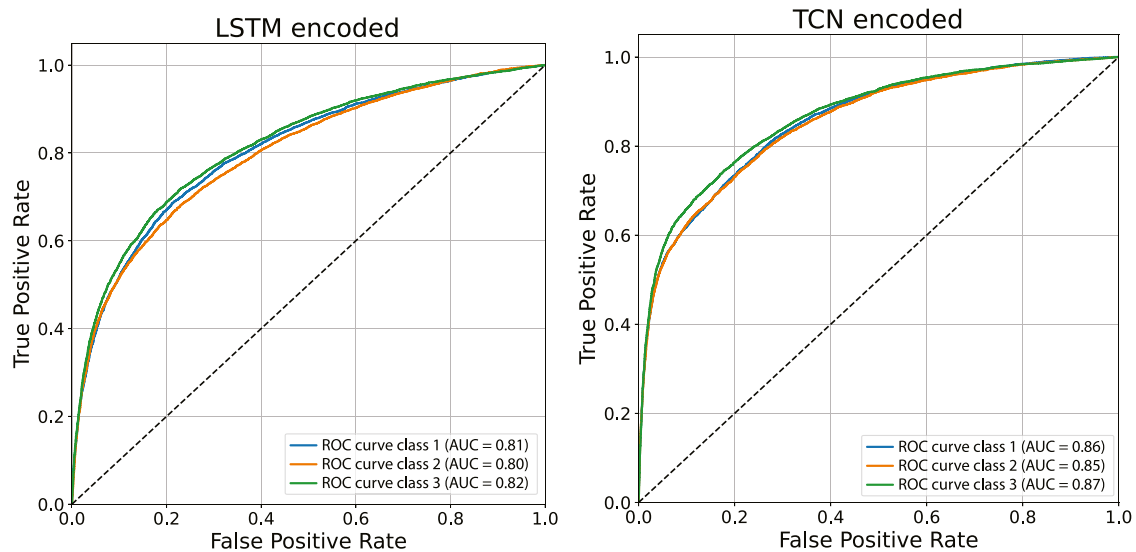


Fig. 19. ROC curve for three-class classification using LSTM- and TCN-encoded features (*M. musculus*).

Let us consider *H. sapiens*: for TCN-encoded features, the deep CNN accuracy of correctly labelling interactions of two-class is 3.50% and 4.77% higher than three and five classes, respectively. For LSTM-encoded features, the classification accuracy of two-class is 1.40% than three and 6.30% higher than five classes.

TCN-encoded (LSTM-encoded) features are those encoded by Conv-TCN-AE (Conv-LSTM-AE), which serve as the input tensor of the deep CNN. The area under the ROC curve (AUC) portrays the discriminative ability of the deep CNN for each class [148]. Considering all three scenarios, AUC is evidently higher when the deep

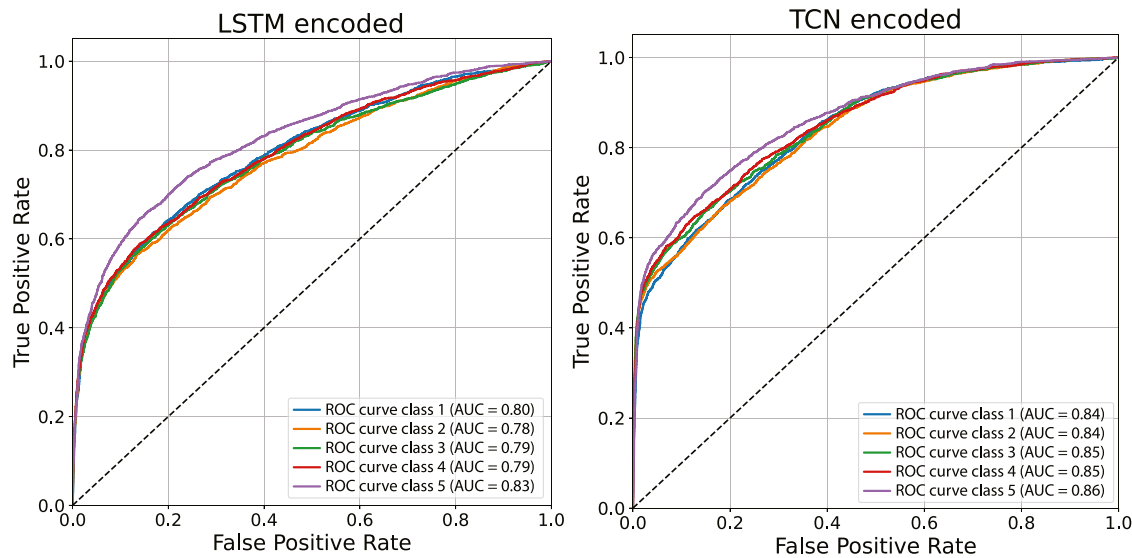


Fig. 20. ROC curve for five-class classification using LSTM- and TCN-encoded features (*M. musculus*).

Table 10
ProtInteract performance of five-fold cross-validation for three different multi-class classification scenarios.

Species	Scenario	Accuracy		AUC		AveragePrecision	
		TCN Encoded	LSTM Encoded	TCN Encoded	LSTM Encoded	TCN Encoded	LSTM Encoded
<i>H. sapiens</i>	2- classes	0.9568	0.9315	0.9600	0.9400	0.9550	0.9400
	3- classes	0.9244	0.9182	0.9340	0.9200	0.8950	0.8600
	5- classes	0.9132	0.8756	0.9060	0.8940	0.7920	0.7620
<i>M. musculus</i>	2- classes	0.9145	0.8456	0.8600	0.8100	0.8450	0.7850
	3- classes	0.8983	0.8337	0.8600	0.8100	0.7870	0.7850
	5- classes	0.8749	0.8216	0.8480	0.7980	0.7000	0.6800

CNN is fed by TCN-encoded features. The deep CNN achieves higher accuracy, precision and AUC, where the input tensor is encoded by Conv-TCN-AE (TCN-encoded) compared to the baseline model, Conv-LSTM-AE (LSTM-encoded). Considering *H. sapiens*, the deep CNN achieves 2.7%, 0.6% and 4.3% higher accuracy when fed by TCN-encoded features than LSTM-encoded features.

A similar pattern appears for *M. musculus*, where, for TCN-encoded features, classification accuracy for two class is 1.80% and 4.50% higher than three and five classes, respectively. For LSTM-encoded features, the deep CNN obtains 1.42% and 2.92% higher accuracy for two-class compared to three and five classes. Finally, considering *M. musculus*, the deep CNN with TCN-encoded features obtains 6.70%, 7.70% and 4% higher accuracy than LSTM-encoded features for all three scenarios, respectively.

Consequently, one may conclude that Conv-TCN-AE architecture is not only considerably faster than Conv-TCN-AE (baseline model) to train but also that ProtInteract’s accuracy of correctly labelling interactions is noticeably higher for protein pairs encoded by Conv-TCN-AE than by Conv-LSTM-AE.

9. Conclusion

Various diseases such as neurodegenerative diseases, infectious diseases, and cancer are closely related to abnormal protein–protein interactions [6, 7, 8]. Therefore, identifying protein–protein interactions helps pave the way towards developing new drugs and targeted therapeutic approaches [9, 10, 11].

In this work, a deep learning framework is developed to predict protein–protein interactions corresponding to sequential patterns encoded for each protein. The prediction problem is expressed as a multi-class classification problem under three scenarios [64]. Each scenario is defined according to different confidence score thresholds of protein interactions [65]. This framework is composed of two components. The first component encodes the amino-acid sequences and reduces the data dimensionality representing each protein while extracting underlying sequential informative features. These tasks are implemented using a novel deep autoencoder architecture called Conv-TCN-AE. We proposed a similar architecture using LSTM layers called Conv-LSTM-AE as a baseline model to compare the performance of TCN layers with a well-known model such as LSTM [93, 94, 117, 127]. The second component is a deep CNN that predicts the class of a given interaction based on the confidence score thresholds from the STRING database.

One of the significant achievements of the proposed framework is its low computational complexity and relatively fast response. Training LSTM and TCN networks as an individual architecture is extremely time-consuming and requires comparably high computational capacity, preferably GPU [149]. However, the proposed autoencoder architecture, namely, Conv-TCN-AE, benefits from the sequential pattern recognition ability of TCN while considerably reducing the training time and computational complexity [117]. The Conv-TCN-AE encodes the dataset with relatively high dimensions into a sequential tensor with a limited number of informative features. Therefore, we can analyse large sets of data for far less time.

To further extend this work, we plan to improve the proposed framework by incorporating a bidirectional architecture and adding a generative architecture to accurately predict the interaction scores. In addition, we will test ProtInteract on other species, including *Saccharomyces cerevisiae*, *Escherichia coli*, *Drosophila melanogaster* and *Caenorhabditis elegans*.

Authorship contributions

- *Conception and design of study*: Farzan Soleymani, Eric Paquet, Herna Lydia Viktor
- *Acquisition of data*: Farzan Soleymani, Eric Paquet, Herna Lydia Viktor
- *Analysis and/or interpretation of data*: Farzan Soleymani, Eric Paquet
- *Drafting the manuscript*: Farzan Soleymani, Eric Paquet, Davide Spinello

Appendix

The LSTM architecture is explained as follows.

A.1. Long Short-Term Memory

As mentioned earlier, RNNs were introduced in order to incorporate sequential information when mapping input to output sequences [35,111]. This means RNNs have the capability of comprehending and interpreting input and output sequences by identifying the correlation and progression of elements within the sequence. This is achieved through the utilisation of a memory component within RNNs, thus enabling them to retain and recall previous information in the sequence. This allows for the formation of decisions based on a comprehensive understanding of the entire sequence, as opposed to just the current input. However, RNNs often suffer from vanishing gradients, limiting the range of context to which they can be applied [150,151]. To cope with this issue, LSTM architecture was introduced [112], as shown in Fig. 21.

The LSTM architecture consists of a set of recurrently connected memory blocks and corresponding control gates, namely, the forget gate f_t , the input gate i_t and the output gate o_t which update and control the cell states [152]. The input and forget gates control the current network memory and a flow of new information. More specifically, as the new information flows into the network, the forget gate manages what information needs to be removed from cell states, while the input gate controls what information will be stored in the cell state. Finally, the output gate determines the encoded information to be forwarded as the input of the next time step.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (A.1a)$$

$$h_t = o_t \otimes \tanh(S_t) \quad (A.1b)$$

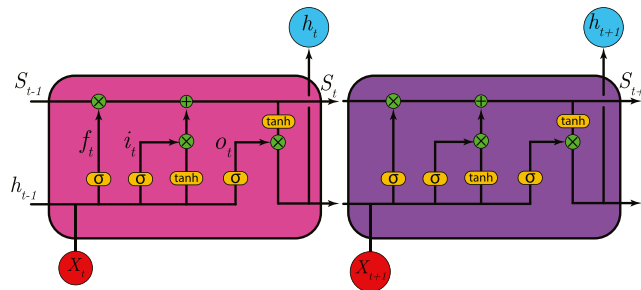


Fig. 21. LSTM architecture.

- *Revising the manuscript critically for important intellectual content*: Eric Paquet, Herna Lydia Viktor, Wojtek Michalowski, Davide Spinello
- *Approval of the version of the manuscript to be published (the names of all authors must be listed)*: Farzan Soleymani, Eric Paquet, Herna Lydia Viktor, Wojtek Michalowski, Davide Spinello

Acknowledgements

All persons who have made substantial contributions to the work reported in the manuscript (e.g., technical help, writing and editing assistance, general support), but who do not meet the criteria for authorship, are named in the Acknowledgements and have given us their written permission to be named. If we have not included an Acknowledgements, then that indicates that we have not received substantial contributions from non-authors.

where σ is the sigmoid activation function, W and b are the weight matrix and bias vector, and $\otimes$ denotes point-wise product. The initial operation is conducted by the forget gate f_t , Eq. (A.1a), which controls which information to keep or to remove. The LSTM architecture contains the hidden state h_t , Eq. (A.1b) that is formed by the sequential information. The next step is to store the new input information in the cell state via the input gate i_t , Eq. (A.2a). Accordingly, the cell state can be modified via candidate values $\hat{S}_t$ by Eq. (A.2b) and Eq. (A.2c). Finally, the LSTM determines the output of each unit by Eq. (A.2d).

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (\text{A.2a})$$

$$\hat{S}_t = \sigma(W_c[h_t - 1, x_t] + b_c) \quad (\text{A.2b})$$

$$S_t = f_t \otimes S_t - 1 + i_t \otimes \hat{S}_t \quad (\text{A.2c})$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (\text{A.2d})$$

The following section illustrates the architecture of the convolutional LSTM autoencoder as the baseline model in this work.

A.2. Convolutional LSTM Autoencoder

The architecture of the convolutional LSTM autoencoder (Conv-LSTM-AE) is shown in Fig. 22. This architecture comprises the well-known LSTM layer, which serves as a baseline model to compare with Conv-TCN-AE in the proposed framework. Similar to Conv-TCN-AE, Conv-LSTM-AE consists of two main components. The encoder receives high-dimensionality data to encode into a lower dimensionality and extracts the sequential patterns using two layers of LSTM. Next, the latent features are extracted from the output of a densely connected layer. The decoder reconstructs the input data via dense, LSTM, and 2-D transposed convolutional layers. Finally, a 2-D convolutional layer maps the data into the proper dimensionality. The pseudocode of this architecture is shown in Alg. 3.

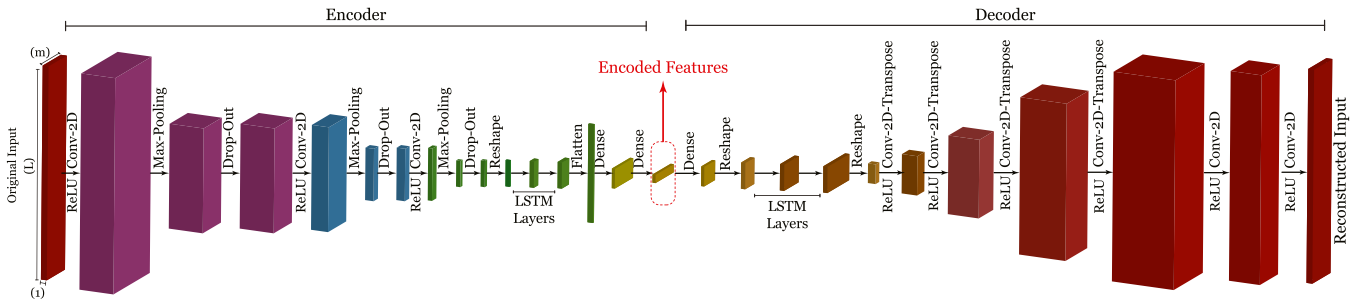


Fig. 22. Conv-LSTM-AE architecture that reduces the dimensionality of input data and extracts informative and uncorrelated features.

Algorithm 3. Convolutional-LSTM Autoencoder [134].

Input: Encoded sequence tensor= $S_t [L \times m \times 1]_{t=1}^T$, encode-size, input-size, num-features

Output: Latent layer - Informative pattern of sequence of amino acids in a protein,

Initialization:

1. **Encoder:**

- **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=64}
- **Max Pooling layer:** {Pool size=(2,2), Padding="same"}
- **Dropout Layer:** [0.3]
- **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=32}
- **Max Pooling layer:** {Pool size=(2,2), Padding="same"}
- **Dropout Layer:** [0.3]
- **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=16}
- **Max Pooling layer:** {Pool size=(2,1), Padding="same"}
- **Reshape:** (to 2-D tensor)
- **LSTM layer:** {act-fcn=ReLU, units=16}
- **LSTM layer:** {act-fcn=ReLU, units=8}
- **Flatten**
- **Dense layer:** {units=2×encode-size, act-fcn=ReLU}
- **Dense layer:** {units=encode-size, act-fcn=ReLU, layer weight regularizers (kernel-bias-activity)}
- *Encoder Output:* Latent layer - low dimensional, highly informative patterns.

2. **Decoder:**

- **Dense layer:** {units=encode-size}
 - **Reshape layer:** {Extend dimension to 2-D tensor}
 - **LSTM layer:** {units=4, act-fcn=ReLU}
 - **LSTM layer:** {units=num-features, act-fcn=ReLU}
 - **Reshape layer:** {Extend dimension to 3-D tensor}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=16, Strides=(2,1)}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=32, Strides=(4,1)}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=64, Strides=(4,1)}
 - **2-D-Transposed convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=128, Strides=(4,1)}
 - **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=8, Padding="same"}
 - **2-D-convolutional layer:** {act-fcn=ReLU, kernel-size=(2,2), num-filters=1, Padding="same"}
 - *Decoder Output:* Reconstructed input vector
 - *Compiler:* {**Optimiser:** Adam [134], **Loss:** Mean Squared Error}
-

References

- [1] Lodish H, Zipursky SL. Molecular cell biology. Biochem Mol Biol Educ 2001;29:126–33.
- [2] A.W. White, A.D. Westwell, G. Brahemi, Protein–protein interactions as targets for small-molecule therapeutics in cancer, Expert reviews in molecular medicine 10.
- [3] Blazer LL, Neubig RR. Small molecule protein–protein interaction inhibitors as cns therapeutic agents: current progress and future hurdles. Neuropsychopharmacology 2009;34(1):126–41.
- [4] Rosell M, Fernandez-Recio J. Hot-spot analysis for drug discovery targeting protein–protein interactions. Expert Opin Drug Discov 2018;13(4):327–38.
- [5] Liu L, Zhu X, Ma Y, Piao H, Yang Y, Hao X, et al. Combining sequence and network information to enhance protein–protein interaction prediction. BMC Bioinforma 2020;21(16):1–13.
- [6] S. Ferrari, F. Pellati, M. Costi, Disruption of protein–protein interfaces (2013).
- [7] Rual J-F, Venkatesan K, Hao T, Hirozane-Kishikawa T, Dricot A, Li N, et al. Towards a proteome-scale map of the human protein–protein interaction network. Nature 2005;437(7062):1173–8.
- [8] Stelzl U, Worm U, Lalowski M, Haenig C, Brembeck FH, Goehler H, et al. A human protein–protein interaction network: a resource for annotating the proteome. Cell 2005;122(6):957–68.
- [9] F. Browne, H. Zheng, H. Wang, F. Azuaje, From experimental approaches to computational techniques: a review on the prediction of protein–protein interactions., Advances in Artificial Intelligence (16877470).

- [10] Skrabanek L, Saini HK, Bader GD, Enright AJ. Computational prediction of protein–protein interactions. *Mol Biotechnol* 2008;38(1):1–17.
- [11] Lu H, Zhou Q, He J, Jiang Z, Peng C, Tong R, et al. Recent advances in the development of protein–protein interactions modulators: mechanisms and clinical trials. *Signal Transduct Target Ther* 2020;5(1):1–23.
- [12] Schwikowski B, Uetz P, Fields S. A network of protein–protein interactions in yeast. *Nat Biotechnol* 2000;18(12):1257–61.
- [13] Ito T, Tashiro K, Muta S, Ozawa R, Chiba T, Nishizawa M, et al. Toward a protein–protein interaction map of the budding yeast: a comprehensive system to examine twohybrid interactions in all possible combinations between the yeast proteins. *Proc Natl Acad Sci* 2000;97(3):1143–7.
- [14] Gavin A-C, B'osche M, Krause R, Grandi P, Marzioch M, Bauer A, et al. Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature* 2002;415(6868):141–7.
- [15] Ho Y, Gruhler A, Heilbut A, Bader GD, Moore L, Adams S-L, et al. Systematic identification of protein complexes in *Saccharomyces cerevisiae* by mass spectrometry. *Nature* 2002;415(6868):180–3.
- [16] Pandey A, Mann M. Proteomics to study genes and genomes. *Nature* 2000;405(6788):837–46.
- [17] Figeys D. Novel approaches to map protein interactions. *Curr Opin Biotechnol* 2003;14(1):119–25.
- [18] Noor Z, Ahn SB, Baker MS, Ranganathan S, Mohamedali A. Mass spectrometry–based protein identification in proteomics—a review. *Brief Bioinforma* 2021;22(2):1620–38.
- [19] Garza KY, Feider CL, Klein DR, Rosenberg JA, Brodbelt JS, Eberlin LS. Desorption electrospray ionization mass spectrometry imaging of proteins directly from biological tissue sections. *Anal Chem* 2018;90(13):7785–9.
- [20] MacBeath G, Schreiber SL. Printing proteins as microarrays for high-throughput function determination. *Science* 2000;289(5485):1760–3.
- [21] Bu'ssow K, Nordhoff E, Lu'bbert C, Lehrach H, Walter G. A human cdna library for high-throughput protein expression screening. *Genomics* 2000;65(1):1–8. <https://doi.org/10.1006/geno.2000.6141>. (URL). <https://www.sciencedirect.com/science/article/pii/S088875430096141X>.
- [22] Brizuela L, Braun P, LaBaer J. Flexgene repository: from sequenced genomes to gene repositories for high-throughput functional biology and proteomics. *Mol Biochem Parasitol* 2001;118(2):155–65.
- [23] Brizuela L, Richardson A, Marsischky G, Labaer J. The flexgene repository: exploiting the fruits of the genome projects by creating a needed resource to face the challenges of the post-genomic era. *Arch Med Res* 2002;33(4):318–24.
- [24] Piehler J. New methodologies for measuring protein interactions in vivo and in vitro. *Curr Opin Struct Biol* 2005;15(1):4–14.
- [25] V.S. Rao, K. Srinivas, G. Sujini, G. Kumar, Protein–protein interaction detection: methods and analysis, *International journal of proteomics* 2014.
- [26] Wu J, Paquet E, Viktor HL, Michalowski W. Paying attention: using a siamese pyramid network for the prediction of protein–protein interactions with folding and self-binding primary sequences. 2021 Int Jt Conf Neural Netw (IJCNN) 2021:1–8. <https://doi.org/10.1109/IJCNN52387.2021.9534212>
- [27] Wang T, Li L, Huang Y-A, Zhang H, Ma Y, Zhou X. Prediction of protein–protein interactions from amino acid sequences based on continuous and discrete wavelet transform features. *Molecules* 2018;23(4):823.
- [28] Ding Z, Kihara D. Computational identification of protein–protein interactions in model plant proteomes. *Sci Rep* 2019;9(1):1–13.
- [29] S. Tsukiyama, M.M. Hasan, S. Fujii, H. Kurata, Lstm-phv: Prediction of human-virus protein–protein interactions by lstm with word2vec, *bioRxiv*.
- [30] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.
- [31] Jangmin O, Lee J, Lee JW, Zhang B-T. Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Inf Sci* 2006;176(15):2121–47.
- [32] Wang Y, You Z, Li L, Chen Z. A survey of current trends in computational predictions of protein–protein interactions. *Front Comput Sci* 2020;14(4):1–12.
- [33] Soleymani F, Paquet E. Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder–deepbreath. *Expert Syst Appl* 2020;156:113456.
- [34] Soleymani F, Paquet E. Deep graph convolutional reinforcement learning for financial portfolio management– deep-pocket. *Expert Syst Appl* 2021;182:115127.
- [35] Bao W, Yue J, Rao Y. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS One* 2017;12(7):e0180944.
- [36] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. *science* 2006;313(5786):504–7.
- [37] C.O.S. Sorzano, J. Vargas, A.P. Montano, A survey of dimensionality reduction techniques, *arXiv preprint arXiv:1403.2877*.
- [38] Gao W, Mahajan SP, Sulam J, Gray JJ. Deep learning in protein structural modeling and design. *Patterns* 2020:100142.
- [39] Chen X-w, Jeong JC. Sequence-based prediction of protein interaction sites with an integrative method. *Bioinformatics* 2009;25(5):585–91.
- [40] Zahiri J, Yaghoubi O, Mohammad-Noori M, Ebrahimpour R, Masoudi-Nejad A. Ppieve: Protein–protein interaction prediction from pssm based evolutionary information. *Genomics* 2013;102(4):237–42.
- [41] You Z-H, Chan KC, Hu P. Predicting protein–protein interactions from primary protein sequences using a novel multi-scale local feature representation scheme and the random forest. *PloS One* 2015;10(5):e0125811.
- [42] F. Soleymani, E. Paquet, H. Viktor, W. Michalowski, D. Spinello, Protein–protein interaction prediction with deep learning: A comprehensive review, *Computational and Structural Biotechnology Journal*.
- [43] Chen H, Zhou H-X. Prediction of interface residues in protein–protein complexes by a consensus neural network method: test against nmr data, *Proteins: Structure. Funct. Bioinforma* 2005;61(1):21–35.
- [44] Rodgers-Melnick E, Culp M, DiFazio SP. Predicting whole genome protein interaction networks from primary sequence data in model and non-model organisms using ents. *BMC Genom* 2013;14(1):1–17.
- [45] Emamjomeh A, Goliaei B, Torkamani A, Ebrahimpour R, Mohammadi N, Parsian A. Protein–protein interaction prediction by combined analysis of genomic and conservation information. *Genes Genet Syst* 2014;89(6):259–72.
- [46] Kotlyar M, Pastrello C, Pivetta F, Lo Sardo A, Cumbaa C, Li H, et al. In silico prediction of physical protein interactions and characterization of interactome orphans. *Nat Methods* 2015;12(1):79–84.
- [47] Chang J-W, Zhou Y-Q, Ul Qamar MT, Chen L-L, Ding Y-D. Prediction of protein–protein interactions by evidence combining methods. *Int J Mol Sci* 2016;17(11):1946.
- [48] B-Q. Li, K-Y. Feng, L. Chen, T. Huang, Y-D. Cai, Prediction of protein–protein interaction sites by random forest algorithm with mrmr and ifs, *PloS one*.
- [49] X. Hu, C. Feng, T. Ling, M. Chen, Deep learning frameworks for protein–protein interaction prediction, *Computational and Structural Biotechnology Journal*.
- [50] Jia J, Li X, Qiu W, Xiao X, Chou K-C. ippi-pseaac (cgr): Identify protein–protein interactions by incorporating chaos game representation into pseaac. *J Theor Biol* 2019;460:195–203.
- [51] Pan J, Li L-P, Yu C-Q, You Z-H, Guan Y-J, Ren Z-H. Sequence-based prediction of plant protein–protein interactions by combining discrete sine transformation with rotation forest. *Evolut Bioinforma* 2021;17. 11769343211050067.
- [52] Du X, Sun S, Hu C, Yao Y, Yan Y, Zhang Y. Deepppi: boosting prediction of protein–protein interactions with deep neural networks. *J Chem Inf Model* 2017;57(6):1499–510.
- [53] Yang X, Yang S, Lian X, Wuchty S, Zhang Z. Transfer learning via multi-scale convolutional neural layers for human–virus protein–protein interaction prediction. *Bioinformatics* 2021;37(24):4771–8.
- [54] Lei Y, Li S, Liu Z, Wan F, Tian T, Li S, et al. A deep-learning framework for multi-level peptide–protein interaction prediction. *Nat Commun* 2021;12(1):1–10.
- [55] Sledzieski S, Singh R, Cowen L, Berger B. D-script translates genome to phenotype with sequence-based, structure-aware, genome-scale predictions of protein–protein interactions. *Cell Syst* 2021;12(10):969–82.
- [56] Bepler T, Berger B. Learning protein sequence embeddings using information from structure. *arXiv Prepr arXiv 1902.08661*.
- [57] Hu X, Feng C, Zhou Y, Harrison A, Chen M. Deeptrio: a ternary prediction system for protein–protein interaction using mask multiple parallel convolutional neural networks. *Bioinformatics* 2022;38(3):694–702.
- [58] Chen M, Ju C-J-T, Zhou G, Chen X, Zhang T, Chang K-W, et al. Multifaceted protein–protein interaction prediction based on siamese residual rcnn. *Bioinformatics* 2019;35(14):i305–14.
- [59] K. Cho, B. VanMerri'enoer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, *arXiv preprint arXiv:1406.1078*.
- [60] Yang F, Fan K, Song D, Lin H. Graph-based prediction of protein–protein interactions with attributed signed graph embedding. *BMC Bioinforma* 2020;21(1):1–16.
- [61] Kipf TN, Welling M. Variational graph auto-encoders. *arXiv Prepr arXiv 1611.07308*.
- [62] Szklarczyk D, Gable AL, Nastou KC, Lyon D, Kirsch R, Pyysalo S, et al. The string database in 2021: customizable protein–protein networks, and functional characterization of user-uploaded gene/measurement sets. *Nucleic Acids Res* 2021;49(D1):D605–12.
- [63] D. Szklarczyk, R. Kirsch, M. Koutrouli, K. Nastou, F. Mehryary, R. Hachilif, A. L. Gable, T. Fang, N. T. Doncheva, S. Pyysalo, et al. The string database in 2023: protein–protein association networks and functional enrichment analyses for any sequenced genome of interest, *Nucleic Acids Research*.
- [64] M. Grandini, E. Bagli, G. Visani, Metrics for multi-class classification: an overview, *arXiv preprint arXiv:2008.05756*.
- [65] Crosara KTB, Moffa EB, Xiao Y, Siqueira WL. Merging in-silico and in vitro salivary protein complex partners using the string database: a tutorial. *J Proteom* 2018;171:87–94.
- [66] Tran L, Hamp T, Rost B. Profpdb: pairs of physical protein–protein interactions predicted for entire proteomes. *PloS One* 2018;13(7):e0199988.
- [67] Ju J, Liu F-A. Multivariate time series data prediction based on att-lstm network. *Appl Sci* 2021;11(20):9373.
- [68] Bandara K, Bergmeir C, Hewamalage H. Lstm-msnet: leveraging forecasts on sets of related time series with multiple seasonal patterns. *IEEE Trans Neural Netw Learn Syst* 2020;32(4):1586–99.
- [69] Bandara K, Bergmeir C, Smyl S. Forecasting across time series databases using recurrent neural networks on groups of similar series: a clustering approach. *Expert Syst Appl* 2020;140:112896.
- [70] Bandara K, Hewamalage H, Liu Y-H, Kang Y, Bergmeir C. Improving the accuracy of global forecasting models using time series data augmentation. *Pattern Recognit* 2021;120:108148.
- [71] Jha K, Saha S. Amalgamation of 3d structure and sequence information for protein–protein interaction prediction. *Sci Rep* 2020;10(1):1–14.
- [72] Guo Y, Yu L, Wen Z, Li M. Using support vector machine combined with auto covariance to predict protein–protein interactions from protein sequences. *Nucleic Acids Res* 2008;36(9):3025–30.
- [73] Kyte J, Doolittle RF. A simple method for displaying the hydropathic character of a protein. *J Mol Biol* 1982;157(1):105–32.

- [74] Biro J. Amino acid size, charge, hydropathy indices and matrices for protein structure analysis. *Theor Biol Med Model* 2006;3(1):1–12.
- [75] Shen J, Zhang J, Luo X, Zhu W, Yu K, Chen K, et al. Predicting protein–protein interactions based only on sequences information. *Proc Natl Acad Sci* 2007;104(11):4337–41.
- [76] Zhang S-W, Hao L-Y, Zhang T-H. Prediction of protein–protein interaction with pairwise kernel support vector machine. *Int J Mol Sci* 2014;15:3220–33.
- [77] Y.-A. Huang, Z.-H. You, X. Gao, L. Wong, L. Wang, Using weighted sparse representation model combined with discrete cosine transformation to predict protein–protein interactions from protein sequence, *BioMed research international* 2015.
- [78] You Z-H, Yu J-Z, Zhu L, Li S, Wen Z-K. A mapreduce based parallel svm for large-scale predicting protein–protein interactions. *Neurocomputing* 2014;145:37–43.
- [79] W. Ma, Y. Cao, W. Bao, B. Yang, Y. Chen, Act-svm: Prediction of protein–protein interactions based on support vector basis model, *Scientific Programming* 2020.
- [80] Sun T, Zhou B, Lai L, Pei J. Sequence-based prediction of protein interaction using a deep-learning algorithm. *BMC Bioinforma* 2017;18(1):1–8.
- [81] S. Sledzieski, R. Singh, L. Cowen, B. Berger, Sequence-based prediction of protein–protein interactions: a structure-aware interpretable deep learning model, *bioRxiv*.
- [82] Su X-R, You Z-H, Chen Z-H, Yi H-C, Guo Z-H. Protein–protein interaction prediction by integrating sequence information and heterogeneous network representation. *International Conference on Intelligent Computing*. Springer; 2021. p. 617–26.
- [83] Das S, Deb T, Dey N, Ashour AS, Bhattacharya D, Tibarewala D. Optimal choice of k-mer in composition vector method for genome sequence comparison. *Genomics* 2018;110(5):263–73.
- [84] Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q. Line: large-scale information network embedding. *Proc 24th Int Conf World wide web* 2015:1067–77.
- [85] Xu W, Gao Y, Wang Y, Guan J. Protein–protein interaction prediction based on ordinal regression and recurrent convolutional neural networks. *BMC Bioinforma* 2021;22(6):1–21.
- [86] Hawkins-Hooker A, Depardieu F, Baur S, Couairon G, Chen A, Bikard D. Generating functional protein variants with variational autoencoders. *PLoS Comput Biol* 2021;17(2):e1008736.
- [87] J. Cao, L. Xiong, Protein sequence classification with improved extreme learning machine algorithms, *BioMed research international* 2014.
- [88] Y. G'ormez, Dimensionality reduction for protein secondary structure prediction, Master's thesis, Abdullah Gu'l Universitesi, Fen Bilimleri Enstitu'su" (2017)."
- [89] Du X, Li Y, Xia Y-L, Ai S-M, Liang J, Sang P, et al. Insights into protein–ligand interactions: mechanisms, models, and methods. *Int J Mol Sci* 2016;17(2):144.
- [90] Gupta A, Mu'ller AT, Huisman BJ, Fuchs JA, Schneider P, Schneider G. Generative recurrent networks for de novo drug design. *Mol Inform* 2018;37(1–2):1700111.
- [91] Lea C, Vidal R, Reiter A, Hager GD. Temporal convolutional networks: a unified approach to action segmentation. *European Conference on Computer Vision*. Springer; 2016. p. 47–54.
- [92] Soleymani F, Paquet E. Long-term financial predictions based on feynman–dirac path integrals, deep bayesian networks and temporal generative adversarial networks. *Mach Learn Appl* 2022;7:100255.
- [93] Bai S, Kolter JZ, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv Prepr arXiv* 1803:01271.
- [94] Wan R, Mei S, Wang J, Liu M, Yang F. Multivariate temporal convolutional network: a deep neural networks approach for multivariate time series forecasting. *Electronics* 2019;8(8):876.
- [95] ElAbd H, Bromberg Y, Hoarfrost A, Lenz T, Franke A, Wendorff M. Amino acid encoding for deep learning applications. *BMC Bioinforma* 2020;21(1):1–14.
- [96] Liu Z, Cui Y, Xiong Z, Nasiri A, Zhang A, Hu J. Deepseqpan, a novel deep convolutional neural network model for pan-specific class i hla-peptide binding affinity prediction. *Sci Rep* 2019;9(1):1–10.
- [97] Prokhorenkova L, Gusev G, Vorobev A, Dorogush AV, Gulina A. Catboost: unbiased boosting with categorical features. *arXiv Prepr arXiv* 1706:09516.
- [98] Johnson JM, Khoshgoftaar TM. Encoding techniques for high-cardinality features and ensemble learners. 2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI). IEEE; 2021. p. 355–61.
- [99] Cohen M, Reichmann D, Neuvirth H, Schreiber G. Similar chemistry, but different bond preferences in inter versus intra-protein interactions. *Proteins* 2008;72(2):741–53.
- [100] Tsai C-J, Lin SL, Wolfson HJ, Nussinov R. Studies of protein–protein interfaces: a statistical analysis of the hydrophobic effect. *Protein Sci* 1997;6(1):53–64.
- [101] Desantis F, Miotto M, Di Rienzo L, Milanetti E, Ruocco G. Spatial organization of hydrophobic and charged residues affects protein thermal stability and binding affinity. *Sci Rep* 2022;12(1):1–13.
- [102] Meiler J, Mu'ller M, Zeidler A, Schma'schke F. Generation and evaluation of dimension-reduced amino acid parameter representations by artificial neural networks. *Mol Model Annu* 2001;7(9):360–9.
- [103] Xu P-S, Luo J, Dou T-Y. Predict protein–protein interactions from protein primary sequences: using wavelet transform combined with stacking algorithm. *PeerJ Prepr* 2017;5:e2964v1.
- [104] L. Yang, Y. Han, H. Zhang, W. Li, Y. Dai, Prediction of protein–protein interactions with local weight-sharing mechanism in deep learning, *BioMed Research International* 2020.
- [105] Ausaf Ali S, Hassan I, Islam A, Ahmad F, et al. A review of methods available to estimate solvent-accessible surface areas of soluble proteins in the folded and unfolded states. *Curr Protein Pept Sci* 2014;15(5):456–76.
- [106] Jha K, Saha S, Tanveer M. Prediction of protein–protein interactions using stacked auto-encoder. *Trans Emerg Telecommun Technol* 2021:e4256.
- [107] S. Debnath, A.F. Mollah, A supervised machine learning approach for sequence based protein–protein interaction (ppi) prediction, *arXiv preprint arXiv:2203.12659*.
- [108] Ellis JJ, Huard FP, Deane CM, Srivastava S, Wood GR. Directionality in protein fold prediction. *BMC Bioinforma* 2010;11(1):1–16.
- [109] J. Wang, B. Wicher, V. Maurizot, I. Huc, Directing the self-assembly of aromatic foldamer helices using acridine appendages and metal coordination, *Chemistry–A European Journal*.
- [110] Idowu AO, Alashi AM, Nwachukwu ID, Fagbemi TN, Aluko RE. Functional properties of sesame (*Sesamum indicum* linn) seed protein fractions, *Food Production. Process Nutr* 2021;3(1):1–16.
- [111] S. Gopali, F. Abri, S. Siarni-Namini, A.S. Namin, A comparative study of detecting anomalies in time series data using lstm and tcn models, *arXiv preprint arXiv:2112.09293*.
- [112] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Comput* 1997;9(8):1735–80.
- [113] Sagheer A, Kotb M. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Sci Rep* 2019;9(1):1–16.
- [114] Palangi H, Deng L, Shen Y, Gao J, He X, Chen J, et al. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans. Audio, Speech, Lang Process* 2016;24(4):694–707.
- [115] Palangi H, Ward R, Deng L. Distributed compressive sensing: a deep learning approach. *IEEE Trans Signal Process* 2016;64(17):4504–18.
- [116] Gopali S, Abri F, Siarni-Namini S, Namin AS. A comparison of tcn and lstm models in detecting anomalies in time series data. 2021 IEEE International Conference on Big Data (Big Data). IEEE; 2021. p. 2415–20.
- [117] Nan M, Tr'asc'au M, Florea AM, Iacob CC. Comparison between recurrent networks and temporal convolutional networks approaches for skeleton-based action recognition. *Sensors* 2021;21(6):2051.
- [118] Y. He, J. Zhao, Temporal convolutional networks for anomaly detection in time series, in: *Journal of Physics: Conference Series*, Vol. 1213, IOP Publishing, 2019, p. 042050.
- [119] Zerze GH, Stillinger FH, DeBenedetti PG. Computational investigation of retro-isomer equilibrium structures: Intrinsically disordered, foldable, and cyclic peptides. *FEBS Lett* 2020;594(1):104–13.
- [120] F. Yu, V. Koltun, Multi-scale context aggregation by dilated convolutions, *arXiv preprint arXiv:1511.07122*.
- [121] Zhao W, Gao Y, Ji T, Wan X, Ye F, Bai G. Deep temporal convolutional networks for short-term traffic flow forecasting. *IEEE Access* 2019;7:114496–507.
- [122] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. *Proc IEEE Conf Comput Vis Pattern Recognit* 2016:770–8.
- [123] Duc TN, Minh CT, Xuan TP, Kamioka E. Convolutional neural networks for continuous qoe prediction in video streaming services. *IEEE Access* 2020;8:116268–78.
- [124] Zhu R, Liao W, Wang Y. Short-term prediction for wind power based on temporal convolutional network. *Energy Rep* 2020;6:424–9.
- [125] Salimans T, Kingma DP. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. *Adv Neural Inf Process Syst* 2016;29:901–9.
- [126] V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, in: *ICML*, 2010.
- [127] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, *arXiv preprint arXiv:1609.03499*.
- [128] Bouatta N, Sorger P, AlQuraishi M. Protein structure prediction by alphafold2: are attention and symmetries all you need? *Acta Crystallogr Sect D: Struct Biol* 2021;77(8):982–91.
- [129] Y. Bengio, L. Yao, G. Alain, P. Vincent, Generalized denoising auto-encoders as generative models, *arXiv preprint arXiv:1305.6663*.
- [130] Bengio Y, Goodfellow IJ, Courville A. Deep learning. *Nature* 2015;521(7553):436–44.
- [131] Y. Zhang, A better autoencoder for image: Convolutional autoencoder, in: *ICONIP17-DCEC*. Available online: <http://users.cecs.anu.edu.au/Tom.Gedeon/conf/ABCs2018/paper/ABCs2018 paper 58. pdf> (accessed on 23 March 2017), 2018.
- [132] Polic M, Krajacic I, Lepora N, Orsag M. Convolutional autoencoder for feature extraction in tactile sensing. *IEEE Robot Autom Lett* 2019;4(4):3671–8.
- [133] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. *J Mach Learn Res* 2014;15(1):1929–58.
- [134] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*.
- [135] Blohm P, Frishman G, Smialowski P, Goebels F, Wachinger B, Ruepp A, et al. Negatome 2.0: a database of non-interacting proteins derived by literature mining, manual annotation and protein structure analysis. *Nucleic Acids Res* 2014;42(D1):D396–400.
- [136] Orchard S, Kerrien S, Abbani S, Aranda B, Bhate J, Bidwell S, et al. Protein interaction data curation: the international molecular exchange (imex) consortium. *Nat Methods* 2012;9(4):345–50.
- [137] Orchard S, Ammari M, Aranda B, Breuza L, Briganti L, Broackes-Carter F, et al. The mintact project—intact as a common curation platform for 11 molecular interaction databases. *Nucleic Acids Res* 2014;42(D1):D358–63.

- [138] Oughtred R, Stark C, Breitkreutz B-J, Rust J, Boucher L, Chang C, et al. The biogrid interaction database: 2019 update. *Nucleic Acids Res* 2019;47(D1):D529–41.
- [139] Bauer-Mehren A, Furlong LI, Sanz F. Pathway databases and tools for their exploitation: benefits, current limitations and challenges. *Mol Syst Biol* 2009;5(1):290.
- [140] Amberger JS, Bocchini CA, Schiettecatte F, Scott AF, Hamosh A. Omim. org: Online mendelian inheritance in man (omim®), an online catalog of human genes and genetic disorders. *Nucleic Acids Res* 2015;43(D1):D789–98.
- [141] Cherry JM, Hong EL, Amundsen C, Balakrishnan R, Binkley G, Chan ET, et al. Saccharomyces genome database: the genomics resource of budding yeast. *Nucleic Acids Res* 2012;40(D1):D700–5.
- [142] Franceschini A, Lin J, von Mering C, Jensen LJ. Svd-phy: improved prediction of protein functional associations through singular value decomposition of phylogenetic profiles. *Bioinformatics* 2016;32(7):1085–7.
- [143] Jarazo J, Barmba K, Modamio J, Saraiva C, Sabat'e-Soler S, Rosety I, et al. Parkinson's disease phenotypes in patient neuronal cultures and brain organoids improved by 2-hydroxypropyl- β -cyclodextrin treatment. *Mov Disord* 2022;37(1):80–94.
- [144] Alibrahim H, Ludwig SA. Hyperparameter optimization: comparing genetic algorithm against grid search and bayesian optimization. 2021 IEEE Congress on Evolutionary Computation (CEC). IEEE; 2021. p. 1551–9.
- [145] Wong T-T, Yeh P-Y. Reliable accuracy estimates from k-fold cross validation. *IEEE Trans Knowl Data Eng* 2019;32(8):1586–94.
- [146] J. Ingraham, V. Garg, R. Barzilay, T. Jaakkola, Generative models for graph-based protein design, *Advances in Neural Information Processing Systems* 32.
- [147] Kim J, Kim J. The impact of imbalanced training data on machine learning for author name disambiguation. *Scientometrics* 2018;117(1):511–26.
- [148] van den Hout WB. The area under an roc curve with limited information. *Med Decis Mak* 2003;23(2):160–6.
- [149] Zhang Y, Wang C, Gong L, Lu Y, Sun F, Xu C, et al. A power-efficient accelerator based on fpgas for lstm network. 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE; 2017. p. 629–30.
- [150] Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans Neural Netw* 1994;5(2):157–66.
- [151] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al., Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001).
- [152] James J, Lam AY, Hill DJ, Li VO. Delay aware intelligent transient stability assessment system. *IEEE Access* 2017;5:17230–9.

Chapter 4

Conclusion

This thesis project focused on developing a computational framework that may be transferred to a wide range of problems involving time and pseudo-time series, such as area coverage control in an evolving environment, financial stock markets, and PPIs. At first glance, these problems may seem unrelated; however, there are several resemblances between them. For instance, the area coverage control problem may be formulated in an environment with time-varying risk densities, which requires an adaptive control scheme to update accordingly. This problem aims to formulate a temporal difference, learning to obtain optimal trajectories for a given performance index later in time, relying on time-dependent measurements along the trajectories. In the stock market, the price of financial instruments may also be assimilated into time series; thus, the underlying distribution associated with financial instruments may vary over time. Analogously, the protein’s primary structure is constructed of a sequence of amino acids that follow a causal directionality posed by N-terminus to C-terminus translation. Therefore, proteins may be modelled and analysed as a pseudo-time series [48, 49, 58].

4.1 Completed Work

The ProtInteract framework is developed to predict PPIs corresponding to sequential patterns encoded for each protein. The prediction problem is expressed as a multi-class classification problem under three scenarios [63]. Each scenario is defined according to different confidence score thresholds that are associated with the potential protein interactions [67]. This framework incorporates two major components. The first component encodes the amino acid sequences and reduces the data dimensionality representing each protein while extracting underlying sequential informative features. These tasks are implemented using a novel deep autoencoder architecture called Conv-TCN-AE. We proposed a similar architecture using LSTM layers, called Conv-LSTM-AE, as a baseline model to compare the performance of TCN layers with a well-known model such as LSTM [55, 65, 68, 69]. The second component is a deep convolutional neural network that predicts the class of a given interaction based on the confidence score thresholds from the STRING database.

One of the significant achievements of the proposed framework is its low computational complexity and relatively fast response. Training LSTM and TCN networks as an individual architecture is extremely time-consuming and requires comparably high computational capacity, especially for the graphics processing unit [70]. However, the proposed autoencoder architecture, namely Conv-TCN-AE, benefits from the sequential pattern recognition ability of TCN while considerably reducing the training time and computational complexity [68]. The Conv-TCN-AE encodes the dataset with relatively high dimensions into a sequential tensor with a limited number of informative features. Therefore, we can analyse large sets of data in far less time.

Proteins support numerous biological functions in the cell, including cell signalling (hormones), biochemical activity (enzymes), mechanical tasks (muscle), and structural roles (cytoskeleton). Proteins execute these functions by direct physicochemical interaction with other molecules, including proteins and peptides, nucleic acids, and small-molecule ligands. Generally, any molecule capable of binding to a protein with high specificity and affinity may be considered a ligand [66]. However, finding a protein-ligand pair with high affinity remains a challenging problem in computational drug discovery. The binding affinity relies heavily on physicochemical properties, size, shape, and constitutional characteristics of both the protein and ligand. The protein-ligand association is rooted in their binding kinetics, which determines their binding rate. Accordingly, fast binding and slow dissociation (unbinding) rates result in high binding affinity. One of the extensions of this work is that ProtInteract may identify protein-ligand pairs as the interaction score among protein pairs falls within the (0.9, 1] range.

The future work is discussed in the next section.

4.2 Future Work

The continuation of this project may be dedicated to developing a reinforcement learning technique based on actor-critic elements to predict the amino acid sequence of a ligand, given the receptor (target protein). Unlike other protein-ligand binding methods founded on known structures or sequences of proteins, the proposed framework learns the policy that will generate the amino acid sequence of a ligand, given a receptor. Owing to a large amount of available data, deep learning methods have immensely influenced different scientific fields by learning patterns and improving the decision-making processes in a specific field such as medicine [71]. The Covid-19 pandemic has revealed that computational methods are particularly necessary to accelerate the production of effective treatments [72, 73, 74]. The proposed framework aims to expedite the development of new treatments and offer feasible solutions for specific pathologies such as enzymatic disorders [75], antibiotic resistance [76], metabolic disorders [77] and Parkinson’s disease [78], to mention a few.

Actor-critic-based reinforcement learning frameworks are developed for optimal control of dynamical systems with uncertain or unknown models, using online neural network estimators to optimise systems while relying on data measured along the trajectories. This type of formulation is typically based on Bellman’s optimality, which in the classical formulation produces optimal adaptive trajectories backwards in time based on the knowledge of the system’s dynamics. In contrast, reinforcement learning solutions of this class of problems can recover optimality by moving forward in time based on data observed along the trajectories. I previously employed reinforcement learning methods, namely state–action–reward–state–action (SARSA) and actor-critic in DeepBreath and DeepPocket frameworks [59, 60] (see Chapter 5). Additionally, I have designed reinforcement learning-based adaptive control algorithms to find an optimal solution for the coverage control problem concerning a group of robots over time-invariant and time-varying environments (see Chapter 6).

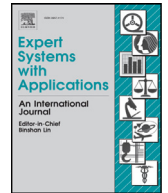
Chapter 5

Appendix-A

This chapter presents three deep learning frameworks developed for portfolio management and stock market prediction that inspired the sequence-based method for PPI prediction.

5.1 Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—DeepBreath

The following publication further explains the financial time series and the proposed deep learning methods for portfolio management and stock market prediction. First, the DeepBreath framework is developed for real-time portfolio management by implementing a settlement system on a Bitcoin blockchain [79]. Moreover, a novel autoencoder architecture is proposed, called restricted stacked autoencoder, which is designed by restricting the latent layer, as well as the output layer. Unlike traditional autoencoders, in this architecture, the decoder partially reconstructs the latent features, leading to generating more informative features while focusing on what was expected as the most important features. The principle behind using an autoencoder to extract meaningful features while reducing the dimension of the input tensor can be used in the PPI prediction problem. This paper is published in the Expert Systems with Applications journal on April 18, 2020.



Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—DeepBreath

Farzan Soleymani, Eric Paquet*

National Research Council, 1200 Montreal Road, Ottawa, ON K1K 2E1, Canada

ARTICLE INFO

Article history:

Received 3 October 2019

Revised 10 February 2020

Accepted 13 April 2020

Available online 18 April 2020

Keywords:

Portfolio management

Deep reinforcement learning

Restricted stacked autoencoder

Online learning

Settlement risk

Blockchain

ABSTRACT

The process of continuously reallocating funds into financial assets, aiming to increase the expected return of investment and minimizing the risk, is known as portfolio management. In this paper, a portfolio management framework is developed based on a deep reinforcement learning framework called DeepBreath. The DeepBreath methodology combines a restricted stacked autoencoder and a convolutional neural network (CNN) into an integrated framework. The restricted stacked autoencoder is employed in order to conduct dimensionality reduction and features selection, thus ensuring that only the most informative abstract features are retained. The CNN is used to learn and enforce the investment policy which consists of reallocating the various assets in order to increase the expected return on investment. The framework consists of both offline and online learning strategies: the former is required to train the CNN while the latter handles concept drifts i.e. a change in the data distribution resulting from unforeseen circumstances. These are based on passive concept drift detection and online stochastic batching. Settlement risk may occur as a result of a delay in between the acquisition of an asset and its payment failing to deliver the terms of a contract. In order to tackle this challenging issue, a blockchain is employed. Finally, the performance of the DeepBreath framework is tested with four test sets over three distinct investment periods. The results show that the return of investment achieved by our approach outperforms current expert investment strategies while minimizing the market risk.

Crown Copyright © 2020 Published by Elsevier Ltd.

This is an open access article under the CC BY-NC-ND license.

(<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

1. Introduction

A collection of financial assets, known as a portfolio, may consist of financial instruments such as stocks, bonds, currencies, and cryptocurrencies. The process of reallocating the various assets forming the portfolio, in order to increase the expected return on investment, is called portfolio management. The financial instruments managed in such a portfolio may be assimilated to time series, which often exhibit complex behaviors, due to external factors such as the global economy and the political climate. As a result, their behaviors are inherently nonlinear, uncertain and non-stationary (Malkiel, 2003; Tsai & Hsiao, 2010).

Generally, the erratic and complex behavior of the financial market is determined by endogenous and exogenous factors (Filimonov & Sornette, 2012). In recent years, due to the constant rise of social media, such as TwitterTM and FacebookTM, it has been

implicitly assumed that market evolution essentially results from these exogenous factors. This assumption finds its origin in the efficient market hypothesis (Fama, 1991; Malkiel & Fama, 1970) in which the price of an asset fully reflects all available information. Nevertheless, various studies have demonstrated the falsehood of this hypothesis showing that these exogenous factors have a limited impact on markets where behavior is essentially driven by endogenous factors (Bouchaud, 2011; Cutler, Poterba, & Summers, 1988; Joulin, Lefevre, Grunberg, & Bouchaud, 2008). The extent to which the market is endogenous is determined by the spectral radius associated with the underlying state-dependent Hawkes process (Filimonov & Sornette, 2012).

When managing a portfolio, one must take into account the volatility and the correlation associated with the various financial instruments. Portfolio management is an investment strategy that aims at maximizing the expected return on investment (ROI) while minimizing the financial risk by continuously reallocating the portfolio assets i.e. by attributing the proper weight to each financial instrument. With recent advances in machine learning and deep learning, it has become possible to predict complex financial be-

* Corresponding author.

haviors and to automatize the decision process; at least up to a certain extent (Jangmin, Lee, Lee, & Zhang, 2006).

Stock market behavior prediction with an artificial neural network (ANN) was proposed for the first time by Atsalakis and Valavanis (2009). Artificial neural networks are data-driven machine learning techniques which do not require a complete prior model of the data generation mechanism. Nonetheless, traditional multilayer neural networks do not take advantage of the time series structure associated with financial data (Eakins & Stansell, 2003; Hussain, Knowles, Lisboa, & El-Deredey, 2008; Lam, 2004).

Moreover, the performance of an ANN is strongly dependent on the network architecture, the learning, and hyper-parameters, as well as the training, set just to mention a few. Therefore, other alternatives have been explored in order to improve the accuracy such as support vector machine (SVM), random forest (RF) and naïve Bayes with seminal work by Patel, Shah, Thakkar, and Kotecha (2015) with non-significant improvement. A major breakthrough came with the introduction of deep reinforcement learning (Jiang, Xu, & Liang, 2017) in which the policy, which determines the actions taken by the agent, is either learned with a convolutional neural network (CNN) or with some kind of recurrent neural network (RNN) such as the long-short-term memory network (LSTM). In the portfolio management context, the actions are the weights attributed to the various assets forming the portfolio.

Financial data are particularly suited for deep learning because of the very large amount of data available for training (Goodfellow, Bengio, & Courville, 2016). Nevertheless, deep learning algorithms performances are affected by multiple factors such as the activation and pooling functions, noise, network structure, which may consist of multiple sub-networks as well as the features selection mechanism. In the financial context, noise reduction has been achieved with wavelet transforms (Bao, Yue, & Rao, 2017) while more informative representations and dimensionality reduction were obtained with Boltzmann machines and autoencoders (Hinton & Salakhutdinov, 2006) among other techniques. For large portfolios, dimensionality reduction is desirable in order to optimize the learning process while avoiding the curse of dimensionality (Sorzano, Vargas, & Montano, 2014). Various types of encoders have been employed such as convolutional autoencoders (Cheng, Sun, Takeuchi, & Katto, 2018) and stacked Autoencoders (Bao et al., 2017) just to mention a few.

In this paper, we propose an online framework for portfolio management based on deep reinforcement learning and restricted stacked autoencoder for feature selection. High-level features are extracted from the input data which contain eleven (11) correlated features. The restricted stacked autoencoder extracts new informative features while reducing the dimensions down to three (3). Therefore, the training may be completed very rapidly making this approach suitable for online learning and online portfolio management. The investment policy is enforced using a CNN. The algorithm is initially trained offline which results in a pre-trained model from historical data. Then, the weights are updated following an online learning scheme in order to follow the evolution of the market. Both offline and online strategies are employed for learning: the former being based on passive concept drift detection (Gama, Indr , Bifet, Pechenizkiy, & Bouchachia, 2014) and the latter being based on online stochastic batching. Using passive concept drift detection, which allows the system to learn new emerging patterns in the market while preserving the knowledge acquired during offline training with historical data. In order to facilitate the audit and to secure the settlement process, a blockchain is employed.

The paper is organized as follows. The underlying mathematical formalism associated with portfolio management is introduced in Section 2. Features normalization and selection are addressed

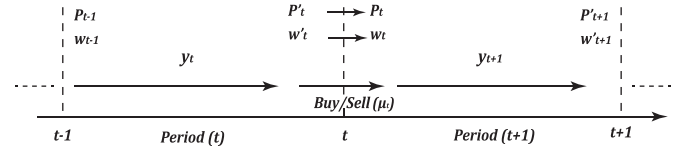


Fig. 1. Weight reallocation process. The portfolio weight vector $\mathbf{w}'_{t-1}$ and the portfolio value P'_{t-1} at the beginning of period t evolve to (t) and P'_t respectively. At that moment, assets are sold and bought in order to increase the expected return on investment. These operations involve commission fees which shrink the portfolio value to P_t which result in new weights $\mathbf{w}_t$.

in Section 3. Our deep reinforcement learning framework, named DeepBreath, is presented in Section 4. This framework integrates passive concept drift detection and online stochastic batching. This is followed, in Section 5, by a description of the blockchain employed for settlement. Our experimental results are presented in Section 6. A conclusion follows in Section 7.

2. Mathematical model for portfolio management

The trading period for financial instruments on the stock exchange is one calendar day. During this period, the price fluctuations associated with a financial instrument may be characterized by four metrics namely the price at the opening of the market, the lowest and the highest prices reached during the day and, the price at the closure of the market. Based on these characteristics, this section provides a mathematical framework for portfolio management. An approach similar to (Jiang et al., 2017), which was introduced earlier by Ormos and Urb n (2013), is followed.

The price vector $\mathbf{V}_t$ consists of the closing price of all assets forming the portfolio at the time in addition to the amount of cash available for trading (here in USD). The normalized price vector $\mathbf{Y}_t$ is defined as the element-wise division in between the price vector at the time t and $t - 1$:

$$\mathbf{Y}_t := \mathbf{V}_t \oslash \mathbf{V}_{t-1} = \left[1, \frac{\mathbf{V}_{1,t}}{\mathbf{V}_{1,t-1}}, \frac{\mathbf{V}_{2,t}}{\mathbf{V}_{2,t-1}}, \dots, \frac{\mathbf{V}_{m,t}}{\mathbf{V}_{m,t-1}} \right]^T. \quad (1)$$

Initially, the portfolio consists solely of cash which means that the weight associated with the cash is one (1), while the weights associated with the other financial instruments are zero (0). This means that no prior assumption is made about the portfolio composition: the weight allocation is entirely performed by our deep reinforcement learning algorithm. Therefore, no other prior assumption about the portfolio composition is made:

$$\mathbf{w}_0 = [1, 0, \dots, 0]^T. \quad (2)$$

In order to buy or to sell an asset, one must pay a commission fee. As illustrated in Fig. 1, owing to prices fluctuations, at the end of period t , the value of the portfolio vector is determined by

$$\mathbf{w}'_t = \frac{\mathbf{Y}_t \odot \mathbf{w}_{t-1}}{\mathbf{Y}_t \cdot \mathbf{w}_{t-1}}, \quad (3)$$

where $\mathbf{w}_{t-1}$ is the portfolio weight vector at the beginning of period t . At that moment, the portfolio is reallocated by buying and selling assets in order to increase the expected return on investment; therefore, the portfolio weight vector evolves from $\mathbf{w}'_t$ to $\mathbf{w}_t$. The portfolio reallocation involves commission fees which shrink the portfolio value by a factor of $\mu \in (0, 1]$,

$$P_t = \mu_t P'_t. \quad (4)$$

The portfolio value at the end of period t is given by

$$P_t = \mu_t P_{t-1} \cdot \mathbf{Y}_t \cdot \mathbf{w}_{t-1}. \quad (5)$$

The return rate for period t and the logarithmic return rate are defined as

$$\rho_t := \frac{P_t}{P_{t-1}} - 1 = \frac{\mu_t \cdot P'_t}{P_{t-1}} - 1 = \mu_t \mathbf{Y}_t \cdot \mathbf{w}_{t-1} - 1, \quad (6a)$$

Table 1
Financial indicators employed as features.

Financial indicators	Definition
Average True Range	The average true range is an indicator that measures market volatility by decomposing the entire range of an asset price for that period.
Commodity Channel Index	The commodity channel index is a momentum-based oscillator used to help determine when an investment vehicle is reaching a condition of being overbought or oversold
Commodity Selection Index	The commodity selection index is a momentum indicator that attempts to identify which commodities are the most suitable for short-term trading.
Demand Index	The demand index is an indicator that uses price and volume to assess buying and selling pressure affecting a security.
Dynamic Momentum Index	The dynamic momentum index is a technical indicator used to determine if an asset is overbought or oversold.
Exponential Moving Average	An exponential moving average is a type of moving average that places a greater weight and significance of the most recent data points.
Hull Moving Average	The hull moving average solves the age old dilemma of making a moving average more responsive to current price activity whilst maintaining curve smoothness.
Momentum	The momentum is the rate of acceleration of a security's price or volume – that is, the speed at which the price is changing.

$$R_t := \ln \frac{P_t}{P_{t-1}} = \ln(\mu_t \mathbf{Y}_t \mathbf{w}_{t-1}) - 1. \quad (6b)$$

Therefore, the final value of the portfolio for a time horizon t_f is given by

$$P_f = P_0 \cdot \exp \left(\sum_{t=1}^{t_f+1} R_t \right) = P_0 \prod_{t=1}^{t_f+1} \mu_t \mathbf{Y}_t \mathbf{w}_{t-1}. \quad (7)$$

Consequently, the total amount of cash gained C_g by selling assets is

$$C_g = (1 - c_s) P'_t \sum_{i=1}^m \text{ReLU}(\mathbf{w}'_{t,i} - \mu_t \mathbf{w}_{t,i}), \quad (8)$$

where the selling commission rate is denoted by $0 < c_s < 1$. As a result, the cash reserve $P'_t \mathbf{w}'_{t,0}$ becomes $\mu_t P'_t \mathbf{w}_{t,0}$ which corresponds to the new amount of cash available for acquiring new assets. On the other hand, the amount of cash spent C_s is

$$\begin{aligned} C_s &= (1 - c_b) \left[\mathbf{w}'_{t,0} + (1 - c_s) \sum_{i=1}^m \text{ReLU}(\mathbf{w}'_{t,i} - \mu_t \mathbf{w}_{t,i}) - \mu_t \mathbf{w}_{t,0} \right] \\ &= \sum_{i=1}^m \text{ReLU}(\mu_t \mathbf{w}_{t,i} - \mathbf{w}'_{t,i}). \end{aligned} \quad (9)$$

while the buying commission rate is given by $0 < c_b < 1$. With the help of $\mathbf{w}'_{t,0} + \sum_{i=1}^m \mathbf{w}'_{t,i} = 1 = \mathbf{w}_{t,0} + \sum_{i=1}^m \mathbf{w}_{t,i}$ and $\text{ReLU}(x - y) - \text{ReLU}(y - x) = x - y$ the Eq. (9) yields:

$$\begin{aligned} \mu_t &= \frac{1}{1 - c_b \mathbf{w}_{t,0}} \\ &\times \left[1 - c_b \mathbf{w}'_{t,0} - (c_s + c_b - c_s c_b) \sum_{i=1}^m \text{ReLU}(\mathbf{w}'_{t,i} - \mu_t \mathbf{w}_{t,i}) \right]. \end{aligned} \quad (10)$$

The transaction factor μ_t is a function of the relative price $\mathbf{Y}$ as well as of the current and previous weight vectors:

$$\mu_t = \mu_t(\mathbf{w}_{t-1}, \mathbf{w}_t, \mathbf{Y}_t). \quad (11)$$

This implicit equation cannot be solved analytically. However, an approximation for the shrinking factor may be obtained through an iterative process. It has been demonstrated (Jiang et al., 2017) that the sequence generated by Eq. (10) always converge to the right solution.

Theorem 1. Given an initial estimate μ_0 , the sequence $\{\tilde{\mu}_t^{(k)}\}$ defined by

$$\{\tilde{\mu}_t^{(k)} | \tilde{\mu}_t^{(0)} = \mu_0 \text{ and } \tilde{\mu}_t^{(k)} = f(\tilde{\mu}_t^{(k-1)}), k \in \mathbb{N}\}, \quad (12)$$

$$\begin{aligned} f(\mu) &:= \frac{1}{1 - c_b \mathbf{w}_{t,0}} \\ &\times \left[1 - c_b \mathbf{w}'_{t,0} - (c_s + c_b - c_s c_b) \sum_{i=1}^m \text{ReLU}(\mathbf{w}'_{t,i} - \mu \mathbf{w}_{t,i}) \right], \end{aligned} \quad (13)$$

always converge to the solution of Eq. (10) for any $\mu_0 \in [0, 1]$.

The convergence rate depends on the initial value of the transaction factor μ_t . According to (Moody, Wu, Liao, & Saffell, 1998), a fast convergence rate may be obtained if the initial transaction factor μ_0 is chosen as

$$\mu_0 = c \sum_{i=1}^m |\mathbf{w}'_{t,i} - \mathbf{w}_{t,i}|. \quad (14)$$

The transaction factor μ_t is a function of the relative price $\mathbf{Y}_t$ as well as the current and previous weight vectors: In the present work, the following assumptions are made:

- it is always possible to buy or to sell an asset at any time.
- the transactions do not affect the values of the financial instruments involved. In other words, the number of assets traded by the agent is small compared to the overall market liquidity.

In order to optimize the learning process, data normalization and feature selection are addressed in the next section.

3. Features normalization and selection

As mentioned earlier, each financial instrument is characterized by four prices: the opening price, the lowest and the highest prices reached during the trading period as well as the price at the closing of the markets. In addition, financial instruments may be described by a plethora of financial indicators such as the average true range, the exponential moving average, the commodity channel index among others (Murphy, 1999): each index characterizing a specific behavior of the underlying asset. The financial indicators employed in our framework are described in Table 1; they are among the most commonly employed in the financial industry (Achelis, 2001).

Features normalization is addressed in the next subsection.

3.1. Features normalization

The portfolio is characterized by twelve (12) feature vectors. Each feature vector refers to a particular feature for all the assets forming the portfolio. The first four vectors contain the opening, low, high and closing price of all the assets respectively. The last eight vectors contain the financial indicators for all the assets as

described in Table 1. In order for the agent to efficiently learn the policy, these features must be normalized (Ross, Mineiro, & Langford, 2013). Because both the extrema and the effective range are unknown a priori, it is not possible to normalize them with the min-max approach. Therefore, the low, high and closing prices are normalized with respect to the opening price:

$$\begin{aligned} \mathbf{V}_t^{(Lo)} &= \left[\frac{Lo(t-n-1)}{Op(t-n-1)}, \dots, \frac{Lo(t-1)}{Op(t-1)} \right]^T, \\ \mathbf{V}_t^{(Cl)} &= \left[\frac{Cl(t-n-1)}{Op(t-n-1)}, \dots, \frac{Cl(t-1)}{Op(t-1)} \right]^T, \\ \mathbf{V}_t^{(Hi)} &= \left[\frac{Hi(t-n-1)}{Op(t-n-1)}, \dots, \frac{Hi(t-1)}{Op(t-1)} \right]^T. \end{aligned} \quad (15)$$

Such normalization is more informative than the min-max normalization scheme (Hussain et al., 2008) or the Z-score as it clearly identifies the market trend (up and down). A similar approach is followed for the financial indicators which are normalized with respect to their previous trading period:

$$\mathbf{V}_t^{(FI)} = \left[\frac{FI(t-n)}{FI(t-n-1)}, \dots, \frac{FI(t)}{FI(t-1)} \right]^T. \quad (16)$$

As a result, the feature tensor consists of three (3) price vectors as defined in Eq. (15) as well as of eight(8) financial indicator vectors as described by Eq. (16).

Policies are more difficult to learn when the feature tensors have a large number of dimensions which in turn has a detrimental effect on the system performance (Choung, Lee, & Jeong, 2017). Such a situation may occur when the portfolio consists of a large number of assets; a commonplace situation in real life. In addition, some of these features may be highly correlated which may result in sub-optimal learning; not to mention that the training process may become computationally prohibitive. In order to alleviate these problems, a smaller number of non-correlated and more informative features must be extracted from the original ones. The

next section describes our features selection approach which is based on an unsupervised stacked restricted autoencoder.

3.2. Features selection

Features selection is performed with an unsupervised stacked restricted autoencoder inspired by Wong and Luo (2018). The input feature vector consists of eleven (11) features namely the normalized low, high and closing prices as well as the eight normalized financial indicators; all taken at the same time t . The network is designed in order to extract low-dimensional uncorrelated features while optimizing the training process. Autoencoders are a type of unsupervised feedforward neural network that reconstructs the output from the input. It is formed of two parts namely the encoder and the decoder. The encoder reduces the dimensionality of the input data in the so-called latent of hidden-layer while the decoder reconstructs the input data from the hidden layer; such a network is said to be under-complete because of the restriction imposed by the latent layer (Bengio, Goodfellow, & Courville, 2015). Autoencoders are capable of learning nonlinear dimension reduction functions, removing redundancies and correlations while extracting highly informative features (Bengio et al., 2015).

As mentioned earlier, we employed a stacked restricted autoencoder. This network consists of seven (7) layers: one input layer, five (5) hidden layers and one output layer. The network structure is illustrated in Fig. 2. The network is asymmetrical: the first hidden layer has (10) neurons, the second (9) neurons while the next three consist of (3) neurons each. The feature extraction is performed at the level of the third hidden layer. The output layer also has only (3) neurons. The network is said to be restricted because the decoder performs a partial reconstruction of the input vector. Indeed, the three neurons of the output vector refer to the normalized low, high and closing prices. Therefore, the financial indicators are not reconstructed by the decoder. Consequently, our encoder is not only under-complete but also constrained: the constraint being the partial reconstruction at the level of the decoder. As later

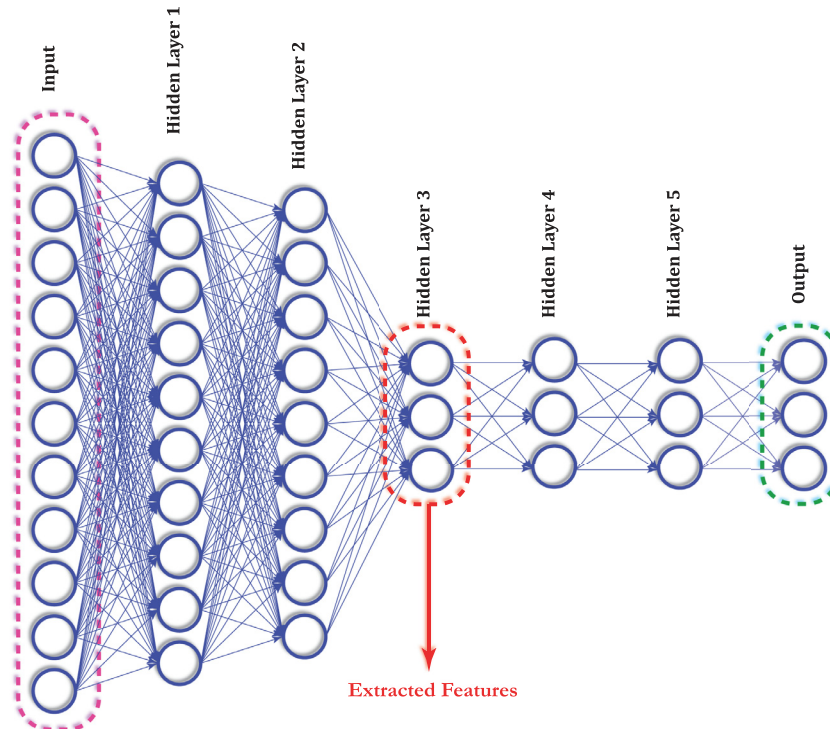


Fig. 2. Structure of the restricted stacked autoencoder.

demonstrated by the experimental results, the partial reconstruction constraint generates more informative feature vectors while putting the emphasis on what was foreseen as the most important features.

Our deep reinforcement learning algorithm is described in the next section.

4. Deep reinforcement learning

A deep reinforcement learning algorithm (Ivanov & D'yakonov, 2019) consists of an agent that takes the best actions in the environment given the state of the latter in order to maximize a reward r . Each action taken by the agent is associated with a reward. The best action is determined by a policy which is learned with deep learning techniques. The agent seeks to maximize the reward by taking proper actions. In our framework the action consists in determining the optimal weight vector for the portfolio in order to increase the expected return on investment:

$$\mathbf{a}_t = \mathbf{w}_t. \quad (17)$$

Due to the relation in between the return rate, Eq. (6a), and the transaction factor, Eq. (11), the current action is partially determined by the previous one. Therefore, the state vector consists of the feature tensor at time t as well as the weight vector at time $t - 1$ (the state of the portfolio at this time):

$$\mathbf{s}_t = [\mathbf{X}_t, \mathbf{w}_{t-1}]^T. \quad (18)$$

The portfolio weight vector $\mathbf{w}_{t-1}$ and the features tensor $\mathbf{X}_t$ correspond to the internal (or latent) and external states respectively. As previously mentioned, the primary objective is to increase the portfolio return on investment which is determined by Eq. (7) over an investment horizon $t_f + 1$. It follows that the reward is associated with the logarithmic accumulated return $\mathcal{R}$, which may be inferred from Eqs. (6b), (7) and (11):

$$\begin{aligned} \mathcal{R}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) &= \frac{1}{t_f} \cdot \ln \left(\frac{P_f}{P_0} \right), \\ &= \frac{1}{t_f} \cdot \sum_{t=1}^{t_f+1} \ln(\mu_t \cdot \mathbf{Y}_t \cdot \mathbf{w}_{t-1}) = \frac{1}{t_f} \cdot \sum_{t=1}^{t_f+1} R_t. \end{aligned} \quad (19)$$

The algorithm employed to learn the policy is described in the next subsection.

4.1. SARSA

Portfolio management is considered a decision making process in which the agent constantly takes actions in order to increase the portfolio value. The SARSA algorithm (Sutton & Barto, 2018) is employed to train a CNN in order to learn the investment policy. This network is described in the next subsection. SARSA is an on-policy which means that the previous actions taken by the agent are taken into account in the learning process. This is in contrast with the Q-learning algorithm in which the optimal policy is learned irrespectively of the actions taken by the agent. Since on-policy methods are simpler and converge faster than their off-policy counterpart, they are more adapted to portfolio management (Sutton & Barto, 2018).

Our implementation of SARSA is described in Algorithm 1. The normalized features tensor is fed to the stacked restricted autoencoder for features selection and dimensionality reduction. These new features are employed to train the convolutional neural network i.e. to determine the weights associated with the kernel. The state tensor and the action vector are initialized based on the input tensor as well as on the previous and current weight vectors. The initial weight vector is given by Eq. (2) and consists only of cash

Algorithm 1 SARSA (ϵ -greedy).

Data: $\alpha \in [0, 1]$ Learning rate

Data: $\epsilon \in [0, 1]$ epsilon value

Data: $\beta_d = 1 \times 10^{-2}$ epsilon decay rate

Data: N_t, B_n, r : Number of iterations, Batch Number, Immediate Reward

Data: π_θ, Q : Policy, state-action value

Data: $\tilde{\mathbf{w}}_t$: Selected action at $(t+1)$

Data: $\mathcal{A}, \mathcal{S}$: Action space, State Space

```

1 Initialization:
  iteration = 0
  initial  $\epsilon \leftarrow 0.6 \leq \epsilon \leq 0.85$ 
  while iteration  $\leq N_t$  do
2   if random_number  $< \epsilon$  then
3      $\tilde{\mathbf{w}}_t \leftarrow \text{random\_action} \in \mathcal{A}$ 
4   else
5      $\tilde{\mathbf{w}}_t \leftarrow \pi_\theta(\mathbf{X}_{t+1}, \mathbf{w}_t)$ 
6   end
7    $\delta \leftarrow r_{t+1} + \gamma \cdot Q(\mathbf{X}_{t+1}, \mathbf{w}_t, \tilde{\mathbf{w}}_t)$ 
8   for  $(\mathbf{X}_{t+1}, \mathbf{w}_t, \tilde{\mathbf{w}}_t) \in \mathcal{S} \times \mathcal{A}$  do
9      $Q(\mathbf{X}_t, \mathbf{w}_{t-1}, \mathbf{w}_t) \leftarrow Q(\mathbf{X}_t, \mathbf{w}_{t-1}, \mathbf{w}_t) + \alpha \cdot (\delta -$ 
10     $Q(\mathbf{X}_t, \mathbf{w}_{t-1}, \mathbf{w}_t)).$ 
11  end
12  Update:
     $[\mathbf{X}_{t+1}, \mathbf{w}_t]^T \leftarrow [\mathbf{X}_t, \mathbf{w}_{t-1}]^T$ 
     $\mathbf{w}_{t+1} \leftarrow \tilde{\mathbf{w}}_t$ 
    Update:
     $\epsilon \leftarrow \epsilon_{\min} + (\epsilon_{\max} - \epsilon_{\min}) \cdot \exp(-\beta_d \cdot B_n)$ 
13 end
14 Return:  $Q$ 

```

which means that the CNN is entirely responsible for the reallocation of the assets. The discount rate γ determines the trade-off in between immediate (or myopic reward) and long term strategy. The Q value corresponds to the average logarithmic accumulated return. The Q value is defined as

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \cdot (r_{t+1} + \gamma \cdot Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q(\mathbf{s}_t, \mathbf{a}_t)), \quad (20)$$

where r_{t+1} is the immediate reward. Therefore, following a transition from state $\mathbf{s}_t$ to $\mathbf{s}_{t+1}$, the Q value must be updated according to Eq. (20). In other words, the Q value is continuously estimated based on the current policy π_θ while the latter is repeatedly optimized in order to increase the Q value. The action and the state are given by Eqs. (17) and (18) respectively. Each action is taken according to an ϵ -greedy strategy: a random number is generated from a uniform distribution: if this number is smaller than a given threshold, a random action is taken; if not, the action is determined by the current policy. This strategy improves the exploration of the action space which results in turn in an optimal solution while avoiding over-fitting and addiction (Garcia & Fernández, 2012). As such, this approach is reminiscent of simulated annealing (Van Laarhoven & Aarts, 1987). The ϵ value is initially chosen close to one and is progressively reduced toward zero as the policy is learned. Thereafter, the policy is updated according to Eq. (20) which is essentially a gradient descent algorithm (Sutton, Barto et al., 1998).

The implementation of the gradient descent algorithm is described in the next section.

4.2. Online stochastic batching

The gradient descent algorithm is implemented with an online stochastic batching approach in order to obtain an unbiased and

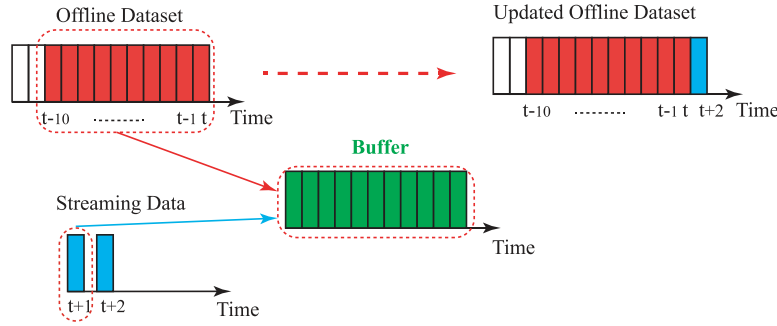


Fig. 3. Online learning batches generation.

optimal estimate of the gradient (Jiang et al., 2017). Financial data are sequential time series. Therefore, training batches must follow the same time ordering as the original data. In addition, different starting points generate completely distinctive batches. As a result, distinct mini-batches may be obtained by simply changing their starting point. For instance, $[t_B, t_B + B_s]$ and $[t_B + 1, t_B + B_s + 1]$ correspond to two distinct batches. In this paper, the online stochastic method, as proposed by Jiang et al. (2017), is employed in order to generate mini-batches. In this approach, it is assumed that the starting points are distributed according to a geometrical distribution:

$$P_\beta(t_B) = \beta(1 - \beta)^{t_B - B_s}, \quad (21)$$

where the decaying rate is given by $\beta \in (0, 1)$ and where the relation in between the starting point and the mini-batch size is given by B_s . This distribution tends to favor more recent events; the extent of which is controlled by the decaying rate. In order to train the CNN, both offline learning and online learning are utilized.

Our learning strategy is described in the next subsection.

4.3. Offline and online learning

The CNN learns the policy offline which means that the policy is learned from historical data. Historical data is any data anterior to the current calendar time. The offline learning technique is based on online stochastic batching which has been described in the previous subsection. Nevertheless, the underlying distribution associated with a financial time series may vary over time. This phenomenon is called concept drift (Gama et al., 2014). In the case of financial instruments, concept drift is usually triggered by unforeseen exogenous factors such as political turmoil, unexpected loss or panic reaction (Cavalcante & Oliveira, 2015). These factors may affect the efficacy of the actions taken by the agent. Essentially, concept drift may be handled in two different ways which are known as the active and the passive approach (Rodríguez & Kuncheva, 2008). In the first case, a concept drift detector attempts to detect the occurrence of a concept drift. This is usually achieved by determining if there is a noteworthy change in

the underlying data distribution (Elwell & Polikar, 2009). Once a concept drift is detected, the agent is trained afresh from the subsequent events (Gama, Medas, Castillo, & Rodrigues, 2004). In the second approach, the agent is initially trained from historical data and then, continuously updated as new data become available.

In our framework, we employ the passive approach which bestows us with feature drift adaptation (Gama et al., 2014). This is mainly motivated by two reasons: firstly, in trading, behaviors or events that occurred in the past are likely to repeat themselves in the future (Hu et al., 2015). By contrast, the active approach does not take advantage of this prior knowledge that exists in a latent in the historical data. Secondly, if the market is unstable, concept drift may occur repeatedly, and the agent might tend to employ short-term investment strategies which may have a detrimental effect on long-term investments (Hu et al., 2015). Nonetheless, passive concept drift detection tends to forget patterns that do not repeat themselves after a certain amount of time (Gama et al., 2004). These so-called anomalies have a relatively small impact on long-term investment strategies.

The online learning is performed for both the restricted stacked autoencoder and the CNN. The batches for online learning are generated as outlined in Fig. 3. A buffer is designed to store the last ten (10) days of offline data along with one time-step (one business day) of streaming data. At the end of each day, the offline dataset is expended by adding the current data point to the offline dataset.

In the next subsection, the convolutional neural network responsible for learning the policy is described

4.4. Convolutional neural network

In our framework, the policy is enforced with a convolutional neural network which is illustrated in Fig. 4. Indeed, convolutional neural networks have demonstrated their ability to learn and implement efficiently complex policies in a deep reinforcement learning framework (Jiang et al., 2017). The policy is initially learned offline from historical data with SARSA in conjunction with on-

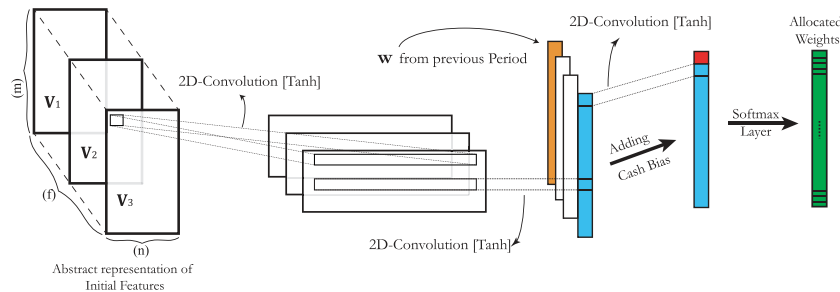


Fig. 4. Structure of the CNN employed to learn the policy.

line stochastic batching. Once the offline training is completed, the policy is continuously updated online as new information becomes available.

The architecture of our CNN is described in Algorithm 2. The

Algorithm 2 Implementation of convolutional neural network.

Data: f, m, n : Number of Features, Number of Assets, Trading Window Length

Data: V_1, V_2, V_3 : Extracted Features

13 Initialization:

Input: $X_t = [V_1, V_2, V_3]^T$

while $i \leq n$ **do**

1. **First Convolution:**

Activation Function: Tanh

Kernel size $\rightarrow (1,3)$

Kernel depth $\rightarrow f$

2. **Second Convolution:**

Activation Function: Tanh

Kernel size $\rightarrow (1,n)$

Kernel depth $\rightarrow f$

3. **Third Convolution:**

Concatenate weights for previous and current time steps (Weight Matrix).

$W = [W_1, W_2, \dots, W_i]^T$

Activation Function: Tanh

Kernel size $\rightarrow (1,1)$

Kernel depth $\rightarrow f + 1$

4. **Adding Cash:**

Concatenate Cash Bias and computed Weight Matrix

5. **Softmax Layer:**

Return: Distributed Portfolio Weights $\leftarrow W_{pf} =$

$[w_c, w_1, w_2, \dots, w_m]^T$

14 end

CNN is fed with the features learned by the restricted stacked autoencoder. These features form a tensor which consists of the three (3) channels. Each channel forms a matrix which is associated with a particular abstract feature. Each matrix contains the values of the abstract feature for the assets (rows) forming the portfolio taken for the last n days (columns). Then, convolution is performed on each channel with a single kernel of size 1×3 followed by a *Tanh* activation function. Once more, a convolution with *Tanh* activation is performed on each of the three channels obtained from the previous convolution with a single kernel of size $1 \times n$ in order to obtain three vectors of size m . Thereafter, a fourth layer channel is added which consists of the weights (action) obtained from the previous iteration. A third and last convolution with activation is performed with a 1×1 kernel (the depth is equal to the number of channels) in order to obtain a single vector of size m . The last two steps constitute an implementation of Eq. (18) which states that the current action is determined by the current state as well as the previous action (weights). Finally, the previous vector is normalized with a function in order to obtain the portfolio new weights i.e. the current action.

The settlement process is addressed in the next section.

5. Settlement with blockchain

So far, we have implicitly assumed that stock exchange transactions may be finalized in real-time. Unfortunately, this is not the case. Indeed, although a sale or a purchase may be performed in a fraction of a second, a transaction settlement may take days, even weeks (Iansiti & Lakhani, 2017). This is at the origin of a major problem called settlement risk which results from a failure, either

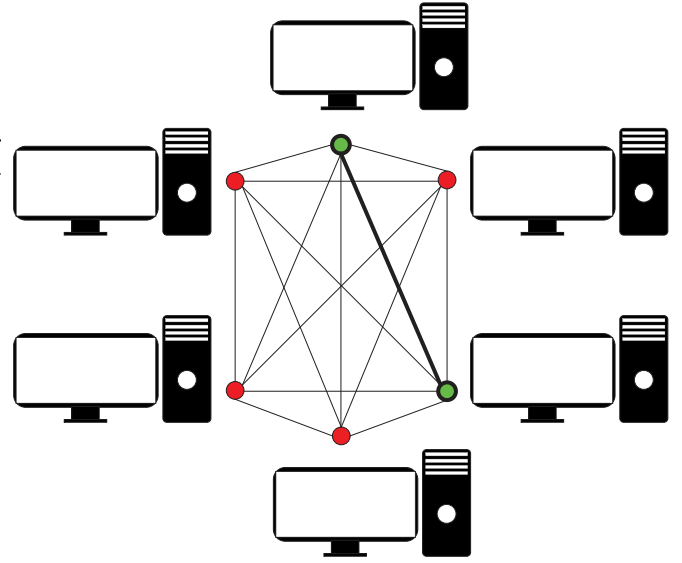


Fig. 5. Distributed network associated with the settlement blockchain.

on the part of the seller or on the part of the buyer, to deliver a sold asset to or to pay for the latter.

Generally speaking, a specialized third-party coordinates legal ownership of securities versus payment: the so-called Central Securities Depository (Schmiedel, Malkamäki, & Tarkka, 2006). In addition, intermediaries such as brokers, custodians and payment agents are involved in the process (Pacces, 2000). Consequently, a typical settlement process is both time, and cost consuming (Benos, Garratt, & Gurrola-Perez, 2017). In order to address this problem, we have implemented a settlement system on a Bitcoin blockchain (Nakamoto et al., 2008). It is assumed that the assets may be exchanged on a digital market: the extent of which is determined by legal considerations rather than by technical ones. The cash and assets associated with the portfolio are stored in a digital wallet (Dwyer, 2015).

Each time the reinforcement learning algorithm sells or buys an asset, a transaction is initiated. Each transaction corresponds to a block in the blockchain. A transaction consists of two primary components; the input object, which contains the information about the original sender (timestamp, balance, signature, sender's public key) and the output object, which contains the nature and the amount of the transactions as well as the address of the receiver. The transaction is not validated by a third party but by a distributed consensus mechanism known as proof-of-work (Nakamoto et al., 2008).

In a consensus mechanism, all the nodes forming the network, such as the one illustrated in Fig. 5, compete to solve a cryptographic problem which is computationally prohibitive to solve but whose solution is easily validated. The nodes or miners may only solve this problem on a trial and error basis which means that it is almost impossible to obtain rapidly the solution. Consequently, it is virtually impossible to predict which miner shall validate the transaction.

The cryptographic problem consists of finding a number which is lower than the hash (Sobti & Geetha, 2012) associated with the new block (target hash) associated with the transaction. Not only this number must be lower than the target hash, but it must be determined within a certain tolerance given by the nonce. The lower the tolerance, the more difficult the problem becomes. The miner that first solves the problem adds the new validated block to the blockchain and receives a reward in terms of either digital or cryptocurrencies as an incentive. The validation consists of determining

Table 2

Composition of our Portfolio for the period in between January 2, 2002 and July 31, 2018.

Sector	Name
Technology	Apple Inc. (AAPL)
	Adobe Inc. (ADBE)
	Cisco Systems, Inc. (CSCO)
	Intel Corporation (INTC)
	International Business Machines Corporation (IBM)
	Microsoft Corporation (MSFT)
Financial Services	NVIDIA Corporation (NVDA)
	Oracle Corporation (ORCL)
	Bank of America Corporation (BAC)
	HSBC Holdings plc (HSBC)
	Royal Bank of Canada (RY)
	The Toronto-Dominion Bank (TD)
Consumer Cyclical	Amazon.com, Inc. (AMZN)
	The Walt Disney Company (DIS)
Consumer Defensive	The Home Depot, Inc. (HD)
	Walmart Inc. (WMT)
Industrials	The Coca-Cola Company (KO)
	Boeing Co (BA)
Healthcare	Johnson & Johnson (JNJ)
	Merck & Co., Inc. (MRK)
Communication Services	Novo Nordisk A/S (NVO)
	Novartis AG (NVS)
	Verizon Communications Inc. (VZ)

if both the money and the asset are available and then, to finalize the transaction. As a result, the settlement problem is solved without any recourse to an intermediary (Wall & Malm, 2016). In addition, such a distributed system may overcome a cyber-attack in which 50% or less of its nodes are overtaken (Sayeed & Marco-Gisbert, 2019).

It is important to point out that the blockchain is not an essential part of our framework in the sense that it does not affect the deep reinforcement learning investment strategy. The role of the blockchain is to mitigate the settlement risk; a risk that is also present in traditional trading. Furthermore, our approach is not limited to digital assets as it may be applied to any financial instrument traded on a stock exchange.

Our experimental results are presented in the next section.

6. Experiment and results

The composition of our portfolio, in terms of stocks, appears in Table 3. The times series starts on January 2, 2002, and ends on July 31, 2018.

In order to evaluate the performance of our system in terms of return on investment, two metrics were employed: namely the Sharpe ratio and the maximum drawdown (MDD). These metrics are among the most commonly employed in portfolio performance evaluation (Bessler, Opfer, & Wolff, 2017). The Sharpe ratio was introduced by Sharpe (1994) and is essentially a ratio in between the

Table 3

Hyper-parameters.

Hyper-parameters	Size	Description
Portfolio Size	23	Number of assets considered to create a portfolio
Trading Window	30	Number of trading days in which agents trained to increase the portfolio value (investment return)
Trading Period	1 Day	Length of time in which price movements occur
Time Interval	4172	Number of Days from 02/01/2002 to 27/07/2018
Offline Learning Rate	10^{-4}	Parameter α of the Adam optimization (Kingma & Ba, 2015) for offline training.
Online Learning Rate	10^{-1}	Learning rate for online training session.
Sample Bias	10^{-5}	Parameter of geometric distribution when selecting online training sample batches. (The parameter β in Eq. (21) of Section 4.2)
Commission Rate	1.5%	Rate of commission fee applied to each transaction
Risk Free Rate	1%	The theoretical investment return rate with zero risk
Regularization Coefficient	10^{-8}	The L2 regularization coefficient.

Table 4

Training and test sets.

ID	Training set	Test set
Test 1	2002-01-02 to 2008-08-15	2009-06-15 to 2009-10-21
Test 2	2002-01-02 to 2011-12-06	2012-10-03 to 2013-02-14
Test 3	2002-01-02 to 2015-04-06	2016-02-01 to 2016-06-09
Test 4	2002-01-02 to 2017-05-19	2018-03-19 to 2018-07-27

Table 5Maximum drawdown (MDD) and Sharpe ratio (S_r) as obtained with our system for the four test sets.

ID	Investment duration	MDD (%) algorithm	S_r Algorithm
Test Set 1	30 Days	1.71	-1.82
	60 Days	4.87	1.57
	90 Days	7.49	2.44
Test Set 2	30 Days	2.43	-1.014
	60 Days	6.91	2.22
	90 Days	8.23	2.20
Test Set 3	30 Days	4.62	2.9
	60 Days	9.44	2.34
	90 Days	17.46	3.47
Test Set 4	30 Days	3.13	3.13
	60 Days	12.52	3.09
	90 Days	12.52	2.26

Table 6

Return on investment for our approach and for the Dow Jones Industrial (Bloomberg, 2019).

ID	Investment duration	RoI (%) algorithm	RoI(%) DJI
Test Set 1	30 Days	0.24	5.62
	60 Days	3.67	10.85
	90 Days	5.74	15.52
Test Set 2	30 Days	0.5	-6.85
	60 Days	4.83	-4.12
	90 Days	6.91	3.88
Test Set 3	30 Days	2.43	4.88
	60 Days	7.4	9.68
	90 Days	18.09	9.33
Test Set 4	30 Days	2.83	-2.07
	60 Days	11.14	2.4
	90 Days	11.93	3.72

expected return on investment and the risk:

$$S_r = \frac{E[R_f - R_b]}{\sigma_d} \quad (22)$$

where the numerator is the expectation on the differential return and where σ_d is the standard deviation associated with the asset excess return (or volatility); the higher the Shape ratio the better. It is generally accepted that a Sharpe ratio below one is sub-optimal; a ratio in between one and two is acceptable while a ratio in between two and three is considered very good. Any ratio over three is regarded as excellent (Maverick, 2019). The maximum draw down metric was introduced by Magdon-Ismail and

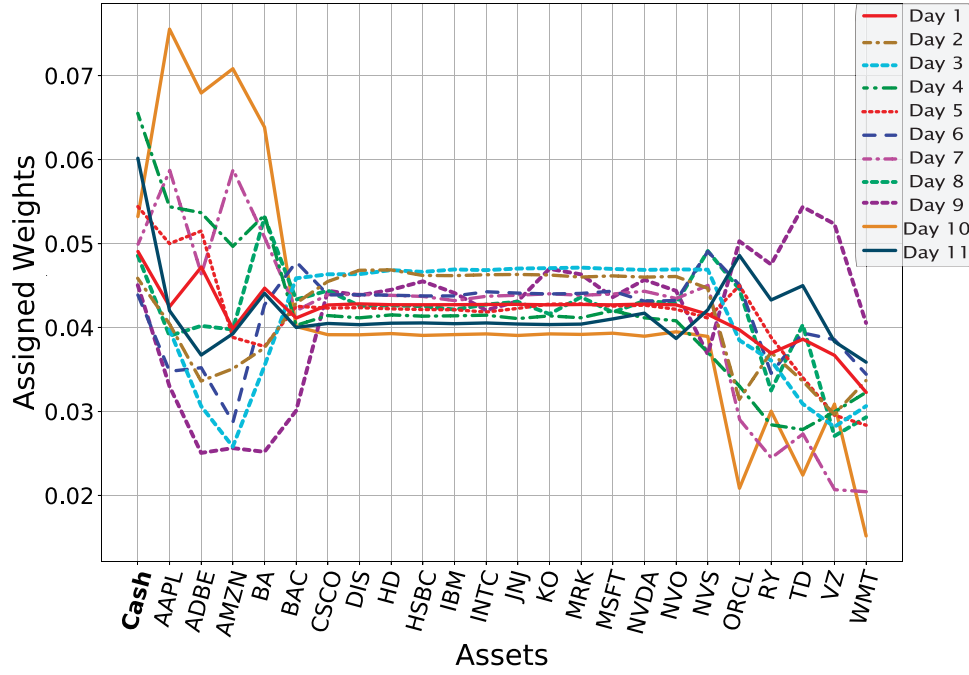


Fig. 6. Evolution of the portfolio weights during the first 11 days of training for the online learning process.

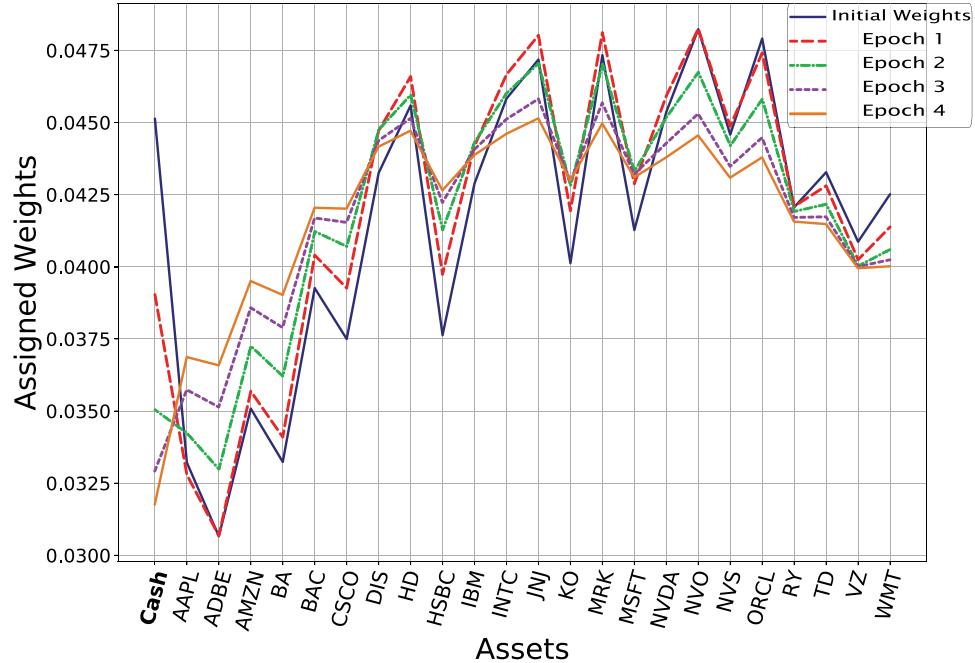


Fig. 7. Final portfolio weights distributions for the first training set.

Atiya (2004). This ratio is a function of the maximum P and minimum value L reached by the portfolio throughout a certain period of time; performant portfolios are characterized by a high MDD.

$$MDD = \frac{P - L}{P}. \quad (23)$$

In order to validate our approach, four training sets were generated. As mentioned earlier, the assets forming the portfolio are described in Table 2. Each training set covers a different period or window: all periods are initiated on January 2, 2002, while ending on August 15, 2008; December 6, 2011; April 6, 2015, and May 19, 2017 respectively. To these four training sets correspond four test sets of 90 days each starting the day following the end of their

respective training sets. The training sets and their corresponding test sets are described in Table 4. Initially, the portfolio consists solely of cash. For each training set, the agent learns the weights associated with the assets in order to increase the expected return on investment. The hyper-parameters associated with the agent appear in Table 3.

The final weights distributions, for the first training set, during training episodes, are shown in Fig. 7. The final weights distributions of the second training set, for the four training episodes, are shown in Fig. 10. As it may be noticed from the figures, the final distribution is affected by the duration of the training period. Indeed, longer periods encompass states or events that do not occur

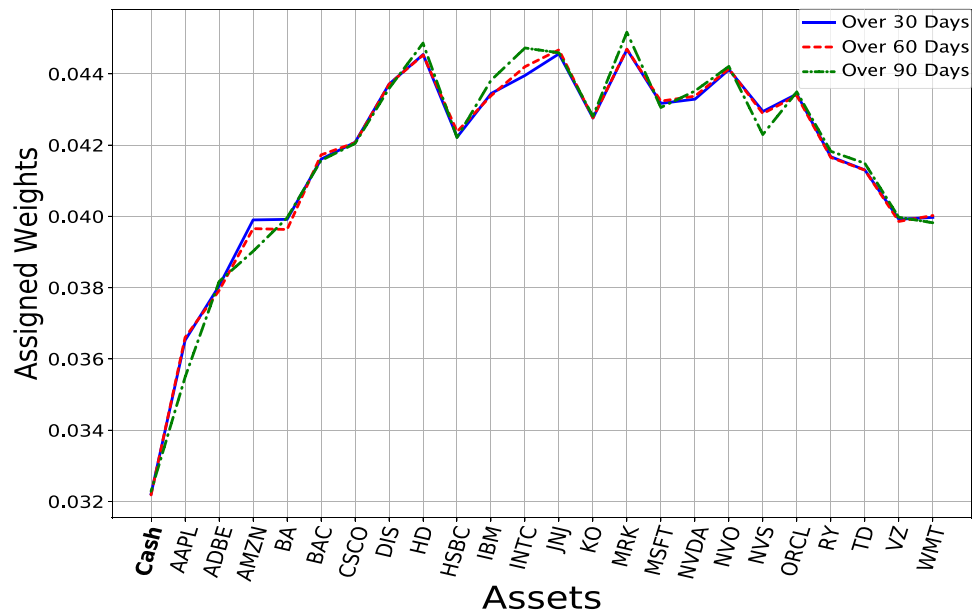


Fig. 8. Final portfolio weights distributions for the first test set over a period of 30, 60 and 90 days.

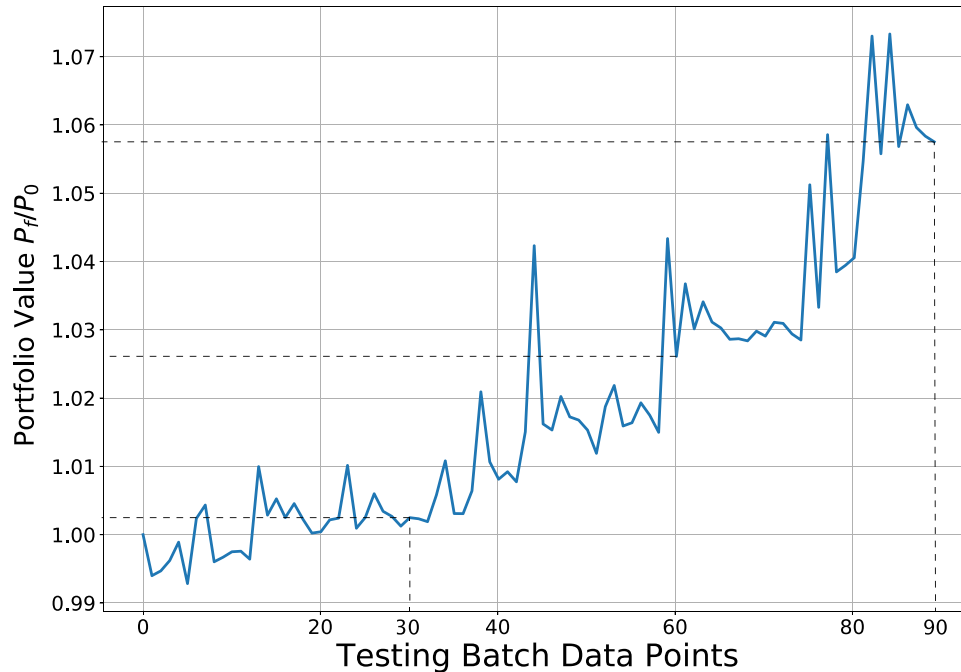


Fig. 9. Relative return on investment for the first testing set over a period of 30, 60 and 90 days.

in shorter periods which in turn affect the decisions taken by the agent. As illustrated by Fig. 6, weights fluctuate during the online learning process. Nevertheless, some weights are subjected to important variations, such as Apple and Amazon, while others, such as HSBC and IBM, are not. This behavior may be explained by the fact that the assets associated with the most stable weights are utilized in order to mitigate the risk (portfolio hedging) while the most volatile are employed for their leverage effect which explains the high variability of their weights.

Fig. 8 shows the final weights distributions, for the first test set, after thirty, sixty and ninety days respectively. Although sometimes subtle, the corrections have nevertheless a large impact on the expected return on investment. Indeed, as illustrated by Fig. 9, the relative return on investment, with respect to the initial amount of

cash, steadily increases over this ninety days period with a relative gain of almost 7%. Despite some fluctuations, there is a continued upward trend.

Likewise, Figs. 10–18 show the final training weights distributions for the last three training sets, the final weights distributions for the last three testing sets as well as their respective relative return on investments. In each set, a clear upward trend for the expected return on investment may be observed. The longer the training period, the more discontinuous is the evolution of the expected return on investment. The origin of the behavior is to be found in the fact that a longer training period allows the agent to learn more complex actions which in turn result in more drastic decisions. Nonetheless, the process remains very efficient as the fi-

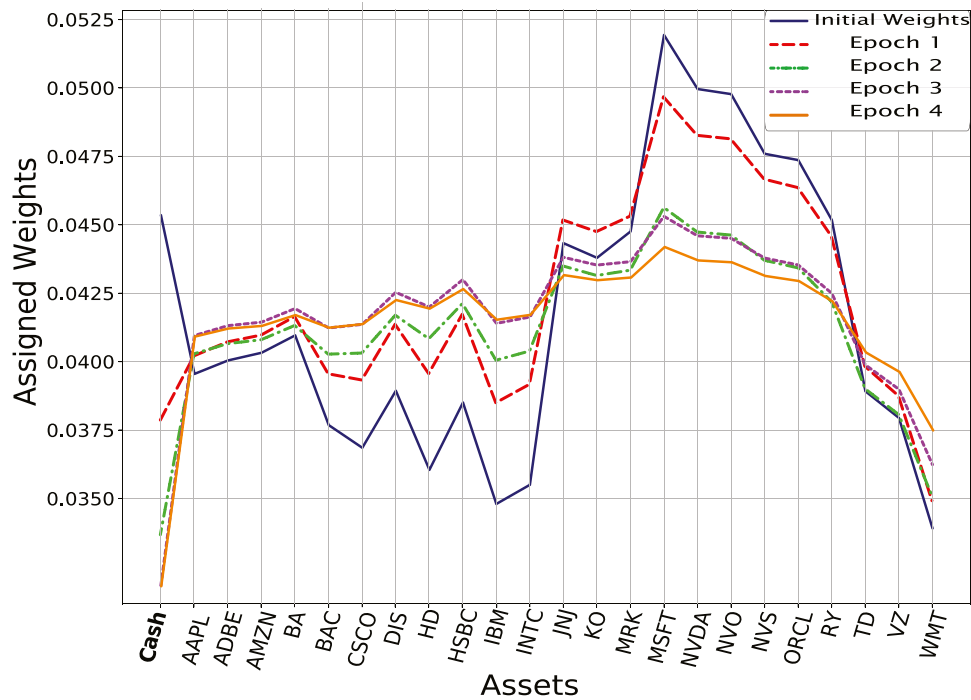


Fig. 10. Final portfolio weights distributions for the second training set.

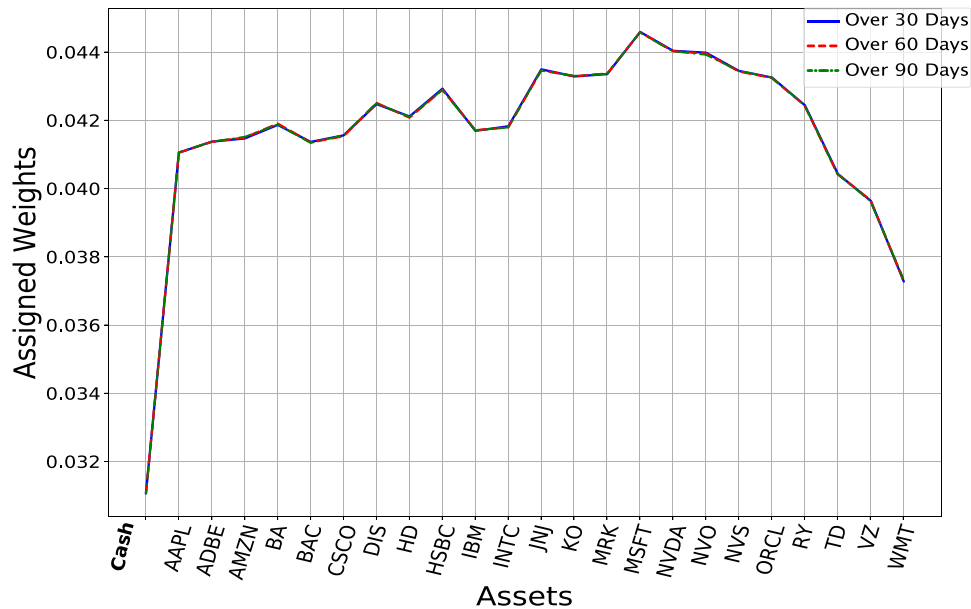


Fig. 11. Final portfolio weights distributions for the second test set.

nal relative returns on investment are approximately 7%, 18% and 11% for the last three test sets respectively.

Our results are summarized in Table 5. From the table, it should be noticed that the Sharpe ratios associated with our approach are either very good or excellent for nine out of the twelve test sets; the latter being described in Table 4. The test set 1 has the shortest training window while Test set 4 as the longest. Therefore, Table 5, clearly demonstrates that the performances of our system are improved if the CNN is trained over a large historical dataset such as Test set 4. In other words, a large amount of historical data is required in order to learn efficiently the policy. Indeed, for Test set 4, the Sharpe ratio is excellent for a time horizon of thirty and sixty days while it is very good for a long-term investment made

over ninety days. These results are corroborated by the maximum drawdown metric.

In order to further ascertain the performances of our system, we compare the expected return on investment of our approach with the expected return on investment associated with the Dow Jones Industrial (DJI); the latter being considered a benchmark in the finance industry (Zhang, Wang, Li, & Shen, 2018). We restrict our discussion to Test set 4 as we have established that the agent must be trained over a large historical dataset in order to properly learn the policy. Our results are reported in Table 6. With our approach, the returns on investment (RoI) are 2.83%, 11.14% and 11.93% over periods of 30, 60 and 90 days respectively while the RoI, for the DJI, is 2.07% (a loss!), 2.4% and 3.72% over the same

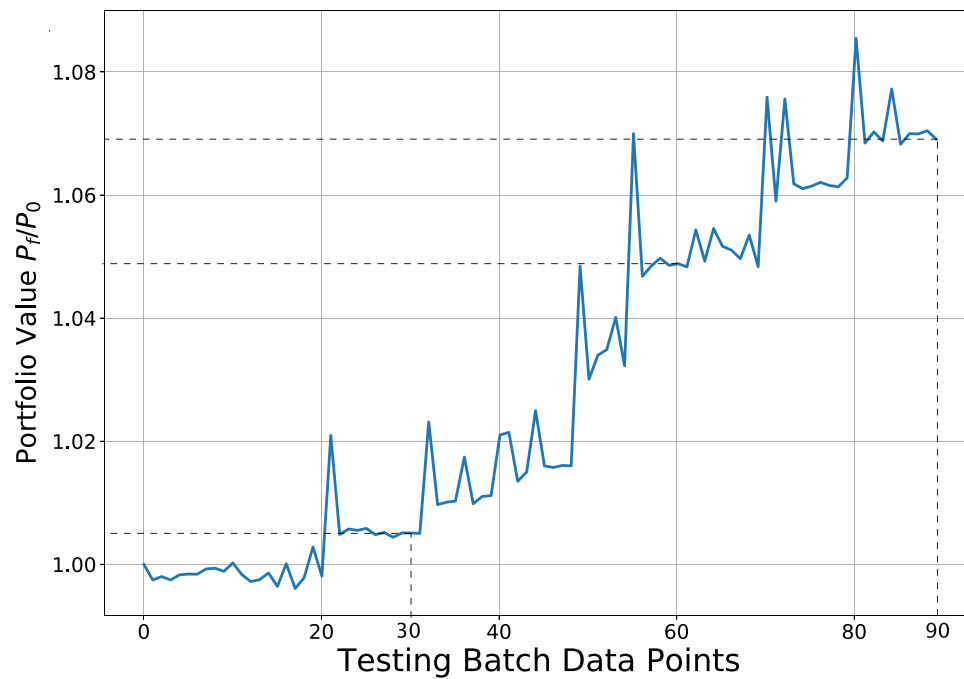


Fig. 12. Relative return on investment for the second testing set over a period of 30, 60 and 90 days.

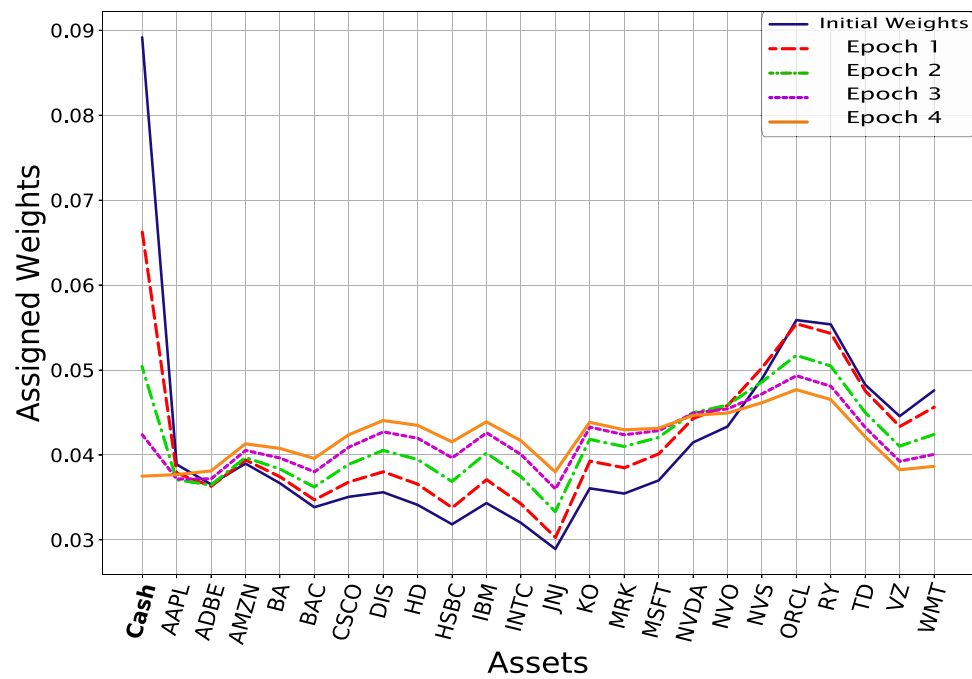


Fig. 13. Final portfolio weights distributions for the third training set.

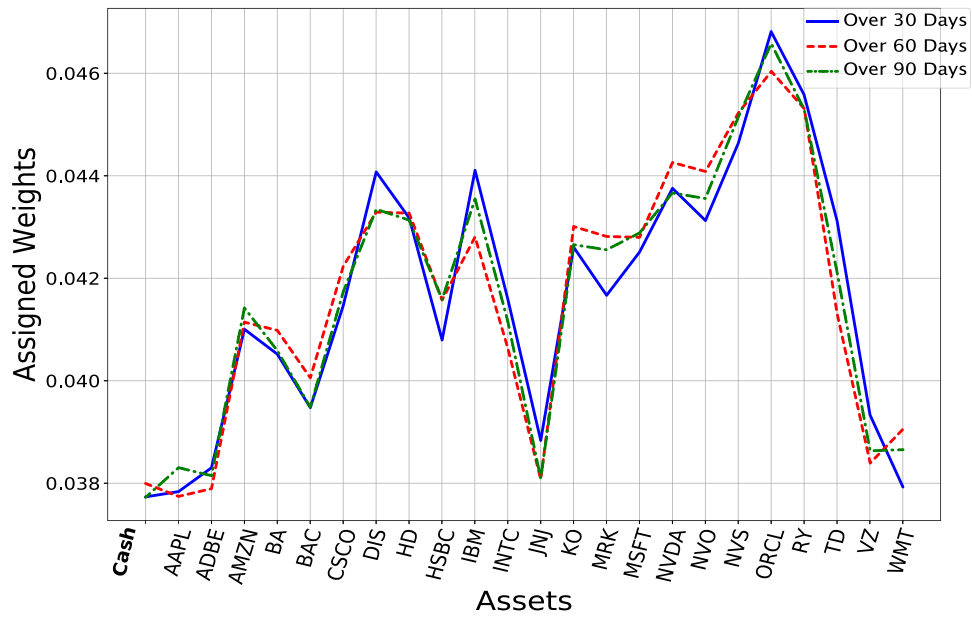


Fig. 14. Final portfolio weights distributions for the third test set.

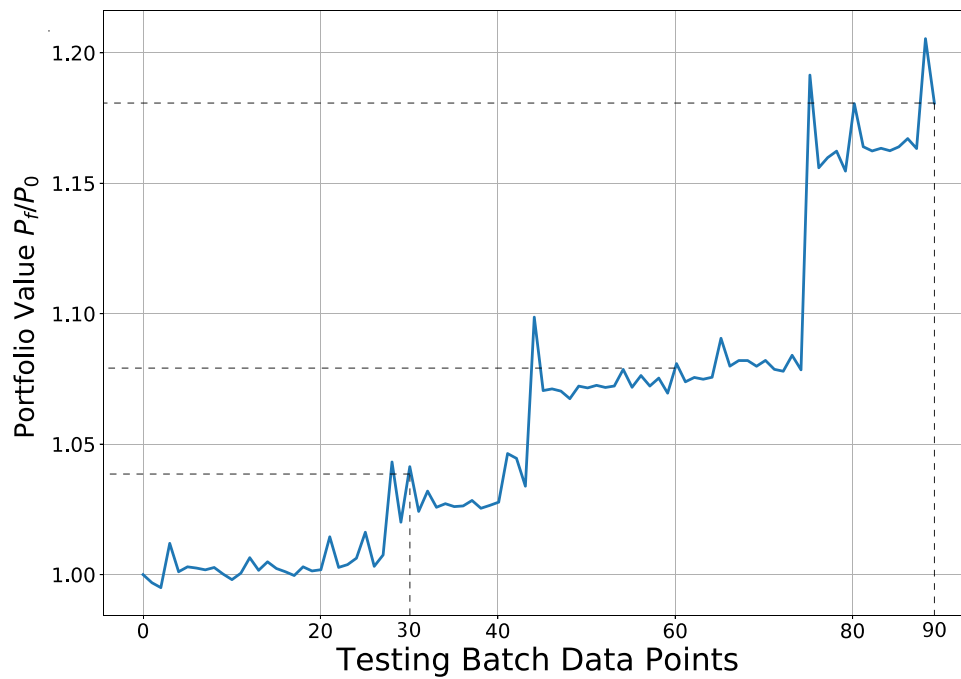


Fig. 15. Relative return on investment for the third testing set over a period of 30, 60 and 90 days.

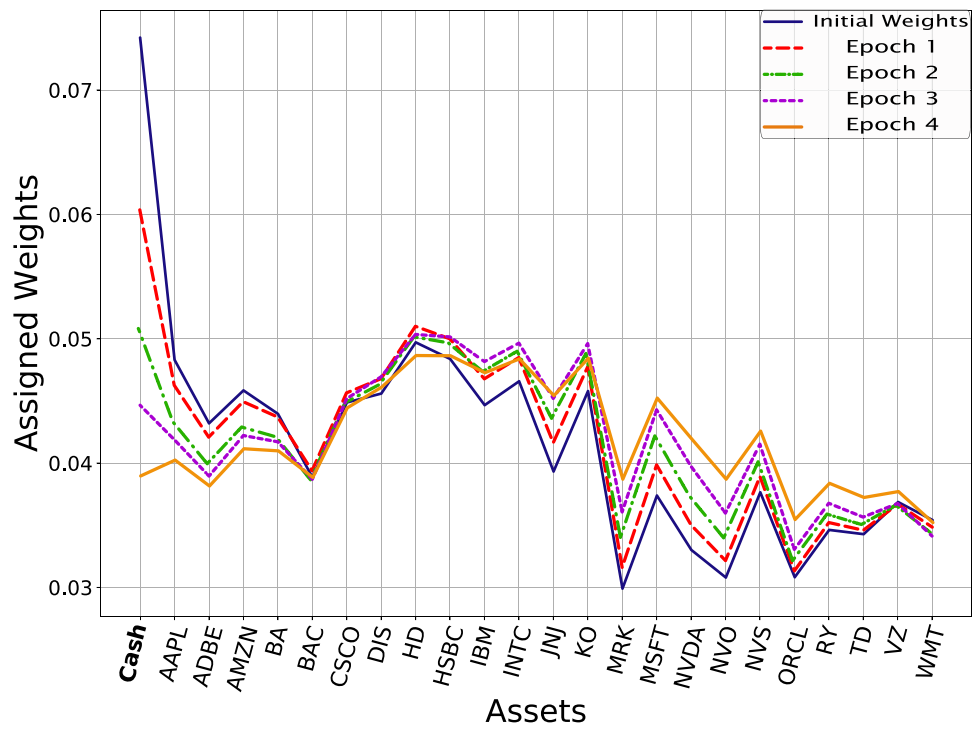


Fig. 16. Final portfolio weights distributions for the fourth training set.

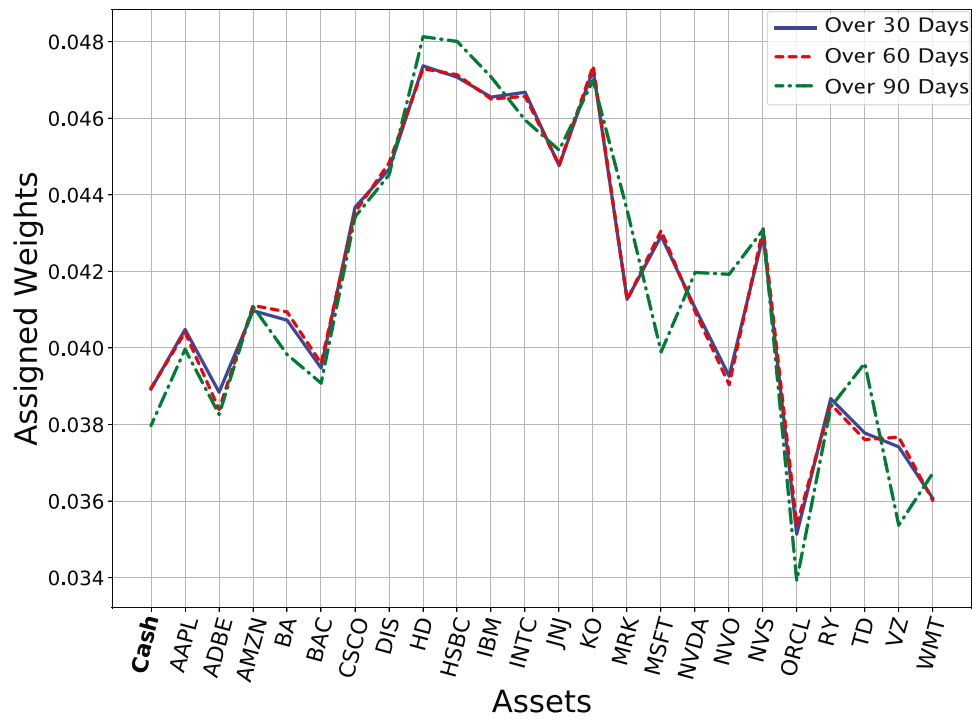


Fig. 17. Final portfolio weights distributions for the fourth test set.

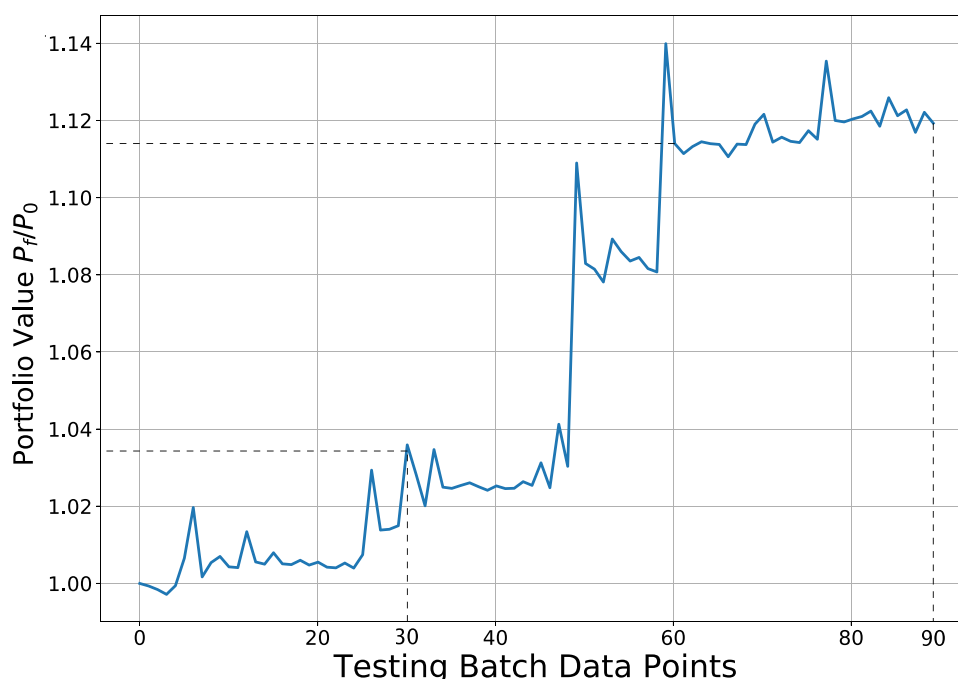


Fig. 18. Relative return on investment for the fourth testing set over a period of 30, 60 and 90 days.

time horizons. These results clearly demonstrate the effectiveness of our system.

7. Conclusion

This paper proposed a framework for portfolio management and optimization based on deep reinforcement learning called DeepBreath. A portfolio contains multiple assets that are traded simultaneously to automatically increase the expected return on investment while minimizing the risk. The investment policy is implemented using CNN; as a result, financial instruments are reallocated by buying and selling shares on the stock market. The neural network, as well as its hyper-parameters, is common to all the assets which means that the computational complexity only increases linearly with the number of financial instruments. In order to reduce the computational complexity associated with the size of the training dataset, a restricted stacked autoencoder was employed in order to obtain non-correlated and highly informative features. The investment policy was learned with the SARSA algorithm, both offline and online, using online stochastic batching and passive online learning respectively. The passive online learning approach handled concept drift resulting from exogenous factors. The settlement risk problem was mitigated with a blockchain. Our experimental results demonstrated the efficiency of our system. In future work, exogenous factors, such as social media, will be integrated into the framework in order to take into account their effect on price fluctuations. In addition, the theoretical model will be improved in order to take into account transactions in which a large volume of assets are traded i.e. when the actions (buying or selling) of the agent may have a large impact on the environment (large variation in prices and market liquidity). Finally, we plan to explore deep reinforcement learning architectures which are more suitable for long-term investment strategies.

Declaration of Competing Interest

All authors have participated in (a) conception and design, or analysis and interpretation of the data; (b) drafting the article or

revising it critically for important intellectual content; and (c) approval of the final version.

This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue.

The authors have no affiliation with any organization with a direct or indirect financial interest in the subject matter discussed in the manuscript

Credit authorship contribution statement

Farzan Soleymani: Conceptualization, Methodology, Writing - original draft, Software, Visualization, Validation. **Eric Paquet:** Supervision, Writing - review & editing, Data curation.

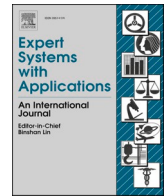
References

- Achelis, S. B. (2001). *Technical analysis from a to z*. McGraw Hill New York.
- Atsalakis, G. S., & Valavanis, K. P. (2009). Surveying stock market forecasting techniques-part II: Soft computing methods. *Expert Systems with Applications*, 36(3), 5932–5941.
- Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS One*, 12(7), e0180944.
- Bengio, Y., Goodfellow, I. J., & Courville, A. (2015). Deep learning. *Nature*, 521(7553), 436–444.
- Benos E, Garratt R, Gurrola-Perez P. The economics of distributed ledger technology for securities settlement. Available at SSRN 3023779. 2017 Aug 22.
- Bessler, W., Opfer, H., & Wolff, D. (2017). Multi-asset portfolio optimization and out-of-sample performance: An evaluation of black-litterman, mean-variance, and naïve diversification approaches. *The European Journal of Finance*, 23(1), 1–30.
- Bloomberg (2019). Dow Jones Industrial (DJI), stock quote. <https://www.bloomberg.com/quote/INDU:IND>. Accessed: 2019-04-15.
- Bouchaud, J.-P. (2011). The endogenous dynamics of markets: Price impact, feedback loops and instabilities. *Lessons from the credit crisis*. Risk Publications.
- Cavalcante, R. C., & Oliveira, A. L. I. (2015). An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection. In *2015 International joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.
- Cheng, Z., Sun, H., Takeuchi, M., & Katto, J. (2018). Deep convolutional autoencoder-based lossy image compression. In *2018 Picture coding symposium (PCS)* (pp. 253–257). IEEE.
- Choung, O.-h., Lee, S. W., & Jeong, Y. (2017). Exploring feature dimensions to learn a new policy in an uninformed reinforcement learning task. *Scientific Reports*, 7(1), 17676.
- Cutler, DM, Poterba, JM, & Summers, LH (1988). What moves stock prices? *National Bureau of Economic Research*, 17 Mar 1.

- Dwyer, G. P. (2015). The economics of bitcoin and similar private digital currencies. *Journal of Financial Stability*, 17, 81–91.
- Eakins, S. G., & Stansell, S. R. (2003). Can value-based stock selection criteria yield superior risk-adjusted returns: An application of neural networks. *International Review of Financial Analysis*, 12(1), 83–97.
- Elwell, R., & Polikar, R. (2009). Incremental learning of variable rate concept drift. In *International workshop on multiple classifier systems* (pp. 142–151). Springer.
- Fama, E. F. (1991). Efficient capital markets: II. *The Journal of Finance*, 46(5), 1575–1617.
- Filimonov, V., & Sornette, D. (2012). Quantifying reflexivity in financial markets: Toward a prediction of flash crashes. *Physical Review E*, 85(5), 056108.
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Brazilian symposium on artificial intelligence* (pp. 286–295). Springer.
- Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4), 44.
- García, J., & Fernández, F. (2012). Safe exploration of state and action spaces in reinforcement learning. *Journal of Artificial Intelligence Research*, 45, 515–564.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 504–507.
- Hu, Y., Liu, K., Zhang, X., Xie, K., Chen, W., Zeng, Y., & Liu, M. (2015). Concept drift mining of portfolio selection factors in stock market. *Electronic Commerce Research and Applications*, 14(6), 444–455.
- Hussain, A. J., Knowles, A., Lisboa, P. J. G., & El-Deredy, W. (2008). Financial time series prediction using polynomial pipelined neural networks. *Expert Systems with Applications*, 35(3), 1186–1199.
- Iansiti, M., & Lakhani, K. R. (2017). The truth about blockchain. *Harvard Business Review*, 95(1), 118–127.
- Ivanov, S., & D'yakonov, A. (2019). Modern deep reinforcement learning algorithms. arXiv preprint arXiv:1906.10025.
- Jangmin, O., Lee, J., Lee, J. W., & Zhang, B.-T. (2006). Adaptive stock trading with dynamic asset allocation using reinforcement learning. *Information Sciences*, 176(15), 2121–2147.
- Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. arXiv:1706.10059.
- Joulin, A., Lefevre, A., Grunberg, D., & Bouchaud, J.-P. (2008). *Stock price jumps: News and volume play a minor role* arXiv:0803.1769.
- Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization (2014). arXiv:1412.6980. 15.
- Lam, M. (2004). Neural network techniques for financial performance prediction: Integrating fundamental and technical analysis. *Decision Support Systems*, 37(4), 567–581.
- Magdon-Ismael, M., & Atiya, A. F. (2004). Maximum drawdown. *Risk Magazine*, 17(10), 99–102.
- Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1), 59–82.
- Malkiel, B. G., & Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2), 383–417.
- Maverick, J. B. (2019). What is a good sharpe ratio? <https://www.investopedia.com/ask/answers/010815/what-good-sharpe-ratio.asp>.
- Moody, J., Wu, L., Liao, Y., & Saffell, M. (1998). Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5–6), 441–470.
- Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.
- Nakamoto, S. Bitcoin. A peer-to-peer electronic cash system. Bitcoin.URL: <https://bitcoin.org/bitcoin.pdf>. 2008.
- Ormos, M., & Urbán, A. (2013). Performance analysis of log-optimal portfolio strategies with transaction costs. *Quantitative Finance*, 13(10), 1587–1597.
- Paccos, A. M. (2000). Financial intermediation in the securities markets law and economics of conduct of business regulation. *International Review of Law and Economics*, 20(4), 479–510.
- Patel, J., Shah, S., Thakkar, P., & Kotecha, K. (2015). Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Systems with Applications*, 42(1), 259–268.
- Rodríguez, J. J., & Kuncheva, L. I. (2008). Combining online classification approaches for changing environments. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)* (pp. 520–529). Springer.
- Ross, S., Mineiro, P., & Langford, J. (2013). *Normalized online learning* arXiv:1305.6646.
- Sayeed, S., & Marco-Gisbert, H. (2019). Assessing blockchain consensus and security mechanisms against the 51% attack. *Applied Sciences*, 9(9), 1788.
- Schmiedel, H., Malkamäki, M., & Tarkka, J. (2006). Economies of scale and technological development in securities depository and settlement systems. *Journal of Banking and Finance*, 30(6), 1783–1806.
- Sharpe, W. F. (1994). The sharpe ratio. *Journal of Portfolio Management*, 21(1), 49–58.
- Sobti, R., & Geetha, G. (2012). Cryptographic hash functions: A review. *International Journal of Computer Science Issues (IJCSI)*, 9(2), 461.
- Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). *A survey of dimensionality reduction techniques* arXiv:1403.2877.
- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.
- Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning: vol. 2*. MIT Press Cambridge.
- Tsai, C.-F., & Hsiao, Y.-C. (2010). Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision Support Systems*, 50(1), 258–269.
- Van Laarhoven, P. J. M., & Aarts, E. H. L. (1987). Simulated annealing. In *Simulated annealing: Theory and applications* (pp. 7–15). Springer.
- Wall, E., & Malm, G. (2016). Using blockchain technology and smart contracts to create a distributed securities depository.
- Wong, T., & Luo, Z. (2018). Recurrent auto-encoder model for large-scale industrial sensor signal analysis. In *Engineering applications of neural networks* (pp. 203–216). Springer International Publishing.
- Zhang, W., Wang, P., Li, X., & Shen, D. (2018). The inefficiency of cryptocurrency and its cross-correlation with dow jones industrial average. *Physica A: Statistical Mechanics and its Applications*, 510, 658–670.

5.2 Deep Graph Convolutional Reinforcement Learning for Financial Portfolio Management - DeepPocket

This publication proposed the DeepPocket framework, which introduces an improvement compared to the DeepBreath framework by replacing the SARSA method with an actor-critic architecture [80]. Additionally, using a graph convolutional neural network (GCN), the DeepPocket framework investigates the interrelation among financial instruments that may be initially conceptualised as statistically independent. Similarly, proteins may be modelled as graph-structured data [81], leading to various applications such as protein solubility prediction [82], genomic analysis [83], and drug discovery [84]. The GCN variations have been used to determine the amino acids of an interacting protein interface by integrating 3-D structures into a GCN [85], to acquire positional information in PPIs [17], and to accurately predict interactions between query proteins entirely from 3-D structural data [86]. For a more in-depth literature review, please refer to Chapter 2. This paper is published in the Expert Systems with Applications journal on April 24, 2021.



Deep graph convolutional reinforcement learning for financial portfolio management – DeepPocket

Farzan Soleymani, Eric Paquet *

National Research Council, 1200 Montreal Road, Ottawa, ON K1K 2E1, Canada

ARTICLE INFO

Keywords:

Portfolio management
Deep reinforcement learning
Restricted stacked autoencoder
Online learning
Actor-critic
Graph convolutional network

ABSTRACT

Portfolio management aims at maximizing the return on investment while minimizing risk by continuously reallocating the assets forming the portfolio. These assets are not independent but correlated during a short time period. A graph convolutional reinforcement learning framework called DeepPocket is proposed whose objective is to exploit the time-varying interrelations between financial instruments. These interrelations are represented by a graph whose nodes correspond to the financial instruments while the edges correspond to a pair-wise correlation function in between assets. DeepPocket consists of a restricted, stacked autoencoder for feature extraction, a convolutional network to collect underlying local information shared among financial instruments and an actor-critic reinforcement learning agent. The actor-critic structure contains two convolutional networks in which the actor learns and enforces an investment policy which is, in turn, evaluated by the critic in order to determine the best course of action by constantly reallocating the various portfolio assets to optimize the expected return on investment. The agent is initially trained offline with online stochastic batching on historical data. As new data become available, it is trained online with a passive concept drift approach to handle unexpected changes in their distributions. DeepPocket is evaluated against five real-life datasets over three distinct investment periods, including during the Covid-19 crisis, and clearly outperformed market indexes.

1. Introduction

The process of selecting and managing a group of financial instruments such as stocks, bonds, and securities, is called portfolio management (Markowitz, 1978). Portfolio management aims at maximizing the return on investment while minimizing the risk. Stock market movements vary with time and reflect social and political trends. As a result, the financial market follows a complex trajectory that is characterized by its volatility as well as by the correlations between financial instruments. One must have a broad knowledge of these movements and erratic behaviors in order to somehow predict market evolution. Various strategies for portfolio management have been proposed in the literature such as Pouya, Solimanpur, and Rezaee (2016) which employs a multiobjective weed optimization method, Yue and Wang (2017) which relies on fuzzy multi-objective, high-order moment portfolio selection, and Omid, Abbasi, and Nazemi (2017) which dynamically solves portfolio selection with uncertainty chance constraints. Constrained portfolio selection has been discussed in García, González-Bueno, Oliver, and Riley (2019) and García, González-Bueno, Oliver, and Tamošiūnienė (2019). In this paper, it is

proposed to learn the portfolio allocation directly from the data with a deep neural network: a free-form, data-driven approach.

Investing in multiple indexes and stocks could be overwhelming as one must assess market evolution while continuously reallocating assets in the most profitable way. Market movements, price fluctuations, and sudden turmoil are known to be interrelated (Park & Shin, 2013). Yet, it is extremely difficult to model the duration and magnitude of these fluctuations (Sarwar & Khan, 2019; Park & Shin, 2013). For instance, the financial market is influenced by exogenous factors related to the globalized economy, such as the merging of 21st Century Fox and Disney, or Peugeot Fiat-Chrysler. Understanding the dependencies and interrelationships among assets helps to understand market evolution while identifying the optimal portfolio composition and risk management strategy (Drozd, Grümmer, Ruf, & Speth, 2001; Elton, Gruber, Brown, & Goetzmann, 2009). Deep learning techniques are well suited for extracting complex patterns from large datasets (Bengio, Goodfellow, & Courville, 2015). In addition, datasets are mostly Euclidean in the sense that the space associated with the data has a Euclidean geometry.

* Corresponding author.

Images are a good example of such a geometry as the pixels form a regular grid which spans a bidimensional Euclidean space. An important benefit of Euclidean space is that it permits the concept of convolution. This is paramount for convolutional networks as their architecture is based on this very concept. Convolutional neural networks (CNNs) have been widely employed for pattern recognition, localization, and parameter reduction on Euclidean datasets, just to mention a few. If the space is not Euclidean, convolution networks cannot be employed in their standard form (Defferrard, Bresson, and Vandergheynst (2016). Moreover, most deep learning algorithms assume that the instances forming the dataset are statistically independent (Gama, Sebastião, & Rodrigues, 2013; Barabási et al., 2016; Wu et al., 2019); but this assumption does not apply to datasets whose instances are interrelated. Still, non-Euclidean data are far from rare. One may think, for instance, of scientific publications and citations (Defferrard et al., 2016; Veličković et al., 2017) in which the various instances are linked to each other, thereby forming a non-Euclidean interconnected network. The same may be said about macromolecular structures, whose importance in medicine cannot be overstressed (Gilmer, Schoenholz, Riley, Vinyals, & Dahl, 2017).

Financial data, such as financial instruments, are no exception as they also define a non-Euclidean space (Wu et al., 2020). Indeed, the reason lies within the interrelation between financial instruments (Lucey & Muckley, 2011). These are correlated (Anthony, 1988) during short periods of time, especially in the presence of exogenous extreme events (Arfaoui & Rejeb, 2017). Their structure may be represented by a graph in which the nodes correspond to the financial instruments, while the edges or links correspond to some pair-wise correlation function. Because the geometry is non-Euclidean, one cannot easily transpose the Euclidean definitions of locality, translation, and compositionality which underpin convolution and, ultimately, are needed to employ convolutional neural networks (Angles & Gutierrez, 2008; Bronstein, Bruna, LeCun, Szlam, & Vandergheynst, 2017). In Euclidean geometry, convolution may be performed directly or with the help of the convolution theorem (Hammond, Vandergheynst, & Gribonval, 2011) by employing the Fourier transform. The latter is well defined in Euclidean space.

In order to apply the convolution theorem in finance, one must define a Fourier transform directly on the non-Euclidean graph. Such a graph Fourier transform was introduced recently by Shuman, Narang, Frossard, Ortega, and Vandergheynst (2013): it allows convolutions on the graph to be evaluated directly with the help of the convolution theorem and the graph Fourier transform. Once defined, the transform makes it possible to apply a convolutional network directly to non-Euclidean graphs, which are henceforth known as graph convolutional networks (GCN) (Shuman et al., 2013; Henaff, Bruna, & LeCun, 2015). As for their Euclidean counterparts, they consist of a bank of kernels (or filters) which are convolved with their input data. Euclidean filters are local in the sense that their extension (dimensionality) is much smaller than that of the signals with which they are convolved. In the non-Euclidean case, because of the Fourier transform, localization is lost, generally speaking. This problem may be overcome if the filters are defined with truncated Chebyshev polynomials, which are characterized by their compact support and low computational complexity (recurrence relation) (Defferrard et al., 2016).

The portfolio management problem is inherently nonlinear, stochastic, and time-dependent. Therefore, it may be portrayed as a decision-making process that yields a sequence of actions. These actions are defined as the amount of funds to be allocated to each asset forming the portfolio to increase the expected return on investment. This is achieved by training an agent which selects a set of actions from all possible actions in the action space, aiming to increase the portfolio return over a certain investment period. This objective may be formulated as a stochastic optimization problem such that the solution yields an optimal chain of actions which aims at maximizing the expected return on investment.

These types of problem can be solved using reinforcement learning (RL) techniques where an agent takes action ($\mathbf{a}_t$) from an action space ($\mathcal{A}$) based on the state ($\mathbf{s}_t$) of the environment which belongs to a state-space ($\mathcal{S}$). Each action can be interpreted as a kind of interaction with the environment, which is associated with a scalar reward r_t . RL determines an optimal path in the action space, which aims at increasing the reward. The path is characterized by a chain of actions that are derived from a policy function $\pi(\mathbf{a}_t|\mathbf{s}_t)$. Therefore, the agent learns a policy based on past experience, which is evaluated by a value function according to some environment dynamics (Andrew, 1999). Since the portfolio management problem is time-dependent, learning must be conducted online.

There are various RL techniques that provide optimal policy and value functions, e.g. Q-learning (Watkins & Dayan, 1992; Bu, Babu, & De Schutter, 2008), SARSA (Zhao, Wang, Shao, & Zhu, 2016), deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015) and actor-critic (Konda & Tsitsiklis, 2000) just to mention a few. Q-learning is an off-policy algorithm meaning that the policy that generates the actions may be unrelated to the improved policy. On the other hand, SARSA is an on-policy approach that aims to improve the policy that dictates the actions (Andrew, 1999). DDPG is a model-free, off-policy approach that learns a deterministic policy in a continuous space (Lillicrap et al., 2015). The advantage of such a policy is its ability to explore all possible actions while learning a deterministic policy (Andrew, 1999). Nonetheless, this exploration is time-consuming as it requires a large number of training epochs. In the DeepBreath framework, a SARSA algorithm was employed in order to learn the optimal policy (Soleymani & Paquet, 2020). However, SARSA may lead to abrupt changes in the policy which are not necessarily suitable for portfolio optimization (Celikyurt & Özekici, 2007) as a result of an erratic change in the policy if the latter performs too poorly (Andrew, 1999).

The actor-critic algorithm consists of two components: one being associated with the policy while the other is associated with the value function. The network responsible for the implementation of the policy is known as the actor. The critic is a learnable network that evaluates the performance index (value function) for each new each state based on the actions previously taken by the actor. Therefore, the actor-critic approach may be implemented with two online, trainable neural networks that work concurrently in tandem. In this work, the actor-critic algorithm is exploited wherein the actor is trained based on experience, corresponding to probabilistic mapping from state actions according to a learnable policy. At the same time, the critic estimates the expected future return through a state-value function while constantly improving the policy (Bhatnagar, Sutton, Ghavamzadeh, & Lee, 2009). Both the actor and the critic are implemented with specially designed convolutional networks where the output of the actor determines the portfolio allocation while the return on investment is approximated by the critic. One of the most important innovations in this work is the introduction of a **graph convolutional network** for the actor and the critic in order to take advantage of the correlation (as well as the non-Euclidean space) existing in between the financial instruments.

A graph convolutional reinforcement learning framework, Deep-Pocket is introduced, consisting of a restricted, stacked autoencoder (RSAE) for feature extraction and dimensionality reduction, a graph convolutional network (GCN) to acquire interrelations among financial instruments, as well as a convolutional network for each of the actor and the critic. Feature extraction generates a low-dimensional and information-rich description which is more suited to machine learning, besides reducing the computational complexity. The GCN captures the correlations existing between financial instruments. The investment policy is enforced, and the return on investment is estimated using the actor-critic algorithm. The framework is initially trained offline; then, the weights of the neural networks are updated online as new data becomes available. The offline training step is conducted using historical data, while online training is based on passive concept-drift detection (Gama, Žliobaitė, Bifet, Pechenizkiy, & Bouchachia, 2014).

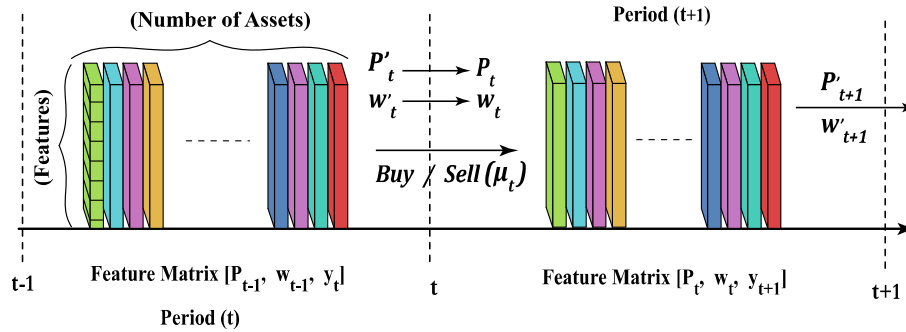


Fig. 1. Weight reallocation process. The portfolio weight vector w'_{t-1} and the portfolio value P'_{t-1} at the beginning of period (t) evolve to (t) and P'_t respectively. At that moment, assets are sold and bought in order to increase the expected return on investment. These operations involve commission fees which shrink the portfolio value to P_t which result in new weights w_t .

The paper is organized as follows. The mathematical model of portfolio management is discussed in Section 2. The architecture of the proposed framework is presented in Section 3, which includes feature normalization in Section 3.1, the restricted stacked autoencoder in Section 3.2, and the graph convolutional networks in Section 4. The reinforcement learning framework is introduced in Section 5. This is followed by a description of our offline and online learning approaches in Section 6. Our experimental results are presented in Section 7. Finally, Section 8 concludes the paper.

2. Mathematical model

A portfolio containing m assets may be managed directly by an individual or by financial professionals through an institution like an investment bank. The investors must select and allocate funds to a group of assets according to an investment strategy, subject to some risk aversion. Our portfolio management model is inspired by an approach introduced by Ormos and Urbán (2013) which introduces an experimental approximation of the log-optimal investment strategy which ensures a near optimal growth rate of investments along with a survivorship bias-free setup. In this model, a trading period corresponds to one business day. The closing price of all assets in the portfolio corresponds to the vector V_t . The vector also contains the amount of available cash. The normalized price vector is defined as

$$Y_t := V_t \oslash V_{t-1} = \left[1, \frac{V_{1,t}}{V_{1,t-1}}, \frac{V_{2,t}}{V_{2,t-1}}, \dots, \frac{V_{m,t}}{V_{m,t-1}} \right]^T. \quad (1)$$

where $(\oslash)$ denotes element-wise division. Initially, there is no assumption regarding the weight distribution: as a result, the portfolio only consists of cash which means that the weight associated with the cash is one (1), while the other weights are zero (0):

$$w_0 = [1, 0, \dots, 0]^T. \quad (2)$$

A transaction factor, which shrinks the portfolio value, is associated with each transaction, such as buying or selling. At the end of a business day, as illustrated in Fig. 1, the portfolio weights vector is equal to:

$$w'_t = \frac{Y_t \odot w_{t-1}}{Y_t \cdot w_{t-1}}, \quad (3)$$

At the beginning of the period (t) , the portfolio weight vector is w_{t-1} . The agent enforces the investment policy in order to reallocate the assets, such as to increase the expected return on investment, resulting, at the end of the trading day, in a new weight vector w'_t . A commission fee $\mu \in (0, 1]$ is applied to the transactions, resulting in the final weight vector w_t .

$$P_t = \mu_t P'_t. \quad (4)$$

The portfolio value at the end of period (t) is obtained by

$$P_t = \mu_t P'_{t-1} \cdot Y_t \cdot w_{t-1}. \quad (5)$$

while the return rate and logarithmic return rate for period (t) are given by

$$\rho_t := \frac{P_t}{P_{t-1}} - 1 = \frac{\mu_t P'_t}{P_{t-1}} - 1 = \mu_t Y_t \cdot w_{t-1} - 1, \quad (6a)$$

$$R_t := \ln \frac{P_t}{P_{t-1}} = \ln(\mu_t Y_t \cdot w_{t-1}) - 1. \quad (6b)$$

Accordingly, the final portfolio value at time horizon t_f is determined by

$$P_f = P_0 \cdot \exp \left(\sum_{t=1}^{t_f} R_t \right) = P_0 \prod_{t=1}^{t_f} \mu_t Y_t \cdot w_{t-1}. \quad (7)$$

Buying and selling assets result in spending and gaining cash (Soleymani & Paquet, 2020), which implies that:

$$C_g = ((1 - c_s)) P'_t \sum_{i=1}^m \cdot \text{ReLU}(w'_{t,i} - \mu_t w_{t,i}) \quad (8)$$

$$C_s = (1 - c_b) \left[w'_{t,0} + (1 - c_s) \sum_{i=1}^m \text{ReLU}(w'_{t,i} - \mu_t w_{t,i}) - \mu_t w_{t,0} \right] \\ = \sum_{i=1}^m \text{ReLU}(\mu_t w_{t,i} - w'_{t,i}) \quad (9)$$

where the selling commission rate belongs to $0 < c_s < 1$, the buying commission rate lies within $0 < c_b < 1$ and $\text{ReLU}(x) = \max(0, x)$ is a rectified linear unit. As a result, the available cash $P'_t w'_{t,0}$ becomes $\mu_t P'_t w'_{t,0}$. Therefore, the transaction factor μ_t is obtained by solving an implicit equation:

$$\mu_t = \frac{1}{1 - c_b w'_{t,0}} \left[1 - c_b w'_{t,0} - (c_s + c_b - c_s c_b) \sum_{i=1}^m \text{ReLU}(w'_{t,i} - \mu_t w_{t,i}) \right] \quad (10a)$$

$$\mu_t = \mu_t (w_{t-1}, w_t, Y_t) \quad (10b)$$

Eq. (10) cannot be solved analytically. A solution may, however, be achieved iteratively, convergence being guaranteed by a theorem demonstrated in Jiang, Xu, and Liang (2017). Our mathematical model relies on two assumptions:

- Trading is possible at any time during business hours.
- The volume of financial instruments traded by the agent is relatively small compared to the total number of assets traded during a day period.

As long as the volume of financial instruments traded by the agent is small compared to the overall traded volume, the latter assumptions remain valid. The architecture of the proposed framework is explained in the following section.

3. System architecture

A profitable trading system must be able to forecast price movements on the stock market. These price movements may be characterized by various financial indicators such as price trend (Nti, Adekoya, & Weyori, 2019), the momentum indicator, and moving average, to mention a few (see Table 1).

The price trend indicates the direction toward which the market evolves (Qiu & Song, 2016). The strength of a trend, as well as the likeliness of its reversal, are measured by the momentum indicator, this being related to the rate at which prices change (Cervelló-Royo & Guijarro, 2020). The moving average indicator measures the average price of a given financial instrument over a certain period of time. Seven financial indicators were selected in order to measure price movement and predict future trends. These indicators include the opening and closing price of the market and the lowest, and the highest prices reached during the trading period, along with financial indicators such as the Hull moving average, average true range, the dynamic momentum index, the commodity selection index, among others (Murphy, 1999). These indicators are described in Table 1.

The proposed portfolio management framework consists of multiple components, namely the normalization module, feature selection with a restricted, stacked autoencoder, and a graph convolutional neural network which extracts information-rich features based on the correlations between financial instruments. The policy is learned with an actor-critic algorithm: the latter comprising two convolutional neural networks as illustrated in Fig. 2. These neural networks are fully described in the next section.

3.1. Feature normalization

Each asset is characterized by a feature vector containing twelve (12) features including opening, closing, low and high prices in addition to financial indicators such as average true range that evaluate the market volatility over a certain period (Soleymani & Paquet, 2020) as illustrated in Table 1.

Since the effective range and extrema of the features are unknown a priori, the min-max approach cannot be applied (Hussain, Knowles, Lisboa, & El-Dereby, 2008; Bhanja & Das, 2018). Instead, the prices are normalized with respect to their values at opening:

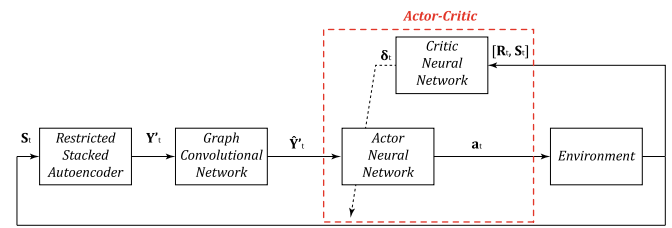


Fig. 2. Global architecture of DeepPocket.

$$\begin{aligned} \mathbf{V}_t^{(Lo)} &= \left[\frac{Lo(t-n-1)}{Op(t-n-1)}, \dots, \frac{Lo(t-1)}{Op(t-1)} \right]^T, \\ \mathbf{V}_t^{(Cl)} &= \left[\frac{Cl(t-n-1)}{Op(t-n-1)}, \dots, \frac{Cl(t-1)}{Op(t-1)} \right]^T, \\ \mathbf{V}_t^{(Hi)} &= \left[\frac{Hi(t-n-1)}{Op(t-n-1)}, \dots, \frac{Hi(t-1)}{Op(t-1)} \right]^T. \end{aligned} \quad (11)$$

while the financial indicators are normalized with respect to their closing value on the previous day:

$$\mathbf{V}_t^{(FI)} = \left[\frac{FI(t-n)}{FI(t-n-1)}, \dots, \frac{FI(t)}{FI(t-1)} \right]^T. \quad (12)$$

Feature extraction is performed on these normalized vectors with a restricted autoencoder.

3.2. Restricted stacked autoencoder

In order to implement machine learning algorithms efficiently, it is important to reduce the computational cost and time required for training (Meng, Catchpole, Skillicom, & Kennedy, 2017). For this purpose, feature extraction is performed to obtain highly informative abstract features with low dimensionality. The input feature vector contains eleven (11) features, namely: normalized low, high, and closing prices along with the eight normalized financial indicators, all taken at the same time t . These features are partially correlated and, as a result, carry redundant information. Feature extraction aims to remove such a redundancy while making the indicators more informative. The restricted, stacked autoencoder transforms recurrent events into low-dimension abstractions (Langr & Bok, 2019).

The restricted autoencoder consists of two parts, namely the encoder and the decoder. The encoder maps the input feature to a lower dimension via multiple stacked layers, collectively called the latent layer, while the decoder reconstructs low-dimensional features at the output. The decoder is designed to reconstruct only a subset of the original features; in our case, the low, close, and high price. In order to

Table 1
Financial Indicators Employed as Features.

Financial Indicators	Definition
Average True Range	The average true range is a technical indicator that assess the volatility of an asset in the market through a certain period.
Commodity Channel Index	The commodity channel index is a momentum-based oscillator used to help determine cyclical trend of an asset price as well as strength and direction of that trend.
Commodity Selection Index	The commodity selection index is a momentum indicator that evaluates the eligibility of an asset for short term investment.
Demand Index	The demand index is an indicator that uses price and volume to assess buying and selling pressure affecting a security.
Dynamic Momentum Index	The dynamic momentum index determines if an asset is an asset is overbought or oversold.
Exponential Moving Average	An exponential moving average is a technical indicator that follow the price of an asset while prioritizing on recent data points.
Hull Moving Average	The hull moving average is a more responsive alternative to moving average indicator that focus on the current price activity whilst maintaining curve smoothness.
Momentum	The momentum in a technical indicator that determines the speed at which the price of an asset is changing.

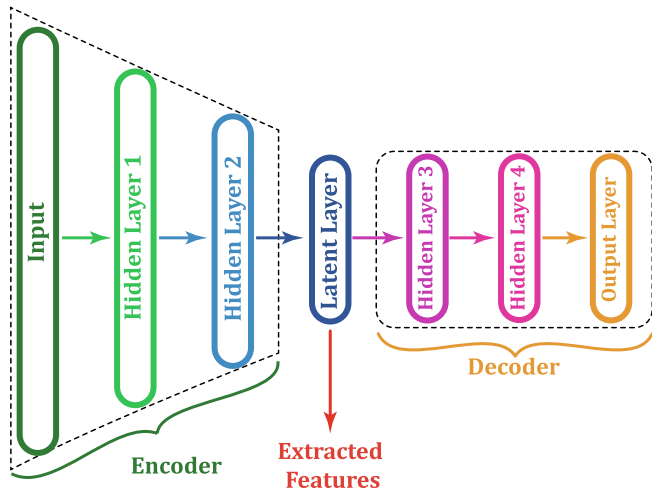


Fig. 3. Restricted stacked autoencoder.

partially reconstruct the normalized input vector, each layer of the decoder has three (3) neurons which restrict the reconstruction to the normalized low, high, and closing prices. The network is under-complete, which means the latent layer has fewer nodes than the input layer (Bengio et al., 2015). The latent layer creates a tensor of abstract features as $([V_1, V_2, V_3])$. The structure of our RSAE is illustrated in Fig. 3 In the next section, the correlation between financial assets is extracted using graph convolutional networks.

4. Spectral graph convolutional networks

Most of the data used in deep learning have a regular grid structure (Bronstein et al., 2017). One may mention, for instance, the regular pixel structure of an image or times series at fixed time intervals. These data may easily be encoded for neural network input as vectors, matrices, or tensors. These structures are readily represented in Euclidean space, exhibit properties such as local-connectivity, shift-invariance, and compositionality, to mention just a few (Stone et al., 2017). As a result, the convolution operation is properly defined, thus providing the basis for convolutional neural networks (Henaff et al., 2015).

In contrast, many entities, such as financial instruments, cannot be represented on a regular grid due to the complex nature of their interrelations. Indeed, financial instruments are correlated among themselves, and their structures may be better represented by a graph in which the nodes correspond to the financial instruments while the edges or links correspond to some pair-wise correlation function. It is worth noting that a single financial instrument, being a time series, embeds naturally in Euclidean space, but a plurality of financial instruments, because of their mutual correlations, resists Euclidean representation and instead is represented as a graph (Wu et al., 2019; Zhang, Tong, Xu, & Maciejewski, 2019). Unfortunately, standard convolution cannot be directly extended to non-Euclidean geometries (Shuman et al., 2013) therefore impeding its applicability to CNN. Nonetheless, as set out by the convolution theorem (Shuman et al., 2013), the convolution may be evaluated with the help of the Fourier transform: firstly, the Fourier transforms of both the input and the filter are evaluated; then, both transformations are multiplied element-by-element (the Hadamard product) and, finally, the inverse Fourier transform of the Hadamard product is taken. The convolution theorem remains valid under non-Euclidean geometry if the Fourier transform is properly defined (Shuman et al., 2013; Henaff et al., 2015), thus allowing the application of CNN to non-Euclidean geometries.

The next section provides a mathematical formulation of spectral graph convolution.

4.1. Graph Fourier transform

Let us consider an undirected weighted graph $G = \{V, E, W\}$ representing, for instance, m financial instruments. This graph consists of $(|V| = m)$ vertices or nodes and $(|E| = n)$ edges or links. The edges represent the interrelations between the nodes while the weights are a pair-wise correlation function between nodes. This graph may be characterized by a weight adjacency matrix $W \in \mathbb{R}^{m \times n}$:

$$W = [w_{ij}] \quad (13)$$

where w_{ij} is the weight between node i and j (Shuman, Ricaud, & Vandergheynst, 2016). Such a graph is dynamic in the sense that the correlations vary at each time interval. The correlation is evaluated over a period consisting of n time intervals. This period should be relatively short as the correlation tends to disappear rapidly due to randomization (Fama, 1965; Rocchi, Tsui, & Saad, 2017). For the correlation, a value of one indicates a perfect correlation while a value of zero indicates its absence. It is desirable that correlated nodes be topologically close to each other while uncorrelated nodes are more aloof. In order to reflect this behavior, the weights are defined as

$$w_{ij} = 1 - \text{corr}(V_i, V_j)_{[t-n+1, t]} \quad (14)$$

where the correlation is evaluated on the interval $[t-n+1, t]$. A graph may be represented by a Laplacian. The standard graph Laplacian is defined as

$$L = D - W \quad (15)$$

where $D \in \mathbb{R}^{n \times n}$ is diagonal degree matrix that indicates the degree of connectivity of each node $d_{ij} = \sum w_{ij}$ and, W is the weight matrix (Chung & Graham, 1997). This Laplacian is a discrete representation of the Laplacian operator which is encountered, for instance, in the heat and Schrödinger equations. As such, the Laplacian completely characterizes a Gaussian random walk on the graph (Felmer, Quaaas, & Tan, 2012). The Laplacian may also be defined as a symmetrical matrix (Kipf & Welling, 2016):

$$L_{\text{sym}} = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}} \quad (16)$$

Symmetry is highly desirable as the eigenvalues $(0 = \lambda_0 \leq \lambda_1 \leq \dots \leq \lambda_{n-1})$ associated with a symmetric matrix will then be real while the eigenvectors are mutually orthogonal, which greatly reduces the computational complexity. The eigendecomposition of a symmetrical Laplacian is given by

$$L_{\text{sym}} = \Phi \Lambda \Phi^T \quad (17)$$

where $\Phi = [\phi_0, \dots, \phi_{n-1}]$ are the column-wise orthogonal eigenvectors:

$$\Phi^T \Phi = I \quad (18)$$

The eigenvalues are represented as a diagonal matrix $\Lambda = \text{diag}(\lambda_0, \lambda_1, \dots, \lambda_{\max}) \in \mathbb{R}^{n \times n}$. The eigenvectors corresponding to the smallest eigenvalues are the smoothest in the sense that $|\phi_i(i) - \phi_i(j)|$ is small for two neighboring nodes i and j (Shuman, Vandergheynst, & Frossard, 2011). As aforementioned, graph convolution may be defined using the Fourier transform and the convolution theorem (Shuman et al., 2013). The graph Fourier transform of a signal x on the graph is defined as

$$\mathcal{F}(x) = \Phi^T x = \tilde{x} \quad (19)$$

where Φ^T indicates the transpose of Φ ; whereas the inverse graph Fourier transform is obtained by

$$\mathcal{F}^{-1}(\tilde{\mathbf{x}}) = \Phi(\tilde{\mathbf{x}}) = \mathbf{x} \quad (20)$$

where $\mathcal{F}$ denotes the Fourier transform, and $(\tilde{\mathbf{x}})$ is the signal in the spectral domain. The latter results from the orthogonality of the eigenvectors. Therefore, the Fourier transform is achieved by projecting the signal onto the transpose of the eigenvector matrix while the inverse transform is obtained by projecting onto the eigenvectors per se. The graph convolution is introduced in the next section.

4.2. Spectral graph convolution

Having defined the graph Fourier transform, one may immediately define the spectral graph convolution with the help of the convolution theorem. (Bronstein et al., 2017). The standard convolution is defined as

$$(\mathbf{f} \star \mathbf{g})(x) = \int_{\Omega} \mathbf{f}(x - x') \mathbf{g}(x') dx' \quad (21)$$

According to the latter, the convolution of a signal $\mathbf{x} \in \mathbb{R}^n$ with a parametric filter (g_{θ}) is defined as

$$\mathbf{x} \star_{G} g_{\theta} = \mathcal{F}^{-1}[\mathcal{F}(\mathbf{x}) \odot \mathcal{F}(g_{\theta})] \quad (22)$$

Where $\theta \in \mathbb{R}^n$ is a parametrization of the filter and $\odot$ is the element-wise or Hadamard product. As stated in the previous section, the Fourier transform and its inverse may be expressed in terms of the eigenvectors of the Laplacian matrix. Therefore, the spectral graph convolution becomes:

$$\mathbf{x} \star_{G} g_{\theta} = \Phi(\Phi^T \mathbf{x}) \odot (\Phi^T g_{\theta}) \quad (23)$$

Because of the orthogonality of the eigenvectors, this equation may be further simplified:

$$\mathbf{y} = \mathbf{x} \star_{G} g_{\theta} = g_{\theta}(\mathbf{L}) \mathbf{x} = \Phi g_{\theta}(\mathbf{\Lambda}) \Phi^T \mathbf{x} = \Phi \underbrace{\text{diag}(g_{\theta})}_{\tilde{g}} \Phi^T \quad (24)$$

where the kernel is given by $g_{\theta}(\mathbf{\Lambda}) = \text{diag}(g_{\theta}(\lambda))$. The filters are parametrized in terms of a truncated power series over the eigenvalue matrix. Because this matrix is diagonal, only the diagonal element needs to be exponentiated, thus reducing the complexity of the calculation from $\mathcal{O}(N^2)$, for a dense matrix, to $\mathcal{O}(N)$:

$$g_{\theta}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k \quad (25)$$

where $\theta_k \in \mathbb{R}^K$ are the filter's parameters (Defferrard et al., 2016). Most filters are not localized because their spectrum has an infinite expansion (Defferrard et al., 2016). This is in contrast with the standard convolution for which the filters associated with CNN are always localized. Nonetheless, localization may be achieved if the filters are expressed in terms of Chebyshev polynomials of the first kind (Hammond et al., 2011; Wu et al., 2019):

$$g_{\theta}(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\mathbf{\Lambda}}) \quad (26)$$

where $\tilde{\mathbf{\Lambda}} = \frac{2\mathbf{\Lambda}}{\lambda_{\max}} - \mathbf{I}_n$ is the rescaled eigenvalue matrix and $T_k(\tilde{\mathbf{\Lambda}})$ is the Chebyshev polynomial of order k . A truncated development of order $K-1$ corresponds to a filter that spans a K -ring neighborhood: from a 1-ring up to a K -ring. Each polynomial corresponds to a particular neighborhood: T_1 corresponds to a 1-ring neighborhood, T_2 corresponds to a 2-ring neighborhood while T_k corresponds to a k -ring neighborhood away (Kipf & Welling, 2016; Hammond et al., 2011). The Chebyshev polynomial may be defined by a recurrence relation, thus reducing the computational complexity. Indeed, given:

$$\tilde{\mathbf{x}}_k = T_k(\tilde{\mathbf{L}}) \mathbf{x} \in \mathbb{R}^n \quad (27)$$

where the scaled Laplacian is defined as

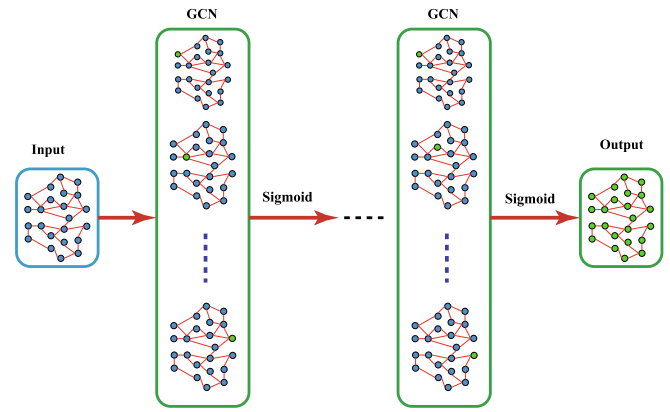


Fig. 4. Graph Convolutional Neural Network.

$$\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}_n \quad (28)$$

the recurrence relation is given by

$$\tilde{\mathbf{x}}_k = 2\tilde{\mathbf{L}}\tilde{\mathbf{x}}_{k-1} - \tilde{\mathbf{x}}_{k-2} \quad (29)$$

With $\tilde{\mathbf{x}}_0 = \mathbf{x}$ and $\tilde{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$. As a result, the complexity associated with the filtering operation $\mathbf{y} = g_{\theta}(\mathbf{L})\mathbf{x} = [\tilde{\mathbf{x}}_0, \tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_{K-1}]$ is $\mathcal{O}(K|E|)$, where $|E|$ is the number of edges.

Our graph convolutional network is described in the next section.

4.3. Graph convolution network

The spectral graph convolution in the non-Euclidean domain is obtained by applying the graph Fourier transform and the convolution theorem to both the input signal and the convolving filter (Bruna, Zaremba, Szlam, & LeCun, 2013) (Fig. 4). The graph convolution extracts underlying local information by collecting the node information in a local neighborhood whose extension is determined by the order of the Chebyshev polynomial. In order to extract multi-scale substructure features, multiple graph convolution layers may be stacked as illustrated in Fig. 4. The propagation rule for the multi-layer Graph Convolutional Network (GCN) is given by

$$\mathbf{f}_i^{t+1} = \sigma \left(\sum_{j=1}^p \Phi \hat{\mathbf{G}}_{ij} \Phi^T \mathbf{f}_j^t \right), \quad i = 1, \dots, q, \quad j = 1, \dots, p \quad (30)$$

where σ denotes the nonlinear activation function ($\mathbf{I}$) and $\hat{\mathbf{G}} = \text{diag}(g_{\theta}(\lambda))$. The number of features is indicated by (q) while the number of assets is given by (p).

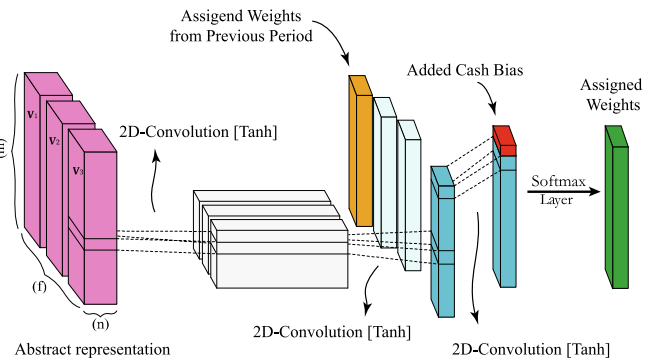


Fig. 5. Architecture of actor.

Algorithm 1. Implementation of Graph Convolutional Network

Input: $\mathbf{x}_t = [v_1, v_2, v_3]^T$
Output: Output feature vector $\leftarrow \mathbf{F} = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^T = [f_1^{t+1}, f_2^{t+1}, f_3^{t+1}]^T$
Data: v_1, v_2, v_3 : Extracted features from restricted stacked autoencoder for each asset
Data: f_i^{t+1}, σ : Transformed feature of layer $t+1$, Activation function [Sigmoid]
1 Initialization:
1. Calculation of the graph Laplacian:
 $\mathbf{L} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$
2. Eigendecomposition of the graph Laplacian:
 $\mathbf{L} = \Phi \Lambda \Phi^T$
 $\tilde{\mathbf{L}} = \frac{2\mathbf{L}}{\lambda_{max}} - \mathbf{I}_n$
3. Chebyshev polynomial ($k, \mathbf{x}_t$):
 $\hat{\mathbf{x}}_0 = \mathbf{x}_t$
if $k == 1$ then
 $\hat{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$
else
 $\hat{\mathbf{x}}_k = \tilde{\mathbf{L}}\hat{\mathbf{x}}_{k-1} - \hat{\mathbf{x}}_{k-2}$
4. Kernel definition:
 $\Phi g_\theta(\Lambda)\Phi^T = \Phi \underbrace{\text{diag}(\hat{g}_\theta)}_{\hat{\mathbf{G}}} \Phi^T$
5. Graph Convolution:
 $f_i^{t+1} = \sigma \left(\sum_{j=1}^p \Phi \hat{\mathbf{G}}_{i,j} \Phi^T f_j^t \right), i = 1, \dots, q, j = 1, \dots, p$

Algorithm 1 describes the graph convolutional network architecture. The output of the GCN is employed to train the actor-critic algorithm. The reinforcement learning approach is explained in detail in the following section.

5. Deep reinforcement learning framework

Deep reinforcement learning algorithms may be divided into three categories: critic-only, actor-only, and actor-critic methods. The critic-only approaches include, among others, Q-learning (Watkins & Dayan, 1992; Bu et al., 2008) and SARSA (Zhao et al., 2016); the latter was employed in DeepBreath framework (Soleymani & Paquet, 2020). Critic-only methods rely on a state-action value function without explicitly defining a function over the policy space. Q-learning is an off-policy RL method in which the generated actions may be unrelated to the improvement of the policy. On the other hand, SARSA is an on-policy RL algorithm that seeks to improve the policy that generates the current actions (Andrew, 1999). On the other hand, actor-only methods optimize the cost over parameter space using a policy gradient approach. The actor-only methods converge much faster than critic-only methods (Grondman, Busoniu, Lopes, & Babuska, 2012). The actions generated by an actor-only method span a continuum, but the variance associated with the gradient is large, which results in a slower learning rate (Konda & Tsitsiklis, 2000; Sutton, McAllester, Singh, & Mansour, 2000). Finally, the actor-critic algorithm was developed in order to combine the advantages of both the actor-only and critic-only methods. Pseudo code for our actor-critic algorithm appears in Section 5.1. Reinforcement learning methods apply a sequence of actions taken by an agent, to ensure that the expected cumulative reward be optimum. In our framework, the actions correspond to the weight vector associated with the portfolio.

$$\mathbf{a}_t = \mathbf{w}_t. \quad (31)$$

While the reward aims to maximize the expected return on investment. When considering the relation between the return rate Eq. (6a) and the transaction factor Eq. (10), it may be concluded that the current action depends on the previous one. The state of the portfolio is obtained by concatenating the internal and external states: the former being the portfolio weight vector at time $t-1$ ($\mathbf{w}_{t-1}$) while the latter is the current feature tensor $\mathbf{X}_t$:

$$\mathbf{s}_t = [\mathbf{X}_t, \mathbf{w}_{t-1}]^T. \quad (32)$$

The performance of each action is evaluated based on the achieved

reward, aiming to increase the reward over a finite investment horizon ($t_f + 1$). The reward is determined by the logarithmic accumulated return $\mathcal{R}$: Eqs. (6b), (7) and 10:

$$\begin{aligned} \mathcal{R}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) &= \frac{1}{t_f} \ln \left(\frac{P_t}{P_0} \right) \\ &= \frac{1}{t_f} \sum_{i=1}^{t_f+1} \ln(\mu_i \cdot \mathbf{Y}_i \cdot \mathbf{w}_{i-1}) = \frac{1}{t_f} \sum_{i=1}^{t_f+1} R_i. \end{aligned} \quad (33)$$

The actor-critic algorithm used to learn the policy is described in the next section.

5.1. Actor-critic architecture

As the name suggests, the actor-critic algorithms-consists of two main components: the actor and the critic. The actor undertakes a sequence of actions following a policy while the critic evaluates the policy by assigning a performance index, called the value function, to each of the actor's actions. The performance is evaluated using temporal differences (TD) (Sutton, 1988). The critic approximates and updates the value function, which indicates the direction in which the actor's policy parameters should be updated to improve performance. In this method, the policy update is derived directly from the value function, while the value function enforces the direction of the policy gradient at each time step (Grondman et al., 2012). Actor-critic algorithms are capable of generating continuous actions while avoiding large policy gradient variance.

In this work, an actor-critic algorithm is employed in order to find the optimal policy for a higher return on investment. The actor undertakes the actions while the critic evaluates the state-value function as a success metric for the actions. The actor updates the policy in the direction suggested by the critic. The actor consists of a deep convolutional neural network, as illustrated in Fig. 5. Convolutional neural networks are known for their ability to learn complex patterns and efficiently implement policies (Jiang et al., 2017). The output of the actor is the learned policy, which determines the portfolio allocation. The network is trained in two stages: initially, offline learning is performed based on a historical dataset using online stochastic batching (Jiang et al., 2017), leading to the offline policy. Then, as new data become available, the policy is updated online. The architecture of the actor is shown in Algorithm 2. The input tensor of this network consists of the high-level features learned from the restricted stacked autoencoder. The tensor consists of three (3) channels, with which three (3) matrices are associated.

The number of rows in each matrix is determined by the number of assets (m), while the number of columns corresponds to the size of the trading window (n). The initial convolution is performed on the input tensor with a kernel of 1×3 size, and then a *Tanh* activation function is applied. The second convolutional layer has a kernel of size $1 \times n$ aiming to obtain a vector of size m . The third layer is connected to the fourth layer through convolution with a kernel of 1×1 size (obtaining one main channel) and a *Tanh* activation function. The third layer consists of the previous actions (portfolio vector), i.e., the current action is computed based on both the current state and previous actions. The cash bias is added to the acquired action vector in the fourth layer, generating a vector of size $m + 1$. The last layer normalizes the action vector using a Softmax function, resulting in a new portfolio weight distribution. All of the dimensions were determined by inspection.

Similarly, the architecture of the critic is described in Fig. 6, and the corresponding pseudocode may be found in Algorithm 3. The input consists of the current feature tensor $\mathbf{X}_t$. The size of the kernels for the first and second convolutional layers, is 1×1 . This is followed by a ReLU activation function. The third convolutional layer has a kernel of size $1 \times m$ and ReLU activation function. Finally, the approximated value function is obtained by means of a densely connected layer.

Algorithm 2. Implementation of Actor Algorithm

Input: $\mathbf{X}_t = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^T$
Output: Distributed Portfolio Weights $\leftarrow \mathbf{W}_{ptf} = [\mathbf{w}_C, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_m]^T$
Data: f, m : Number of Features, Number of Assets

- 1 **Initialization:**
while $i \leq n$ do
 1. **First Convolution:**
Activation Function: Tanh
Kernel size $\rightarrow (1,3)$
Kernel depth $\rightarrow f$
 2. **Second Convolution:**
Activation Function: Tanh
Kernel size $\rightarrow (1,n)$
Kernel depth $\rightarrow f$
 3. **Third Convolution:**
Concatenate weights for previous and current time steps (Weight Matrix).
 $\mathbf{W} = [\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_i]^T$
Activation Function: Tanh
Kernel size $\rightarrow (1,1)$
Kernel depth $\rightarrow f+1$
 4. **Adding Cash bias:**
Concatenate Cash Bias and computed Weight Matrix
 5. **Softmax Layer:**
Distributed Portfolio Weights

The actor is implemented with convolutional neural networks, including three convolutional layers and a final Softmax layer that generates the weights distribution vector. The current action at time t is a function of the action at time $t-1$. The loss function of this network is evaluated based on the log-normal distribution of the policy π_θ and the temporal difference error (TD error) calculated by the critic network. The actor neural network follows a policy gradient approach, which updates the weight using a one-step TD error from Eq. (35).

Algorithm 3. Implementation of Critic Algorithm

Input: $\mathbf{X}_t = [\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3]^T$
Output: Approximated value function V_π
Data: f, m, n : Number of Features, Number of Assets, Trading Window Length

- 1 **Initialization:**
while $i \leq n$ do
 1. **First Convolution:**
Activation Function: ReLU
Kernel size $\rightarrow (1,1)$
Kernel depth $\rightarrow f$
 2. **Second Convolution:**
Activation Function: ReLU
Kernel size $\rightarrow (1,1)$
Kernel depth $\rightarrow n$
 3. **Third Convolution:**
Activation Function: ReLU
Kernel size $\rightarrow (1,m)$
Kernel depth $\rightarrow 1$
 4. **Dense Layer**
Value (V_π) Approximation
 5. **Training**
Compute the TD error using Eq. 35.
Minimize the loss
Compute loss using Eq. 34

The critic consists of three convolutional neural networks. The critic network is updated aiming to minimize, in Eq. (34), the mean square error (MSE) between the approximated value function V_π and target value V^π .

$$MSE = \|V^\pi - V_\pi\|^2 \quad (34)$$

The weights of the critic network are updated based on the gradient descent method, as shown in Algorithm 3. The actor-critic framework is detailed in Algorithm 4.

$$\delta_v = r(\mathbf{s}, \mathbf{a}) + \gamma V_\pi(\mathbf{s}') - V_\pi(\mathbf{s}) \quad (35)$$

where $r(\mathbf{s}, \mathbf{a})$ denotes immediate reward at state $\mathbf{s}$ resulting from action $\mathbf{a}$. The trade-off in between immediate and long term strategy is established by γ . The weights of the critic network are updated based on the gradient descent method as shown in Algorithm 3. The actor-critic framework is detailed in Algorithm 4.

Algorithm 4. Actor-Critic Algorithm

Input: Initial network weights: $\theta_{a,init}, w_{c,init}$
Data: $\alpha_a, \alpha_c \in [0, 1]$: Actor and critic learning rate
Data: θ_a, w_c : Actor and critic neural network weights
Data: N_t : Number of iterations

- 1 **Initialization:**
iteration = 0
 $\pi_\theta(\mathbf{s}, \mathbf{a}) \in \mathcal{A}$
while iteration $\leq N_t$ do
 - 2 $\Delta_w = \alpha_c \delta_v \nabla_w V(\mathbf{s})$
 $w_c \leftarrow w_c + \Delta_w$
 $\Delta_\theta = \alpha_a \delta_v \nabla_\theta \log \pi_\theta(\mathbf{s}, \mathbf{a})$
 $\theta_a \leftarrow \theta_a + \Delta_\theta$
Update state and action:
 $\mathbf{s} \leftarrow \mathbf{s}'$
 $\mathbf{a} \leftarrow \mathbf{a}'$

The actor-critic algorithm uses the Adam optimization method (Kingma & Ba, 2015) to update the weights for both the actor and the critic. The actor and the critic act alternately. At first, the weights associated with the networks are initialized randomly, and an initial admissible policy π_θ is calculated. The portfolio state is determined with respect to the initial action, and the value function is approximated. Then, the weights of the critic network are updated. Afterward, the weights of the actor network are updated using the TD error measured by the critic network. Finally, the state ($\mathbf{s}$) and action ($\mathbf{a}$) are updated. In other words, at each step, the actor network yields an action ($\mathbf{a}$) at state ($\mathbf{s}$), following the direction enforced by the critic network, in order to minimize the value function (V_π).

6. Online learning

The proposed framework is trained using historical data (prior to current calendar time) and the current flow of data pertaining to offline and online learning, respectively. The offline learning is performed using online stochastic batching, described in Soleymani and Paquet (2020). Financial data in the stock market may be assimilated into time series. As a result, unanticipated changes in their underlying distribution may occur, a phenomenon known as concept drift (Gama et al., 2014). These may be generated by exogenous factors such as panic reactions (e.g. the Covid-19 pandemic) (Ashraf, 2020; Zhang, Hu, & Ji, 2020), and political disputes (e.g. US-China trade war) (Cavalcante & Oliveira, 2015). Such factors may profoundly affect the actions taken by investors. In this work, concept drift is addressed using a passive detection approach where the agent is first trained based on historical data, and then the learned policy is continuously updated as new data arrive, thus allowing for concept drift without discarding useful knowledge (Soleymani & Paquet, 2020).

The motivation of the present approach is twofold: extreme events are very often recurrent, which means that a solution to a current financial crisis may be brought from past crises (Hu et al., 2015). Secondly, the passive detection approach discards historical patterns that are not repeated after a certain amount of time (Gama, Medas, Castillo, & Rodrigues, 2004), thus preventing any unstable, short-term investment strategies based on outliers. All of the neural networks in our portfolio management framework (RSAE, GCN, actor-critic), are first trained offline on historical data. Then, the weights are updated online

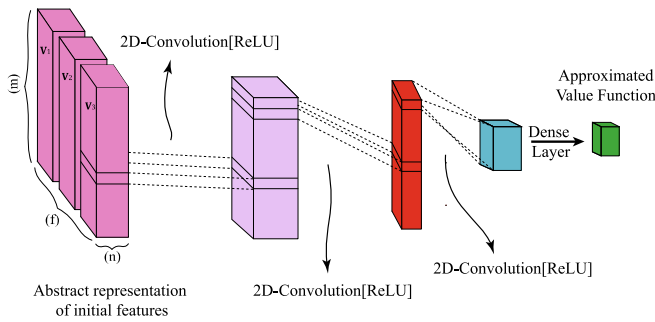


Fig. 6. Critic architecture based on deep convolutional neural network.

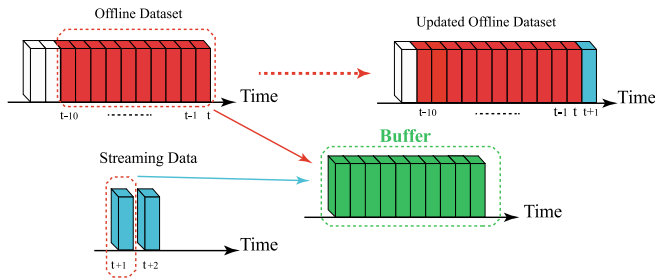


Fig. 7. Structure of the buffer that stores data from the offline dataset as well as the flow of online data for online learning.

as new data become available. The online learning is managed by a buffer, as shown in Fig. 7. The buffer stores the financial data from the last ten days, along with the current business day. Then the stored set containing data from eleven days is used to train and update the networks. At the end of the business transaction (current day), the current financial data are added to the offline dataset.

7. Experimental results

The assets forming our portfolio appear in Table 2. This dataset consists of various financial time series which cover a period ranging from January 2, 2002, up to March 24, 2020; details about each time series may be found in Table 3. In reinforcement learning, actions are taken to minimize the risk while increasing the return on investment. Therefore, to further investigate the risk involved with the investment policy, the performance of the proposed framework is evaluated using three of the most celebrated risk evaluation metrics (Bessler, Opfer, & Wolff, 2017): namely maximum drawdown (MDD), Sharpe ratio (S_r), and Conditional-Value-at-Risk (CVaR) (Rockafellar et al., 2000). The maximum drawdown metric was first proposed by Magdon-Ismaïl and Atiya (2004). This metric is defined as the difference in between the maximum peak value P and the minimum value L reached by the portfolio for a certain period of time before the next peak occurs. Technically, this indicator measures the amplitude of a loss over a period of time. A low maximum drawdown indicates a small loss.

$$MDD = \frac{P - L}{P}. \quad (36)$$

The Sharpe ratio was introduced by Sharpe (1994). It measures the volatility of investment as a ratio between the expected return on investment and the risk:

$$S_r = \frac{E[R_f - R_b]}{\sigma_d}. \quad (37)$$

Table 2

Composition of our Portfolio for the period in between January 2, 2002, and February 10, 2020.

Sector	Name
Technology	Apple Inc. (AAPL)
	Cisco Systems Inc. (CSCO)
	Intel Corporation (INTC)
	Oracle Corporation (ORCL)
	Microsoft Corporation (MSFT)
	International Business Machines Corporation (IBM)
	Honeywell International Inc. (HON)
	Verizon Communications Inc. (VZ)
Financial Services	Manulife Financial Corporation (MFC)
	JPMorgan Chase & Co. (JPM)
	Bank of America Corporation (BAC)
	The Toronto-Dominion Bank (TD)
Industries	3 M Company (MMM)
	Caterpillar Inc. (CAT)
	The Boeing Company (BA)
	General Electric Company (GE)
Consumer Defensive	Walmart Inc. (WMT)
	The Coca-Cola Company (KO)
Consumer Cyclical	The Home Depot, Inc. (HD)
	Amazon.com, Inc. (AMZN)
Healthcare	Johnson & Johnson (JNJ)
	Merck & Co., Inc. (MRK)
	Pfizer Inc. (PFE)
	Gilead Sciences, Inc. (GILD)
Energy	Enbridge Inc. (ENB)
	Chevron Corporation (CVX)
	BP p.l.c. (BP)
	Royal Dutch Shell plc (RDSA.L)

where σ_d is the standard deviation associated with the asset excess return (or volatility) where $(E[R_f - R_a])$ is the expected differential return. A high return on investment is achieved when the Shape ratio is greater than one. In other words, this metric helps investors to better identify the returns associated with high-risk actions.

In order to further analyze the risk associated with portfolio optimization, risk assessment criteria called Value-at-Risk (VaR) and Conditional-Value-at-Risk (CVaR) are adopted here (see Fig. 13). The former measures and control the risk with respect to percentiles of the loss distribution, while later estimates the tail risk of a portfolio (Alexander, Coleman, & Li, 2006). Nonetheless, the use of VaR is limited due to lack of subadditivity and being non-convex and non-smooth with multiple local minima (Acerbi, 2002). On the other hand, the CVaR renders a convex optimization problem (Cornuejols & Tütüncü, 2006).

$$CVaR_\alpha(X) = E[X|X \geq VaR_\alpha(X)] \quad (38)$$

The historical training set consists of four different time series as shown in Table 3. Each training set spans a different period, allowing our

Table 3

Training and test sets.

ID	Training Set	Test Set
Test 1	2002-01-02 to 2009-04-16	2010-03-15 to 2010-07-21
Test 2	2002-01-02 to 2012-12-06	2013-11-04 to 2014-03-14
Test 3	2002-01-02 to 2016-08-01	2017-06-28 to 2017-11-02
Test 4	2002-01-02 to 2018-10-19	2019-06-09 to 2019-10-16
Test 5	2002-01-02 to 2019-06-23	2019-11-12 to 2020-03-24

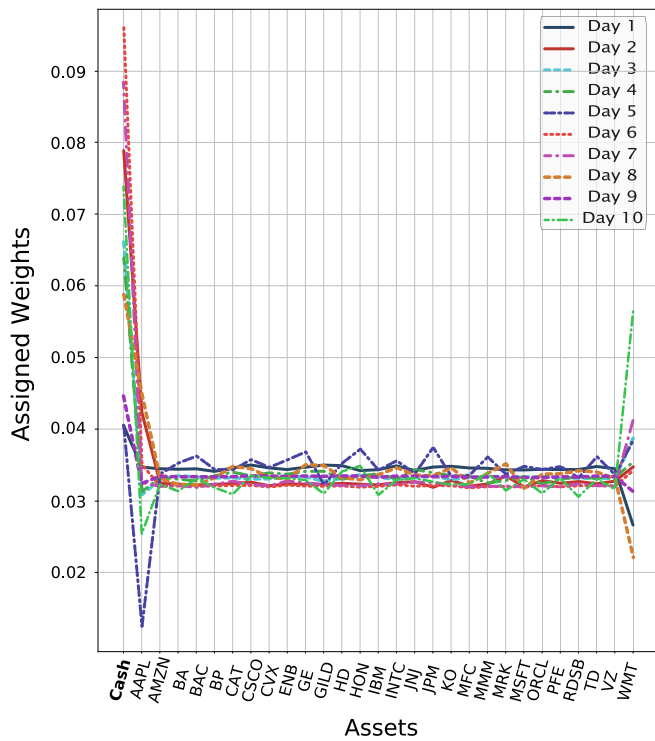


Fig. 8. Evolution of the portfolio weights during the first 10 days of the online learning process.

system to learn from various historical contexts. Each training batch covers 90 consecutive days. Initially, the portfolio consists only of cash, so the portfolio composition is determined entirely by the agent from the start. As mentioned earlier, the assets forming the portfolio appear in Table 2. These assets were selected in order to reflect various segments of the economy, such as the pharmaceuticals industry, financial services, manufacturing, healthcare, energy, and consumer services. This portfolio allows us to further investigate the interrelation among financial instruments that may be perceived as statistically independent at first sight.

Our online learning results are shown in Fig. 8, where the pre-trained model from the last training set is employed, and the weights for the online model are updated as new data becomes available (for each trading day). The learning is conducted over 10 trading days from 2020-03-24 to 2020-04-06. Fig. 8 illustrates the weight fluctuations during the online learning process. Initially, the lowest weight corresponds to WMT, while the rest of the assets are uniformly distributed. Then, as the process unfolds, the weights associated with assets such as APPL and WMT are subject to great variations, while others, such as ORCL, PFE, and RDSB, are not. The most stable assets, in terms of their weights in the portfolio, are employed by the agent to achieve a stable return on investment by hedging the risk. More volatile assets, in terms of their weights, are exploited by the agent for their leverage effect (Bouchaud, Matacz, & Potters, 2001).

Fig. 9 illustrates the weight distribution for the four test sets, as defined in Table 3, after respectively 30, 60, and 90 days of investment. One may notice that the weights are more equally distributed for the last testing set. The behavior may be explained by the fact that the agent aims to mitigate risk by distributing the funds among the various assets.

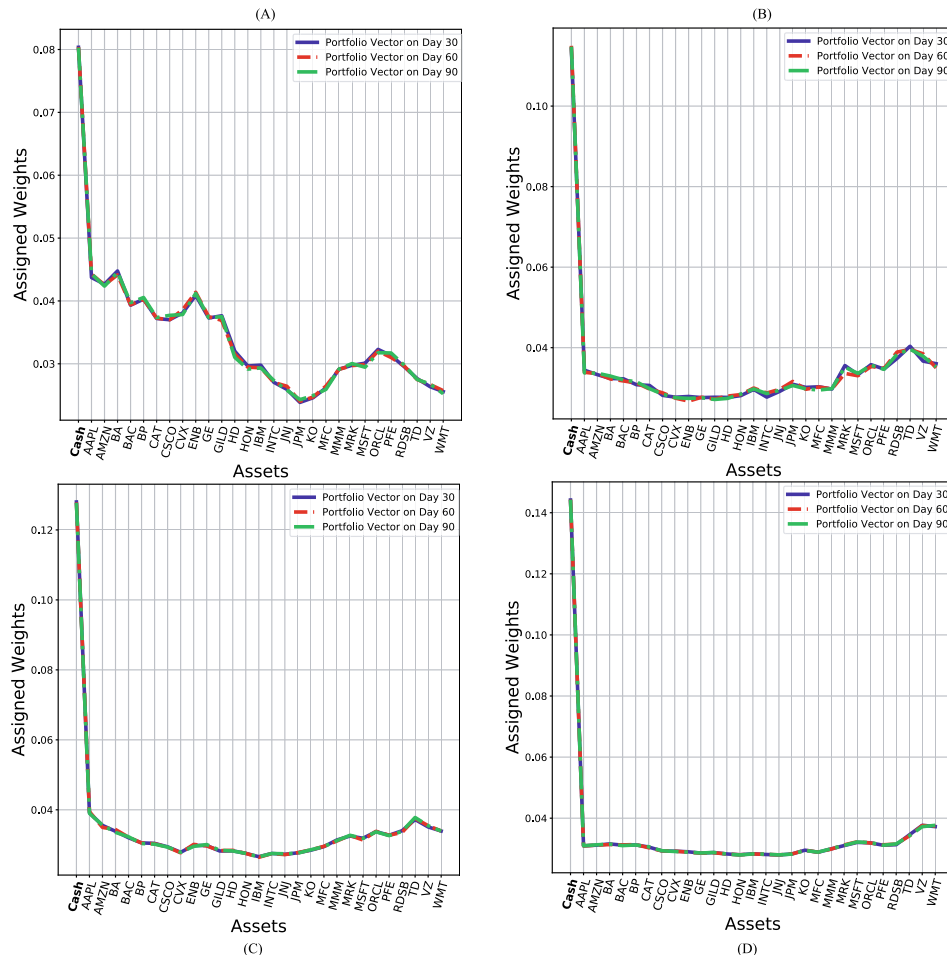


Fig. 9. Portfolio weight distributions at the end of the investment horizon for four test sets.

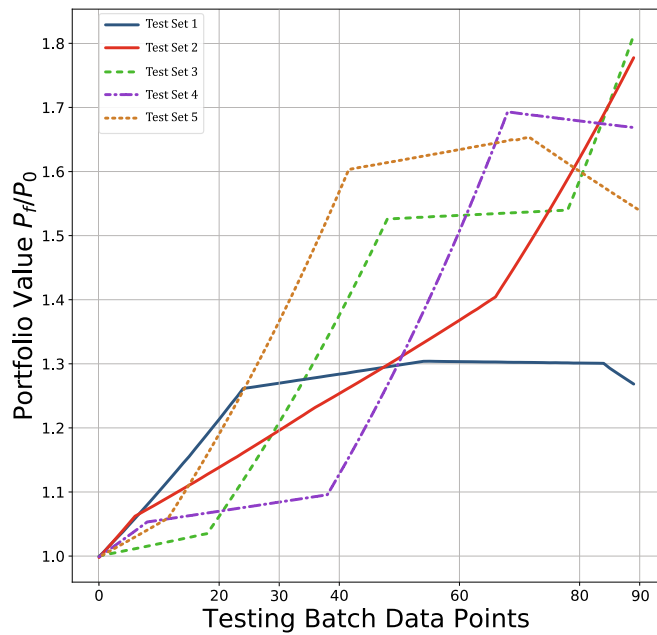


Fig. 10. Relative portfolio values after respectively 30, 60, and 90 days of trading for all five test sets.

The resulting portfolio values are reported in Fig. 10. The portfolio value for the first test set is highlighted in blue. During the first 23 days, the portfolio value increases by 23.8% and then, after 60 days, climbs gradually to 30.33%. Then, the portfolio value remains stable while, during the last five days, the growth decreases to 26.23% after 90 days of trading. The evolution of the portfolio for the second test set consistently increases, showing a growth of 79.57% after 90 days. As for the third test, the growth reaches 83.97% after 90 days, the highest among all test sets. Finally, the fourth test set achieves 68.8% growth after being 68

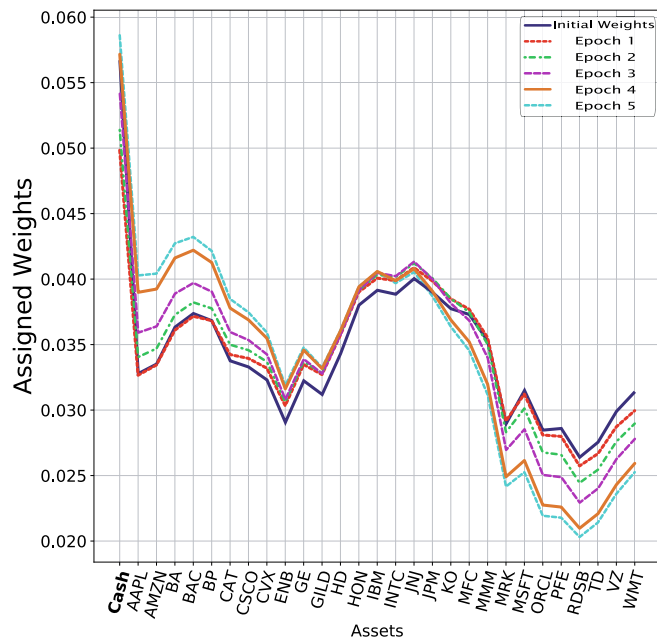


Fig. 11. Final portfolio weights distributions for the fifth training set.

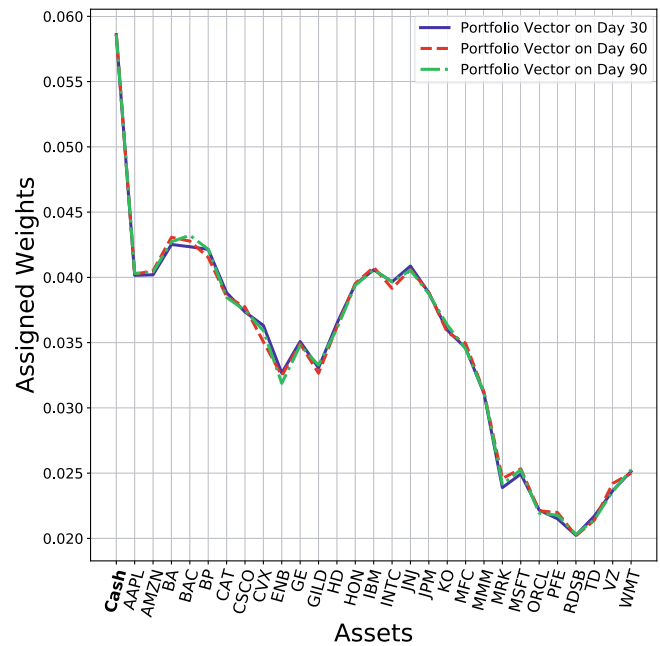


Fig. 12. Final portfolio weight distributions for the fifth test set.

days on the market before gradually dropping to 60.81% after 90 trading days. From Fig. 10, one may conclude that the proposed framework is particularly suitable for medium-term investments (see Figs. 11 and 12).

Our results are summarized in Table 4. With the exception of Test Set 1, the Sharpe ratio is either very good or excellent for all four remaining test sets. In general, it is recognized that a Sharpe ratio below one is sub-optimal; a ratio between one and two is fair, and a ratio between two and three is considered very good. Any ratio over three is considered excellent (Maverick, 2019). In general, online training over a larger time window results in a more efficient policy in terms of return on investment. Our frameworks seems particularly adapted for medium-term

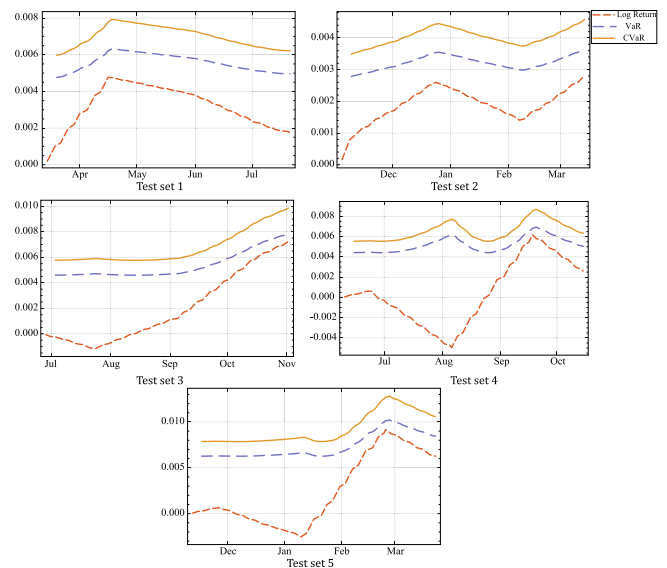


Fig. 13. Risk assessment using CVaR and VaR criteria for all five test sets.

Table 4Maximum drawdown (MDD) and Sharpe ratio (S_r) for DeepPocket, the Dow Jones Industrial (DJI), Euro Stoxx 50, Nasdaq, and S&P500 for all five test sets.

ID	Investment Duration	MDD (%) Algorithm	S_r Algorithm	S_r DJI	S_r FEZ	S_r Nasdaq	S_r S&P500
Test Set 1	30 Days	21.16	2.89	2.101	-3.80	1.766	1.66
	60 Days	23.31	3.11	-1.44	-2.133	-1.62	-1.55
	90 Days	23.31	2.95	-1.458	-1.722	-1.362	-1.42
Test Set 2	30 Days	16.096	3.41	2.752	-1.151	1.746	1.66
	60 Days	26.64	3.507	-1.518	-2.814	0.222	-1.65
	90 Days	43.74	3.84	0.949	-0.418	2.261	1.83
Test Set 3	30 Days	16.18	3.18	0.755	-1.94	-0.3914	-1.070
	60 Days	34.68	2.53	2.23	-0.987	0.755	1.27
	90 Days	44.83	3.39	3.018	1.65	2.819	2.43
Test Set 4	30 Days	7.71	3.24	1.75	1.93	1.951	2.031
	60 Days	32.70	3.67	-0.520	-2.518	-0.0521	-0.285
	90 Days	40.93	2.47	1.6134	0.114	0.981	1.42
Test set 5	30 Days	26.34	3.107	2.51	1.65	3.300	3.014
	60 Days	38.794	2.64	2.59	0.757	3.265	3.03
	90 Days	39.53	2.301	-3.78	-3.678	-2.314	-3.21

investments as the value of the portfolio usually peaks after 80 to 90 trading days.

In order to test the robustness of our framework, DeepPocket managed the portfolio in the depths of the coronavirus crisis (Covid-19). The investments were performed from 11 February 2020, 7 days before the crisis, until 23 March 2020, while the stock market was witnessing record falls. The weight distribution during this time period is shown in Fig. 14 and Table 5. The evolution of the portfolio is illustrated in Fig. 15. The objective of this test was to evaluate the performance of the agent in unprecedented circumstances as the stock market was entering a bear market (Barsky & De Long, 1990). As shown in Fig. 15, despite the debacle in worldwide financial markets, DeepPocket managed to increase the value of the portfolio by 4.46%. This is even more remarkable

when considering that the agent was trained online on a bull market during a sharp rise. This test shows the agent's adaptation capacity under the worst possible conditions. During this period, some assets were more affected than others: indeed, while IBM, GILD, JPM, and HON were in sharp decline, AMZN and WMT were rapidly recovering.

In comparison, as illustrated by Fig. 15, had one invested evenly across these assets, the portfolio value would have shrunk by 17.83%. Moreover, during the same period, the Dow Jones Industrial (DJI) lost 29.36% of its value. Therefore, DeepPocket clearly outperforms uniform-investment and benchmark indexes in times of crisis. Our framework has the ability to remember remote events because of, among

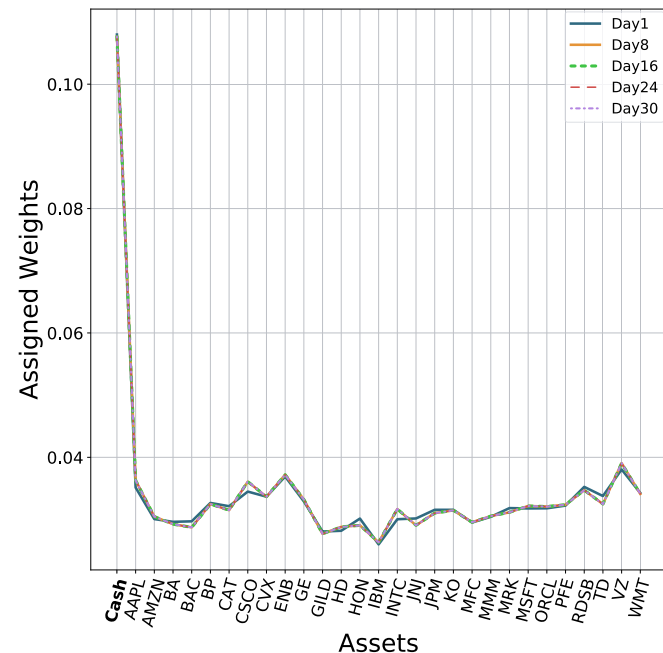


Fig. 14. Evolution of the portfolio weights over a period of 30 days following the Covid-19 crisis.

Table 5

Portfolio weights distribution over a period of 30 days following the Covid-19 crisis(from Fig. 14).

ID	Day 1	Day 8	Day 16	Day 24	Day 30
Cash	0.10905	0.10878	0.10845	0.10807	0.10735
AAPL	0.0349	0.0361	0.0362	0.0365	0.037
AMZN	0.0299	0.0305	0.03052	0.03055	0.03055
BA	0.0296	0.0292	0.02918	0.02918	0.02918
BAC	0.0297	0.0286	0.0287	0.0288	0.02885
BP	0.0327	0.0322	0.0324	0.0322	0.0322
CAT	0.0322	0.03164	0.0315	0.03185	0.03185
CSCO	0.0344	0.03616	0.03605	0.03602	0.03602
CVX	0.03368	0.03367	0.03367	0.03367	0.03367
ENB	0.0367	0.037	0.037	0.03724	0.03724
GE	0.0328	0.033	0.0328	0.0326	0.0326
GILD	0.0283	0.0279	0.0278	0.0279	0.02792
HD	0.0281	0.0288	0.0291	0.0292	0.0292
HON	0.0301	0.0291	0.029	0.0289	0.0289
IBM	0.0259	0.0261	0.0262	0.0266	0.0266
INTC	0.0299	0.0317	0.0316	0.03158	0.03158
JNJ	0.0302	0.02906	0.02908	0.02915	0.02915
JPM	0.0316	0.03097	0.031	0.0312	0.0312
KO	0.0315	0.03148	0.03143	0.03141	0.03141
MFC	0.0297	0.0297	0.0297	0.0295	0.0295
MMM	0.0303	0.03046	0.03058	0.03062	0.03062
MRK	0.0318	0.03125	0.0313	0.0313	0.0313
MSFT	0.03172	0.0321	0.03211	0.0322	0.0322
ORCL	0.0321	0.03208	0.03208	0.03207	0.03207
PFE	0.03225	0.03235	0.0325	0.0321	0.0321
RDSB	0.0351	0.0348	0.03475	0.03471	0.03471
TD	0.0338	0.0324	0.0325	0.0326	0.03275
VZ	0.0378	0.039	0.0389	0.0389	0.0389
WMT	0.0342	0.0339	0.0339	0.03338	0.03338

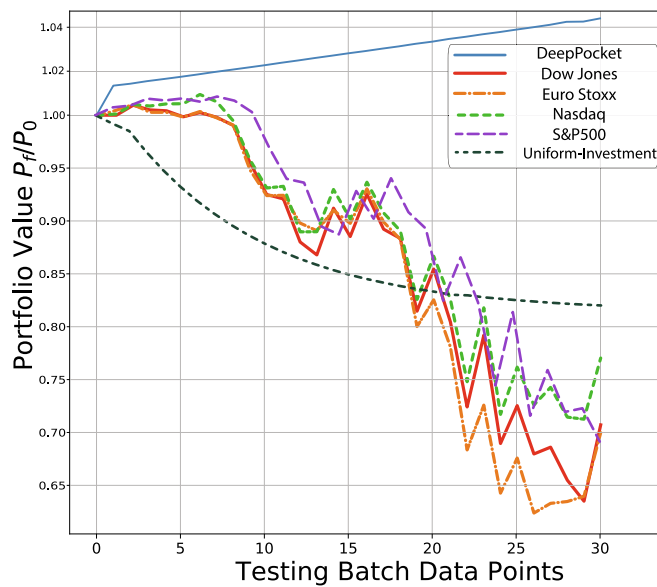


Fig. 15. Evolution of the portfolio value over a period of 30 days following the Covid-19 crisis.

other reasons, gradual concept drift (Gama et al., 2014). The fact that DeepPocket could manage the coronavirus crisis is to be found, in part, in such a long-term memory mechanism. Indeed, the offline training, which was performed on historical data, involved the early 2000's recession where the market reached a low point in 2002 as well as the 2009 market crisis (Langdon, McMenamin, & Krolik, 2002; Farmer, 2012). The knowledge gained by DeepPocket over these two crisis periods was instrumental in the management of the coronavirus crisis.

In order to further assess the performance of the proposed framework, the return on investment of DeepBreath was compared with four benchmark portfolios namely the Dow Jones Industrial (DJI), the Euro Stoxx 50 ETF, Nasdaq, and S&P500 (Serletis & Rosenberg, 2009). Our results are reported in Table 6. For instance, for test set 5, our framework achieved a 37.66%, 63.53%, and 55.36% return on investment (ROI) over thirty, sixty, and ninety days, respectively. Meanwhile, for the same investment periods, the ROI was 2.97%, 5.095%, and -32.86% for DJI, and 2.616%, 1.584%, -35.018% for Euro Stoxx 50 ETF, 5.772%, 12.478%, -18.722% for Nasdaq and finally 4.417%, 7.797%, -27.522% for S&P500.

Table 6

Return on investment for DeepPocket, the Dow Jones Industrial, Euro Stoxx 50, Nasdaq, and S&P500 for all five test sets.

ID	Investment Duration	ROI(%) DeepPocket	ROI(%) DJI	ROI(%) FEZ	ROI(%) Nasdaq	ROI(%) S&P500
Test Set 1	30 Days	26.98	3.28	-6.48	4.625	2.88
	60 Days	30.33	-6.98	-21.88	-8.608	-8.24
	90 Days	26.23	16.908	-15.45	-7.403	-4.94
Test Set 2	30 Days	19.77	1.509	-2.08	2.212	0.739
	60 Days	36.91	0.38	-2.28	4.249	0.829
	90 Days	79.57	12.79	0.467	7.844	5.14
Test Set 3	30 Days	20.85	1.814	-0.807	-0.281	-0.101
	60 Days	53.16	4.17	0.58	3.087	2.521
	90 Days	83.97	20.20	4.539	7.70	6.029
Test Set 4	30 Days	8.5	4.30	3.55	5.280	4.675
	60 Days	50.78	-0.78	-4.54	0.923	0.013
	90 Days	60.81	4.71	0.619	4.866	4.837
Test set 5	30 Days	37.66	2.97	2.616	5.772	4.417
	60 Days	63.53	5.095	1.584	12.478	7.797
	90 Days	55.36	-32.86	-35.018	-18.722	-27.522

8. Discussion

DeepPocket was evaluated against five real-life datasets over three distinct investment periods including during the Covid-19 crisis, for which it clearly outperformed market indexes. The results obtained during the Covid-19 crisis are particularly remarkable as DeepPocket managed to generate a profit by exploiting profitable assets, such as Amazon and Facebook, in addition to exploiting offline knowledge such as what it learned from the 2008 global financial recession. This performance is in contrast with market indexes, which were collapsing during the same period. The reason is that our deep neural network learns the underlying rules, connections and predictable patterns directly from the data. The network adapts as new information becomes available, which allows the system to make better predictions based on insight acquired from previously analyzed information. The amount of complex information that it can process far surpasses largely human capacities. The network may detect anomalies such as unexpected spikes, drops, level shifts and trend changes. In contrast to traders, DeepPocket is not prone to such a high degree of personal biases, conflicts of interest, and emotional decisions. Indeed, as reported recently in,¹ robot-analysis systems, like DeepPocket, tend to outperform traditional investment strategies.

8.1. Conclusion

In this paper, we proposed DeepPocket, a portfolio management framework that aims to increase the expected return on investment while hedging the risk. This framework exploits the time-varying correlations between financial instruments. These correlations are represented in a graph whose nodes correspond to the financial instruments and whose edges correspond to pair-wise correlation functions between assets. DeepPocket consists of a restricted, stacked autoencoder (RSAE) for feature extraction, and an actor-critic reinforcement learning agent. The actor-critic framework consists of two convolutional networks in which the actor learns and enforces an investment policy, and the critic, in turn, evaluates the policy to establish the best course of action by constantly reallocating the various portfolio assets to maximize the expected return on investment. The agent is initially trained offline with online stochastic batching on historical data. As new data becomes available, the agent is trained online with a passive concept-drift approach to handle unexpected changes in the underlying data distribution.

The approach has many benefits for practitioners such as seeking

¹ <https://www.bloomberg.com/news/articles/2020-02-11/robot-analyst-s-outwit-humans-in-study-of-profit-from-stock-calls>.

investment opportunities, identifying market patterns, evaluating trading strategies, and automating workflows, just to mention a few. By finding predictable patterns, structures and trends, the system may provide valuable recommendations and insight to traders. Risk modeling and forecasting, as well as their accuracy, may be improved using the insight acquired from the data. Traditional risk models have demonstrated their ineffectiveness in coping with global financial crisis. As shown by our results, DeepPocket successfully managed such risks during the Covid-19 crisis: a characteristic which has direct repercussions for portfolio risk management and compliance.

DeepPocket assumes a liquid market which means that many buyers and sellers are available, while price variations remain relatively small. We plan to target this problem in a future work, in order to address situations in which buyers and sellers are in short supply. Besides portfolio allocation, we plan to have DeepPocket select the stocks directly in addition to developing deep learning models more suitable for long-term investment strategies.

CRedit authorship contribution statement

Farzan Soleymani: Conceptualization, Data curation, Formal analysis, Methodology, Software, Validation, Visualization, Writing - original draft, Writing - review & editing. **Eric Paquet:** Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- Acerbi, C. (2002). Spectral measures of risk: A coherent representation of subjective risk aversion. *Journal of Banking & Finance*, 26, 1505–1518.
- Alexander, S., Coleman, T. F., & Li, Y. (2006). Minimizing cvar and var for a portfolio of derivatives. *Journal of Banking & Finance*, 30, 583–605.
- Andrew, A.M. (1999). Reinforcement learning: An introduction by richard s. Sutton and Andrew G. Barto, adaptive computation and machine learning series, MIT Press (Bradford book), Cambridge, Mass., 1998, xviii+ 322 pp, isbn 0-262-19398-1, (hardback, £ 31.95). *Robotica*, 17, 229–235.
- Angles, R., & Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40, 1–39.
- Anthony, J. H. (1988). The interrelation of stock and options market trading-volume data. *The Journal of Finance*, 43, 949–964.
- Arfaoui, M., & Rejeb, A. B. (2017). Oil, gold, us dollar and stock market interdependencies: a global analytical insight. *European Journal of Management and Business Economics*.
- Ashraf, B.N. (2020). Stock markets' reaction to covid-19: Cases or fatalities? Research in International Business and Finance, (p. 101249).
- Barabási, A.-L., et al. (2016). *Network science*. Cambridge University Press.
- Barsky, R. B., & De Long, J. B. (1990). Bull and bear markets in the twentieth century. *The Journal of Economic History*, 50, 265–281.
- Bengio, Y., Goodfellow, I. J., & Courville, A. (2015). Deep learning. *Nature*, 521, 436–444.
- Bessler, W., Opfer, H., & Wolff, D. (2017). Multi-asset portfolio optimization and out-of-sample performance: an evaluation of black-litterman, mean-variance, and naive diversification approaches. *The European Journal of Finance*, 23, 1–30.
- Bhanja, S., & Das, A. (2018). Impact of data normalization on deep neural network for time series forecasting. arXiv preprint arXiv:1812.05519.
- Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., & Lee, M. (2009). Natural actor-critic algorithms. *Automatica*, 45, 2471–2482.
- Bouchaud, J.-P., Matacz, A., & Potters, M. (2001). Leverage effect in financial markets: The retarded volatility model. *Physical review letters*, 87, Article 228701.
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34, 18–42.
- Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.
- Bu, L., Babu, R., De Schutter, B., et al. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38, 156–172.
- Cavalcante, R. C., & Oliveira, A. L. (2015). An approach to handle concept drift in financial time series based on extreme learning machines and explicit drift detection. In *2015 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.
- Celikyurt, U., & Özekici, S. (2007). Multiperiod portfolio optimization models in stochastic markets using the mean–variance approach. *European Journal of Operational Research*, 179, 186–202.
- Cervelló-Royo, R., & Guíjarro, F. (2020). Forecasting stock market trend: A comparison of machine learning algorithms. *Finance, Markets and Valuation*, 6, 37–49.
- Chung, F. R., & Graham, F. C. (1997). *Spectral graph theory*, 92. American Mathematical Soc.
- Cornuejols, G., & Tütüncü, R. (2006). *Optimization methods in finance* (Vol. 5). Cambridge University Press.
- Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems* (pp. 3844–3852).
- Drozd, S., Grümm, F., Ruf, F., & Speth, J. (2001). Towards identifying the world stock market cross-correlations: Dax versus dow jones. *Physica A: Statistical Mechanics and its Applications*, 294, 226–234.
- Elton, E. J., Gruber, M. J., Brown, S. J., & Goetzmann, W. N. (2009). *Modern portfolio theory and investment analysis*. John Wiley & Sons.
- Fama, E. F. (1965). The behavior of stock-market prices. *The Journal of Business*, 38, 34–105.
- Farmer, R. E. (2012). The stock market crash of 2008 caused the great recession: Theory and evidence. *Journal of Economic Dynamics and Control*, 36, 693–707.
- Felmer, P., Quaa, A., & Tan, J. (2012). Positive solutions of the nonlinear schrödinger equation with the fractional laplacian. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, 142, 1237–1262.
- Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Brazilian symposium on artificial intelligence* (pp. 286–295). Springer.
- Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine learning*, 90, 317–346.
- Gama, J., Žilobaite, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46, 44.
- García, F., González-Bueno, J., Oliver, J., & Riley, N. (2019). Selecting socially responsible portfolios: a fuzzy multicriteria approach. *Sustainability*, 11, 2496.
- García, F., González-Bueno, J., Oliver, J., & Tamosiūnienė, R. (2019). A credibilistic mean-semivariance-per portfolio selection model for latin america. *Journal of Business Economics and Management*, 20, 225–243.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., & Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 1263–1272). JMLR. org.
- Grondman, I., Busoniu, L., Lopes, G. A., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42, 1291–1307.
- Hammond, D. K., Vandergheynst, P., & Gribonval, R. (2011). Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30, 129–150.
- Henaff, M., Bruna, J., & LeCun, Y. (2015). Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163.
- Hu, Y., Liu, K., Zhang, X., Xie, K., Chen, W., Zeng, Y., & Liu, M. (2015). Concept drift mining of portfolio selection factors in stock market. *Electronic Commerce Research and Applications*, 14, 444–455.
- Hussain, A. J., Knowles, A., Lisboa, P. J., & El-Dereby, W. (2008). Financial time series prediction using polynomial pipelined neural networks. *Expert Systems with Applications*, 35, 1186–1199.
- Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. arXiv preprint arXiv:1706.10059.
- Kingma, D., & Ba, J. (2015). Adam: a method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980, 15.
- Kipf, T.N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.
- Konda, V.R., & Tsitsiklis, J.N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008–1014).
- Langdon, D. S., McMenamin, T. M., & Krolak, T. J. (2002). Us labor market in 2001: Economy enters a recession. *Monthly Labor Review*, 125, 3.
- Langr, J., & Bok, V. (2019). *GANs in action: Deep learning with generative adversarial networks*. Manning Publications.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
- Lucey, B. M., & Muckley, C. (2011). Robust global stock market interdependencies. *International Review of Financial Analysis*, 20, 215–224.
- Magdon-Ismael, M., & Atiya, A. F. (2004). Maximum drawdown. *Risk Magazine*, 17, 99–102.
- Markowitz, H. M. (1978). *Portfolio selection*. New York: Wiley.
- Maverick, J. (2019). What is a good sharpe ratio? URL: <https://www.investopedia.com/ask/answers/010815/what-good-sharpe-ratio.asp>.
- Meng, Q., Catchpole, D., Skillicom, D., & Kennedy, P. J. (2017). Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 364–371). IEEE.
- Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.
- Nti, I. K., Adekoya, A. F., & Weyori, B. A. (2019). A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review*, 1–51.
- Omid, F., Abbasi, B., & Nazemi, A. (2017). An efficient dynamic model for solving a portfolio selection with uncertain chance constraint models. *Journal of Computational and Applied Mathematics*, 319, 43–55.
- Ormos, M., & Urbán, A. (2013). Performance analysis of log-optimal portfolio strategies with transaction costs. *Quantitative Finance*, 13, 1587–1597.

- Park, K., & Shin, H. (2013). Stock price prediction based on a complex interrelation network of economic factors. *Engineering Applications of Artificial Intelligence*, 26, 1550–1561.
- Pouya, A. R., Solimanpur, M., & Rezaee, M. J. (2016). Solving multi-objective portfolio optimization problem using invasive weed optimization. *Swarm and Evolutionary Computation*, 28, 42–57.
- Qiu, M., & Song, Y. (2016). Predicting the direction of stock market index movement using an optimized artificial neural network model. *PloS one*, 11, Article e0155133.
- Rocchi, J., Tsui, E. Y. L., & Saad, D. (2017). Emerging interdependence between stock values during financial crashes. *PloS one*, 12.
- Rockafellar, R. T., Uryasev, S., et al. (2000). Optimization of conditional value-at-risk. *Journal of risk*, 2, 21–42.
- Sarwar, G., & Khan, W. (2019). Interrelations of us market fears and emerging markets returns: Global evidence. *International Journal of Finance & Economics*, 24, 527–539.
- Serletis, A., & Rosenberg, A. A. (2009). Mean reversion in the us stock market. *Chaos, Solitons & Fractals*, 40, 2007–2015.
- Sharpe, W. F. (1994). The sharpe ratio. *Journal of portfolio management*, 21, 49–58.
- Shuman, D. I., Narang, S. K., Frossard, P., Ortega, A., & Vandergheynst, P. (2013). The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30, 83–98.
- Shuman, D. I., Ricaud, B., & Vandergheynst, P. (2016). Vertex-frequency analysis on graphs. *Applied and Computational Harmonic Analysis*, 40, 260–291.
- Shuman, D. I., Vandergheynst, P., & Frossard, P. (2011). Chebyshev polynomial approximation for distributed signal processing. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)* (pp. 1–8). IEEE.
- Soleymani, F., & Paquet, E. (2020). Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder-deepbreath. *Expert Systems with Applications*, (p. 113456).
- Stone, A., Wang, H., Stark, M., Liu, Y., Scott Phoenix, D., & George, D. (2017). Teaching compositionality to cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5058–5067).
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 9–44.
- Sutton, R.S., McAllester, D.A., Singh, S.P., & Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (pp. 1057–1063).
- Velicković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.
- Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, 8, 279–292.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Philip, S. Y. (2020). A comprehensive survey on graph neural networks. In *IEEE Transactions on Neural Networks and Learning Systems*.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2019). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.
- Yue, W., & Wang, Y. (2017). A new fuzzy multi-objective higher order moment portfolio selection model for diversified portfolios. *Physica A: Statistical Mechanics and its Applications*, 465, 124–140.
- Zhang, D., Hu, M., & Ji, Q. (2020). Financial markets under the global pandemic of covid-19. *Finance Research Letters*, (p. 101528).
- Zhang, S., Tong, H., Xu, J., & Maciejewski, R. (2019). Graph convolutional networks: A comprehensive review. *Computational Social Networks*, 6, 11.
- Zhao, D., Wang, H., Shao, K., & Zhu, Y. (2016). Deep reinforcement learning with experience replay based on sarsa. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–6). IEEE.

5.3 Long-term financial predictions based on Feynman–Dirac path integrals, deep Bayesian networks and temporal generative adversarial networks

This paper presents a new deep learning framework, QuantumPath, for long-term stock price prediction. This approach is established based on stochastic equations, the Feynman–Dirac path integral, deep Bayesian networks, and temporal generative adversarial neural networks (temporal GAN). The expected financial trajectory is estimated with a Feynman–Dirac path integral by summing all possible financial trajectories that *could* have been taken by the financial instrument. The temporal GAN generates these trajectories and assigns a probability to each point on each path. The probability is a function of the Lagrangian, which is derived from a stochastic equation describing the temporal evolution of the stock. Since the evolution of a stock’s price is isomorphic to a time series, the temporal GAN consists of LSTM neural networks, which introduce a memory mechanism, and TCNs, which ensure causality.

In the same manner, the primary structure of proteins follows a causal directionality (see Chapter 3), which is a challenge for traditional neural networks to manage[51, 52]. Similar to QuantumPath, LSTM and TCN architectures help capture sequential characteristics of the protein primary structure. Generative models have been widely used for protein sequence and structure design and protein function prediction (see Section. 7 in Chapter 2). This paper is published in the Machine Learning with Applications journal on January 10, 2022.



Long-term financial predictions based on Feynman–Dirac path integrals, deep Bayesian networks and temporal generative adversarial networks

Farzan Soleymani^a, Eric Paquet^{b,*}

^a Department of Mechanical Engineering, University of Ottawa, Ottawa, ON, Canada

^b National Research Council, 1200 Montreal Road, Ottawa, ON, K1A 0R6, Canada

ARTICLE INFO

Keywords:

Temporal generative adversarial network
Time series
Financial predictions
Long short-term memory
Temporal convolutional network

ABSTRACT

This paper presents a new deep learning framework, QuantumPath, for long-term stock price prediction, which is of great significance in portfolio management and risk mitigation, especially when the market becomes volatile due to unpredictable circumstances such as a pandemic. Our approach is based on stochastic equations, the Feynman–Dirac path integral, deep Bayesian networks, and temporal generative adversarial neural networks (t-GAN). The expected financial trajectory is evaluated with a Feynman–Dirac path integral. The latter involves summing all possible financial trajectories that *could* have been taken by the financial instrument. These trajectories are generated with a t-GAN. A probability is attributed to each point of each path. The probability is a function of the Lagrangian, which is derived from a stochastic equation describing the temporal evolution of the stock. The drift and the volatility at each point, which are required in order to evaluate the Lagrangian, are predicted with a deep Bayesian neural network. Given that the evolution of a stock's price is isomorphic to a time series, our temporal GAN consists of long short-term memory (LSTM) neural networks, which introduce a memory mechanism, and temporal convolutional neural networks (TCN), which ensure causality. Stock prices are predicted over periods of twenty and thirty days for nine stocks, eight of which are included in the S&P 500 index. Our experimental results clearly demonstrate the efficiency of our approach.

The stock market is known for its erratic behaviour (Ziemba et al., 2017) with a 2020 episode resulting from the pandemic (Ashraf, 2020; Zhang et al., 2020). During the first quarter of 2020, while numerous companies experienced a sharp decline in their value, others such as Amazon, Walmart, and Tesla were sharply rising. As a result, those who invested in these companies prior to the pandemic made substantial profits while others suffered heavy losses (Nicola et al., 2020). In such contexts, stock price prediction may contribute to improving the return on investment while also mitigating the risks associated with uncertainty (Zhou et al., 2018). Generative models have the ability to learn the probability density function associated with a dataset (e.g., audio and images) (Lucic et al., 2018) while the posterior distribution may be inferred from discriminative models (Ulusoy & Bishop, 2006). Generative adversarial networks (GAN) are generative models that were first introduced by Goodfellow et al. (2014). They aim to generate synthetic data with the same probability distribution function as the training dataset. They consist of a generator that produces synthetic data and a discriminator, which evaluates the discrepancy between the synthetic data and the real data.

These networks are notably difficult to train and tend to suffer from mode collapse (He et al., 2019; Thanh-Tung & Tran, 2020). High-frequency stock market prediction based on GAN was first proposed

by Zhou et al. (2018). Their network consisted of a long short-term memory (LSTM) network for the generator and a convolutional neural network (CNN) for the discriminator. Mode collapsing was reduced in Saatci and Wilson (2017) by employing Bayesian inference. Bayesian neural networks were employed both for the generator and for the discriminator, and the loss function was defined in terms of maximum likelihood. However, this approach did not guarantee convergence (Kononenko, 1989; Saatci & Wilson, 2017). Additionally, a stock market prediction framework based on GAN was proposed in Zhang et al. (2019), where the discriminator and the generator were implemented with a multi-layer perceptron (MLP) and a LSTM, respectively. The objective was to predict the daily closing prices of various stocks from the S&P 500.

In quantum mechanics, quantum state evolution has been tackled by renowned scientists such as Schrödinger, Dirac, and Feynman. Initially, Schrödinger derived a wave equation from the Hamiltonian of the system under consideration (Andreev, 2006). Later on, Dirac and Feynman proposed an alternative, but completely equivalent, formulation, based on the Lagrangian. In Dirac and Feynman's approach, time evolution is expressed in terms of a path integral which is evaluated as a weighted

* Corresponding author.

sum over all possible paths or trajectories between the initial and the final states, the weight being determined by a functional of the Lagrangian associated with the system under consideration (Perepelitsa, 2018). The path integral is discrete in nature and may be evaluated with a truncated expansion (Kakushadze, 2015; Perepelitsa, 2018). Such path integrals have been employed in numerous applications, including protein folding, quantum mechanics, stochastic dynamics, quantum field theories, to mention just a few (Ingold, 2002; Linetsky, 1997).

Temporal convolutional networks (TCNs) were originally introduced by Lea et al. (2016) for video-based action segmentation. They allow a hierarchical, bi-level analysis of time series (Lea et al., 2016). Low-level features are initially extracted with a CNN, and high-level features are subsequently captured by a recurrent neural network (RNN) (Lea et al., 2016). TCNs tend to outperform standard recurrent networks when temporal series or data streams are involved (Bai et al., 2018; Chung et al., 2014; Jozefowicz et al., 2015; Pascanu et al., 2013). In addition, TCNs tend to have a longer long-term memory than RNNs, making them more suitable when remote past events are required in order to obtain accurate predictions (Bai et al., 2018; Oord et al., 2016; Wan et al., 2019).

In this paper, a new framework for financial predictions, called QuantumPath, is proposed. The expected financial trajectory for a given stock is obtained from a Feynman–Dirac path integral derived from a generic stochastic equation, this latter describing the temporal evolution of the financial instrument. The drifts and the volatilities, which are the parameters required to evaluate the probability associated with a particular realisation of a given path, are predicted by a deep Bayesian neural network. The various trajectories which result from sampling the data distribution and which are required in order to evaluate the path integral with Monte Carlo techniques are predicted using a temporal GAN. Three architectures are proposed for the t-GAN: a dense network for both the generator and the discriminator, a dense network for the generator and a LSTM network for the discriminator, and a TCN for both the generator and the discriminator.

The paper is organised as follows. Feynman–Dirac path integrals for financial predictions are introduced in Section 1. The prediction of the path integral parameters, using deep Bayesian networks, is addressed in Section 2. Financial path generations and data distribution sampling with a t-GAN are exposed in Section 3. The architecture of our t-GAN is described in Section 4. Our experimental results are reported in Section 5, while Section 6 concludes the paper.

1. Financial path prediction using stochastic equations and the Feynman–Dirac path integral

Essentially, there are two formulations of quantum mechanics: the Schrödinger (Goldin, 2008) equation, and the Feynman–Dirac equations (Truman, 1978) (The Heisenberg equation (Esteve, 2002) is an alternative formulation of the Schrödinger equation (Aizenman et al., 2006) in which the state vectors are stationary while the operators are time-dependent.) The Schrödinger and the Feynman–Dirac formulations are entirely equivalent (Wu^{thrich}, 2010). The Schrödinger formulation is based on the time-dependent Schrödinger equation:

$$i\hbar \frac{\partial |S(t)\rangle}{\partial t} = \hat{\mathcal{H}} |S(t)\rangle \quad (1)$$

in which $|S(t)\rangle$ is the time-dependent state vector and $\hat{\mathcal{H}}$ is the Hamiltonian operator; the latter expressing the system's total energy. Whereas most quantum computing algorithms are based on the Schrödinger formulation, here we formulate the quantum algorithm in terms of Feynman–Dirac path integrals. In contrast to most approaches, the quantum formulation is not imposed *ex cathedra* but, rather, derived directly from the stochastic equations associated with the temporal evolution of the financial instruments. The temporal evolution of a

security S may be characterised by a stochastic equation (Linetsky, 1997):

$$\frac{dS(t)}{dt} = \mu(t)S(t) + \sigma(t)S(t)\xi(t) \quad (2)$$

where $\mu(t)$ and $\sigma(t)$ are the drift and volatility, respectively. It is worth nothing that no hypothesis is made about the nature of these functions. The stochastic noise is defined as

$$E[\xi(t)] = 0,$$

$$E[\xi(t)\xi(t')] = \delta(t - t'), \quad (3)$$

$$T_i \leq t, t' \leq T_f$$

where $E[\cdot]$ is the mathematical expectation (Strukov & Timan, 1977) while T_i and T_f define the investment horizon. Naturally, the evolution of a security is a discrete process characterised by a 'tick,' ϵ , which is the smallest temporal movement admissible:

$$\frac{S(t_{n+1}) - S(t_n)}{t_{n+1} - t_n} \simeq \alpha(t_n)S(t_n) + \sigma(t_n)S(t_n)\xi(t_n), \quad (4)$$

$$t_{n+1} - t_n \equiv \epsilon \quad \forall n.$$

Instead of considering the evolution of a state vector, the Feynman–Dirac equation considers all possible financial trajectories associated with a security: a probability, determined by the Lagrangian of the financial process, is associated with every realisation. This is to be contrasted with the Hamiltonian-based Schrödinger equation; the Lagrangian and the Hamiltonian are related by a Legendre transformation (Tulczyjew, 1977). As the various realisations of a security are determined by the stochastic noise, the Feynman–Dirac equation considers all possible noise realisations. A detailed description of path integrals may be found in Masujima (2008) and Perepelitsa (2018). Therefore, the Feynman–Dirac integration metric is given by

$$\int \mathcal{D}\xi \triangleq \prod_{t=T_i}^{T_f} \int_{-\infty}^{\infty} d\xi(t). \quad (5)$$

where the integration is carried out over all possible noise values up to the investment horizon. According to the Feynman–Dirac equation, the probability of a given noise realisation is

$$\int \mathcal{D}\xi p[\xi] = \frac{1}{Z} \int \mathcal{D}\xi \exp[\mathcal{A}[\xi]] \quad (6)$$

where Z is a normalisation factor:

$$Z = \int \mathcal{D}\xi \exp[\mathcal{A}[\xi]] \quad (7)$$

and

$$\mathcal{A}[\xi] = \int_{T_i}^{T_f} dt \mathcal{L}[\xi]. \quad (8)$$

is the action associated with the stochastic noise, i.e. the time integral of the Lagrangian (Dirac, 2005). The Lagrangian may be defined as a quadratic function:

$$\mathcal{L}[\xi] \triangleq -\frac{1}{2} \xi(t) M(t, t') \xi(t'), \quad (9)$$

$M(t, t') = (E[\xi(t)\xi(t')])^{-1} \Rightarrow \mathcal{L}[\xi] = -\frac{1}{2} \xi^2(t).$

Let us define

$$z(t) \equiv \ln S(t). \quad (10)$$

Then, Eq. (2) becomes:

$$\frac{dz(t)}{dt} = -\mu(t) - \sigma(t)\xi(t). \quad (11)$$

The path integral of the Dirac delta function (Linetsky, 1997) of Eq. (11) is hence

$$\int \mathcal{D}z \delta \left(\frac{1}{\sigma(t)} \left(\frac{\partial z(t)}{\partial t} + \mu(t) \right) + \xi(t) \right) = 1. \quad (12)$$

As this path integral is equal to the identity, Eq. (12) may be inserted into Eq. (7) without changing its value:

$$Z = \int D\xi p[\xi] = \int Dz D\xi \delta \left(\frac{1}{\sigma(t)} \left(\frac{\partial z(t)}{\partial t} + \mu(t) \right) + \xi(t) \right) \exp[\mathcal{A}[\xi]]. \quad (13)$$

This expression may now be simplified by integrating over the stochastic noise:

$$\begin{aligned} Dz p[z] &= \frac{1}{Z'} Dz \exp[\mathcal{A}[z]], \\ \mathcal{A}[z] &= \int_{T_i}^{T_f} dt \mathcal{L}[z], \\ \mathcal{L}[z] &= -\frac{1}{2} \left[\frac{\frac{\partial z(t)}{\partial t} + \mu(t)}{\sigma(t)} \right]^2, \\ \int Dz &= \prod_{t=T_i}^{T_f} \int_{-\infty}^{\infty} dz(t), \\ Z' &= \int Dz \exp[\mathcal{A}[z]]. \end{aligned} \quad (14)$$

Such a path integral is illustrated in Fig. 1. It follows that the expectation value of a security, which is the prediction, becomes

$$E[z] \triangleq \int Dz p[z] z = \frac{1}{Z} \int Dz \exp[\mathcal{A}[z]] z \quad (15)$$

where all the terms are defined in Eq. (14). This path integral may be evaluated with Monte Carlo techniques (Hurtado & Barbat, 1998):

$$E(z(t)) = \frac{1}{\sum_{i=1}^N p(z^{(i)}(t))} \sum_{i=1}^K z^{(i)}(t) p(z^{(i)}(t)), \quad (16)$$

$$p(z^{(i)}(t)) = \frac{1}{\sigma(t)\sqrt{2\pi}} \exp \left[-\frac{1}{2} \left(\frac{\frac{\partial z^{(i)}(t)}{\partial t} - \mu(t)}{\sigma(t)} \right)^2 \right] \quad (17)$$

where $z^{(i)}(t)$ is a particular realisation of the financial instrument $z(t)$. The expected value, which is also the observable, corresponds to a weighted sum over all possible financial trajectories; the weights have Gaussian probability distribution functions. In order to apply this quantum model, one should be able to predict, at each time step, the drift and the volatility as well as generating the various financial paths. Instead of utilising an *ad hoc* model, we propose to generate the financial paths with a temporal generative adversarial network, and to predict the drift and the volatility with a deep Bayesian network. Applying Monte Carlo techniques entails sampling the probability distribution associated with the data; the better the sampling, the faster the convergence, and the higher the accuracy that may be achieved. Unfortunately, these distributions are notoriously difficult to sample because of their high dimensionality (Hurtado & Barbat, 1998). By employing generative models, which learn the data distribution, we expect that both fast convergence and accuracy may be achieved with a relatively small number of paths. Experimental results shall indeed demonstrate that it is the case. Deep Bayesian neural networks are introduced in the next section.

The parameters of the path integral are predicted using deep Bayesian neural networks.

2. Deep Bayesian neural network for predicting drift and the volatility

Bayesian neural networks were introduced by David (1992). They are essentially stochastic networks in which the parameter estimation is done by Bayesian inference (Jospin et al., 2020). A Bayesian neural network may be defined as

$$p(\mathbf{y} | \mathbf{x}, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y} | \mathbf{x}, \omega) p(\omega | \mathbf{X}, \mathbf{Y}) d\omega \quad (18)$$

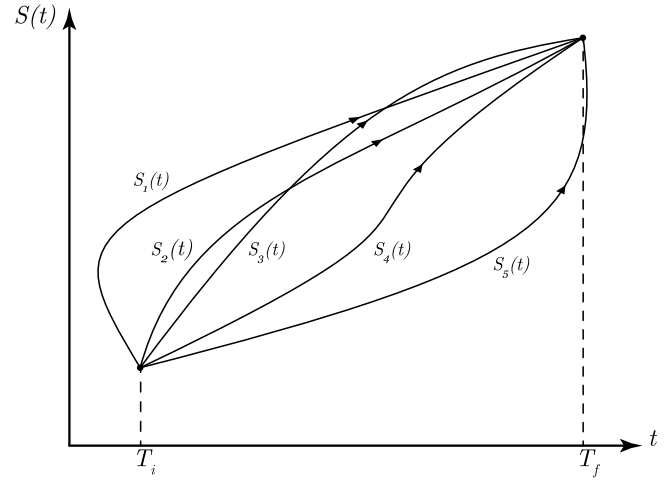


Fig. 1. Some of the paths, or trajectories, involved in evaluating a Feynman-Dirac path integral.

where $\mathbf{x}$ is the input, $\mathbf{y}$ the corresponding output, ω represents the parameters of the neural network, and $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ and $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^N$ are the input and output training sets, respectively. Here, $p(\mathbf{y} | \mathbf{x}, \mathbf{X}, \mathbf{Y})$ is the conditional probability of the output given the input and the training set, $p(\mathbf{y} | \mathbf{x}, \omega)$ is the conditional probability of the output given the input and the parameters, and $p(\omega | \mathbf{X}, \mathbf{Y})$ is the posterior distribution over the training set. As opposed to a predictive standard neural network, the output of a predictive Bayesian neural network is a density probability function rather than just a single value. It thus allows for the prediction of drift and volatility. Our Bayesian neural network is a dense deep neural network, which means that

$$\omega = \{\mathbf{W}_i\}_{i=1}^L \quad (19)$$

where $\{\mathbf{W}_i\}_{i=1}^L$ are the weights associated with the various layers. The weights priors are all assumed to be Gaussian:

$$p_0(\mathbf{W}_i) = \mathcal{N}(\mathbf{W}_i; \mathbf{0}, \mathbf{I}). \quad (20)$$

The weights posterior is not tractable, in general, but may be approximated by variational inference (Blei et al., 2017; Zhang et al., 2018). As a result, the real posterior weights posterior $p(\omega | \mathbf{X}, \mathbf{Y})$ is approximated with a variational distribution $q_\theta(\omega)$. The variational parameters θ are obtained by minimising the Kullback-Leibler divergence $\text{KL}[q \| p]$ between the variational distribution and the real posterior distribution (Amin, 2019; Chen & Shao, 1997). The Kullback-Leibler divergence is zero when both distributions are identical. Assuming that the data are statistically independent, one obtains:

$$\begin{aligned} \text{KL}[q \| p] &= E_q \left[\ln \frac{q}{p} \right] = \text{KL}[q_\theta(\omega) \| p(\mathbf{y} | \mathbf{X}, \omega)] \\ &\propto - \int q_\theta(\omega) \ln p(\mathbf{y} | \mathbf{X}, \omega) d\omega + \text{KL}[q_\theta(\omega) \| p_0(\omega)] \\ &= - \sum_{i=1}^n \int q_\theta(\omega) \ln p(\mathbf{y}_i | f^\omega(\mathbf{x}_i)) d\omega + \text{KL}[q_\theta(\omega) \| p_0(\omega)] \end{aligned} \quad (21)$$

where $f^\omega(\cdot)$ is a dense deep neural network with

$$\theta = \{\mathbf{M}_i\}_{i=1}^L \quad (22)$$

being the corresponding weights. Deterministic neural networks with stochastic regularisation, such as dropouts (Ba & Frey, 2013), may be interpreted as Bayesian neural network approximations (Kononenko, 1989; Springenberg et al., 2016). Indeed, applying a stochastic reg-

ularisation technique is equivalent to multiplying the neural network weights by random noise:

$$\mathbf{W}_i = \varepsilon_i \mathbf{M}_i. \quad (23)$$

The resulting stochastic matrix samples the posterior distribution over the Bayesian neural network weights (Kononenko, 1989; Mullachery et al., 2018). The $\text{KL}[q \parallel p]$ divergence may be generalised by means of Amari's α -divergence (Cai et al., 2020; Hernandez-Lobato et al., 2016), this latter being defined as

$$D_\alpha[p \parallel q] = \frac{1}{\alpha(1-\alpha)} \left(1 - \int p(\omega)^\alpha q(\omega)^{1-\alpha} d\omega \right). \quad (24)$$

Amari's α -divergence encompasses numerous metrics which include the $\text{KL}[q \parallel p]$ divergence ($\alpha = 0$), and the Hellinger distance ($\alpha = 1/2$), amongst others. Therefore, the Kullback–Leibler divergence in Eq. (21) may be replaced with Amari's α -divergence. The resulting energy function (a cost or loss function) is called a black-box α energy function (Hernandez-Lobato et al., 2016). Such an energy function provides a better approximation for the marginal likelihood, resulting in a model that is less biased and which better fits the data distribution (Habek, 2012). To make the text more readable, the following notation shall be adopted henceforth:

$$p(\omega) = \frac{1}{Z} p_0(\omega) \prod_n f_n(\omega), \quad (25a)$$

$$p(\omega) \equiv p(\omega \mid \mathbf{X}, \mathbf{Y}) \quad (25b)$$

$$f_n(\omega) \equiv p(\mathbf{y}_n \mid \mathbf{x}_n, \omega), \quad (25c)$$

$$Z = p(\mathbf{Y} \mid \mathbf{X}). \quad (25d)$$

In the most general case, the black-box α energy function is given by

$$\text{KL}[q \parallel p] \rightarrow D_\alpha[q \parallel p] \Rightarrow \mathcal{L}_\alpha(q) = -\frac{1}{\alpha} \sum_n \ln E_q \left[\left(\frac{f_n(\omega) p_0(\omega)^{1/N}}{q(\omega)^{1/N}} \right)^\alpha \right]. \quad (26)$$

This equation reduces to Eq. (21) when $\alpha \rightarrow 0$:

$$\lim_{\alpha \rightarrow 0} \mathcal{L}_\alpha(q) \equiv \mathcal{L}_0(q) = \text{KL}[q \parallel p_0] - \sum_n E_q [\ln f_n(\omega)]. \quad (27)$$

Yet, Eq. (26) is computationally prohibitive ($\mathcal{O}(N)$). In order to solve this problem, Santana and Hernández-Lobato (2020) proposed a reparametrisation of the black-box α energy function in order to reduce the computational burden. Firstly, the posterior distribution is approximated with a free-form cavity distribution:

$$q(\omega) = \frac{1}{Z_q} \tilde{q}(\omega) \left(\frac{\tilde{q}(\omega)}{p_0(\omega)} \right)^{\frac{\alpha}{N-\alpha}}. \quad (28)$$

When this approximation is substituted in Eq. (26), the energy function reduces to

$$\mathcal{L}_\alpha(q) = -\frac{1}{\alpha} \sum_n \ln \int \left(\frac{1}{Z_q} \tilde{q}(\omega) \left(\frac{\tilde{q}(\omega)}{p_0(\omega)} \right)^{\frac{\alpha}{N-\alpha}} \right)^{1-\frac{\alpha}{N}} \times p_0(\omega)^{\alpha/N} f_n(\omega)^\alpha d\omega. \quad (29)$$

As $\alpha/N \simeq 0$ in general, this expression may be further simplified:

$$\lim_{\frac{\alpha}{N} \rightarrow 0} \mathcal{L}_\alpha(q) \approx \tilde{\mathcal{L}}_\alpha(q) = \text{KL}[q \parallel p_0] - \frac{1}{\alpha} \sum_n E_q [\ln f_n(\omega)^\alpha]. \quad (30)$$

Since our Bayesian network is employed in a regression/prediction context, we have

$$f_n(\omega) \propto p(\mathbf{y}_n \mid \mathbf{x}_n, \omega) \propto \exp[-\ell(\mathbf{y}_n, f^\omega(\mathbf{x}_n))] \quad (31)$$

where $\ell(\mathbf{y}_n, f^\omega(\mathbf{x}_n))$ is a loss function which measures the discrepancy between the prediction and the real data. The energy function may be further simplified by assuming a quadratic loss function:

$$\ell(\mathbf{y}_n, f^\omega(\mathbf{x}_n)) = \frac{\tau}{2} \|\mathbf{y}_n - f^\omega(\mathbf{x}_n)\|_2^2 \quad (32)$$

and a Gaussian likelihood

$$\mathbf{y} \sim \mathcal{N}(\mathbf{y}_n; f^\omega(\mathbf{x}_n), \tau^{-1} \mathbf{I}). \quad (33)$$

If this function is sampled with a Monte Carlo technique, one finally obtains:

$$\mathcal{L}_\alpha^{\text{MC}}(q) = -\frac{1}{\alpha} \sum_n \ln \frac{1}{K} \sum_k \exp\left[-\frac{\alpha\tau}{2} \|\mathbf{y}_n - f^{\tilde{\omega}_k}(\mathbf{x}_n)\|_2^2\right] + \frac{ND}{2} \log \tau + \sum_i p_i \|\mathbf{M}_i\|_2^2, \quad (34)$$

$$\tilde{\omega}_k \sim q(\omega)$$

where the $\{\tilde{\omega}_k\}$ are sampled from the variational distribution $q(\omega)$, and the $\{\mathbf{M}_i\}$ are the dropout matrices as defined in Eq. (23). The financial trajectories are also needed to evaluate the path integral; these are generated using a t-GAN.

3. Generating financial trajectories with a temporal GAN

In order to evaluate the path integral, multiple financial trajectories must be generated. Instead of assuming an ad hoc model, we propose to learn this model, and to generate these trajectories with a temporal generative adversarial network (Goodfellow et al., 2014; Zhao et al., 2016). Indeed, as the evaluation of the path integral with a Monte Carlo technique entails sampling the underlying data distribution, we propose to learn and sample this distribution with a GAN. As the generated trajectories will be more realistic, the Monte Carlo calculation is more likely to converge rapidly with high accuracy. In its original form, generative neural networks consists of a generator and a discriminator. The generator creates financial trajectories from a uniform random function, while the discriminator ensures that these trajectories are as realistic as possible. In order to meet these requirements, the energy function was originally defined (Goodfellow et al., 2014) as

$$\mathcal{L}_G = \min_{\theta_G} \max_{\theta_D} E_{\mathbf{x} \sim p_D} [\ln D_{\theta_D}(\mathbf{x})] + E_{\mathbf{z} \sim \mathcal{U}} [\ln(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))] \quad (35)$$

$$\mathbf{z} \sim \mathcal{U}$$

where the discriminator $D_{\theta_D}(\cdot)$ is a deep neural network with parameters θ_D , $G_{\theta_G}(\cdot)$ is the generator, a deep neural network with parameters θ_G , and $\mathbf{z}$ is a random vector sampled from a uniform distribution $\mathcal{U}$. The two networks may be assimilated to players in a minimax game (Du & Pardalos, 2013; Wang et al., 2017). These networks are notoriously difficult to train, and tend to be unstable (Thanh-Tung & Tran, 2020). Therefore, the Wasserstein generative adversarial network (Arjovsky et al., 2017) has been proposed as an alternative. During the learning process, this network minimises an approximation of the earth mover's distance (Weng, 2019) based on the Kantorovich–Rubinstein duality (Adler & Lunz, 2018; Deshpande et al., 2018). These networks tend to be easier to train, and more stable, while avoiding mode dropping (Bau et al., 2019). The discriminator of the Wasserstein generative adversarial network must satisfy a K -Lipschitz constraint (Arjovsky et al., 2017; Liu, 2018):

$$|D(\mathbf{x}_i) - D(\mathbf{x}_j)| \leq K \|\mathbf{x}_i - \mathbf{x}_j\|, \forall \mathbf{x}_i, \mathbf{x}_j \quad (36)$$

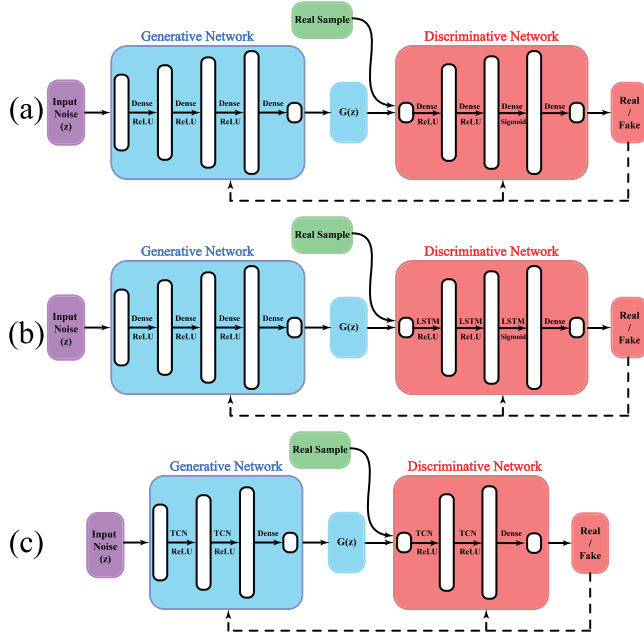


Fig. 2. GAN architectures: (a) Dense network–dense network, (b) Dense network–LSTM, (c) TCN–TCN.

where K is a constant. The Wasserstein loss (cost or objective) function (Dukler et al., 2019; Frogner et al., 2015), which approximates the earth mover's distance, has the form:

$$\mathcal{L}_W = \min_{\theta_G} \max_{\theta_D} E_{\mathbf{x} \sim p_D} [D(\mathbf{x})] + E_{\mathbf{z} \sim p_G} [D(G(\mathbf{z}))], \quad (37)$$

$$\mathbf{z} \sim \mathcal{U}$$

If K is set to one, the K -Lipschitz condition may be enforced by clipping the weight matrices' singular values to one (Saito et al., 2017):

$$\mathbf{W}_i = \mathbf{U}_i \Sigma_i \mathbf{V}_i^\dagger, \quad (38)$$

$$\Sigma = \text{diag}(\sigma_1, \dots, \sigma_n)$$

$$\sigma_j > 1 \Rightarrow \sigma_j = 1$$

which implies that the spectral norms of the weights matrices are always smaller than, or equal to, one.

Financial trajectories are a particular form of time series. In their original form, generative adversarial networks were designed to handle static data such as images, not for temporally varying entities such as time series. For this reason, we propose to employ a temporal generative adversarial network for generating the financial paths (Kakushadze, 2015; Linetsky, 1997). Let us assume that the price of the securities are generated over the discrete-time interval $[t_1, t_2, \dots, t_T]$, which results in T realisations of the security being considered. The t-GAN consists of two generators. The first generator, named the temporal generator, takes a latent variable $\mathbf{z}_0$ as an argument, and generates a latent-variables time series $[\mathbf{z}_1^1, \dots, \mathbf{z}_1^T]$, in which each temporal latent variable corresponds to a particular realisation of the financial trajectory. The latent variable $\mathbf{z}_0$ is drawn from a uniform distribution $\mathcal{U}$. The second generator, named the financial generator, generates a particular instance of the financial trajectory from the latent variable $\mathbf{z}_0$ and from a particular instance of the temporal latent variable: $[G_1(\mathbf{z}_0, \mathbf{z}_1^1), \dots, G_1(\mathbf{z}_0, \mathbf{z}_1^T)]$.

As a result, the Wasserstein loss function associated with the t-GAN becomes:

$$\mathcal{L}_T = \min_{\theta_{G_0}, \theta_{G_1}} \max_{\theta_D} E_{[\mathbf{x}^1, \dots, \mathbf{x}^T] \sim p_D} [D([\mathbf{x}^1, \dots, \mathbf{x}^T])] + E_{\mathbf{z}_0 \sim p_{G_0}} [D([G_1(\mathbf{z}_0, \mathbf{z}_1^1), \dots, G_1(\mathbf{z}_0, \mathbf{z}_1^T)])], \quad (39)$$

$$\mathbf{z}_0 \sim \mathcal{U},$$

$$G(\mathbf{z}_0) = [\mathbf{z}_1^1, \dots, \mathbf{z}_1^T]$$

It should be noted that, while $\mathbf{z}_1$ varies in time, $\mathbf{z}_0$ is stationary in order to ensure consistency in between generated instances. As mentioned earlier, the K -Lipschitz condition is enforced with singular value clipping (Saito et al., 2017). In the next section, three architectures are proposed for the t-GAN.

4. GAN architectures

Three architectures are proposed for the t-GAN, illustrated in Fig. 2. The first architecture consists of two dense networks: one for the generator and one for the discriminator. The second architecture involves a dense network and a LSTM. The last architecture combines two temporal convolutional networks (TCN).

As for the Dense–Dense GAN, both the generator and the discriminator consist of four fully connected layers having 5, 10, 20, and 1 neurons respectively. The generator creates financial trajectories from a uniform random function, while the discriminator ensures that these trajectories are as realistic as possible. The discriminator is trained with a binary cross-entropy loss function, which aims to distinguish between fake and real samples.

Algorithm 1 Dense–Dense GAN

Input: financial time series, uniform distribution.

Output: Predicted trajectory for financial instrument

Initialization:

while $i \leq n$ **do**

• Generative Network

1. **Dense layer**
{units = 5, act-fcn = ReLU}
2. **Dense layer**
{units = 10, act-fcn = ReLU}
3. **Dense layer**
{units = 20, act-fcn = ReLU}
4. **Dense layer**
{units = 1 }

• Discriminative Network

1. **Dense layer**
{units = 5, act-fcn = ReLU}
2. **Dense layer**
{units = 10, act-fcn = ReLU}
3. **Dense layer**
{units = 20, act-fcn = ReLU}
4. **Dense layer**
{units = 1, act-fcn = Sigmoid}
5. **Model**
{loss = binary crossentropy, optimizer = Adam}

In the case of the Dense–LSTM architecture, the GAN consists of a generator (comprising four fully connected layers consisting of 10, 20, 40 and 1 neurons respectively) and a discriminator (comprising three LSTM layers having 10, 40 and 80 neurons and of an output neuron). As for the Dense–LSTM architecture, the discriminator distinguishes between fake and real samples. The LSTM layers are capable of capturing sequential patterns in financial time series, thus forcing the generator to create realistic fake samples.

Algorithm 2 Dense-LSTM GAN**Input:** financial time series, uniform distribution.**Output:** Predicted trajectory for financial instrument**Initialization:****while** $i \leq n$ **do**• **Generative Network**

1. **Dense layer**
{units = 10, act-fcn = ReLU}
2. **Dense layer**
{units = 20, act-fcn = ReLU}
3. **Dense layer**
{units = 40, act-fcn = ReLU}
4. **Dense layer**
{units = 1}

• **Discriminative Network**

1. **Dense layer**
{units = 10, act-fcn = ReLU}
2. **Dense layer**
{units = 40, act-fcn = ReLU}
3. **Dense layer**
{units = 80, act-fcn = ReLU}
4. **Dense layer**
{units = 1, act-fcn = Sigmoid}
5. **Model**
{loss = binary crossentropy, optimizer = Adam}

In this architecture, the generator has two TCN layers, consisting of 10 neurons each, and an output neuron. In contrast, the discriminator's two TCN layers consist of 6 neurons each, followed by an output neuron. Both networks employ causal padding to prevent information leakage. As for LSTM, TCNs learn sequential patterns and long-term behaviours. When compared to LSTMs, TCNs tend to have a greater long-term memory. In our study, they outperform the other networks for long-term predictions (20 and 30 days).

Algorithm 3 TCN-TCN GAN**Input:** financial time series, uniform distribution.**Output:** Predicted trajectory for financial instrument**Initialization:****while** $i \leq n$ **do**• **Generative Network**

1. **TCN layer**
{units = 10, act-fcn = ReLU, padding = causal}
2. **TCN layer**
{units = 10, act-fcn = ReLU, padding = causal}
3. **Dense layer**
{units = 1}

• **Discriminative Network**

1. **TCN layer**
{units = 6, act-fcn = ReLU, padding = causal}
2. **TCN layer**
{units = 6, act-fcn = ReLU, padding = causal}
3. **Dense layer**
{units = 1, act-fcn = Sigmoid}
4. **Model**
{loss = binary crossentropy, optimizer = Adam}

In a time series, such as a financial trajectory, time flows from the past to the future. This causal directionality, which is of fundamental importance, is inherently absent from a standard convolutional neural

network, so that the future may influence the past! This problem may be circumvented by employing a TCN (Wan et al., 2019). Our TCN, which is inspired by Bai et al. (2018), consists of two dilated causal convolution layers as described in Fig. 3. Combining residual and dilated convolution layers provides the ability to make predictions from both recent and past events (long-term memory) while enforcing causality (Bai et al., 2018). The dilated convolution operator $F(s)$ over a time series s is defined as (Bai et al., 2018)

$$F(s) = (\mathbf{x}_s \cdot f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \mathbf{x}_{s-i \cdot d} \quad (40)$$

where d is the dilation factor, k is the filter size, and $s - i \cdot d$ explains the direction of the past. The dilation factor determines the scale of the convolution stride. For instance, $d = 1$ corresponds to a standard neural network. In order to avoid a vanishing or an explosion of the gradient, the TCN employs a residual module (Bai et al., 2018), this latter being illustrated in Fig. 3(b). Essentially, the residual module consists of a dilated causal convolution, followed by weight normalisation and a non-linear activation function (ReLU).

The whole procedure for price predictions is summarised in Algorithm 4. Stocks are first selected while their prices, over a given time period, are retrieved. For each stock, the time series is partitioned into a training set and a test set; the test set consists of either the last twenty or last thirty days of the time series. A set of 1024 randomly generated Bayesian networks, as described in Section 2, are trained with Eq. (34). The best network is selected to predict the drift and volatility associated with the prices. From the drifts and the volatilities, the Lagrangian is formed according to Eq. (14), the probability associated with each point of each financial path being a function of the Lagrangian. The predictions, which correspond to the expectation values, are obtained from Eq. (15). The path integral is evaluated according to Monte Carlo techniques as defined in Eq. (16): as the various paths are generated with a t-GAN, the probability distribution associated with the stocks is sampled directly by the GAN.

5. Experimental results

Our dataset consisted of nine stocks, eight of which are included in the S&P 500 index, namely Apple (AAPL), Advanced Micro Devices (AMD), Amazon (AMZN), Alphabet, better known as Google (GOOG), Microsoft (MSFT), Nvidia (NVDA), Pfizer (PFE), Shopify (SHOP), and Walmart (WMT). These stocks are drawn from various sectors, such as high-technology, retail, e-commerce, and pharmaceutical, among others. Therefore, the proposed approach may be evaluated against diverse financial ecosystems. The only parameters are the start and end dates which were set to 1 January 2019 and 31 December 2020 respectively. Therefore, the time series contain data from both before and during the Covid-19 pandemic. For each tick, the data consist of open, close, high and low prices. The low, high and closing prices are normalised with respect to the opening price (Soleymani & Paquet, 2020). Our aim is to predict stock prices over periods of twenty days and thirty days. For the twenty-day prediction period, the training set consisted of the first 479 trading days, while the last twenty days' records were assigned to the test set. For the thirty-day prediction period, the training dataset consisted of the first 469 trading days, and the last thirty days' records constituted the test set. The chosen trading period was during the Covid-19 crisis, which makes the prediction task more challenging due to the economical and financial instability induced by the pandemic. The financial time series were retrieved directly with Mathematica with the FinancialData function.

The calculations were performed as follows: a total of 1024 dense Bayesian neural networks, as described in Section 2, were generated randomly. Each of these networks consisted of 1 to 5 layers with 10 to 200 neurons per layer, 3 choices for the activation function (ramp, SELU, and tanh), and either a homoscedastic (all the random variables have the same variance) or heteroscedastic (the random variables do

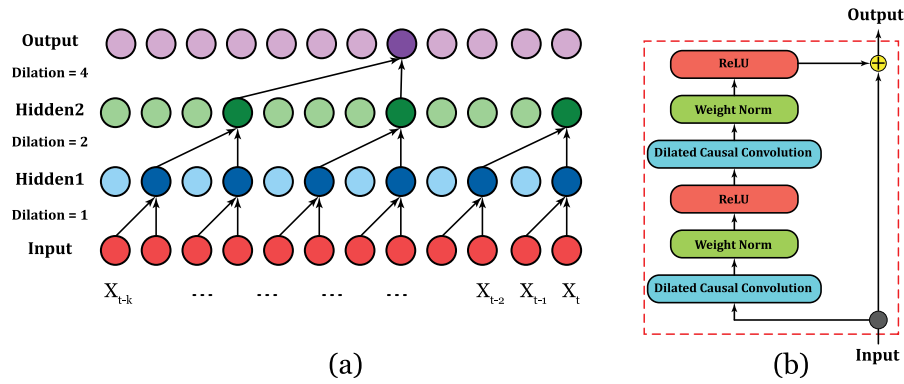


Fig. 3. (a) Illustration of a temporal convolution with dilation factors $d = 1, 2, 4$ and a filter size of $k = 2$ (b) Architecture of the residual block associated with the temporal convolutional network.

Algorithm 4 QuantumPath

Input: Prediction period T

Output: predicted financial trajectory

Data: m : number of neurons per layer (BNN): 10–200

Data: l : number of hidden layers (BNN): 1–5

Data: n : number of randomly generated Bayesian network architectures: 1024

Data: err_mod : error model (BNN): homoscedastic or heteroscedastic

Data: act_fcn : activation function (BNN): ramp, SELU and Tanh

1. Prediction of drifts and volatilities with a deep Bayesian network:
 - (a) Generate randomly n BNN architectures by varying the number of:
 - Neurons per layer ($1-m$);
 - The number of hidden layers ($1-l$);
 - The activation functions (act_fcn);
 - The error model (err_mod)
 - (b) Train each generated network with Adam, with the loss function described by Eq. 34.
 - (c) Select the most accurate network.
 - (d) Employ the network to predict the drifts and volatilities.
 2. Prediction of the financial trajectory:
 - (a) Train the t-GAN with Adam, employing the loss function described by Eq. 39.
 - (b) Generate financial trajectories with the t-GAN (sampling of the data distribution).
 - (c) Evaluate the expected predicted financial trajectory, with a Feynman–Dirac path integral (Eq. 14–16), with Monte Carlo techniques (Eq. 16–17), from the financial trajectories generated by the t-GAN, and the drifts and volatilities predicted by the BNN.
-

not share the same variance) error model. The models were trained in parallel on an HP Apollo System with 40 Dual Intel Xeon Gold 6149 processors, one Nvidia Tesla V100 (32 GB) and 968 GB of memory; one model for each stock. For each particular stock, the best model was selected for predicting the drifts and volatilities over the test periods, that is, the last twenty and thirty days, respectively.

The deep Bayesian network was implemented with Mathematica while the GANs were implemented with TensorFlow 2.2 and Python 3.7. The code for the GANs was executed on a HP Z6 workstation with two Intel Xeon Silver 4114 CPUs, 176 GB of RAM and a NVIDIA GeForce RTX 3090 GPU with 24 GB of RAM. The total execution time was 0.2 h including both training and predictions.

The drifts and the volatilities were substituted into the Lagrangian in order to evaluate the occurrence probability for each day of each financial trajectory with the help of Eq. (14). Afterwards, all three GANs were trained with the training set using the Adam algorithm (Kingma & Ba, 2015). Once trained, they were employed to generate one thousand financial trajectories or paths for the evaluation of the path integral. The mathematical expectations of the stock prices over the test period, which are inherently the predictions made by our QuantumPath framework, were evaluated with the help of Eqs. (14) and (15). The relative errors for the predictions over a period of twenty

and thirty days, for all three GAN architectures, are reported in Figs. 4 and 5 respectively. These results demonstrate the efficiency of our QuantumPath framework.

The maximum relative errors, for all three GAN architectures, are reported in Fig. 6 for prediction periods of both 20 and 30 days. From Figs. 4–6, it may be concluded that the TCN–TCN GAN slightly outperforms the others, while the dense network–LSTM GAN presents the lowest performances. These results point to the importance of the long-term memory for price predictions.

The proposed method has been evaluated in low and high volatility conditions. Indeed, because of the start and end dates, 1 January 2019 and 31 December 2020 respectively, the networks were trained both with pre-pandemic and pandemic time series while the predictions were made during a volatile and uncertain period strongly affected by exogenous factors such as Covid-19 (Cervelló-Royo & Guijarro, 2020; He et al., 2020; Nti et al., 2020; Soleymani & Paquet, 2021; Zhang et al., 2020). In order to determine the impact of the time horizon on the accuracy, the RMSV was evaluated for each stock over periods of 5, 10, 15, and 20 days. The results are reported in Tables 1–3. The tables show that the RMSV remains relatively stable, or at least increases only slowly, as the investment horizon increases.

In order to compare our approach to classical ones, the financial predictions have also been performed with an autoregressive integrated

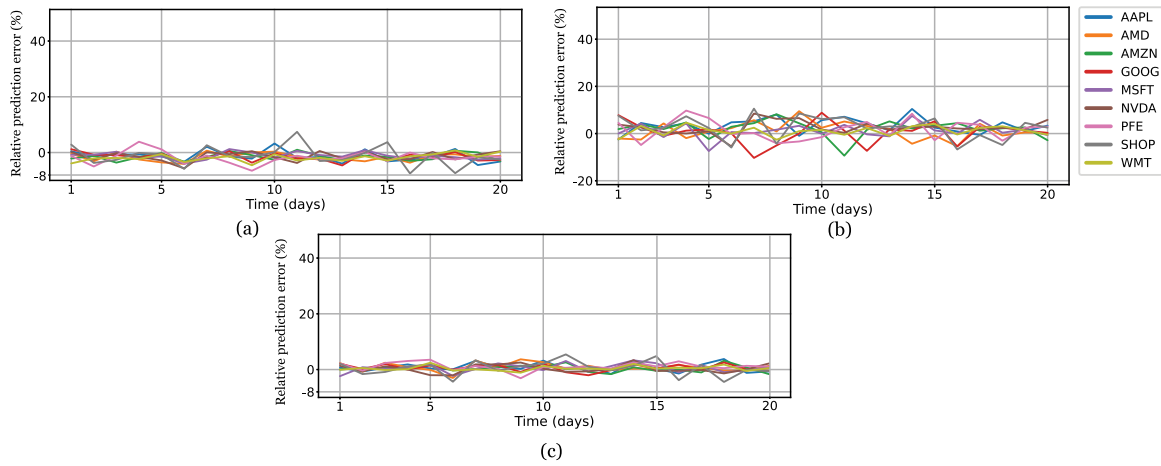


Fig. 4. Relative errors for price prediction for all nine stocks over a period of 20 days: (a) Dense network-dense network GAN, (b) Dense network-LSTM GAN, (c) TCN-TCN GAN.

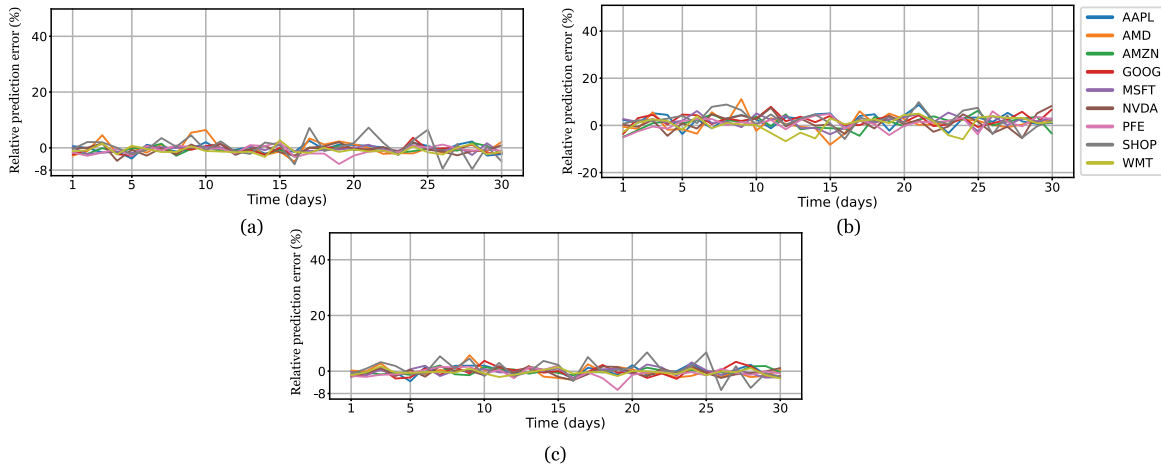


Fig. 5. Relative errors for price prediction for all nine stocks over a period of 30 days: (a) Dense network-dense network GAN, (b) Dense network-LSTM GAN, (c) TCN-TCN GAN.

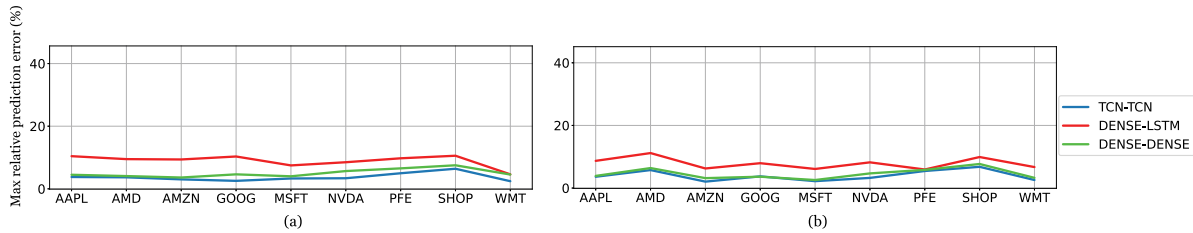


Fig. 6. Maximum relative error for price predictions for the three GAN architectures: (a) 20day period, (b) 30-day period.

Table 1

RMSV on 5, 10, 15, 20 and 30-day predictions for the Dense-Dense GAN architecture.

Stocks/Days	5	10	15	20	30
AAPL	0.025	0.035	0.035	0.040	0.308
AMD	0.026	0.021	0.024	0.032	0.269
AMZN	0.023	0.029	0.025	0.030	0.310
GOOG	0.017	0.017	0.017	0.029	0.314
MSFT	0.012	0.019	0.025	0.032	0.344
NVDA	0.011	0.035	0.041	0.035	0.364
PFE	0.011	0.022	0.037	0.042	0.340
SHOP	0.043	0.074	0.079	0.081	0.385
WMT	0.009	0.019	0.021	0.048	0.291

Table 2

RMSV on 5, 10, 15, 20 and 30-day predictions for the Dense-LSTM GAN architecture.

Stocks/Days	5	10	15	20	30
AAPL	0.094	0.060	0.063	0.148	0.081
AMD	0.076	0.075	0.081	0.080	0.139
AMZN	0.094	0.045	0.133	0.072	0.091
GOOG	0.083	0.069	0.088	0.076	0.128
MSFT	0.150	0.055	0.078	0.079	0.077
NVDA	0.075	0.057	0.097	0.096	0.096
PFE	0.140	0.059	0.132	0.062	0.091
SHOP	0.125	0.092	0.102	0.151	0.100
WMT	0.065	0.060	0.118	0.088	0.114

Table 3

RMSV on 5, 10, 15, 20 and 30-day predictions for the TCN-TCN GAN architecture.

Stocks/Days	5	10	15	20	30
AAPL	0.022	0.032	0.028	0.031	0.047
AMD	0.020	0.027	0.030	0.052	0.054
AMZN	0.021	0.025	0.026	0.025	0.041
GOOG	0.015	0.024	0.020	0.027	0.030
MSFT	0.011	0.018	0.026	0.024	0.033
NVDA	0.016	0.042	0.023	0.038	0.044
PFE	0.016	0.018	0.033	0.055	0.051
SHOP	0.038	0.075	0.068	0.070	0.096
WMT	0.009	0.028	0.017	0.021	0.045

Table 4

RMSV on 30-day predictions for GARCH, ARIMA, OU, and QuantumPath.

Stocks/Algorithm	QuantumPath	GARCH	ARIMA	OU
AAPL	0.308	23.008	1.016	2.070
AMD	0.269	16.920	1.146	1.617
AMZN	0.310	586.176	10.887	35.085
GOOGL	0.314	325.065	7.836	19.530
MSFT	0.344	39.834	0.846	1.972
NVDA	0.364	24.417	1.203	1.360
PFE	0.340	7.113	0.603	0.829
SHOP	0.385	203.662	31.288	31.209
WMT	0.291	27.182	1.392	0.967

moving average (ARIMA) model (Li et al., 2020), a generalised autoregressive conditional heteroscedasticity (GARCH) model (Li et al., 2020), and a Markov chain Monte Carlo (MCMC) Ornstein–Uhlenbeck (OU) stochastic process (Donado et al., 2017). All these methods are commonly employed in time series financial predictions. The parameters for ARIMA and GARCH were evaluated with the method of moments, while the parameters for the OU process were evaluated with nested sampling (Sivia & Skilling, 2006), a state-of-the-art Bayesian inference technique. Their predictions, as well as those obtained with the proposed quantum-hybrid approach, were compared over a period of thirty trading days with the root mean square value (RMSV) metric. The results are reported in Table 4. QuantumPath outperforms OU, ARIMA and GARCH.

6. Summary and conclusions

This paper proposed a new framework, called QuantumPath, for long-term financial predictions. The expected financial trajectory is predicted with a Feynman–Dirac path integral which results from the stochastic equation associated with the financial instruments. The path integral parameters – the drifts and the volatilities at all times – are predicted with a deep Bayesian network. The Monte Carlo evaluation of the path integral, which involved sampling the data distribution, is performed with a t-GAN which incorporates mechanisms for short and long-term memory as well as causality. Prices for nine stocks, including eight stocks from the S&P 500, were predicted over periods of twenty and thirty days with high precision.

Typically, the temporal evolution of a financial instrument is described with stochastic equations (Linetsky, 1997). In this paper, it has been demonstrated that such an equation may be formulated in terms of a Feynman–Dirac path integral. Quantum computing is not, therefore, introduced *ad hoc*, but is rather the result of the stochastic nature of the stocks: a data-centric approach. These path integrals make it possible to predict the expected value of a financial instrument at a later time. The Monte Carlo evaluation of the path integrals involves sampling the data distribution. In order to optimise the sampling and the precision, and to minimise the computational complexity and the number of trajectories generated, the trajectories are generated with a t-GAN, which means that only the most probable paths are considered. Therefore, the sampling mechanism is learned and not assumed, while being optimal in the Wasserstein sense. The same may be said about

the parameters which are learned and predicted with a deep Bayesian neural network rather than being fixed by inspection. In addition to financial predictions, this QuantumPath framework may be applied to risk mitigation, portfolio management, and crisis management. Risk mitigation and portfolio management are closely related. Indeed, our approach allows us to predict stock prices over a given period of time. By knowing the behaviour of the stocks forming a portfolio, it becomes possible to reallocate the assets in order to maximise the return on investment, thus mitigating the risk. As for crisis management, a sudden decline of stock prices may be an indicator of an emerging crisis (Liu et al., 2021; Samimi & Samimi, 2021). By knowing in advance, it is possible to plan a better management strategy. We plan to apply this approach to other financial instruments such as options and forward and future contracts, as well as for forecasting the evolution of the pandemic from the financial market.

CRedit authorship contribution statement

Farzan Soleymani: Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing. **Eric Paquet:** Conception and design of study, Acquisition of data, Analysis and/or interpretation of data, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

All authors approved the version of the manuscript to be published.

References

- Adler, J., & Lunz, S. (2018). Banach wasserstein gan. In *Advances in neural information processing systems* (pp. 6754–6763).
- Aizenman, M., Sims, R., & Warzel, S. (2006). Stability of the absolutely continuous spectrum of random Schrödinger operators on tree graphs. *Probability Theory and Related Fields*, 136, 363–394.
- Amin, A. A. (2019). Kullback-leibler divergence to evaluate posterior sensitivity to different priors for autoregressive time series models. *Communications in Statistics. Simulation and Computation*, 48, 1277–1291.
- Andreev, A. (2006). Schrödinger equation. In *Atomic spectroscopy: introduction to the theory of hyperfine structure* (pp. 13–42).
- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. arXiv preprint arXiv:1701.07875.
- Ashraf, B. N. (2020). Stock markets' reaction to covid-19: cases or fatalities? *Research in International Business and Finance*, Article 101249.
- Ba, J., & Frey, B. (2013). Adaptive dropout for training deep neural networks. *Advances in Neural Information Processing Systems*, 26, 3084–3092.
- Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271.
- Bau, D., Zhu, J.-Y., Wulff, J., Peebles, W., Strobelt, H., Zhou, B., & Torralba, A. (2019). Seeing what a gan cannot generate. In *Proceedings of the IEEE international conference on computer vision* (pp. 4502–4511).
- Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112, 859–877.
- Cai, L., Chen, Y., Cai, N., Cheng, W., & Wang, H. (2020). Utilizing amari-alpha divergence to stabilize the training of generative adversarial networks. *Entropy*, 22(410).
- Cervelló-Royo, R., & Guijarro, F. (2020). Forecasting stock market trend: A comparison of machine learning algorithms. *Finance, Markets and Valuation*, 6, 37–49.
- Chen, M.-H., & Shao, Q.-M. (1997). Performance study of marginal posterior density estimation via kullback-leibler divergence. *Test*, 6, 321–350.
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.
- David, J. M. (1992). A practical bayesian framework for backprop networks. *Neural Computation*.

- Deshpande, I., Zhang, Z., & Schwing, A. G. (2018). Generative modeling using the sliced wasserstein distance. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3483–3491).
- Dirac, P. A. (2005). The lagrangian in quantum mechanics. In *Feynman's thesis—a new approach to quantum theory* (pp. 111–119). World Scientific.
- Donado, F., Moctezuma, R., López-Flores, L., Medina-Noyola, M., & Arauz-Lara, J. (2017). Brownian motion in non-equilibrium systems and the ornstein-uhlenbeck stochastic process. *Scientific Reports*, 7, 1–7.
- Du, D.-Z., & Pardalos, P. M. (2013). *Minimax and applications*, Vol. 4. Springer Science & Business Media.
- Dukler, Y., Li, W., Lin, A., & Montufar, G. (2019). Wasserstein of wasserstein loss for learning generative models. In *International conference on machine learning* (pp. 1716–1725). PMLR.
- Esteve, J. (2002). Origin of the anomalies: The modified heisenberg equation. *Physical Review D*, 66, Article 125013.
- Frogner, C., Zhang, C., Mobahi, H., Araya, M., & Poggio, T. A. (2015). Learning with a wasserstein loss. *Advances in Neural Information Processing Systems*, 28, 2053–2061.
- Goldin, G. (2008). Nonlinear quantum mechanics: Results and open questions. *Physics of Atomic Nuclei*, 71(884).
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).
- Habek, M. (2012). Evaluation of marginal likelihoods via the density of states. In *Artificial intelligence and statistics* (pp. 486–494).
- He, P., Sun, Y., Zhang, Y., & Li, T. (2020). Covid-19's impact on stock prices across different sectors—an event study based on the chinese stock market. *Emerging Markets Finance and Trade*, 56, 2198–2212.
- He, H., Wang, H., Lee, G.-H., & Tian, Y. (2019). Probgan: Towards probabilistic gan with theoretical guarantees. In *ICLR (poster)*.
- Hernandez-Lobato, J., Li, Y., Rowland, M., Bui, T., Hernández-Lobato, D., & Turner, R. (2016). Black-box alpha divergence minimization. In *International conference on machine learning* (pp. 1511–1520). PMLR.
- Hurtado, J., & Barbat, A. H. (1998). Monte carlo techniques in computational stochastic mechanics. *Archives of Computational Methods in Engineering*, 5, 3–29.
- Ingold, G.-L. (2002). Path integrals and their application to dissipative quantum systems. In *Coherent evolution in noisy environments* (pp. 1–53). Springer.
- Jospin, L. V., Buntine, W., Boussaid, F., Laga, H., & Bennamoun, M. (2020). Hands-on bayesian neural networks—a tutorial for deep learning users. *arXiv preprint arXiv:2007.06823*.
- Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International conference on machine learning* (pp. 2342–2350). PMLR.
- Kakushadze, Z. (2015). Path integral and asset pricing. *Quantitative Finance*, 15, 1759–1771.
- Kingma, D., & Ba, J. (2015). Adam: a method for stochastic optimization 2014. (p. 15). *arXiv preprint arXiv:1412.6980*.
- Kononenko, I. (1989). Bayesian neural networks. *Biological Cybernetics*, 61, 361–370.
- Lea, C., Vidal, R., Reiter, A., & Hager, G. D. (2016). Temporal convolutional networks: A unified approach to action segmentation. In *European conference on computer vision* (pp. 47–54). Springer.
- Li, T., Zhong, J., & Huang, Z. (2020). Potential dependence of financial cycles between emerging and developed countries: Based on arima-garch copula model. *Emerging Markets Finance and Trade*, 56, 1237–1250.
- Linetsky, V. (1997). The path integral approach to financial modeling and options pricing. *Computational Economics*, 11, 129–163.
- Liu, K. (2018). Varying k-lipschitz constraint for generative adversarial networks. *ArXiv*, arXiv:1803.
- Liu, Y., Wei, Y., Wang, Q., & Liu, Y. (2021). International stock market risk contagion during the covid-19 pandemic. *Finance Research Letters*, Article 102145.
- Lucic, M., Kurach, K., Michalski, M., Gelly, S., & Bousquet, O. (2018). Are gans created equal? a large-scale study. *Advances in Neural Information Processing Systems*, 31, 700–709.
- Masujima, M. (2008). *Path integral quantization and stochastic quantization*. Springer Science & Business Media.
- Mullachery, V., Khera, A., & Husain, A. (2018). Bayesian neural networks. *arXiv preprint arXiv:1801.07710*.
- Nicola, M., Alsafi, Z., Sohrabi, C., Kerwan, A., Al-Jabir, A., Iosifidis, C., Agha, M., & Agha, R. (2020). The socio-economic implications of the coronavirus pandemic (covid-19): A review. *International Journal of Surgery (London, England)*, 78(185).
- Nti, I. K., Adekoya, A. F., & Weyori, B. A. (2020). A systematic review of fundamental and technical analysis of stock market predictions. *Artificial Intelligence Review*, 53, 3007–3057.
- Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.
- Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2013). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.
- Perepelitsa, D. V. (2018). Path integrals in quantum mechanics. MIT Department of Physics.
- Saatci, Y., & Wilson, A. G. (2017). Bayesian gan. *Advances in Neural Information Processing Systems*, 30, 3622–3631.
- Saito, M., Matsumoto, E., & Saito, S. (2017). Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE international conference on computer vision* (pp. 2830–2839).
- Samimi, A., & Samimi, M. (2021). Investigating the strategic concept of risk management in the stock market and investing. *Journal of Engineering in Industrial Research*, 2.
- Santana, S. R., & Hernández-Lobato, D. (2020). Adversarial α -divergence minimization for bayesian approximate inference. *Neurocomputing*.
- Sivia, D., & Skilling, J. (2006). *Data analysis: A bayesian tutorial*. USA: Oxford University Press.
- Soleymani, F., & Paquet, E. (2020). Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder-deepbreath. *Expert Systems with Applications*, Article 113456.
- Soleymani, Farzan, & Paquet, Eric (2021). Deep graph convolutional reinforcement learning for financial portfolio management-deeppocket. *Expert Systems with Applications*, 115127. <http://dx.doi.org/10.1016/j.eswa.2021.115127>.
- Springenberg, J. T., Klein, A., Falkner, S., & Hutter, F. (2016). Bayesian optimization with robust bayesian neural networks. *Advances in Neural Information Processing Systems*, 29, 4134–4142.
- Strukov, L., & Timan, A. F. (1977). Mathematical expectation of continuous functions of random variables, smoothness and variance. *Siberian Mathematical Journal*, 18, 469–474.
- Thanh-Tung, H., & Tran, T. (2020). Catastrophic forgetting and mode collapse in gans. In *2020 international joint conference on neural networks (ijcnn)* (pp. 1–10). IEEE.
- Truman, A. (1978). Some applications of vector space measures to non-relativistic quantum mechanics. In *Vector space measures and applications i* (pp. 418–441). Springer.
- Tulczyjew, W. M. (1977). The legendre transformation. In *Annales de l'ihp physique théorique*, Vol. 27 (pp. 101–114).
- Ulusoy, I., & Bishop, C. M. (2006). Comparison of generative and discriminative techniques for object detection and classification. In *Toward category-level object recognition* (pp. 173–195). Springer.
- Wan, R., Mei, S., Wang, J., Liu, M., & Yang, F. (2019). Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting. *Electronics*, 8(876).
- Wang, J., Yu, L., Zhang, W., Gong, Y., Xu, Y., Wang, B., Zhang, P., & Zhang, D. (2017).
- Weng, L. (2019). From gan to wgan. *arXiv preprint arXiv:1904.08994*.
- Wu'trich, A. (2010). *The genesis of feynman diagrams* (pp. 51–64). Springer.
- Zhang, C., Bu'tepage, J., Kjellström, H., & Mandt, S. (2018). Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41, 2008–2026.
- Zhang, D., Hu, M., & Ji, Q. (2020). Financial markets under the global pandemic of covid-19. *Finance Research Letters*, Article 101528.
- Zhang, K., Zhong, G., Dong, J., Wang, S., & Wang, Y. (2019). Stock market prediction based on generative adversarial network. *Procedia Computer Science*, 147, 400–406.
- Zhao, J., Mathieu, M., & LeCun, Y. (2016). Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*.
- Zhou, X., Pan, Z., Hu, G., Tang, S., & Zhao, C. (2018). Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018.
- Ziemba, W. T., Zhitlukhin, M., & Lleo, S. (2017). *Stock market crashes: Predictable and unpredictable and what to do about them*, Vol. 13. World Scientific.

Chapter 6

Appendix-B

In this chapter, the area coverage control problem using multi-agent systems is discussed. The area coverage control problem may be formulated in a time-invariant or time-varying environment. The evolution of the environment is caused by the presence of mobile external objects (risk densities). This problem is formulated by defining a coverage metric that encodes the evolving environment with respect to risk distribution and sensing performances of the agents. The area coverage control aims to find an agent's spatial distribution that locally maximises the coverage metric. Initially, a state-feedback controller method is proposed to address area coverage control over an evolving environment [87]. To further improve this work, the coverage control problem is formulated using Bellman's principle of optimality, which led to designing an adaptive control policy using a reinforcement learning method called actor-critic. The performance index is defined as an additive contribution of a quadratic term in the control effort and of a positive definite term that depends on the coverage metric. The critic network approximates the performance index, and the actor network approximates the optimal policy to obtain maximum coverage over the environment. Initially, this work is implemented in a time-invariant environment (see Section 6.2). This approach is further developed for a time-varying environment by collecting sensing data along trajectories. Additionally, a feedforward network is proposed to work alongside the actor network (actor-feedforward-critic network) to track the Voronoi centroids. This network helps reduce the value function compared to the actor-critic architecture and leads to zero error convergence (see Section 6.3).

6.1 Non-autonomous State–Feedback to Stabilise the Error Dynamics in Time-Varying Area Coverage Control Problems

In this article, a non-autonomous state-feedback controller is proposed to asymptotically drive the agents to their respective time-varying Voronoi centroids, therefore positioning them in the optimal configuration with respect to the performance measured by the coverage metric. This paper is accepted and presented at the 2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE) conference.

Non-autonomous State–Feedback to Stabilize the Error Dynamics in Time-Varying Area Coverage Control Problems

Farzan Soleymani¹, Md Suruz Miah², and Davide Spinello¹

¹*Mechanical Engineering Department, University of Ottawa, Canada,*

²*Electrical and Computer Engineering Department, Bradley University, USA,*

Abstract—We propose a state-feedback control scheme for area coverage problems using a group of agents operating in a time varying environment. The coverage metric defining the optimal control problem encodes a time-varying risk density that models an evolving environment in which the agents operate. The evolution of the environment is caused by the presence of mobile external objects (targets). Maximum coverage can be accomplished by deploying more (less) agents to the part of the area marked with a high (low) risk density, resulting into non-uniform agents' distributions that adapt to the environment. The workspace is partitioned according to a Voronoi tessellation with respect to a distance that quantifies robots' sensing performances. For every initial configuration of the group of agents in the workspace, the proposed non-autonomous state-feedback controller asymptotically drives the agents to time varying centroids of the Voronoi tessellation, therefore positioning them in the optimal configuration with respect to the performance measured by the coverage metric. The proposed control scheme is illustrated by simulations.

Index Terms—Multi-Agent System, Voronoi Tessellation, Non-Autonomous Coverage Control; Time-Varying Risk Density.

I. INTRODUCTION

The use of multi-agent teams in area coverage control has gained prominence in recent years. This is attributed to the fact that multi-agent teams can outperform a single agent for the same task while expending less actuator energy in less time. Multi-agent teams can be used in a variety of area coverage missions including surveillance in hostile environments (i.e. areas contaminated with biological, chemical or even nuclear wastes), environmental monitoring (i.e. air quality monitoring, forest monitoring) and law enforcement missions (i.e. border patrol), etc. With technological advances and developments of relatively inexpensive communication, computation and sensing devices, this topic has received considerable attention over the last two decades. Researchers have proposed various solutions to many interesting sensor network coverage problems based on the work of [1]. Recent contributions and meaningful extensions of the framework formulated in [2] have been recommended in the literature [3], [4], [5]. The work described in this paper is similar to the coverage control problem presented in [6], [7], [8] which uses the geometrical notion of a Voronoi partition to assign the agents to different parts of an environment.

Control of multi-agent systems in area coverage problems is typically posed as an optimal control problem in which the trajectories of the agents maximize an area coverage metric encoding a risk density function that can be thought as a description of the environment/workspace, and the individual sensing performances of the agents. When the environment is static, system's equilibria that locally maximize the coverage metric are found by the well known Lloyd's algorithm [9], in which the search for such equilibria is performed by an iterative procedure that converges the agent's positions to centroids of a Voronoi tessellation. This elegant solution procedure does not apply directly to the case of a time varying environment, in which the risk function evolves due to time varying events that are not causally related to the evolution of the agents. This framework leads to a non-autonomous coverage metric, which required additional attention in order to establish convergence and optimality. The main difference between the autonomous and the non-autonomous case is that equilibria of the former are points in the workspace, whereas equilibria of the latter are trajectories in the workspace.

Nonuniform coverage control problems with time-varying density has been addressed in [6], [10], [11]. Cortes et. al in [10] considered a class of time-varying density functions such that the coverage metric is conserved, and therefore the resulting non-autonomous coverage would be a negative definite Lyapunov function along their proposed time-varying feedback law. Lee et al. in [6] addressed several issues related to coverage control problems with time-varying density functions and showed promising results with a real-time implementation using Khepera III robots operating on the ground. They proposed a motion control algorithm whose trajectories are optimal with respect to non-autonomous coverage metrics under the hypothesis that as the initial condition the agents are overlapped with Voronoi centroids. In general, there is no guarantee of optimality if this hypothesis is violated. Another case of dynamic coverage in a time-varying environment has been studied by Ghaffarkhah et al. in [12], where a mobile robot continuously transmitting the acquired data from the points of interest remote base station while moving along periodic trajectories. A general feedback law for non-autonomous coverage problems has been proposed in [13], [14], by considering a time varying environment respectively due to an evolving risk field associated to moving

targets in the workspace, and an evolving risk due to a linear diffusive process from the boundaries of the workspace. The proposed feedback law does not require hypotheses on the initial conditions, and it generates optimal trajectories with respect to the non-autonomous coverage metric by tracking time varying Voronoi centroids. Optimality is established by invoking Barbalat's lemma, which essentially poses stronger conditions than La Salle's principle by adding constraints on the boundedness of the second time derivative of the Lyapunov function.

In this paper, we consider again the problem of non-autonomous coverage control with multi-agent systems, by considering the error dynamics between agents and Voronoi centroids related to a Voronoi tessellation that evolves based on initial conditions and evolving environment. For the error dynamics, we propose a non-autonomous feedback law that stabilizes the errors to the origin, therefore resulting in the agents tracking time-varying centroids. In [13] we have established that systems' trajectories in which the agents are overlapped to centroids correspond to time-varying minima of the coverage metric, therefore solving the non-autonomous optimization problem. The performance of the proposed feedback law is illustrated in the simulation.

This paper is organized as follows: In section II, the problem setup is represented in the context of coverage metric that assess the effect of in maximizing the coverage over the area of interest. This also includes the formulation of the non-autonomous feedback law and error dynamics. The simulation of the proposed algorithm is illustrated in the section IV. Section V summarizes the work.

II. MODELING AND PROBLEM FORMULATION OF AREA COVERAGE

Suppose $N > 0$ number of agents are deployed in a two-dimensional area. The position of agent i , $1 \leq i \leq N$, in Cartesian coordinates is denoted by $\mathbf{p}^{[i]} = [x^{[i]}, y^{[i]}]^T$ in an area $\Omega \in \mathbb{R}^2$. A first order kinematic model of the agents is

$$\dot{\mathbf{p}}_i = \mathbf{u}_i \quad (1)$$

where $\mathbf{u}_i$ represents the agent's velocity. A possible set up for the class of problems considered in this paper is illustrated in Fig. 2, where the shaded gray area represents the density plot of a non-uniform risk function $\phi(\mathbf{q}, t) > 0$, with $\mathbf{q} \in \Omega$. The area is partitioned into four Voronoi cells associated with four agents located in the workspace.

Higher risk (lighter color in the figure) indicates higher likelihood of events of interests to occur in the corresponding region, and therefore a meaningful area coverage control algorithm tends to clusterize more agents/resources around high risk regions. The non-autonomous coverage metric H as a local optimization function can then be defined by

$$H(\mathbf{p}, \mathcal{V}, t) = \sum_{i=1}^n \int_{\mathcal{V}_i} f(r_i) \phi(\mathbf{q}, t) d\mathbf{q}, \quad (2)$$

where the distance $r_i = \|\mathbf{q} - \mathbf{p}_i\|$, $\forall \mathbf{q} \in \Omega$, $f: \mathbb{R} \rightarrow \mathbb{R}$ is the agent's sensor performance function (in the context of

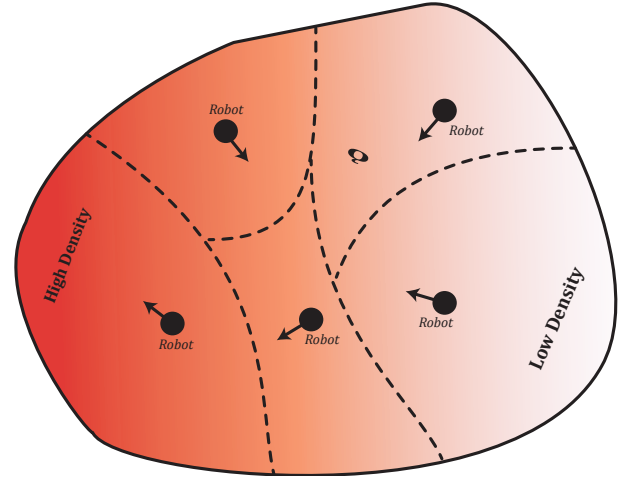


Fig. 1. Schematics of coordinated area coverage control.

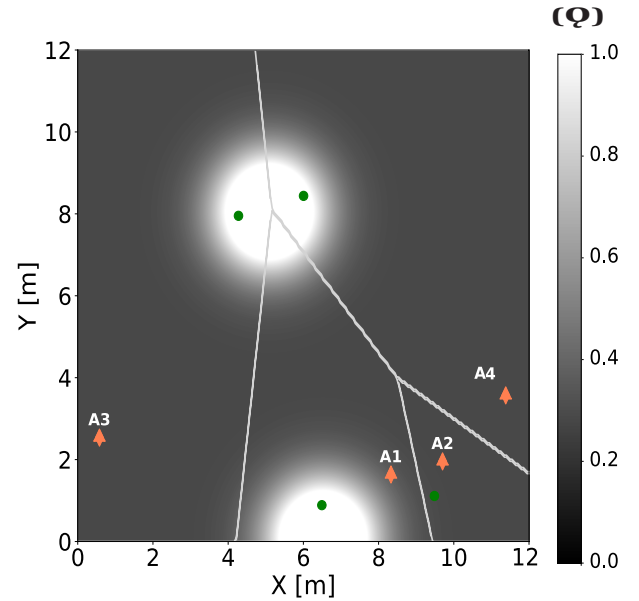


Fig. 2. Initial Partitioning of Environment Using the Voronoi Tessellation Technique

this work, it is chosen as a strictly decreasing function), $\mathcal{V}_i$ (called herein the i th Voronoi cell) is the set of points that are closer to the i th agent than all other agents with respect to the sensing function with respect to the distance f , i.e.,

$$\mathcal{V}_i = \{\mathbf{q} \in \Omega | f(r_i) \leq f(r_j), i \neq j, i \in \mathcal{I}\}.$$

Clearly, $\Omega = \bigcup_{i=1}^n \mathcal{V}_i$. The coverage metric (2) characterizes the performance of the coverage task, which, an increase in the value of H signifies the corresponding spatial configurations of agents achieve better coverage of the area.

Given a time varying risk density $\phi(\mathbf{q}, t) > 0$, for $\mathbf{q} \in \Omega$ and time $t > 0$, we seek to spatially distribute agents such that the coverage H is maximized:

$$\sup_{\mathbf{p}} H(\mathbf{p}, \mathcal{V}, t), \quad \text{subject to (1) as } t \rightarrow \infty \quad (3)$$

where $\mathbf{p} \equiv [\mathbf{p}_1^T, \mathbf{p}_2^T, \dots, \mathbf{p}_n^T]$. The Voronoi partitions are characterized using generalized mass and centroid that defined as follows:

$$m_{\mathcal{V}_i} = \int_{\mathcal{V}_i} \tilde{\phi}(\mathbf{q}, t) d\mathbf{q}, \quad (4a)$$

$$\mathbf{c}_{\mathcal{V}_i} = \frac{1}{\tilde{m}_{\mathcal{V}_i}} \int_{\mathcal{V}_i} \mathbf{q} \tilde{\phi}(\mathbf{q}, t) d\mathbf{q}, \quad (4b)$$

where $\tilde{\phi}(\mathbf{q}, \mathbf{p}_i) = -2\phi(\mathbf{q}, t) (\partial f(r_i)/\partial r_i^2)$.

III. NON-AUTONOMOUS FEEDBACK LAW TO STABILIZE THE ERROR DYNAMICS

A. Non-Autonomous Error Dynamics

The agent's position error is defined as

$$\mathbf{e}_i = \mathbf{c}_{\mathcal{V}_i}(\mathbf{p}_i, t) - \mathbf{p}_i. \quad (5)$$

The centroid is a function of the system's state $\mathbf{p}$ and of time t through the risk density ϕ , see (4b). Thus, the time derivative of the centroid includes a non-autonomous term:

$$\begin{aligned} \dot{\mathbf{e}}_i &= \dot{\mathbf{c}}_{\mathcal{V}_i} - \dot{\mathbf{p}}_i \\ &= \nabla_{\mathbf{p}_i} \mathbf{c}_{\mathcal{V}_i} \dot{\mathbf{p}}_i - \dot{\mathbf{p}}_i + \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \\ &= (\nabla_{\mathbf{p}_i} \mathbf{c}_{\mathcal{V}_i} - \mathbf{I}) \mathbf{u}_i + \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \end{aligned} \quad (6)$$

By introducing the following functions

$$\mathbf{f}(\mathbf{e}_i, t) := \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \quad (7a)$$

$$\mathbf{G}(\mathbf{e}_i, t) := \nabla_{\mathbf{p}_i} \mathbf{c}_{\mathcal{V}_i} - \mathbf{I} \quad (7b)$$

the error dynamics is rewritten as:

$$\dot{\mathbf{e}}_i = \mathbf{f}(\mathbf{e}_i, t) + \mathbf{G}(\mathbf{e}_i, t) \mathbf{u}_i \quad (8)$$

B. Feedback Law

In area coverage problems, the agents' trajectories are typically chosen to locally maximize the coverage metric (2), so that the local equilibria (trajectories in the case of evolving environment) correspond to optimal configuration with respect to the coverage index. For static environments, that is, risk density ϕ stationary, local equilibria correspond to agents' converging to the centroids of a centroidal Voronoi tessellations. This has been generalized in [13] for a time varying environment with prescribed evolution, by proposing a non-autonomous feedback law that converges agents' trajectories to time varying centroids, while guaranteeing the maximization of the coverage metric.

In this work we explore a different approach, by proposing a non-autonomous feedback law that stabilizes the agents' trajectories around time-varying centroids of a Voronoi partition of the environment. The stabilization can be proven by a standard Lyapunov argument. This result is proven in the following Proposition.

Proposition 1. *The error dynamics (8) is globally asymptotically stable along the trajectories generated by the following non-autonomous feedback law*

$$\mathbf{u}_i = -(\nabla_{\mathbf{p}_i} \mathbf{c}_{\mathcal{V}_i} - \mathbf{I})^{-1} \left(\mathbf{K}_i \mathbf{e}_i + \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \right) \quad (9)$$

where $\mathbf{K}_i$ is a 2×2 symmetric positive definite matrix.

Proof. Substitution of the feedback law (9) into (8) gives

$$\begin{aligned} \dot{\mathbf{e}}_i &= \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \\ &\quad - (\nabla_{\mathbf{p}_i} \mathbf{c}_{\mathcal{V}_i} - \mathbf{I}) (\nabla_{\mathbf{p}_i} \mathbf{c}_{\mathcal{V}_i} - \mathbf{I})^{-1} \left(\mathbf{K}_i \mathbf{e}_i + \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \right) \\ &= \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} - \left(\mathbf{K}_i \mathbf{e}_i + \frac{\partial \mathbf{c}_{\mathcal{V}_i}}{\partial t} \right) \\ &= -\mathbf{K}_i \mathbf{e}_i \end{aligned} \quad (10)$$

Consider the Lyapunov function $V = \frac{1}{2} \|\mathbf{e}_i\|^2$. The time derivative of V along the trajectories generated by the feedback law (8) give

$$\dot{V} = \mathbf{e}_i \cdot \dot{\mathbf{e}}_i = -\mathbf{e}_i \cdot \mathbf{K}_i \mathbf{e}_i \quad (11)$$

which is negative for every nonzero error $\mathbf{e}_i$, and it is zero only when the error is zero. This completes proof. $\square$

IV. ILLUSTRATION BY SIMULATION

A. Model of the evolving environment

The performance of the proposed non-autonomous control scheme (8) in tracking time-varying Voronoi centroids is illustrated through the simulation of a distributed area coverage control scenario, in which a group of kinematic mobile agents is deployed in a two-dimensional environment. A time-varying Voronoi partition of the environment assigns sub-regions to each agent, and the proposed feedback law generates velocities that track the time-varying trajectories of the Voronoi centroids, assigned each one of the agents.

The evolution of the environment is described by the time-varying risk density $\phi(\mathbf{q}, t)$ in (2), which has a static component $\phi_0(\mathbf{q})$, and a time-varying component that depends on the motion of external threats/targets entering the environment:

$$\begin{aligned} \phi(\mathbf{q}, t) &= \phi_0(\mathbf{q}) \\ &\quad + \sum_{j=1}^m \exp \left(-\frac{1}{2} \left(\mathbf{q} - \mathbf{s}^{[j]}(t) \right) \cdot \boldsymbol{\beta}^{-1} \left(\mathbf{q} - \mathbf{s}^{[j]}(t) \right) \right) \end{aligned} \quad (12)$$

where $\mathbf{s}^{[j]}(t) \in \mathbb{R}^2$ for all t , is the trajectory of the j th target in the environment. Note that since these trajectories are uncontrollable by the agents, the resulting risk density has time varying component that characterizes the coverage metric as non-autonomous. The fundamental modeling assumption in (12) is that each target moving in the environment modifies the risk around itself according to a Gaussian shaped distribution centered at the target's current position, with squared deviations β_x and β_y being the diagonal entries of the matrix $\boldsymbol{\beta} = \text{diag}(\beta_x^2, \beta_y^2)$.

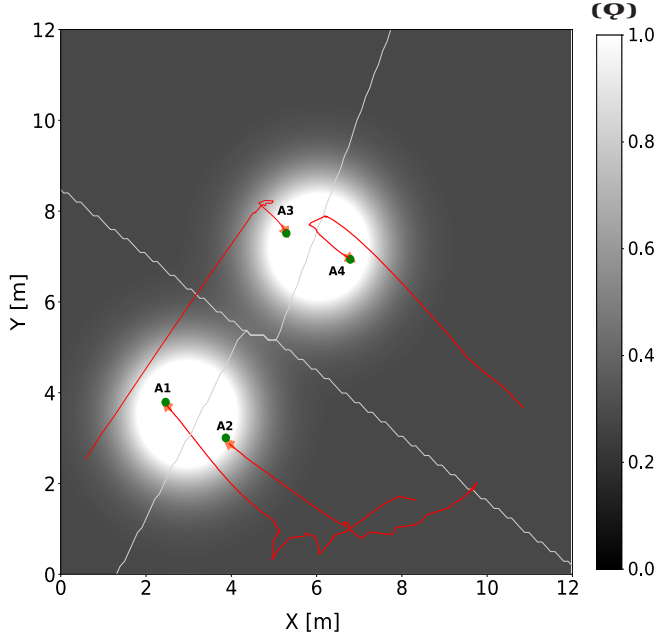


Fig. 3. Final Coverage Mapping of the agents

B. Simulation of the group's evolution under the control law (8)

We consider a square-shaped workspace Ω of side 12m. Four agents are initially placed at (2.49, 3.71)m, (3.91, 2.875)m, (5.26, 7.55)m, and (6.77, 6.96)m. The initial risk density is a Gaussian function

$$\phi_0(\mathbf{q}) = \sum_{j=1}^2 \exp\left(-\frac{1}{2} \left(\mathbf{q} - \mathbf{s}^{[j]}(0)\right) \cdot \beta^{-1} \left(\mathbf{q} - \mathbf{s}^{[j]}(0)\right)\right) \quad (13)$$

about two randomly selected target locations as $\mathbf{s}^{[1]}(0) = (6.52, 0.56)m$, $\mathbf{s}^{[2]}(0) = (5.09, 8.048)m$, with the highest density in these locations. Arbitrary trajectories are selected for targets (center of risk distributions).

The agents' configuration at time $t = 30$ s is shown in Fig. 3. The change of Voronoi partitions and agents convergence to centroids at time 30 s can be seen by comparing Figs. 2 and Fig. 3. The theoretical results in Section II is illustrated in Fig. 4, which shows asymptotic convergence to zero of the errors generated by the proposed non-autonomous feedback law, confirming the theoretical prediction. Convergence to time varying centroids and tracking yield the maximum coverage. Results in 4 show that each agent converges to the centroid of its respective Voronoi cell in a finite time, and it tracks the time varying centroid with evolution induced by the time varying environment.

The time-varying risk density is shown in Fig. 5 at four time instants. The risk-distribution evolves over the workspace based on the targets' trajectories. The evolution of the coverage metric illustrated in Fig. 6 shows that the system's performance increases as the agents converge and track the time

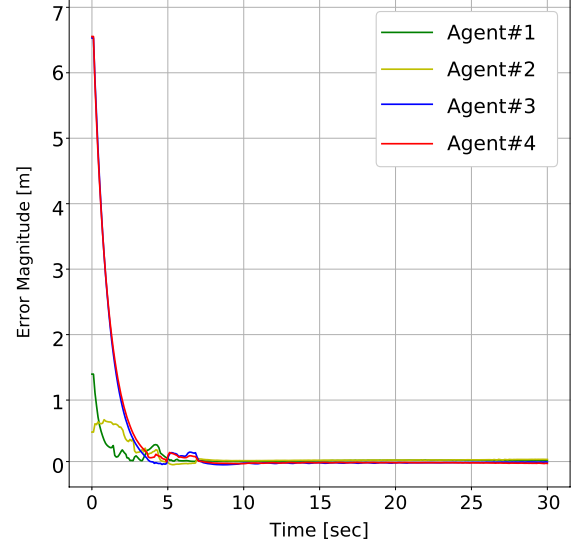


Fig. 4. Evolution of the errors ($\mathbf{e}_i = \mathbf{c}_{V_i} - \mathbf{p}_i$)

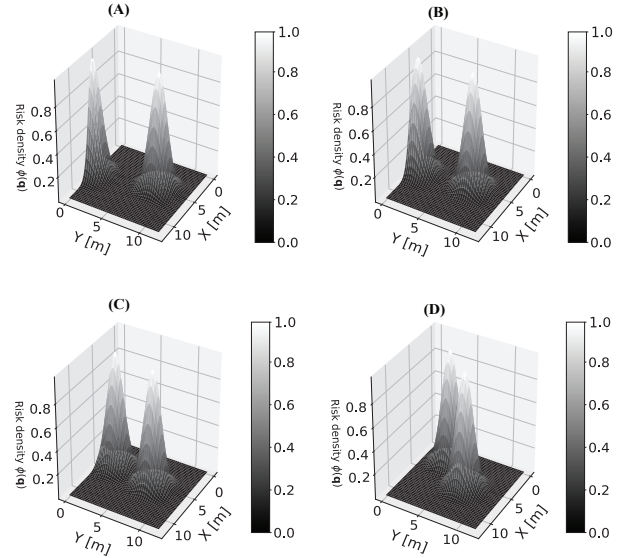


Fig. 5. Evolution of the risk map $\phi(\mathbf{q}, t)$; snapshots (A) at $t = 0$ s, (B) at $t = 10$ s, (C) at $t = 20$ s, (D) at $t = 30$ s.

varying Voronoi centroids, adapting to the evolution of the risk/environment induced by the motion of the targets..

V. SUMMARY AND CONCLUSION

We proposed a non-autonomous state-feedback control to track time varying Voronoi centroids in area coverage problems with time-varying risk density map, modeling an evolving environment to be covered by a group of agents. The evolution of the risk map is induced by the motion of moving targets in the workspace. It is shown that the proposed non-autonomous feedback law generates agents' trajectories that asymptotically

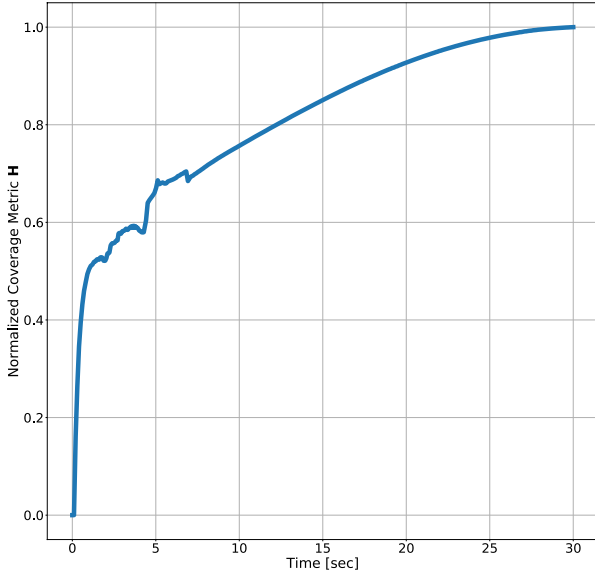


Fig. 6. Coverage metric H as agents converge to their respective Voronoi cell centroids.

stabilize around time varying Voronoi centroids of a Voronoi partition of the area of interest. In order to implement the non-autonomous state-feedback control scheme, the knowledge of the uncontrolled dynamics of the targets in the area is required, and of the state of the other agents in the team, which in this paper have been assumed to be known.

REFERENCES

- [1] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2, pp. 1327–1332, 2002.
- [2] J. Cortes, S. Martinez, T. Karatas, and Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, April 2004.
- [3] F. Bullo, J. Cortes, and S. Martinez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, Applied Mathematics Series, 2009.
- [4] S. Martinez, F. Bullo, J. Cortes, and E. Frazzoli, "On synchronous robotic networks—part I: Models, tasks, and complexity," *IEEE Trans. on Automatic Control*, vol. 52, no. 12, pp. 2199–2213, Dec. 2007.
- [5] S. Martinez, J. Cortes, and F. Bullo, "Motion coordination with distributed information," *IEEE Control Systems Magazine*, vol. 27, no. 4, pp. 75–88, Aug 2007.
- [6] S. Lee, Y. Diaz-Mercado, and M. Egerstedt, "Multirobot control using time-varying density functions," *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 489–493, April 2015.
- [7] S. Miah, B. Nguyen, F.-A. Bourque, and D. Spinello, "Nonuniform deployment of autonomous agents in harbor-like environments," *Unmanned Systems*, vol. 2, no. 04, pp. 377–389, 2014.
- [8] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2nd ed. John Wiley & Sons, LTM, 2000.
- [9] S. Lloyd, "Least squares quantization in pcm," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar 1982.
- [10] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks: Variations on a theme," in *10th Mediterranean Conference on Control and Automation*, vol. 1, 2002.
- [11] F. Lekien and N. E. Leonard, "Nonuniform coverage and cartograms," *SIAM Journal of Control and Optimization*, vol. 48, no. 1, pp. 351–372, February 2009.
- [12] A. Ghaffarkhah, Y. Yan, and Y. Mostofi, "Dynamic coverage of time-varying environments using a mobile robot a communication-aware perspective," pp. 1297–1302, Dec 2011.
- [13] S. Miah, A. Y. Panah, M. M. H. Fallah, and D. Spinello, "Generalized non-autonomous metric optimization for area coverage problems with mobile autonomous agents," *Automatica*, vol. 80, no. 1, pp. 295–299, Jun 2017.
- [14] S. Miah, M. M. H. Fallah, and D. Spinello, "Non-autonomous coverage control with diffusive evolving density," *IEEE Transactions on Automatic Control*, vol. 62, no. 10, pp. 5262–5268, Oct 2017.

APPENDIX

Algorithm 1 Area Coverage Control

Data: $\mathcal{V}_i$: Voronoi Partition

Data: $\mathcal{Q} = \bigcup_{i=1}^n \mathcal{V}_i$

Data: $\mathbf{P}_A, \mathbf{C}_{\mathcal{V}}, t_s$: Position of Agents, Centroid of Voronoi Partition, time step

Data: $r_i = \|\mathbf{q} - \mathbf{P}_{A_i}\|, \forall \mathbf{q} \in \mathcal{Q}$,

Data: $\mathbf{s}_t, \phi_t$: Position of Targets, Risk Distribution over Voronoi Partition (Gaussian Distribution)

Data: $H(\mathbf{P}, \mathcal{V}, t)$: Coverage Metric

Data: $f : \mathbb{R} \rightarrow \mathbb{R}$: Sensor Performance

1 Initialization:

$\mathbf{P}_0 = \{\mathbf{P}_{A_1} \cdots, \mathbf{P}_{A_n}\} \leftarrow$ random initialization

2 Voronoi Partitions:

$\mathcal{V}_i = \{\mathbf{q} \in \mathcal{Q} | f(r_i) \leq f(r_j), i \neq j, i \in \mathcal{I}\}.$

3 $\mathbf{s}_t = \{\mathbf{s}_1 \cdots, \mathbf{s}_m\} \leftarrow$ Random Target Trajectory

4 Generalized Mass of Voronoi Partition:

$\tilde{m}_{\mathcal{V}_i} = \int_{\mathcal{V}_i} \tilde{\phi}(\mathbf{q}, t) d\mathbf{q}$

5 Centroid of Voronoi Partition:

$\mathbf{C}_{\mathcal{V}_i} = \frac{1}{\tilde{m}_{\mathcal{V}_i}} \int_{\mathcal{V}_i} \mathbf{q} \tilde{\phi}(\mathbf{q}, t) d\mathbf{q}$

6 $\tilde{\phi}(\mathbf{q}, \mathbf{P}_i) = -2\phi(\mathbf{q}, t) (\partial f(r_i) / \partial r_i^2).$

7 $\phi(\mathbf{q}, t) = \phi_0(\mathbf{q}) + \sum_{j=1}^m \exp\left(-\frac{1}{2}(\mathbf{q} - \mathbf{s}^{[j]}(t)) \cdot \beta^{-1}(\mathbf{q} - \mathbf{s}^{[j]}(t))\right)$

8 Calculating Initial Centroids:

9 $\mathbf{C}_{\mathcal{V}_0} = \{\mathbf{C}_{\mathcal{V}_1} \cdots, \mathbf{C}_{\mathcal{V}_n}\}$

10 $\dot{\mathbf{P}}_i = \mathbf{u}_i$

11 $H(\mathbf{P}, \mathcal{V}, t) = \sum_{i=1}^n \int_{\mathcal{V}_i} f(r_i) \phi(\mathbf{q}, t) d\mathbf{q}$

12 $\mathbf{e}_i = \mathbf{C}_{\mathcal{V}_i}(\mathbf{P}_i, t) - \mathbf{P}_i.$

13 $\mathbf{u}_i = -(\nabla_{\mathbf{P}_i} \mathbf{C}_{\mathcal{V}_i} - \mathbf{I})^{-1} \left(\mathbf{K}_i \mathbf{e}_i + \frac{\partial \mathbf{C}_{\mathcal{V}_i}}{\partial t} \right)$

14 while $e > 0$ do

15 State update:

16 $\mathbf{P}_t = \mathbf{P}_{t-1} + t_s * \mathbf{u}_t$

17 Update Voronoi Partition:

18 $\mathcal{V}_i = \{\mathbf{q} \in \mathcal{Q} | f(r_i) \leq f(r_j), i \neq j, i \in \mathcal{I}\}.$

19 Calculate centroids:

20 $\mathbf{C}_{\mathcal{V}_t} = \{\mathbf{C}_{\mathcal{V}_1} \cdots, \mathbf{C}_{\mathcal{V}_n}\}$

21 $\mathbf{e}_t = \mathbf{C}_{\mathcal{V}_t}(\mathbf{P}_t, t) - \mathbf{P}_t.$

22 end

6.2 Temporal Difference Learning of Area Coverage Control with Multi-Agent Systems

In this article, a time-invariant problem is investigated using the actor-critic method. This paper is accepted and presented at the 2022 IEEE International Symposium on Robotic and Sensors Environments (ROSE) conference.

Temporal Difference Learning of Area Coverage Control with Multi-Agent Systems

Farzan Soleymani
Mechanical Engineering
University of Ottawa, Canada

Suruz Miah, *Senior Member, IEEE*
Electrical and Computer Engineering
Bradley University, Peoria, IL, USA

Davide Spinello*, *Senior Member, IEEE*
Mechanical Engineering
University of Ottawa, Canada

Abstract—We formulate an area coverage control problem with multi-agent systems by using Bellman's principle of optimality. The performance index is composed of the additive contributions of a term quadratic in the control effort and of a positive definite term that depends on the coverage metric. In this way, the reward encodes optimality in the sense of the classical Lloyd's algorithm, where the term depending on the coverage metric weights the energy of the state, and the term depending on the control weights the effort energy. Quasi optimality is achieved by an adaptive control policy using an actor-critic neural networks based reinforcement learning strategy, with quadratic function approximations for the value function and the control policy. Optimal configurations for a team of agents correspond to centroidal Voronoi partitions of the workspace, with each agent converging to the centroid of the respective generalized Voronoi cell. The system's dynamics is written in discrete time form, and the temporal difference form of Bellman's equation is used in the policy iteration learning scheme to train critic and actor weights. Remarkably, the obtained class of solutions is consistent with the one obtained with Lloyd's algorithm, with the advantage that the reinforcement learning formulation allows for a model-free implementation based on data measured along a system's trajectory. By storing an appropriate time history of control actions, the gradient of the value function is numerically approximated, allowing one to run the policy approximation without knowledge of the input dynamics. Direct comparisons with Lloyd's algorithm show the expected slower convergence since Lloyd's trajectories are optimal for the continuous time system.

Index Terms—actor-critic, area coverage, optimal control, reinforcement learning

I. INTRODUCTION

Multi-agent systems have received a thorough attention due to their capability of performing complex tasks in a distributed, scalable, and robust manner. Applications of such systems include but not limited to harbor patrolling, environmental monitoring, perimeter surveillance, search and rescue, and co-operative estimation [1]–[3]. In area coverage control, a team of mobile agents spatially configure themselves over an area of interest for maximizing a coverage metric which encodes the collective sensing performance of the agents. A scalar field defined in the area (*i.e.*, spatial environment), called risk density, represents a quantity of interest weighting the decisions

of the agents. The implementation of area coverage control algorithms requires the knowledge of the state of neighbor agents and the knowledge of the risk density/environment, which can pose a severe challenge in real-time implementation of robotic systems, especially when the environment itself is evolving [4]–[8].

To overcome the uncertainties induced by control design and implementation with partially unknown parameters, a model-free adaptive approach can be beneficial by relying on sensor data to implement control policies based on machine learning algorithms. In this work, the area coverage problem is formulated in discrete-time in terms of a Bellman equation [9, Ch. 6]. The reward is composed of the contributions of the coverage metric performance index and of a quadratic term of the control input so that optimality is stated consistently in terms of maximization of the coverage metric. An adaptive control policy based on reinforcement learning with an actor-critic structure iterates the control, searching for optimality. The policy iteration method is classified as part of the family of reinforcement learning approaches and is composed of two stages, namely policy evaluation and policy improvement [10], [11].

In policy iteration methods, given a control policy, the current cost is estimated in the direction of true cost value, while the control policy is updated following a greedy approach. The application of policy iteration methods for nonlinear systems has been investigated in [12]. Approximate dynamic programming was proposed in [13], [14] to solve optimal control problems forward in time along a specific system trajectory. Approximate dynamic programming is a value iteration method which is performed without requiring any initial admissible control, aiming to find the optimal value function and calculate the control policy in one step. Adaptive dynamic programming [15] is a value iteration algorithm based on approximate dynamic programming; it couples value iteration to estimate the cost forward, with hard computation of all possible trajectories at each time step emanating from the current state. The algorithm does not require the knowledge of the system dynamics; however, it requires the full exploration of the state space at each iteration, which is equivalent to the knowledge of the system dynamics.

Policy iteration methods are typically implemented using two neural networks, with the critic network approximating

This research has been partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), grant number RGPIN-2016-05783.

*Corresponding author

the value function and the actor network approximating the control policy. The neural networks are finite dimensional basis functions along which the value function and the policy are projected. This is done by introducing a set of unknown coefficients (neural network weights) that are determined iteratively, for example, by taking measurements. The projection introduces approximated versions of the value and the policy, and the weights are updated (trained) to minimize suitable metrics of such errors. In the context of control systems, this effectively corresponds to a direct adaptive control. An online policy iteration algorithm is proposed in [16], [17] based on a sequential update of the critic and actor neural networks, *i.e.*, networks are updated one at a time while the others are kept fixed. These methods are employed regardless of explicit knowledge of internal system dynamics. An adaptive sensor coverage over an unknown environment has been formulated in [18], using a robust learning method that drives a team of robots to perform various distributed sensing tasks. Robots' aim is to converge to positions that locally minimize a cost function, which is defined as the sensing quality of the network.

The contribution of this paper consists of a model-free formulation for area coverage control problems in the terms of Bellman's principle of optimality. The performance index includes a term that depends on the coverage metric and a control energy term. Minimizing trajectories converge the agents to the centroids of Voronoi cells (*i.e.*, centroidal Voronoi tessellation), as in the standard coverage control. The established equivalence between the two optimal control frameworks, model-based and model-free, has important implications for the implementation in real systems, where online model-free adaptive controls may be preferable in highly uncertain environments and varying process dynamics. We limit the scope by considering static environments, which imply drift-free error dynamics. This allows us to focus on the reinforcement learning formulation and approximate dynamic programming solution, showing the analogies with classical Lloyd's algorithms.

II. AREA COVERAGE WITH MOBILE AGENTS

Consider a group of mobile agents which are deployed in a spatial environment, with an induced partition of the environment into subdomains that map one-to-one with networked agents. In a two-dimensional environment, a typical tessellation technique is its Voronoi partition, with common boundaries emerging from the intersection of agents' spheres of influence. We consider exogenous targets modelled as risk sources on the environment, where higher risk implies a higher likelihood of events of interest occurring in the corresponding area. Area coverage control algorithms account for non-uniform risk by attracting more agents toward high-risk neighborhoods.

We consider a set of n kinematic agents with first-order dynamics given by

$$\dot{\mathbf{p}}_i = \mathbf{u}_i, \quad (1)$$

where $\mathbf{u}_i$ is the i th agent velocity and $\mathbf{p}_i \in \mathbb{R}^2$ is the position vector. The area coverage control is formulated as an optimal control problem aiming at finding an agents' spatial distribution that locally maximizes the coverage metric H defined by

$$H(\mathbf{p}, \mathcal{V}) = \sum_{i=1}^n \int_{\mathcal{V}_i} f(r_i) \phi(\mathbf{q}) d\Omega, \quad (2)$$

where $f: \mathbb{R} \rightarrow \mathbb{R}$ is the agents' sensing performance, assumed to be strictly decreasing in the argument $r_i = \|\mathbf{q} - \mathbf{p}_i\|$, $\forall \mathbf{q} \in \Omega \subset \mathbb{R}^2$, $\mathbf{p}$ denotes the collection of the position vectors for n agents, *i.e.*, $\mathbf{p} = [\mathbf{p}_1^T, \mathbf{p}_2^T, \dots, \mathbf{p}_n^T]^T \in \mathbb{R}^{2n}$, and $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_n)$ is the collection of Voronoi cells $\mathcal{V}_i$, which is assigned to the i th agent. Note that the Voronoi cell $\mathcal{V}_i$ is defined as the set of points in Ω that are best sensed by i with respect to the function f [19]:

$$\mathcal{V}_i := \{\mathbf{q} \in \Omega | f(r_i) \geq f(r_j), \forall j \in \{1, \dots, n\}, j \neq i\}. \quad (3)$$

Introduce the generalized density

$$\tilde{\phi}(\mathbf{q}, \mathbf{p}_i) = -2\phi(\mathbf{q}) \frac{\partial f(r_i)}{\partial r_i^2} \quad (4)$$

to define the generalized mass and Voronoi centroid

$$\tilde{m}_i = \int_{\mathcal{V}_i} \tilde{\phi}(\mathbf{q}, \mathbf{p}_i) d\Omega \quad \text{and} \quad \tilde{\mathbf{c}}_i = \frac{1}{\tilde{m}_i} \int_{\mathcal{V}_i} \mathbf{q} \tilde{\phi}(\mathbf{q}, \mathbf{p}_i) d\Omega. \quad (5)$$

The implementation of an area coverage control algorithm based on the metric H requires the estimation of the density $\phi(\mathbf{q})$ and of the state of a subset of the other agents in the group to be able to compute the Voronoi tessellation and the generalized centroids. Local optimality in terms of agents' spatial distribution is reached by convergence to the set of generalized centroids, with induced centroidal Voronoi tessellation as area partition [1], [4]. The error between agent i and the related centroid of the i th Voronoi cell is given by:

$$\mathbf{e}_i = \tilde{\mathbf{c}}_i - \mathbf{p}_i \quad (6)$$

By taking the time derivative and substituting from (1) and (6) we obtain the following error dynamics

$$\dot{\mathbf{e}}_i = \dot{\tilde{\mathbf{c}}}_i - \dot{\mathbf{p}}_i = \nabla_{\mathbf{p}_i} \tilde{\mathbf{c}}_i \dot{\mathbf{p}}_i - \dot{\mathbf{p}}_i + \frac{\partial \tilde{\mathbf{c}}_i}{\partial t} = (\nabla_{\mathbf{p}_i} \tilde{\mathbf{c}}_i - \mathbf{I}) \mathbf{u}_i \quad (7)$$

where $\nabla_{\mathbf{p}_i}$ is the gradient with respect to $\mathbf{p}_i$, and the drift dynamics is zero since the density ϕ is static. In the general, non-autonomous case with time varying risk, the error dynamics can be rewritten in non-linear affine input form by introducing

$$\mathbf{F}(\mathbf{e}_i, t) := \frac{\partial \tilde{\mathbf{c}}_i}{\partial t}, \quad (8a)$$

$$\mathbf{G}(\mathbf{e}_i, t) := \nabla_{\mathbf{p}_i} \tilde{\mathbf{c}}_i - \mathbf{I}, \quad (8b)$$

from which it follows that (7) can be rewritten in the standard form

$$\dot{\mathbf{e}}_i = \mathbf{F}(\mathbf{e}_i, t) + \mathbf{G}(\mathbf{e}_i, t) \mathbf{u}_i. \quad (9)$$

where $\mathbf{u}_i \in \mathbb{R}^2$ acts as the input to the non-autonomous dynamics of the error $\mathbf{e}_i$. Under the condition that the domain

Ω is bounded, the terms $\partial \mathbf{c}_i / \partial t$ and $\nabla_{\mathbf{p}_i} \tilde{\mathbf{c}}_i - \mathbf{I}$ are bounded [20], and therefore the error dynamics is bounded as long as the control input is finite. When the error dynamics is autonomous, the control is just a feedback on the error; with time-varying risk, a feed-forward component of the control is necessary to track time-varying centroids. For the rest of the paper, we focus on the autonomous case; ongoing work considers the generalization to non-autonomous risk. Since optimal configurations are centroidal, with $\mathbf{e}_i = \mathbf{0}$, the controls $\mathbf{u}_i$ drives the system to optimality in the sense of the area coverage metric H . By recalling that the sensing performance f_i depends on the argument $r_i^2 = \|\mathbf{q} - \mathbf{p}_i\|^2$, from the definition of generalized centroid (5) we have

$$\begin{aligned} \nabla_{\mathbf{p}_i} \tilde{\mathbf{c}}_i &= \\ &= -\frac{1}{\tilde{m}_i^2} [\nabla_{\mathbf{p}_i} \tilde{m}_i] \int_{\mathcal{V}_i} \tilde{\phi}_i \mathbf{q} d\Omega + \frac{1}{\tilde{m}_i} \int_{\mathcal{V}_i} \mathbf{q} [\nabla_{\mathbf{p}_i} \tilde{\phi}_i] d\Omega \quad (10) \\ &= \frac{1}{\tilde{m}_i} \int_{\mathcal{V}_i} -2\phi \frac{\partial^2 f_i}{(\partial r_i^2)^2} (\mathbf{q} - \tilde{\mathbf{c}}_i) (\mathbf{q} - \mathbf{p}_i)^\top d\Omega = \frac{\mathbf{J}_i}{\tilde{m}_i} \quad (11) \end{aligned}$$

after introducing the inertia matrix $\mathbf{J}_i = \int_{\mathcal{V}_i} \frac{\partial \tilde{\phi}_i}{\partial r_i^2} (\mathbf{q} - \tilde{\mathbf{c}}_i) (\mathbf{q} - \mathbf{p}_i)^\top d\Omega$. By considering the sensing performances to be described by the radially symmetric sensing functions $f_i = \alpha \exp(-\beta r_i^2)$ [4], [8], [21], where $\alpha > 0$ and $\beta > 0$ are shape parameters, from the definition of generalized density (4) we have $\tilde{\phi}_i = 2\beta f_i \phi$, $\frac{\partial \tilde{\phi}_i}{\partial r_i^2} = -2\beta^2 \phi f_i$, and the moment of inertia matrix becomes

$$\mathbf{J}_i = 4\alpha\beta^2 \int_{\mathcal{V}_i} \phi \exp(-\beta \|\mathbf{q} - \mathbf{p}_i\|^2) (\mathbf{q} - \tilde{\mathbf{c}}_i) (\mathbf{q} - \mathbf{p}_i)^\top d\Omega \quad (12)$$

which is symmetrical at a centroidal configuration $\mathbf{p}_i = \tilde{\mathbf{c}}_i$. The reinforcement learning method described below is equivalent to Lloyd's algorithm when the risk function is not time-varying. With the time-varying risk function, the reinforcement learning algorithm regularizes the error around the origin, therefore driving the system to optimality in the sense of the area coverage metric H .

III. STATIC AREA COVERAGE CONTROL

We adopt a discrete-time implementation for the actor-critic formulation that follows. Previously introduced variables will be indicated with the same symbol, and the integer κ will be used to label the time instant as $t_\kappa = \kappa T$, where T is a constant time interval. The autonomous error dynamics from (9) can be formulated in discrete time as

$$\mathbf{e}_i[\kappa + 1] = \bar{\mathbf{F}}(\mathbf{e}_i[\kappa]) + \bar{\mathbf{G}}(\mathbf{e}_i[\kappa], \kappa) \mathbf{v}_i[\kappa] \quad (13)$$

where $\bar{\mathbf{F}} = \mathbf{e}_i[\kappa]$ and $\bar{\mathbf{G}} = T\mathbf{G}$. In optimal control, the value function is the index of optimality. In area coverage problems with continuous-time dynamics in which each agent has full knowledge of the workspace through the risk function ϕ , and has full knowledge of the state of the group of agents through their states $\mathbf{p}$, the positive definite function H^{-1} was shown to be a Lyapunov function along whose trajectories the agents

would maximize the coverage metric H [4]. We leverage this idea to pose the value function as

$$V(\mathbf{e}[\kappa]) = \sum_{j=\kappa}^{\infty} \gamma^{j-\kappa} \left(H(\mathbf{e}[j])^{-1} + \frac{1}{2} \mathbf{u}^T[j] \mathbf{R} \mathbf{u}[j] \right) \quad (14)$$

where H denotes the coverage metric from (2) and $\mathbb{R}^{2n \times 2n} \ni \mathbf{R} = \mathbf{R}^T > 0$ is a symmetric positive definite matrix, penalizing the action $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n)$, and having the block diagonal structure $\mathbf{R} = \text{diag}(\mathbf{R}_1, \dots, \mathbf{R}_n)$, with $\mathbf{R}_i \in \mathbb{R}^{2 \times 2}$. The parameter $0 < \gamma \leq 1$ is the discount factor, that establishes a balance between current and future values of V [9, chapter 11.5]. For the coverage metric H , the notation has been modified by dropping the explicit dependence on $\mathcal{V}$, and by substituting $\mathbf{p} = \mathbf{e} + \tilde{\mathbf{c}}$, with the dependency on $\mathbf{c}$ expressed through the time index j . Since $H > 0$, the value function is positive definite as usually prescribed in optimization problems [9, chapter 11.5]. Explicit writing of the first term in the summation in (14) gives

$$\begin{aligned} V(\mathbf{e}[\kappa]) &= H(\mathbf{e}[\kappa])^{-1} + \frac{1}{2} \mathbf{u}^T[\kappa] \mathbf{R} \mathbf{u}[\kappa] \\ &+ \sum_{j=\kappa+1}^{\infty} \gamma^{j-\kappa} \left(H(\mathbf{e}[j])^{-1} + \frac{1}{2} \mathbf{u}^T[j] \mathbf{R} \mathbf{u}[j] \right) \quad (15) \end{aligned}$$

By using the definition of value function, the right hand side of (15) can be rewritten as

$$V(\mathbf{e}[\kappa]) = H(\mathbf{e}[\kappa])^{-1} + \frac{1}{2} \mathbf{u}^T[\kappa] \mathbf{R} \mathbf{u}[\kappa] + \gamma V(\mathbf{e}[\kappa + 1]) \quad (16)$$

Based on temporal difference learning [22], (16) is a deterministic Bellman equation. The temporal difference method offers a solution to Bellman's equation without requiring knowledge of the true system dynamics (13), but rather relying on its estimation; in other words, it is a model-free approach. Considering the discrete time system in (13), the optimal control problem can be formulated as: given a set of admissible control policies (actions) $\mathbf{v} \in \mathbb{V}$ and the value function (14), find an optimal control policy that minimizes the performance index (14) of the system (13) [23]. Following Bellman's principle of optimality [9, chapter 11.5], the optimal control inputs that maximize the performance metric H are found by solving the minimization problem

$$\begin{aligned} V^*(\mathbf{e}[\kappa]) &= \\ \min_{\mathbf{v} \in \mathbb{V}} &\left\{ H(\mathbf{e}[\kappa])^{-1} + \frac{1}{2} \mathbf{u}^T[\kappa] \mathbf{R} \mathbf{u}[\kappa] + \gamma V^*(\mathbf{e}[\kappa + 1]) \right\} \quad (17) \end{aligned}$$

Equation (17) is known as the Bellman optimality equation or discrete-time Hamiltonian-Jacobi-Bellman equation [9]. A control policy corresponding to an optimal solution of (17) is given by $\mathbf{u}^*(\mathbf{e}[\kappa]) = \underset{\mathbf{u} \in \mathbb{V}}{\text{argmin}} V^*(\mathbf{e}[\kappa])$. Therefore the search of an optimal control policy is equivalent to a root searching problem in terms of the gradient of (16)

$$\begin{aligned} \frac{\partial V^*(\mathbf{e}[\kappa])}{\partial \mathbf{u}[\kappa]} &= \\ &= -\frac{\frac{\partial H(\mathbf{e}[\kappa], \kappa)}{\partial \mathbf{u}[\kappa]}}{(H(\mathbf{e}[\kappa], \kappa))^2} + \mathbf{R} \mathbf{u}[\kappa] + \gamma \frac{\partial V^*(\mathbf{e}[\kappa + 1])}{\partial \mathbf{e}[\kappa]} = \mathbf{0} \quad (18) \end{aligned}$$

This expression can be simplified by considering

$$\frac{\partial H(e[\kappa], \kappa)}{\partial \mathbf{u}_i[\kappa]} = \frac{\partial H(e[\kappa], \kappa)}{\partial e_i[\kappa]} \frac{\partial e_i[\kappa]}{\partial \mathbf{u}_i[\kappa]} \quad (19a)$$

$$\frac{\partial H(e[\kappa], \kappa)}{\partial e_i[\kappa]} = m_i e_i[\kappa], \quad \frac{\partial e_i[\kappa]}{\partial \mathbf{u}_i[\kappa]} = \mathbf{0} \quad (19b)$$

where the last expression accounts for the fact that the current error $e_i[\kappa]$ does not depend on the current control input $\mathbf{u}_i[\kappa]$. The explicit calculation of the gradient of the value function from (16) gives the optimal control policy

$$\mathbf{u}_i^*(e[\kappa+1]) = -\gamma \mathbf{R}_i^{-1} \frac{\partial V(e[\kappa+1])}{\partial \mathbf{u}_i[\kappa]}. \quad (20)$$

This expression is typically rewritten in terms of $\mathbf{G}$ by $\frac{\partial V(e[\kappa+1])}{\partial \mathbf{u}_i[\kappa]} = \mathbf{G}(e[\kappa]) \frac{\partial V(e[\kappa+1])}{\partial e_i[\kappa+1]}$; however, here we consider a finite difference approximation to calculate it directly in terms of the control $\mathbf{u}$

$$\frac{\partial V(e[\kappa+1])}{\partial \mathbf{u}_i[\kappa]} \approx \frac{V(e[\kappa+1]) - V(e[\kappa])}{(\mathbf{u}_i[\kappa] - \mathbf{u}_i[\kappa-1]) + \epsilon \text{sign}(\mathbf{u}_i[\kappa] - \mathbf{u}_i[\kappa-1])} \quad (21)$$

where the small parameter $\epsilon = 10^{-5}$ is added to the denominator to avoid singularities that would be generated by steady control, and sign is the “signum” function. The calculation requires the storage of two values of the control, which is considered as part of the measurement.

IV. ACTOR-CRITIC NEURAL NETWORK APPROXIMATIONS

In this section, the optimal control solution is obtained by using reinforcement learning. The hardware implementation of this algorithm would require only data collected along the trajectories to sense the environment (function ϕ) and a set of neighbouring agents (position data can be acquired through direct communication). The proposed solution is based on the policy iteration methods that operate iteratively through two steps, namely (i) policy evaluation and (ii) policy improvement [9], [10]. The actor and critic neural network approximations operate in tandem, as illustrated in Fig. 1, with the actor network determining the control policy $\mathbf{u}_i$, and the critic network estimating the value function V . Since the control policy is obtained based on an estimation of the value function V , the critic network must converge faster than the actor network.

A. Critic Network Structure

The value function approximation is defined as

$$\hat{V}(e_i[\kappa]) = \mathbf{w}_c^T \boldsymbol{\psi}_c(e_i[\kappa]), \quad (22)$$

where the vector $\mathbf{w}_c \in \mathbb{R}^{n_c}$ with entries

$$\mathbf{w}_c = [w_{c,1}, w_{c,2}, \dots, w_{c,n_c}]^T. \quad (23)$$

includes the weights of the critic network at the n_c nodes. The collection of activation functions of the neural network

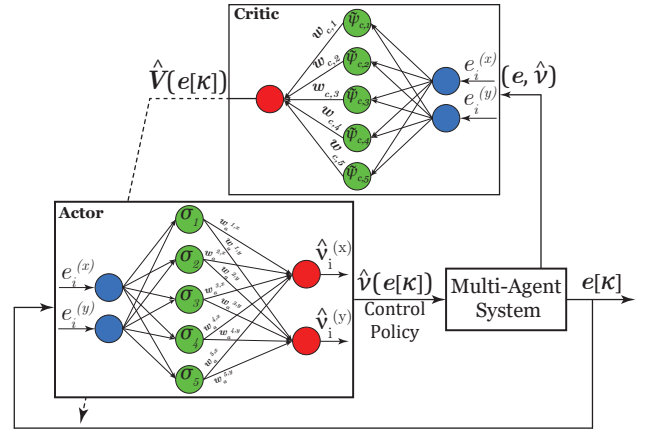


Fig. 1. Actor-Critic scheme.

is denoted by $\boldsymbol{\psi}_c(e_i[\kappa])$. The activation function of the critic network at the i^{th} node is then given by

$$\boldsymbol{\psi}_c(e_i[\kappa]) = \begin{bmatrix} e_i^x[\kappa], e_i^y[\kappa], (e_i^x[\kappa])^2, e_i^x[\kappa]e_i^y[\kappa], (e_i^y[\kappa])^2 \end{bmatrix}^T \quad (24)$$

with the introduction of the Cartesian components e_i^x and e_i^y of the error. The number of elements in the set of basis functions determines the number of nodes in the critic network.

We introduce the compact notation $\nu_\kappa = \nu(\mathbf{u}(e_i[\kappa]))$. The activation functions in the hidden layer are derived from the temporal difference equation (16), by introducing the basis function projection (22)

$$\begin{aligned} \nu_\kappa &= V(e_i[\kappa]) - \gamma V(e_i[\kappa+1]) \\ &\approx \mathbf{w}_c^T \boldsymbol{\psi}_c(e_i[\kappa]) - \gamma \mathbf{w}_c^T \boldsymbol{\psi}_c(e_i[\kappa+1]) =: \mathbf{w}_c^T \tilde{\boldsymbol{\psi}}_c[\kappa] \end{aligned}$$

where $\tilde{\boldsymbol{\psi}}_c[\kappa] = \boldsymbol{\psi}_c(e_i[\kappa]) - \gamma \boldsymbol{\psi}_c(e_i[\kappa+1])$. The critic neural network shown in Fig. 1 estimates the value function $V(e_i[\kappa])$ using the basis function vector $\tilde{\boldsymbol{\psi}}_c[k] \in \mathbb{R}^5$ that minimizes the critic least-square error defined by

$$\delta_c = \sum_{n=0}^{\eta-1} \frac{1}{2} (\nu_{\kappa+n} - \mathbf{w}_c^T \tilde{\boldsymbol{\psi}}_c[\kappa+n])^2 = \frac{1}{2} \|\boldsymbol{\nu} - \boldsymbol{\Lambda} \mathbf{w}_c\|^2 \quad (25)$$

where the entries of the matrix $\boldsymbol{\Lambda} \in \mathbb{R}^{\eta \times 5}$ are $[\boldsymbol{\Lambda}]_n = \tilde{\boldsymbol{\psi}}_c^T[\kappa+n] \in \mathbb{R}^{1 \times 5}$, and the η -vector $\boldsymbol{\nu}$ is the collection of the scalars $\nu_{\kappa+n}$ for $n = 0, 1, \dots, \eta-1$. The weights of the critic network are updated based on a gradient descent scheme, that for $\eta \geq 5$ data samples are given by

$$\mathbf{w}_c^{[\kappa+1]} = \mathbf{w}_c^{[\kappa]} - \alpha_c \frac{\partial \delta_c}{\partial \mathbf{w}_c} = \mathbf{w}_c^{[\kappa]} - \alpha_c \boldsymbol{\Lambda}^T (\boldsymbol{\Lambda} \mathbf{w}_c^{[\kappa]} - \boldsymbol{\nu}) \quad (26)$$

where α_c denotes the learning rate of critic neural network.

B. Actor Network Structure

The policy improvement is executed using the actor network, which approximates the control policy. The structure of

the approximated optimal control (20) in terms actor network's output is

$$\hat{u}(e_i[\kappa]) = \mathbf{W}_a^T \boldsymbol{\sigma}(e_i[\kappa]) \quad (27)$$

where the activation function set $\boldsymbol{\sigma}(e_i[\kappa])$ of the actor neural network in this case is same as the critic basis functions. The weight matrix of the actor network $\mathbf{W}_a \in \mathbb{R}^{5 \times 2}$ is defined as

$$\mathbf{W}_a^T = \begin{bmatrix} w_a^{1,x} & w_a^{2,x} & w_a^{3,x} & w_a^{4,x} & w_a^{5,x} \\ w_a^{1,y} & w_a^{2,y} & w_a^{3,y} & w_a^{4,y} & w_a^{5,y} \end{bmatrix} \quad (28)$$

where the superscripts x and y denote Cartesian components, and w_a^x and w_a^y are the actor's weights.

The training samples for the actor network weights are determined by generating a sequence $\{e_i[k + \kappa], \mathbf{u}(e_i[k + \kappa])\}$, $\kappa = 0, 1, \dots, \hat{n} - 1$, resulting in the following set of linear equations $\boldsymbol{\sigma}^T(e_i[i])\mathbf{W}_a = \hat{\mathbf{u}}_i^T[i]$, for $i = k, \dots, k + \hat{n} - 1$, that are solved for the weight matrix $\mathbf{W}_a$. This linear system can be represented in matrix form as $\boldsymbol{\Sigma}\mathbf{W}_a = \mathbf{z}$, where the entries of the $\hat{n} \times 5$ matrix $\boldsymbol{\Sigma}$ are $[\boldsymbol{\Sigma}]_{k+\kappa} = \boldsymbol{\sigma}^T(e_i[k + \kappa])$, and the entries of the $\hat{n} \times 2$ vector $\mathbf{z}$ are given by $[\mathbf{z}]_{k+\kappa} = \hat{\mathbf{u}}_i^T[k + \kappa]$, for $\kappa = 0, 1, \dots, \hat{n} - 1$. A gradient descent scheme is adopted to update $\mathbf{W}_a$ such that the sum of squared errors $\delta_a = \text{Tr}[(\mathbf{z} - \boldsymbol{\Sigma}\mathbf{W}_a)^T(\mathbf{z} - \boldsymbol{\Sigma}\mathbf{W}_a)]/2$ is minimized, where Tr is the trace operator. The partial derivative of the sum of the squared error is given by $\frac{\partial \delta_a}{\partial \mathbf{W}_a} = \boldsymbol{\Sigma}^T \boldsymbol{\Sigma} \mathbf{W}_a - \boldsymbol{\Sigma}^T \mathbf{z}$. Using this expression, the actor network weights are updated according to the following iterative gradient descent scheme

$$\mathbf{W}_a^{[n+1]} = \mathbf{W}_a^{[n]} - \alpha_a \frac{\partial \delta_a}{\partial \mathbf{W}_a} = \mathbf{W}_a^{[n]} - \alpha_a \boldsymbol{\Sigma}^T (\boldsymbol{\Sigma} \mathbf{W}_a^{[n]} - \mathbf{z}) \quad (29)$$

where α_a is the learning rate of the actor network, and n is the iterator, which stops when the error converges according to $\|e_i[k + n + 1] - e_i[k + n]\| < \varepsilon_v$, and ε_v is a tolerance. The architecture of the proposed model-free coverage control is presented in Algorithm 1. The weights of the actor and critic networks are initiated randomly.

Algorithm 1 Actor-Critic algorithm

Input: R , Policy Iteration Inner-loop length K , sampling time T

Output: Error trajectory of i^{th} agent: $e_i[\kappa]$

Initialize:

$\mathbf{W}_a^n, \mathbf{w}_c^\kappa$ /* Discrete time, critic and actor indices, weights of actor and critic neural networks */

- 1 Initialize $e_i[0]$ from initial state $\mathbf{p}_i[0]$ using (6)
- 2 Compute initial control input $\mathbf{u}_i[0]$ using (27).

3 **for** $n = 0$ to N_a **do**

4 **for** $\kappa = nK$ to $(n+1)K$ **do**

5 Update agents' states by $\mathbf{p}_i[\kappa + 1] = \mathbf{p}_i[\kappa] + T\mathbf{u}_i[\kappa]$

6 Calculate new centroids $\tilde{\mathbf{c}}_i[\kappa + 1]$ using (5)

7 Find $e_i[\kappa + 1]$ using (6)

8 Compute value function $\hat{V}(e_i[\kappa + 1])$ using (22)

9 **if** $\|e_i[k + n + 1] - e_i[k + n]\| > \varepsilon_v$ **then**

10 Compute control policy $\mathbf{u}_i[\kappa + 1]$ using (27)

11 **else if** $\|e_i[k + n + 1] - e_i[k + n]\| \leq \varepsilon_v$ **then**

12 Break

13 Update critic network weights $\mathbf{w}_c^{[\kappa+1]}$ using (26)

14 **end**

15 Update actor network weights $\mathbf{W}_a^{[n+1]}$ using (29)

16 **end**

The error and control input are updated at each step of the inner-loop, followed by the approximation of the value function $\hat{V}(e_i[\kappa + 1])$. Next, the critic network $\mathbf{w}_c^{[\kappa+1]}$ is updated for the new approximated value function. At the end of the inner-loop, the error $e_i[K]$, control input $\mathbf{u}_i[K]$, and value function $\hat{V}(e_i[K])$ are stored. Finally, the weights $\mathbf{W}_a^{[n+1]}$ of the actor network are updated by using the finite-difference from (21), stored error $e_i[K]$, the control input $\mathbf{u}_i[K]$ and the approximated value function $\hat{V}(e_i[K])$. The hyper-parameters of Algorithm 1 are given in Table I. It is worth noting that, for the numerical simulation, the actor-critic enforces control policy as the agents converge to a vicinity of their respective centroids, i.e., as long as $\|e_i[\kappa + 1] - e_i[\kappa]\| > \varepsilon_v$. Subsequently, as the error falls below the tolerance ε_v , the system reaches a centroidal Voronoi configuration which is optimal with respect to the area coverage index.

V. NUMERICAL RESULTS

We consider a square-shaped workspace Ω of side 20 m. Four agents are initially placed randomly at (9.022, 4.82) m, (4.42, 15.64) m, (7.38, 16.22) m, and (5.81, 8.015) m. The risk density is described by the Gaussian function

$$\phi(\mathbf{q}, \kappa) = \sum_{j=1}^3 e^{(-\frac{1}{2}(\mathbf{q} - \mathbf{s}^{[j]}[\kappa]) \cdot \boldsymbol{\beta}^{-1}(\mathbf{q} - \mathbf{s}^{[j]}[\kappa]))} \quad (30)$$

about three randomly selected static locations at $\mathbf{s}^{[1]} = (3.05, 8.1)$ m, $\mathbf{s}^{[2]} = (5.1, 12.53)$ m, $\mathbf{s}^{[3]} = (8.82, 8.3)$ m corresponding to the peaks of the Gaussian. This formalism is often used to model target tracking scenarios, in which external

threats entering an area to be monitored are considered as risk sources. Hyper-parameters used in simulations are listed in Table I, with learning rates obtained through manual tuning via different trials until convergence times of the algorithm were considered acceptable. Although we did not carry out a systematic investigation to determine suitable training parameters, based on the heuristic tuning we can affirm that results are sensitive to learning rates.

The simulation is performed for $t = 20$ s. Algorithm 1 is executed by two computational loops. The boundary condition applied on the value function determines the length of the internal loop. To compare the performance of the proposed algorithm with a well-established framework for area coverage control, we also present results from Lloyd's algorithm [6] with time-invariant risk. Computational times for Lloyd's and for algorithm 1 are comparable when run on the same machine.

TABLE I
HYPER-PARAMETERS OF THE MODEL-FREE COVERAGE CONTROL

Hyper-Parameters	Size
Simulation time steps	2000
Inner-loop length K	4
Outer-loop length N_a	500
Critic learning rate α_c	5×10^{-3}
Actor learning rate α_a	5×10^{-4}
Value function convergence tolerance ε_v	5×10^{-2}
Time step T	0.01 s
Test duration t	20 s

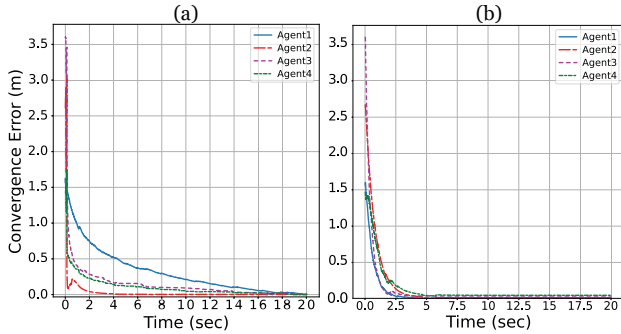


Fig. 2. Error dynamics. (a) Actor-critic algorithm, (b) Lloyd's algorithm

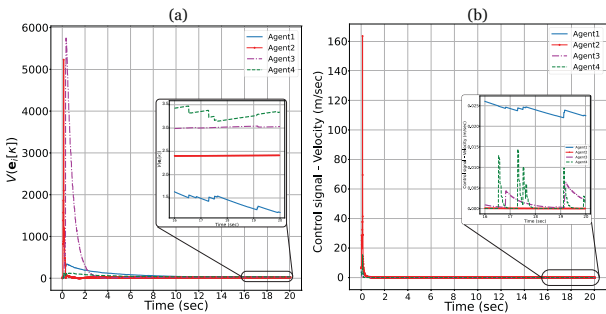


Fig. 3. (a) Evolution of the approximated value function, (b) Control policy.

The error dynamics is illustrated in Fig. 2, which shows asymptotic convergence to zero of the errors generated by the proposed control policy and from Lloyd's algorithm. The proposed approach shows an expected slower response compared to Lloyd's algorithm, which is implemented in continuous time. The value function is monotonically decreasing, as depicted in Fig. 3(a), with a control signal generated by the algorithm shown in Fig. 3(b). The evolution of critic and actor weights are illustrated in Fig. 4 and Fig. 5 respectively.

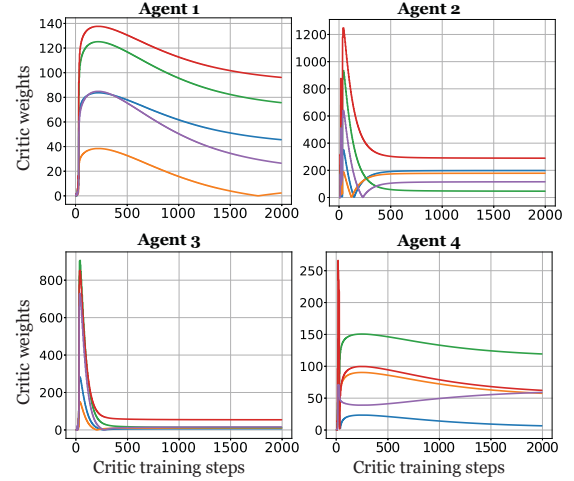


Fig. 4. Weights of the critic networks (time-invariant risk).

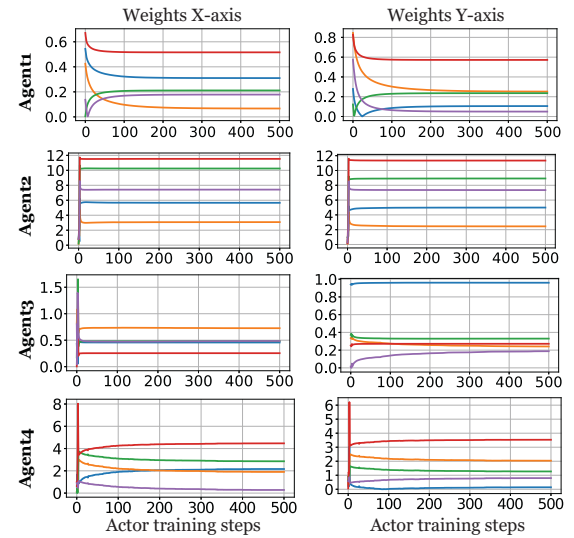


Fig. 5. Weights of the actor networks (time-invariant risk).

A performance comparison of Lloyd's algorithm and actor-critic in the time-invariant environment is illustrated in Fig. 6 by showing the evolution of the coverage metric with respect to the maximum coverage obtained by Lloyd's algorithm, where the faster convergence of Lloyd's algorithm emerges again, due to the fact that it generates the optimal trajectories in a continuous time implementation, climbing the gradient of the coverage metric.

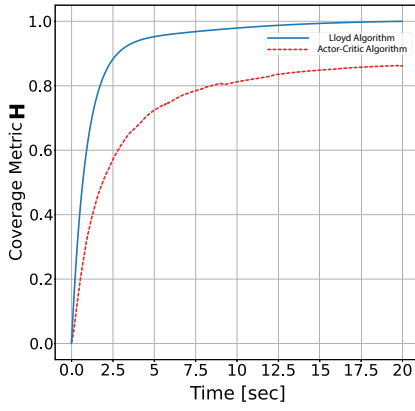


Fig. 6. A comparison of the obtained coverage metric H using actor-critic algorithm and Lloyd's algorithm. Both coverage metrics are normalized with respect to maximum coverage acquired by Lloyd's algorithm in a time-invariant environment.

VI. SUMMARY AND CONCLUSION

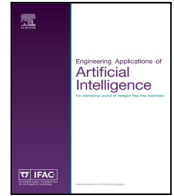
We presented an online temporal difference learning technology for addressing area coverage control problem with multi-agent systems in static environments. Bellman's principle of optimality was applied to a performance index based on the typical coverage metric that encodes the environment and the agents' sensing performance. The optimal solution is obtained by an adaptive model-free control policy based on reinforcement learning with an actor-critic structure. The adaptive control policy aims to maximize the coverage metric index while driving the system error dynamics to zero, which can be achieved as the agents converge to their respective Voronoi centroid. For time-invariant environments, the proposed framework is the approximate dynamic programming solution analogous to the classical Lloyd's algorithm for area coverage control. Ongoing and future work involves the extension to evolving environments, with drift dynamics inducing feed-forward controls that require additional neural network projections for a model-free implementation along the trajectories.

REFERENCES

- [1] F. Bullo, J. Cortes, and S. Martinez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, Applied Mathematics Series, 2009.
- [2] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-Agent Systems: A Survey," *IEEE Access*, vol. 6, pp. 28 573–28 593, 2018.
- [3] F. Soleymani, M. S. Miah, and D. Spinello, "Non-autonomous state-feedback to stabilize the error dynamics in time-varying area coverage control problems," in *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. IEEE, 2019, pp. 1–5.
- [4] M. S. Miah, B. Nguyen, A. Bourque, and D. Spinello, "Nonuniform Coverage Control With Stochastic Intermittent Communication," *IEEE Trans. on Automatic Control*, vol. 60, no. 7, pp. 1981–1986, Jul. 2015.
- [5] F. Lekien and N. E. Leonard, "Nonuniform Coverage and Cartograms," *SIAM J. Control Optim.*, vol. 48, no. 1, pp. 351–372, Feb. 2009.
- [6] J. Cortés, S. Martínez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Trans. on Robotics and Automation*, vol. 20, no. 2, pp. 243–255, Apr. 2004.
- [7] A. Maheshwari and N. Chand, "A Survey on Wireless Sensor Networks Coverage Problems," in *Proceedings of 2nd International Conference on Communication, Computing and Networking*, ser. Lecture Notes in Networks and Systems, C. R. Krishna, M. Dutta, and R. Kumar, Eds. Singapore: Springer, 2019, pp. 153–164.
- [8] S. Miah, A. Y. Panah, M. M. H. Fallah, and D. Spinello, "Generalized non-autonomous metric optimization for area coverage problems with mobile autonomous agents," *Automatica*, vol. 80, pp. 295–299, Jun. 2017.
- [9] F. L. Lewis, D. Vrabie, and V. L. Syrmos, *Optimal Control*. John Wiley & Sons, 2012.
- [10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, second edition ed., ser. Adaptive Computation and Machine Learning Series. Cambridge, Massachusetts: The MIT Press, 2018.
- [11] F. Soleymani and E. Paquet, "Deep graph convolutional reinforcement learning for financial portfolio management–deppocket," *Expert Systems with Applications*, vol. 182, p. 115127, 2021.
- [12] R. W. Beard, G. N. Saridis, and J. T. Wen, "Galerkin approximations of the generalized Hamilton-Jacobi-Bellman equation," *Automatica*, vol. 33, no. 12, pp. 2159–2177, 1997.
- [13] P. Werbos, "Approximate dynamic programming for real time control and neural modelling," *Handbook of intelligent control: neural, fuzzy and adaptive approaches*, pp. 493–525, 1992.
- [14] A. Heydari, "Revisiting approximate dynamic programming and its convergence," *IEEE transactions on cybernetics*, vol. 44, no. 12, pp. 2733–2743, 2014.
- [15] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 32, no. 2, pp. 140–153, 2002.
- [16] D. Vrabie and F. L. Lewis, "Adaptive optimal control algorithm for continuous-time nonlinear systems based on policy iteration," in *2008 47th IEEE Conference on Decision and Control*. IEEE, 2008, pp. 73–79.
- [17] D. Vrabie, K. Vamvoudakis, and F. Lewis, "Adaptive optimal controllers based on generalized policy iteration in a continuous-time framework," in *2009 17th Mediterranean Conference on Control and Automation*. IEEE, 2009, pp. 1402–1409.
- [18] M. Schwager, M. P. Vitus, S. Powers, D. Rus, and C. J. Tomlin, "Robust adaptive coverage control for robotic sensor networks," *IEEE Transactions on Control of Network Systems*, vol. 4, no. 3, pp. 462–476, 2015.
- [19] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2nd ed. John Wiley & Sons, LTM, 2000.
- [20] S. Lee, Y. Diaz-Mercado, and M. Egerstedt, "Multirobot Control Using Time-Varying Density Functions," *IEEE Trans. on Robotics*, vol. 31, no. 2, pp. 489–493, Apr. 2015.
- [21] F. Bullo, R. Carli, and P. Frasca, "Gossip coverage control for robotic networks: Dynamical systems on the space of partitions," *SIAM Journal on Control and Optimization*, vol. 50, no. 1, pp. 419–447, 2012.
- [22] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.
- [23] K. G. Vamvoudakis and F. L. Lewis, "Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem," *Automatica*, vol. 46, no. 5, pp. 878–888, 2010.

6.3 Optimal Non-Autonomous Area Coverage Control with Adaptive Reinforcement Learning

This article is under revision in the Engineering Applications of Artificial Intelligence journal, addressing the time-varying coverage control problem using an actor-feedforward-critic framework. The actor network approximates the optimal control policy to maximise the coverage metric approximated by the critic network. The feedforward network approximates the control policy that tracks the Voronoi centroids, helping to drive the convergence error to zero.



Optimal non-autonomous area coverage control with adaptive reinforcement learning

Farzan Soleymani^a, Md Suruz Miah^b, Davide Spinello^{a,*}

^a Department of Mechanical Engineering, University of Ottawa, Ottawa, ON, Canada

^b Department of Electrical and Computer Engineering, Bradley University, Peoria, IL, USA

ARTICLE INFO

Keywords:

Area coverage
Optimal control
Reinforcement learning
Multi-agent system

ABSTRACT

Area coverage control with multi-agent systems is formulated here in the framework of Bellman's optimality. In the literature, optimal configurations are obtained by Lloyd's algorithm, which iteratively converges the agents to centroidal Voronoi configurations that correspond to local maxima of a coverage metric, where metric sub-domains are assigned to each agent according to Voronoi tessellations. In this work, optimal tracking is achieved by an adaptive control policy using an actor-critic neural network-based reinforcement learning technique, which rewards actions that decrease a Lyapunov function based on the area coverage metric to drive the error dynamics to zero, and uses a feed-forward neural network to approximate the tracking control signal. The implementation of the area-coverage control is model-free in the sense that it relies on neural networks that interpolate local data gathered and shared by each agent, without relying on a global model of the system. We prove that optimality is achieved when the agents converge to Voronoi centroids, therefore maximizing the coverage metric, with the important implication that the obtained class of solutions is consistent with the ones obtained with Lloyd's algorithm and its extension to non-autonomous systems.

1. Introduction

Multi-agent systems are widely studied by the scientific and technological communities due to their capability to perform complex tasks in a distributed, scalable, and robust way. Examples of distributed spatial missions include harbour patrolling (Simetti and Cresta, 2007; Miah et al., 2014), environmental monitoring (Purvis et al., 2003; Elmokadem, 2019), perimeter surveillance (Pimenta et al., 2013; Zhang et al., 2013; Zhu and Ren, 2020), search and rescue missions (Hu et al., 2013), and cooperative estimation (Spinello and Stilwell, 2014). Fuzzy-based adaptive control techniques have been developed for multi-agent systems to handle uncertainties associated to the parametric complexity of some of the models. A finite-time robust precision bipartite consensus tracking for a multi-agent system with various uncertainties, using finite-time performance function for a fuzzy fault-tolerant distributed cooperative control scheme, is obtained by (Liang et al., 2022). The time-varying formation control for nonstrict-feedback nonlinear multi-agent systems with input saturation is studied in Cao et al. (2022), where a high-gain fuzzy logic system-based state observer is employed to estimate the unmeasured states.

An important class of applications can be classified as area coverage, in which a team of mobile agents spatially configure themselves over an area of interest to maximize a coverage performance index. Therefore, area coverage control problems are typically posed as optimization

problems, with the coverage performance index encoding the collective sensing performance of the agents (typically with spatial dependence) and a scalar field defined on the workspace (Soleymani et al., 2019). The scalar field, called herein risk density, represents a quantity of interest weighting the decisions of the agents with respect to how to distribute in the domain. The implementation of area coverage control algorithms requires the knowledge of the state of neighbour agents and the knowledge of the risk density/environment, which can be challenging to achieve in real-time implementation of robotic systems, especially when the environment itself is evolving (Miah et al., 2015; Cortés et al., 2004; Kantaros et al., 2015; Cortes et al., 2002; Lekien and Leonard, 2009; Lee et al., 2015; Miah et al., 2017).

Different approaches to area coverage control in evolving environments can be found for example in Lee et al. (2015), Miah et al. (2017), Diaz-Mercado et al. (2017) and Xu and Diaz-Mercado (2020). A distributed multi-agent area coverage algorithm is proposed in Xiao et al. (2020) to obtain a global optimal coverage path using reinforcement learning. Two scenarios are reflected on this approach; first, the agents may broadcast their communications over the whole target area, enabling them to know the whole dynamic coverage process and run a global optimal coverage strategy; second, communications may be spatially localized with respect to the whole workspace, in which case a local optimal coverage strategy has to be devised. The coverage

* Corresponding author.

control problem is addressed in [Duca and Bugeja \(2020\)](#) via a partitioning approach which finds the optimal segments of the environment considering energy constraints on a finite number of robots. Another novel partitioning method is introduced in [Papatheodorou et al. \(2018\)](#) called Additively Weighted Guaranteed Voronoi diagram (AWGV). This approach implements a gradient-based distributed control considering the positioning uncertainty of a group of heterogeneous agents and their sensing performance. This method aims to conserve the energies of the robots over the path travelled by each. A model predictive approach is proposed in [Carron and Zeilinger \(2020\)](#) to address the coverage control problem with nonlinear dynamics and constraints on states and inputs. A cooperative control problem is studied in [Basu and Yoon \(2020\)](#) considering a linear multi-agent system under the constraint that the agents cannot independently estimate the exosystem states. This problem is addressed by developing a distributed observer on the exosystem states. An indirect/direct learning method is proposed in [Liu et al. \(2021\)](#) to estimate the risk density function by using sparse sensory data and interpolation via Gaussian basis functions. Connectivity constraints in coverage control problems to avoid local minima of the coverage metric are discussed in [Meng and Kan \(2021\)](#) and [Kawajiri et al. \(2021\)](#). Optimal coverage with obstacle avoidance in the presence of time-varying uncertainties is presented in [Bai et al. \(2021\)](#).

A model-free, adaptive approach can be beneficial in allowing the use of sensor data to implement control policies based on machine learning algorithms that do not rely on the system model. With this motivation, we formulate the area coverage problem in the framework of Bellman's optimality ([Lewis et al., 2012](#), Ch. 6). The value function is formed by the additive contributions of a positive definite term that depends on the coverage metric performance index, and of a quadratic term of the control input. Optimality in Bellman's sense is achieved by an adaptive control policy based on reinforcement learning with an actor-critic structure that iterates the policy (control input) searching for optimality ([Chu et al., 2008](#); [Kiumarsi and Lewis, 2015](#); [Lawhead and Gosavi, 2019](#); [Pane et al., 2019](#); [Zhou and Ho, 2022](#)). The policy iteration method is composed of two stages, namely policy evaluation and policy improvement ([Sutton and Barto, 2018](#); [Serrano, 2022](#); [Soleymani and Paquet, 2020, 2021](#)).

In policy iteration methods, given a control policy, the current cost is estimated in the direction of true cost value, while the control policy is updated following a greedy approach. The application of policy iteration methods for nonlinear systems has been investigated in [Beard et al. \(1997\)](#), where the policy evaluation step is defined in terms of nonlinear Lyapunov equations. A conventional method for solving optimization problems with finite dimensional states is dynamic programming ([Bertsekas, 1995](#)). However, a drawback of this method is the curse of dimensionality, i.e., the performance degradation as the dimension of data increases. To overcome this issue, approximate dynamic programming was proposed in [Werbos \(1992\)](#) and later modified in [Heydari \(2014\)](#). Approximate dynamic programming is a value iteration method, which is performed without requiring any initial admissible control, aiming to find the optimal value function and calculate the control policy in one step. Both value iteration and policy iteration methods belong to generalized policy iteration methods ([Lewis et al., 2012](#)).

Policy iteration methods are implemented using two neural networks, namely actor and critic. The critic network approximates the value function while the actor network approximates the control policy. An adaptive critic network for discrete-time is discussed in [Werbos \(1989\)](#) which inspired several studies on continuous-time adaptive critic networks ([Baird, 1994](#); [Hanselmann et al., 2007](#); [Vrabie et al., 2009a](#)). An online policy iteration algorithm is proposed in [Vrabie and Lewis \(2008\)](#), [Vrabie et al. \(2009b\)](#) based on a sequential update of the critic and actor neural networks, i.e., networks are updated one at a time while the others are kept fixed. These methods can be implemented without explicit knowledge of the internal system

dynamics. An adaptive sensor coverage over an unknown environment has been formulated in [Schwager et al. \(2015\)](#), using a robust learning method that drives a team of robots to perform various distributed sensing tasks. The aim of robots is to converge to positions that locally minimize a cost function, which is defined as the sensing quality of the network.

The contribution of this paper consists on a model-free formulation for area coverage control problems over evolving environments, and the proof of convergence to centroidal Voronoi configurations. The model-free framework presented herein uses an online actor-critic structure with the addition of a feed-forward neural network to approximate the tracking control action. In this iterative cascaded process, the critic neural network approximates the value function encoding the coverage metric and the control effort, the actor neural network approximates the regulator control policy using the approximated value function in discrete time, and the feed-forward neural network approximates the feed-forward control policy using the approximated value function and the approximated regulator. We prove that the control policy enforced by the actor-network solves the optimal regulator control problem adaptively. In this way, area coverage optimality is reached in the sense of a Bellman's formulation, with minimizing trajectories tracking the centroids of Voronoi cells, as in the standard coverage control. The established equivalence between the two optimal control frameworks – model-based and model-free – has important implications for the implementation in real systems, where online model-free adaptive controls may be preferable in highly uncertain environments, and varying process dynamics.

Our proof of optimality builds on the work ([Liu and Wei, 2014](#)), where it is shown that a policy iteration algorithm drives to optimality a nonlinear system evolving in discrete time, under the hypotheses of controllability and existence of admissible controls to initiate the iterators. We constructively prove the existence of a class of admissible controls for the system's nonlinear error dynamics. The error convergence around the origin follows, implying that the online reinforcement learning iterator solves the area coverage regulator control problem consistently with classical model-based algorithms reducible to Lloyd's algorithm and generalizations. The additive decomposition of the control into regulator and feed-forward or tracking parts allows to extend the analysis to non-autonomous systems induced by time evolving environments.

The rest of the paper is organized as follows: In Section 2, the optimal area coverage control problem is briefly discussed to introduce necessary nomenclature and to provide the background. The coverage control is formulated as an optimal control in Section 3. Theoretical results presented in Section 4 show that a policy iterator drives the agents' dynao876543 mics to optimality, corresponding to centroidal Voronoi partitions of the workspace, establishing the equivalence with classical area coverage control results. The reinforcement learning implementation of the optimal tracking problem with three neural network approximators is presented in Section 5. Simulation results of the proposed framework are reported in Section 6, showing the capability of the algorithm in minimizing the error in finite time while tracking time evolving centroidal, and therefore optimal, Voronoi configurations. Conclusion and summary are presented Section 7.

2. Optimal area coverage with mobile agents

In a standard area coverage control framework, a group of mobile agents is deployed in a spatial environment. [Fig. 1](#) shows a possible configuration of a group of mobile agents operating in a two-dimensional area, which is partitioned using the Voronoi tessellation technique.

A scalar distribution on the environment, typically referred to as risk function, quantifies the relative importance attributed to points in the workspace with respect to the resources to be allocated, which in this case translates into more agents assigned to a region with higher risk. We consider the possibility of exogenous agents/targets navigating the workspace, which are modelled as risk sources as portrayed in [Fig. 1](#). Area coverage control algorithms account for non-uniform risk by attracting more agents towards high-risk zones.

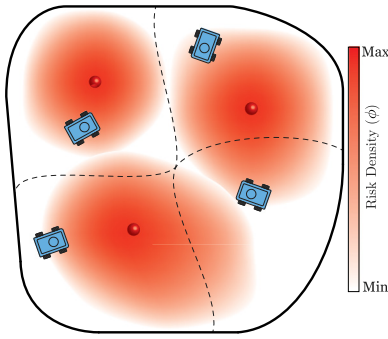


Fig. 1. Schematics of coverage control problem in presence of multiple time varying risk densities.

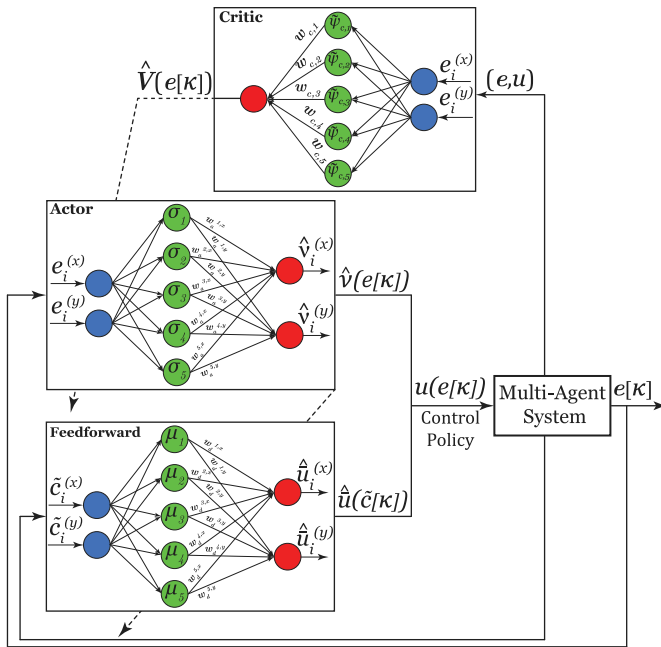


Fig. 2. The proposed actor-critic feed-forward scheme.

2.1. Agent model and area coverage setup

Without loss of generality, we consider a set of n kinematic agents with first-order dynamics given by

$$\dot{p}_i(t) = u_i(t), \quad (1)$$

where $u_i(t) \in \mathbb{R}^2$ is the motion control (velocity) input of the i th agent and $p_i(t) \in \mathbb{R}^2$ is its position vector at time $t \geq 0$. From now on, we shall drop the time argument (t) for clarity and when no ambiguity arises. In the specialized literature, the area coverage control is formulated as an optimal control problem aiming at finding an agents' spatial distribution that locally maximizes the coverage metric H defined by

$$H(p, \mathcal{V}, t) = \sum_{i=1}^l \int_{\mathcal{V}_i} f(r_i) \phi(q, t) d\Omega, \quad (2)$$

where $q \in \Omega \subset \mathbb{R}^2$ is a point in the workspace Ω , $f: \mathbb{R} \rightarrow \mathbb{R}$ is the agents' sensing performance, assumed to be strictly decreasing in the argument $r_i = \|q - p_i\|$, p denotes the collection of the position vectors for l agents, i.e., $p = [p_1^T, p_2^T, \dots, p_l^T]^T \in \mathbb{R}^{2l}$, and $\mathcal{V} = (\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_l)$ is the collection of Voronoi cells $\mathcal{V}_i$ assigned to each agent i , and defined as the set of points in Ω that are best sensed by agent i with respect to the function f (Okabe et al., 2000):

$$\mathcal{V}_i := \{q \in \Omega | f(r_i) \geq f(r_j), \forall j \in \{1, \dots, l\}, j \neq i\}. \quad (3)$$

By introducing the generalized density

$$\tilde{\phi}(q, p_i, t) = -2\phi(q, t) \frac{\partial f(r_i)}{\partial r_i^2}, \quad (4)$$

one can define the generalized mass and generalized Voronoi centroids by

$$\tilde{m}_i = \int_{\mathcal{V}_i} \tilde{\phi}(q, p_i, t) d\Omega, \quad (5a)$$

$$\tilde{c}_i = \frac{1}{\tilde{m}_i} \int_{\mathcal{V}_i} q \tilde{\phi}(q, p_i, t) d\Omega. \quad (5b)$$

The implementation of an area coverage control algorithm based on the metric H requires the knowledge of the time-varying density $\phi(q, t)$ and of the state of a subset of the other agents in the group to be able to compute the Voronoi tessellation and the generalized centroids. Local optimality in terms of agents' spatial distribution is reached by convergence to the set of generalized centroids, with induced centroidal Voronoi tessellation as area partition (Bullo et al., 2009; Miah et al., 2015).

2.2. Error dynamics

The error between agent i and the related centroid of the i th Voronoi cell is given by

$$e_i = \tilde{c}_i - p_i \quad (6)$$

By taking the time derivative and substituting from (1) and (6), the following error dynamics is obtained

$$\begin{aligned} \dot{e}_i &= \dot{\tilde{c}}_i - \dot{p}_i = \nabla_{p_i} \tilde{c}_i \dot{p}_i - \dot{p}_i + \frac{\partial \tilde{c}_i}{\partial t} \\ &= (\nabla_{p_i} \tilde{c}_i - I) \dot{p}_i + \frac{\partial \tilde{c}_i}{\partial t} \end{aligned} \quad (7)$$

where ∇_{p_i} is the gradient with respect to p_i and I is the identity matrix of appropriate dimension. To rewrite the error dynamics more compactly, we define the following functions

$$F(e_i, t) := \frac{\partial \tilde{c}_i}{\partial t}, \quad (8a)$$

$$G(e_i, t) := \nabla_{p_i} \tilde{c}_i - I, \quad (8b)$$

from which it follows that (7) can be rewritten in the standard affine input form

$$\dot{e}_i = F(e_i, t) + G(e_i, t) u_i. \quad (9)$$

where $u_i \in \mathbb{R}^2$ acts as the input to the non-autonomous dynamics of the error e_i . Under the condition that the domain Ω is bounded, the terms $\partial \tilde{c}_i / \partial t$ and $\nabla_{p_i} \tilde{c}_i - I$ are bounded (Lee et al., 2015), and therefore the error dynamics is bounded as long as the control input is finite.

2.3. Tracking control problem

Since the error dynamics (9) is non-autonomous, a non-zero control input $\bar{u}_i$ is generally necessary to keep the error at zero. This term generally depends on the reference signal, in this case, $\tilde{c}$, and it is referred as the tracking part of the control action. The tracking input $\bar{u}_i$ can be obtained from the steady-state error dynamics at the origin

$$0 = F(0, t) + G(0, t) \bar{u}_i \quad (10)$$

from which it follows that the input necessary to keep the system in this state is given by

$$\bar{u}_i = -G(0, t)^{-1} F(0, t). \quad (11)$$

This requires the knowledge of the drift dynamics F and of the input dynamics G .

The optimal control problem can be conveniently decomposed into a regulation problem for the error and a tracking part with the additive

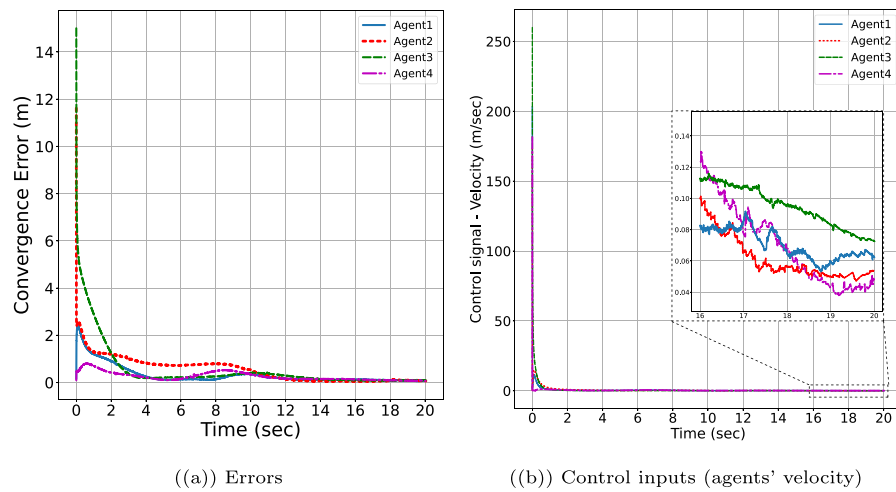


Fig. 3. Evolution of the errors and control inputs from simulations of the non-autonomous area coverage induced by time varying risk.

decomposition of the control input into $u_i = v_i + \bar{u}_i$, splitting the signal into the transient part v_i which drives the agents to the centroids of the corresponding Voronoi cells, and the tracking part $\bar{u}_i$, which keeps the systems in optimal centroidal configurations. Since optimal configurations are centroidal with $e_i = 0$, controls v_i can also be interpreted as the ones responsible for driving the system to optimality in the sense of the area coverage metric H , whereas controls $\bar{u}_i$ are responsible for keeping the system in a configuration corresponding to optimal area coverage. From the definition of $\bar{u}_i$ it follows that $v_i = 0$ when $e_i = 0$. By recalling that the sensing performance f_i depends on the argument $r_i^2 = \|q - p_i\|^2$, from the definition of generalized centroid (5b) we have

$$\begin{aligned} \nabla_{p_i} \tilde{c}_i &= -\frac{1}{\tilde{m}_i^2} \left[\nabla_{p_i} \tilde{m}_i \right] \int_{V_i} \tilde{\phi}_i q d\Omega + \frac{1}{\tilde{m}_i} \int_{V_i} q \left[\nabla_{p_i} \tilde{\phi}_i \right] d\Omega \\ &= \frac{1}{\tilde{m}_i} \int_{V_i} 4\phi \frac{\partial^2 f_i}{(\partial r_i^2)^2} (q - \tilde{c}_i) (q - p_i)^\top d\Omega = \frac{J_i}{\tilde{m}_i} \end{aligned} \quad (12)$$

after introducing the inertia matrix $J_i = \int_{V_i} 4\phi \frac{\partial^2 f_i}{(\partial r_i^2)^2} (q - \tilde{c}_i) (q - p_i)^\top d\Omega$. By considering the sensing performances to be described by the radially symmetric sensing functions $f_i = \alpha \exp(-\beta r_i^2)$ (Bullo et al., 2012; Miah et al., 2015, 2017), where $\alpha > 0$ and $\beta > 0$ are shape parameters, then we have $\partial^2 f_i / (\partial r_i^2)^2 = \beta^2 f_i$, and the moment of inertia matrix becomes

$$J_i = \alpha \beta^2 \int_{V_i} \phi \exp(-\beta \|q - p_i\|^2) (q - \tilde{c}_i) (q - p_i)^\top d\Omega \quad (13)$$

which is symmetrical at a centroidal configuration $p_i = \tilde{c}_i$, where $\nabla_{p_i} \tilde{c}_i$ can be interpreted as the gyration tensor for the specific cell. At a centroidal steady-state configuration, the steady-state tracking input can be locally computed by explicitly writing definition (10) in terms of the quantities just introduced

$$\bar{u}_i = - \left(\frac{J_i}{\tilde{m}_i} - I \right)^{-1} \frac{\partial \tilde{c}_i}{\partial t}, \quad (14a)$$

$$\begin{aligned} \frac{\partial \tilde{c}_i}{\partial t} &= \frac{1}{\tilde{m}_i} \int_{V_i} \frac{\partial \tilde{\phi}_i}{\partial t} (q - \tilde{c}_i)^\top d\Omega \\ &= -\frac{2\alpha}{\tilde{m}_i} \int_{V_i} \frac{\partial \phi}{\partial t} \exp(-\beta \|q - p_i\|^2) (q - \tilde{c}_i) d\Omega \end{aligned} \quad (14b)$$

If the risk function is stationary, $\frac{\partial \phi}{\partial t} = 0$, the tracking control input is zero, and the problem reduces to the regulation of the error at the

origin. The case of stationary workspace was first solved by the classical Lloyd's algorithm, which converges the agents to a set of stationary centroids that maximize the coverage metric in a continuous time setting (Lloyd, 1982; Cortes et al., 2002).

Below, we use an approximate dynamic programming approach to achieve local optimality with time-varying centroids, where an actor-critic structure regulates the error dynamics to the origin, while an additional neural network, called the feed-forward neural network, approximates the feed-forward control $\bar{u}_i$ to track centroids' trajectories. The inverse $G^{-1} = (\nabla_{p_i} \tilde{c}_i - I)^{-1}$ exists if the risk density is log-concave, and it is continuously differentiable in its arguments (Lee et al., 2015). These conditions are satisfied by the settings in this work, where the sensing functions f_i and the risk density ϕ are radially symmetric exponential functions strictly decreasing in their argument, as detailed below and in Section 6.

The reinforcement learning method in this paper is equivalent to Lloyd's algorithm when the risk function is not time-varying. With the time-varying risk function, the reinforcement learning algorithm regularizes the error around the origin, therefore driving the system to optimality in the sense of the area coverage metric H .

3. Area coverage as Bellman's optimality

The action space of reinforcement learning methods can be classified into discrete and continuous. Due to the dimensionality of the action space, parameter tuning is generally an easier process for the discrete case. In addition, discrete policies outperform continuous policies for problems with complex dynamics (Tang and Agrawal, 2019). Therefore, herein we adopt a discrete-time implementation for the actor-critic and feed-forward formulation that follows. Previously introduced variables will be indicated with the same symbol, and the integer κ will be used to label the time instant as $t_\kappa = \kappa T$, where T is a time interval assumed to be constant for the time being. Thus, the error dynamics from (9) can be formulated in discrete time as

$$e_i[\kappa + 1] = \bar{F}(e_i[\kappa], \kappa) + \bar{G}(e_i[\kappa], \kappa) u_i[\kappa]. \quad (15)$$

where $\bar{F} = T F + e_i$ and $\bar{G} = T G$.

The optimal tracking problem is separated into an optimal regulator problem for the homogeneous action v_i , and into the determination of the feed-forward control action $\bar{u}_i$, so that the optimal control u_i^* is given by (Kiumarsi and Lewis, 2015; Dierks and Jagannathan, 2009)

$$u_i^* = v_i^* + \bar{u}_i. \quad (16)$$

In optimal control, the value function is the index of optimality. In area coverage problems with continuous-time dynamics in which each agent

has full knowledge of the workspace through the risk function ϕ , and has full knowledge of the state of the group of agents through their states $\mathbf{p}$, the positive definite function H^{-1} was shown to be a Lyapunov function along whose trajectories the agents maximize the coverage metric H (Miah et al., 2015). We leverage this idea to formulate the optimal regulator problem for $\mathbf{v}$ by posing the value function as

$$V(e[k]) = \sum_{j=k}^{\infty} \gamma^{j-k} \left(H(e[j], j)^{-1} + \frac{1}{2} \mathbf{v}^T[j] \mathbf{R} \mathbf{v}[j] \right) \quad (17)$$

where H denotes the coverage metric from (2), $e[k] \in \mathbb{R}^{2l}$ is the augmented error vector, and $\mathbb{R}^{2l \times 2l} \ni \mathbf{R} = \mathbf{R}^T > 0$ is a symmetric positive definite matrix, penalizing the action $\mathbf{v} = (v_1, v_2, \dots, v_l)$, and having the block diagonal structure $\mathbf{R} = \text{diag}(\mathbf{R}_1, \dots, \mathbf{R}_l)$, with $\mathbf{R}_i \in \mathbb{R}^{2 \times 2}$. The parameter $0 < \gamma \leq 1$ is the discount factor, that establishes a balance between current and future values of V (Lewis et al., 2012, chapter 11.5). For the coverage metric H , the notation has been modified by dropping the explicit dependence on $\mathcal{V}$, and by substituting $\mathbf{p} = \mathbf{e} + \tilde{\mathbf{e}}$ from (6), with the non-autonomous dependency on $\tilde{\mathbf{e}}$ expressed through the time index j . Since $H > 0$, the value function is positive definite as usually prescribed in optimization problems (Lewis et al., 2012, chapter 11.5). Explicit writing of the first term in the summation in (17) gives

$$V(e[k]) = H(e[k], k)^{-1} + \frac{1}{2} \mathbf{v}^T[k] \mathbf{R} \mathbf{v}[k] + \sum_{j=k+1}^{\infty} \gamma^{j-k} \left(H(e[j], j)^{-1} + \frac{1}{2} \mathbf{v}^T[j] \mathbf{R} \mathbf{v}[j] \right) \quad (18)$$

By using the definition of value function, the right hand side of (18) can be rewritten as

$$V(e[k]) = H(e[k], k)^{-1} + \frac{1}{2} \mathbf{v}^T[k] \mathbf{R} \mathbf{v}[k] + \gamma V(e[k+1]) \quad (19)$$

Based on temporal difference learning (Sutton, 1988), (19) is a deterministic Bellman's equation (Lewis et al., 2012). The temporal difference learning method offers a solution to Bellman's equation without requiring knowledge of the true system's dynamics (15), but rather relying on its estimation; in other words, it is a model-free approach. Considering the discrete time system in (15), the optimal control problem can be formulated as: given a set of admissible control policies (actions) $\mathbf{v} \in \mathbb{V}$ and the value function (17), find an optimal control policy that minimizes the performance index (17) of the system (15) (Vamvoudakis and Lewis, 2010). Following Bellman's principle of optimality (Lewis et al., 2012, chapter 11.5), the optimal control inputs that maximize the performance metric H are found by solving the minimization problem

$$V^*(e[k]) = \min_{\mathbf{v} \in \mathbb{V}} \left\{ H(e[k], k)^{-1} + \frac{1}{2} \mathbf{v}^T[k] \mathbf{R} \mathbf{v}[k] + \gamma V^*(e[k+1]) \right\} \quad (20)$$

Eq. (20) is known as Bellman's optimality equation or discrete-time Hamiltonian–Jacobi–Bellman equation (Lewis et al., 2012). A control policy corresponding to an optimal solution of (20) is given by

$$\mathbf{v}^*(e[k]) = \argmin_{\mathbf{v} \in \mathbb{V}} \left(H(e[k], k)^{-1} + \frac{1}{2} \mathbf{v}^T[k] \mathbf{R} \mathbf{v}[k] + \gamma V^*(e[k+1]) \right) \quad (21)$$

Therefore, the search of an optimal control policy is equivalent to a root searching problem in terms of the gradient of (19)

$$\frac{\partial V^*(e[k])}{\partial \mathbf{v}[k]} = - \frac{\frac{\partial H(e[k], k)}{\partial \mathbf{v}[k]}}{(H(e[k], k))^2} + \mathbf{R} \mathbf{v}[k] + \gamma \frac{\partial V^*(e[k+1])}{\partial \mathbf{v}[k]} = \mathbf{0} \quad (22)$$

This expression can be simplified by considering that

$$\frac{\partial H(e[k], k)}{\partial \mathbf{v}_i[k]} = \frac{\partial H(e[k], k)}{\partial e_i[k]} \frac{\partial e_i[k]}{\partial \mathbf{v}_i[k]}, \quad (23a)$$

$$\frac{\partial H(e[k], k)}{\partial e_i[k]} = m_i e_i[k], \quad (23b)$$

$$\frac{\partial e_i[k]}{\partial \mathbf{v}_i[k]} = \mathbf{0} \quad (23c)$$

where the last expression accounts for the fact that the current error $e_i[k]$ does not depend on the current control input $\mathbf{v}_i[k]$. The explicit calculation of the gradient of the value function from (19) gives the optimal regulator

$$\mathbf{v}_i^*[k] = -\gamma \mathbf{R}_i^{-1} \frac{\partial V(e[k+1])}{\partial \mathbf{v}_i[k]} = -\gamma \mathbf{R}_i^{-1} \mathbf{G}(e[k]) \frac{\partial V(e[k+1])}{\partial e_i[k+1]} \quad (24)$$

and the optimal control is given by (16).

By adopting a reinforcement learning approach to solve Bellman's optimality equation (20), we overcome the necessity of knowing the system's drift dynamics $\bar{\mathbf{F}}$ in Eq. (15) by adopting an actor–critic structure with neural network approximators respectively for the control (actor) and for the value function (critic). This solves the regulation problem of driving the error to the origin. However, the tracking part of the control needs to be calculated at each updating step. To consistently avoid the use of the drift dynamics, a third neural network will be used to approximate the feed-forward control $\bar{\mathbf{u}}_i$ (Dierks and Jagannathan, 2009). In this way, optimal control policies (16) can be approximated in a model-free fashion.

4. Convergence results for the optimal regulator

A policy iteration algorithm constructs an iterative control law $\omega^j(e[k])$, $j = 1, 2, \dots$, to drive the error $e[k]$ from an initial value $e[0]$ to the equilibrium $\mathbf{0}$, which corresponds to a local optimum of the area coverage problem as discussed above. The iterative control ω^j is an approximation of the optimal control policy $\mathbf{v}^*$, from which we can construct the sub-optimal iterative performance index $W_j(e[k])$ satisfying the temporal difference equation

$$W_j(e[k]) = H(e[k], k)^{-1} + \frac{1}{2} \omega^{jT} \mathbf{R} \omega^j + \gamma W_j(e[k+1]) \quad (25)$$

Therefore, $W_j(e[k])$ is an approximation of $V(e[k])$, and the policy iteration algorithm updates the iterative control at a fixed $e[k]$ according to

$$\omega^{j+1}[k] = \argmin_{\omega^j[k]} \left\{ H(e[k], k)^{-1} + \frac{1}{2} \omega^{jT} \mathbf{R} \omega^j + \gamma W_j(e[k+1]) \right\} \quad (26)$$

Since the value $W_j(e[k]) \neq V(e[k])$, it is important to determine if the iterative value W_j and associated policy ω^j converge to their optimal versions $V^*(e[k])$ and $\mathbf{v}^*[k]$ for $j = 0, 1, \dots$. In the remaining part of this section, we will proceed to define admissible controls with respect to the performance index (18) that stabilize the error dynamics (15), therefore driving the system to local optima of the coverage metric H , and show that at least one admissible control exists. By initializing the policy iteration algorithm with the admissible control, the policy iteration asymptotically approximates the optimal policy and the optimal value for all k , that is, as $j \rightarrow \infty$ we have $\omega^j(e[k]) \rightarrow \mathbf{v}^*(e[k])$ and $W_j(e[k]) \rightarrow V^*(e[k])$.

Definition 1 (Liu and Wei, 2014). A control policy $\mathbf{v}(e[k])$ is defined to be admissible with respect to the performance index (18) if it is continuous in Ω , $\mathbf{v}(\mathbf{0}) = \mathbf{0}$, and $V(e[0])$ is finite for any initial condition $e[0]$.

With the following Lemma, we show that the system admits at least one admissible control policy, and we give an explicit expression in terms of the dynamics.

Lemma 1. Assume that the sensing functions f_i and the risk density ϕ in (2) are smooth in Ω . Then the control action

$$\mathbf{v}_i(e_i[k]) = -\mathbf{G}(e_i[k])^{-1} (\bar{\mathbf{F}}(e_i[k], k) + \lambda e_i[k]) - \bar{\mathbf{u}}_i[k] \quad (27)$$

is admissible with respect to (18) for any constant $\lambda > 0$.

Proof. From the definition of $\mathbf{F}$ in (8a) and from $\bar{\mathbf{F}} = \mathbf{e} + T\mathbf{F}$, it follows that $\mathbf{v}(e[k])$ is continuous under the hypotheses of smoothness of the sensing functions f_i and of the risk function ϕ . Also, from the

definition (14) of the feed-forward control, and from Definition (27) we have $v(0) = 0$. To complete the proof, we observe that the smoothness of f_i and ϕ imply the integrability of the coverage metric H (2), from which it follows that the value function (18) is bounded for all initial conditions $e[0] = \tilde{e}[0] - p[0]$.

Knowing that the policy iteration algorithm (25) can be initialized with an admissible control policy, we can prove that the policy iteration asymptotically converges to the optimum at every time step, and therefore it generates a sequence of iterators converging to optimality in the sense of Bellman.

Theorem 2. *Let $W_j(e[k])$ and $\omega^j(e[k])$ respectively be the value function and the control policy generated by the policy iteration algorithm (25). Then as $j \rightarrow \infty$, $W_j(e[k])$ converges to the optimal performance index $V^*(e[k])$ from (21) and the control policy $\omega^j(e[k])$ converges to the optimal control $v^*(e[k])$ from (21).*

Proof. As established in Lemma 1, we can initialize the policy iteration algorithm with an admissible control policy $\omega^0(e[k])$, from which it follows Liu and Wei (2014, Lemma 3.1) that the iterator $\omega^j(e[k])$ stabilizes the error dynamics (15), and that Liu and Wei (2014, Theorem 3.1) the performance index of the policy iteration $W_j(e[k])$ is a monotonically decreasing sequence with respect to the policy iteration index j , that is $W_{j+1}(e[k]) \leq W_j(e[k])$. As a corollary of this result, it follows that by initializing the policy iteration algorithm with an admissible control policy ω^0 , then ω^j is admissible for $j > 0$. Finally Liu and Wei (2014, Theorem 3.2), the iterative performance index and the iterative control policy asymptotically converge to the optimal ones

$$\lim_{j \rightarrow \infty} W_j(e[k]) = V^*(e[k])$$

$$\lim_{j \rightarrow \infty} \omega^j(e[k]) = v^*(e[k])$$

Therefore, the policy iteration algorithm with initialized admissible control converges to Bellman's optimality of the area coverage problem.

Finally, we investigate whether or not the policy iteration drives the error dynamics to zero, i.e., optimal area coverage configurations. This is a direct consequence of the fact that the control law generated by the policy iteration algorithm is admissible when initialized by an admissible control law, that we know it exists for this class of problems since we can construct at least one, as shown in Lemma 1. Therefore we have the following

Corollary 3. *The policy iteration algorithm (25) initialized with an admissible control law, regularizes the error dynamics at the origin, that is $\lim_{k \rightarrow \infty} e[k] = 0$.*

The important implication of this set of results is that the policy iteration algorithm (25)–(26) solves the area coverage problem by maximizing the area coverage H , therefore solving the area coverage problem in the same sense as the class of continuous-time algorithms proposed in the literature for both autonomous and non-autonomous metrics. In the case of a time-invariant risk density function, the policy iteration algorithm reconstructs Lloyd's algorithm solutions by finding fixed points of the coverage metric. The crucial point is that the policy iteration algorithm can be implemented in a model-free fashion (except for the initialization), as described in the next section where we use actor-critic neural network approximators and a feed-forward neural network.

5. Neural network approximations of the optimal tracking problem

In this section, approximated optimal tracking control solutions are obtained by using conventional reinforcement learning and neural network approximations to solve the temporal difference learning form of Bellman's equation discussed above. The aim of temporal difference

learning formulations is to obtain optimal trajectories with respect to a given performance index forward in time, relying on measurements along the system's trajectories. This may be achieved by adopting an actor-critic architecture without knowledge of the drift dynamics F and thus, in this case, without knowledge of the non-autonomous term $\frac{\partial c}{\partial t}$ (see (8a)), which according to (14b), depends on the rate of the time-varying part of the risk function $\frac{\partial \phi}{\partial t}$. This ultimately describes the uncontrolled evolution of the environment.

A reinforcement learning approach allows to address a very important aspect related to the implementability of non-autonomous area coverage control algorithms, which is the estimation of the evolution of the environment (Miah et al., 2015, 2017). To implement the algorithm without the knowledge of the input dynamics G , one may adopt a Q-learning formulation; however, we present below a finite difference scheme approximation to numerically calculate the derivative $\partial V / \partial v_i$ in (24). This requires storing two values of the control and value function, which is considered as part of the approximation:

$$\frac{\partial V(e[k+1])}{\partial v_i[k]} \simeq \frac{V(e[k+1]) - V(e[k])}{v_i[k] - v_i[k-1]} (v_i[k] - v_i[k-1]) \quad (28)$$

where $v_i = \|v_i\|$, and the singularity at the denominator is avoided by the fact that $(v_i[k] - v_i[k-1]) / (v_i[k] - v_i[k-1])$ has unit norm.

The policy iteration for the optimal regulator is schematized in Fig. 2, illustrating how the actor and critic neural network operate in tandem respectively in approximating the control policy v_i^* (actor), and in estimating the value function V^* (critic) to evaluate the effect of a given policy and to reward it accordingly. Since the control policy is obtained based on an estimation of the value function V , the critic network must converge faster than the actor network. The loop is completed by using the converged actor and critic gains to iterate the feed-forward adaptive gains, allowing to update the control action according to approximations of (16).

A crucial aspect of implementing the algorithm is the definition of the necessary measurements, which are ultimately dictated by the quantities that are necessary in the implementation. Each agent needs to calculate its own Voronoi cell at each updating time. Following the comprehensive discussion in Cortés et al. (2004), this computation depends on the agent's state p_i and on the states of the Voronoi neighbours (agents whose Voronoi cell shares an edge or a vertex). This knowledge can be acquired based on passive sensing (for example, onboard sensors measuring distance and bearings with the neighbours) or by direct communication (data packets sharing). Here, we assume that the necessary locational information to compute Voronoi cells is available to each agent when necessary for computations, leaving for future work more sophisticated treatments that could include the communication network topology encoded in the performance index and possible disturbances associated with stochastic packet losses or sensing failures. For the purpose of this work, a measurement at time k is the set

$$(p_i, \mathcal{N}_i, \phi(\mathcal{V}_i)) \quad (29)$$

composed of the agent's state p_i , the states of its Voronoi neighbours $\mathcal{N}_i$, and the risk distribution ϕ in the Voronoi cell. Note that the risk distribution could be the result of an interpolation from point measurements or fused from data communicated from neighbours, for example.

5.1. Critic network structure

In this section, we discuss the critic network structure and the policy evaluation task, i.e., approximating the value function and determining the direction in which the actor networks' weights are updated. We define the value function approximation of the i th agent as

$$\hat{V}(e_i[k]) = w_{e_i}^T \psi_{e_i}(e_i[k]), \quad (30)$$

where the vector $\mathbf{w}_{c_i} \in \mathbb{R}^{n_c}$ with entries

$$\mathbf{w}_{c_i} = [w_{c_i,1}, w_{c_i,2}, \dots, w_{c_i,n_c}]^T. \quad (31)$$

includes the weights of the critic network at the n_c nodes. The collection of activation functions of the neural network is denoted by $\psi_c(e_i[k])$. The activation function of the critic network is then given by

$$\psi_c(e_i[k]) = [e_i^x[k], e_i^y[k], (e_i^x[k])^2, e_i^x[k]e_i^y[k], (e_i^y[k])^2]^T \quad (32)$$

with the introduction of the Cartesian components e_i^x and e_i^y of the error. The number of elements in the set of basis functions determines the number of nodes in the critic network, thus $n_c = 5$. This quadratic structure of the function approximation is particularly suitable in approximate dynamic programming, since the policy improvement requires the gradient of the value function with respect to the state (Lewis et al., 2012).

We introduce the compact notation $v_k = v(\mathbf{u}(e_i[k]))$. The activation functions in the hidden layer are derived from the temporal difference equation (19), by introducing the basis function projection (30)

$$\begin{aligned} v_k &= V(e_i[k]) - \gamma V(e_i[k+1]) \\ &\approx \mathbf{w}_{c_i}^T \psi_c(e_i[k]) - \gamma \mathbf{w}_{c_i}^T \psi_c(e_i[k+1]) =: \mathbf{w}_{c_i}^T \tilde{\psi}_c[k] \end{aligned}$$

where $\tilde{\psi}_c[k] = \psi_c(e_i[k]) - \gamma \psi_c(e_i[k+1])$. The critic neural network schematized in Fig. 2 estimates the value function $V(e_i[k])$ using the basis function vector $\tilde{\psi}_c[k] \in \mathbb{R}^{n_c}$ that minimizes the critic least-square error defined by

$$\delta_c = \sum_{n=0}^{\eta-1} \frac{1}{2} (v_{k+n} - \mathbf{w}_{c_i}^T \tilde{\psi}_c[k+n])^2 = \frac{1}{2} \|\mathbf{v} - \mathbf{A} \mathbf{w}_{c_i}\|^2 \quad (33)$$

where the entries of the matrix $\mathbf{A} \in \mathbb{R}^{\eta \times n_c}$ are $[\mathbf{A}]_n = \tilde{\psi}_c^T[k+n] \in \mathbb{R}^{1 \times n_c}$, and the η -vector $\mathbf{v}$ is the collection of the scalars v_{k+n} for $n = 0, 1, \dots, \eta-1$. The weights of the critic network are updated based on a gradient descent scheme given by

$$\mathbf{w}_{c_i}^{[k+1]} = \mathbf{w}_{c_i}^{[k]} - \alpha_c \frac{\partial \delta_c}{\partial \mathbf{w}_{c_i}} = \mathbf{w}_{c_i}^{[k]} - \alpha_c \mathbf{A}^T (\mathbf{A} \mathbf{w}_{c_i}^{[k]} - \mathbf{v}) \quad (34)$$

where α_c denotes the learning rate of critic neural network.

5.2. Actor network structure

The second task of the policy iteration algorithm, namely, policy improvement, is executed using the actor network, which approximates the optimal control policy. The architecture of the actor network is illustrated in Fig. 2. The structure of the approximated optimal control (24) in terms of actor network's output for the i th agent is

$$\hat{\mathbf{u}}_i(e_i[k]) = \mathbf{W}_{a_i}^T \sigma(e_i[k]) \quad (35)$$

where the activation function $\sigma(e_i[k])$ of the actor neural network is constructed using the same quadratic set

$$\sigma(e_i[k]) = [e_i^x[k], e_i^y[k], (e_i^x[k])^2, e_i^x[k]e_i^y[k], (e_i^y[k])^2]^T \quad (36)$$

The weight matrix of the actor network, for the i th agent, $\mathbf{W}_{a_i} \in \mathbb{R}^{n_a \times 2}$ is defined as

$$\mathbf{W}_{a_i}^T = \begin{bmatrix} w_{a_i}^{1,x} & w_{a_i}^{2,x} & w_{a_i}^{3,x} & w_{a_i}^{4,x} & w_{a_i}^{5,x} \\ w_{a_i}^{1,y} & w_{a_i}^{2,y} & w_{a_i}^{3,y} & w_{a_i}^{4,y} & w_{a_i}^{5,y} \end{bmatrix} \quad (37)$$

where the superscripts x and y denote Cartesian components, $n_a = 5$ is the number of nodes in actor network, and w_a^x and w_a^y are the actor's weights.

The training samples for the actor network weights are determined by generating a sequence $\{e_i[k+n], v(e_i[k+n])\}$, $n = 0, 1, \dots, \hat{n}-1$, resulting in the following set of linear equations that are solved for the weight matrix $\mathbf{W}_{a_i}$ as explained below

$$\sigma^T(e_i[k]) \mathbf{W}_{a_i} = \hat{\mathbf{b}}_i^T[k] \quad (38a)$$

$$\sigma^T(e_i[k+1]) \mathbf{W}_{a_i} = \hat{\mathbf{b}}_i^T[k+1] \quad (38b)$$

$\vdots$

$$\sigma^T(e_i[k+\hat{n}-1]) \mathbf{W}_{a_i} = \hat{\mathbf{b}}_i^T[k+\hat{n}-1] \quad (38c)$$

This linear system can be represented in matrix form as

$$\Sigma \mathbf{W}_{a_i} = \mathbf{z} \quad (39)$$

where the entries of the $\hat{n} \times n_a$ matrix Σ are $[\Sigma]_{k+n} = \sigma^T(e_i[k+n])$, and the entries of the $\hat{n} \times 2$ vector $\mathbf{z}$ are given by $[\mathbf{z}]_{k+n} = \hat{\mathbf{b}}_i^T[k+n]$, for $n = 0, 1, \dots, \hat{n}-1$. A gradient descent scheme is adopted to update $\mathbf{W}_{a_i}$ such that the sum of squared errors δ_a is minimized

$$\delta_a = \frac{1}{2} \|\mathbf{z} - \Sigma \mathbf{W}_{a_i}\|^2 = \frac{1}{2} \text{Tr}[(\mathbf{z} - \Sigma \mathbf{W}_{a_i})^T (\mathbf{z} - \Sigma \mathbf{W}_{a_i})] \quad (40)$$

where Tr is the trace operator. The partial derivative of the sum of the squared error is given by

$$\frac{\partial \delta_a}{\partial \mathbf{W}_{a_i}} = \frac{1}{2} \frac{\partial}{\partial \mathbf{W}_{a_i}} \text{Tr}[(\mathbf{z} - \Sigma \mathbf{W}_{a_i})^T (\mathbf{z} - \Sigma \mathbf{W}_{a_i})] = \Sigma^T \Sigma \mathbf{W}_{a_i} - \Sigma^T \mathbf{z} \quad (41)$$

Using (41) the actor network weights are updated according to the following iterative gradient descent scheme

$$\mathbf{W}_{a_i}^{[n+1]} = \mathbf{W}_{a_i}^{[n]} - \alpha_a \frac{\partial \delta_a}{\partial \mathbf{W}_{a_i}} = \mathbf{W}_{a_i}^{[n]} - \alpha_a (\Sigma \mathbf{W}_{a_i}^{[n]} - \mathbf{z})^T \Sigma \quad (42)$$

where α_a is the learning rate of the actor network, and n is the iterator, which stops when the error converges according to $\|e_i[k+n+1] - e_i[k+n]\| < \varepsilon_v$, and ε_v is a tolerance.

5.3. Feed-forward neural network

The control $\mathbf{v}$ regulates the error to the origin, driving the agents to a centroidal configuration corresponding to a local maximum of the coverage metric H . However, the control action $\mathbf{u}_i$ requires the feed-forward component $\bar{\mathbf{u}}_i = -\mathbf{G}^{-1}(\mathbf{0}, t) \mathbf{F}(\mathbf{0}, t)$ defined in (10) to track time-varying centroids, where the non-autonomous part depends on the uncontrolled dynamics of the environment through the rate $\frac{\partial \phi}{\partial t}$. Clearly the calculation of the feed-forward control requires the knowledge of the drift dynamics $\mathbf{F}$, which defeats the purpose of temporal learning relying only on measurements along a specific system's trajectory. This motivates the introduction of a third neural network to approximate the feed-forward control. Following the ideas delineated in Dierks and Jagannathan (2009), let

$$\hat{\bar{\mathbf{u}}}_i = \mathbf{W}_{d_i}^T \mu(\tilde{\mathbf{c}}_i[k]) \quad (43)$$

where μ is the activation functions of the feed-forward neural network, given by

$$\mu(\tilde{\mathbf{c}}_i[k]) = [\tilde{c}_i^x[k], \tilde{c}_i^y[k], (\tilde{c}_i^x[k])^2, \tilde{c}_i^x[k]\tilde{c}_i^y[k], (\tilde{c}_i^y[k])^2]^T \quad (44)$$

Notice that they depend on the Voronoi centroid, which is the reference signal to be tracked by the agent. To determine the gradient descent based training rule for the feed-forward weights $\mathbf{W}_{d_i}$ for the i th agent, define the error

$$\delta_{d_i} = \bar{\mathbf{F}}(e_i[k], k) + \mathbf{G}(e_i[k], k) \hat{\bar{\mathbf{u}}}_i - \bar{\mathbf{F}}(e_i[k], k) + \mathbf{G}(e_i[k], k) \mathbf{W}_{d_i}^T \mu(\tilde{\mathbf{c}}_i[k]) \quad (45)$$

By using the error dynamics with function approximations

$$e_i[k+1] = \bar{\mathbf{F}}(e_i, k) + \mathbf{G}(e_i[k], k) (\mathbf{W}_{a_i}^T \sigma(e_i[k]) + \mathbf{W}_{d_i}^T \mu(\tilde{\mathbf{c}}_i[k]))$$

this can be also written as

$$\delta_{d_i} = e_i[k+1] - \mathbf{G}(e_i[k], k) \mathbf{W}_{a_i}^T \sigma(e_i[k]) \quad (46)$$

which is computable along the trajectories given some value of the actor's weights. The gradient descent learning rule induced by the requirement of minimizing the squared error $\|\delta_{d_i}\|^2/2$ is

$$\mathbf{W}_{d_i}^{[n+1]} = \mathbf{W}_{d_i}^{[n]} - \alpha_d \delta_{d_i}^T \frac{\partial \delta_{d_i}}{\partial \mathbf{W}_{d_i}}$$

$$= \mathbf{W}_{d_i}^{[n]} - \alpha_d (e_i[k+1] - \mathbf{G}(e_i[k], k) \mathbf{W}_{a_i}^T \sigma(e_i[k]))^T \mathbf{G}(e_i[k], k) \mu(\tilde{c}_i) \quad (47)$$

Note that the feed-forward weights do not update when the error is zero, that is, when the system is in a centroidal configuration. Following (16), the approximated optimal control is obtained from (35) and (43) as follows:

$$\hat{\mathbf{u}}_i = \hat{\mathbf{v}}(e_i[k]) + \hat{\mathbf{u}}_i(\tilde{c}_i[k]) \quad (48)$$

5.4. Algorithm

The architecture of the proposed model-free coverage control is presented in Algorithm 1. The weights of the actor, critic and feed-forward networks are initiated randomly. Following, the control policy is initiated by an admissible one according to Lemma 1.

Algorithm 1 Model-free reinforcement learning solution for coverage control problems using an actor-critic-feed-forward structure

Input: $\mathbf{R}$, Policy Iteration inner-loop length K , sampling time T

Output: Error trajectory of i th agent: $e_i[k]$ for $i = 1, \dots, l$

Initialize:

$\mathbf{w}_c^k, \mathbf{W}_{a_i}^n, \mathbf{W}_d^n$ /* Discrete time, critic, actor and feed-forward indices, critic weights, actor and feed-forward weights. */

```

1  Given agents' and targets' initial states, compute  $e_i[0]$  using (6).
   Compute initial admissible control input  $\mathbf{u}_i^*[0]$  using (16)
2  for  $n = 0$  to  $N_a$  do
3    for  $\kappa = nK$  to  $(n+1)K$  do
4      Update the agents' states  $\mathbf{p}_i[\kappa+1] = \mathbf{p}_i[\kappa] + \mathbf{u}_i^*[\kappa]T$  via the initial
       admissible controls  $\mathbf{u}_i^*[\kappa]$ 
5      Update Voronoi's generalized centroids  $\tilde{c}_i[\kappa]$  by using (5b)
6      Calculate the updated error  $e_i[\kappa+1]$  by using (6)
7      Approximate the value function  $\hat{V}(e_i[\kappa+1])$  by using (30)
8      if  $\|e_i[\kappa+n+1] - e_i[\kappa+n]\| > \epsilon_v$  then
9        Update the control policy  $\hat{\mathbf{u}}_i[\kappa+1]$  by using (48)
10     else if  $\|e_i[\kappa+n+1] - e_i[\kappa+n]\| \leq \epsilon_v$  then
11       Break
12     Update the critic's weights  $\mathbf{w}_c^{[\kappa+1]}$  by using (34)
13   end
14   Update the actor's weights  $\mathbf{W}_{a_i}^{[n+1]}$  by using (42)
15   Update the feed-forward's weights  $\mathbf{W}_d^{[n+1]}$  by using (47)
16 end

```

Agents are assigned with their initial position, and the centroid of their Voronoi partition is calculated using (5b), and thus the initial error is obtained by (6). Consequently, the admissible control signal is approximated by using a set of random actor network weights and the initial calculated error. This control signal drives the state of the agents in the environment. The error and control input are updated at each step of the inner-loop, followed by the approximation of the value function $\hat{V}(e_i[\kappa+1])$. Next, the critic weights $\mathbf{w}_c^{[\kappa+1]}$ are updated for the new approximated value function. Following (28), at the end of the inner-loop, the error $e_i[K]$, two approximated control inputs $\mathbf{v}_i[K-2]$ and $\mathbf{v}_i[K-1]$, and the last two approximated value function values $\hat{V}(e_i[K-1])$ and $\hat{V}(e_i[K])$ are stored to estimate the input dynamics $\mathbf{G}$. In the outer-loop, the actor weights $\mathbf{W}_{a_i}^{[n+1]}$ are updated by using the stored error $e_i[K]$, the control input $\mathbf{v}_i[K]$ and the approximated value function $\hat{V}(e_i[K])$. Finally, the feed-forward weights $\mathbf{W}_d^{[n+1]}$ are updated from the actor weights and from $\mathbf{G}(e_i[K-1])$, calculated using (28). The hyper-parameters of Algorithm 1 are given in Table 1, resulting from the tuning briefly described in the next section. The length of the inner loop K has been chosen heuristically to balance the number of updates of the critic weights (inner loop) and of the actor and feed-forward weights (outer loop), with the latter two determining the control action update.

Table 1

Hyper-parameters of the model-free coverage control.

Hyper-Parameters	Size
Number of agents (l)	4
Simulation time steps	6000
Inner-loop length (K)	4
Outer-loop length (N_a)	1500
Critic learning rate (α_c)	1×10^{-5}
Actor learning rate (α_a)	1×10^{-6}
Feed-forward learning rate (α_d)	1×10^{-9}
ϵ_v	5×10^{-2}
Time step (T)	0.0033
Finite difference singularity parameter (ϵ)	10^{-5}

5.5. Tuning considerations

The hyper-parameters reported in Table 1 are tuned by starting from guessed values and by testing the algorithm to convergence with different trials/episodes. A general tuning criterion for the learning rate can be based on the consideration that since the control policy is obtained based on an estimation of the value function V , the critic network must converge faster than the actor network.

Additionally, the $\mathbf{R}$ matrices in (17) are initialized as identity matrices. Further each matrix is tuned via different trials based on the convergence error and feed-forward control signal. The resulting $\mathbf{R}$ matrices in simulation are

$$\mathbf{R}_1 = \begin{bmatrix} 126 & 0 \\ 0 & 125 \end{bmatrix}, \quad \mathbf{R}_2 = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.7 \end{bmatrix}$$

$$\mathbf{R}_3 = \begin{bmatrix} 3.72 & 0 \\ 0 & 3.7 \end{bmatrix}, \quad \mathbf{R}_4 = \begin{bmatrix} 2.4 & 0 \\ 0 & 2.45 \end{bmatrix}$$

6. Simulation results

Unless otherwise specified, the unit of length is metre, and the unit of time is second.

6.1. Non-autonomous area coverage: optimal tracking in an evolving environment

We illustrate the optimal tracking reinforcement learning algorithm by simulating the system in a square-shaped workspace Ω of side 20. Four agents are initially placed at coordinates (0.5, 0), (0, 1), (1, 0), and (0, 0.5). The time-varying part of the risk density is the Gaussian function

$$\phi(q, \kappa) = \sum_{j=1}^2 \exp\left(-\frac{1}{2} (q - s^{[j]}[\kappa]) \cdot \beta^{-1} (q - s^{[j]}[\kappa])\right) \quad (49)$$

about two randomly selected initial locations at $s^{[1]}[0] = (3.2, 6)$, $s^{[2]}[0] = (18.5, 13.5)$ corresponding to the peaks of the Gaussian. This formalism is often used to model target tracking scenarios, in which external threats entering an area to be monitored are considered as moving risk sources. In this example, the targets move on a straight line emanating from the initial states, reaching the final locations (7.5, 14.5) and (8.5, 7), with respective constant speeds $s^{[1]} = 0.45$ and $s^{[2]} = 0.55$. The initial risk density distribution is illustrated in Fig. 9(a), which depicts the initial state of the workspace including the agents' initial positions.

6.1.1. Actor-critic and feed-forward: quasi optimal tracking

The agents' convergence to the generalized centroids of the associated Voronoi cells in the presence of time-varying risk is illustrated in Fig. 3(a), which shows convergence to zero of the errors generated by the proposed control policy, confirming the theoretical prediction. The control inputs/policies, having the meaning of agents' velocities, are plotted in Fig. 3(b). After the error converges, the control policies are still nonzero due to the tracking action of the feed-forward control;

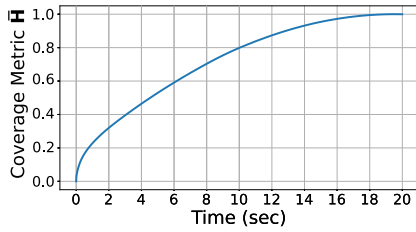


Fig. 4. Normalized coverage metric $\bar{H}$ as agents converge to their respective Voronoi cell centroids.

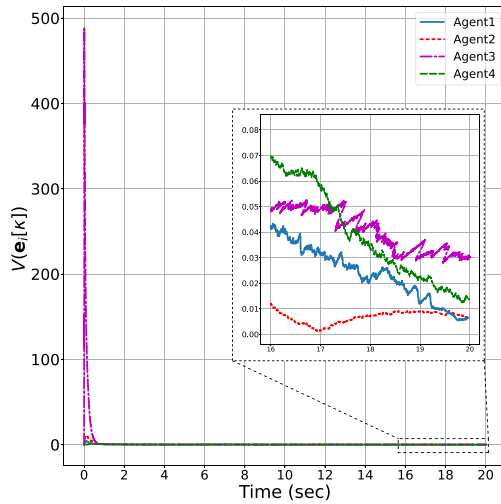


Fig. 5. Evolution of the approximated value function.

the feed-forward actions are small as compared to the initial regulator actions, and therefore they are visualized by a zooming window in Fig. 3(b).

The normalized coverage metric $\bar{H} = (H - H_{min}) / (H_{max} - H_{min})$ is plotted in Fig. 4, where the maximum and minimum coverage metric values recorded are $H_{max} = 12923.97$ and $H_{min} = 8207.07$, respectively. As agents track and converge to their respective time-varying Voronoi centroids, the coverage metric increases, i.e., agents adapt to the evolution of the risk/environment on account of the targets' motion. The application of the optimal tracking algorithm based on the three coupled neural network generates a monotonically increasing coverage metric, consistently with continuous time generalized Lloyd's algorithms (Miah et al., 2017).

As a result of the regulator optimization driving the errors to zero, the value function is minimized in finite time, as depicted in Fig. 5. While the agents move in the environment, the critic network weights are updated, and the new value function is approximated at each step as described by the model (30), using the new measured error by (32). The evolution of critic weights is depicted in Fig. 6, showing some relatively large variation due to the initialization of the coverage metric to a small, suboptimal value induced by the initial state of the agents which are clustered in small region of the workspace.

Thereafter, the actor network is updated based on the estimated value function. The actor and feed-forward networks are placed in the outer loop, as detailed in algorithm 1. The evolution of weights corresponding to actor and feed-forward networks are illustrated in Figs. 7 and 8, respectively. Some of the feed-forward weights may not converge to a constant value (at least at the scale of the simulation presented here) as the feed-forward control action tracks time varying centroids.

The agents' configuration and trajectories converging to their associated centroids at time $t = 20$ is shown in Fig. 9(b).

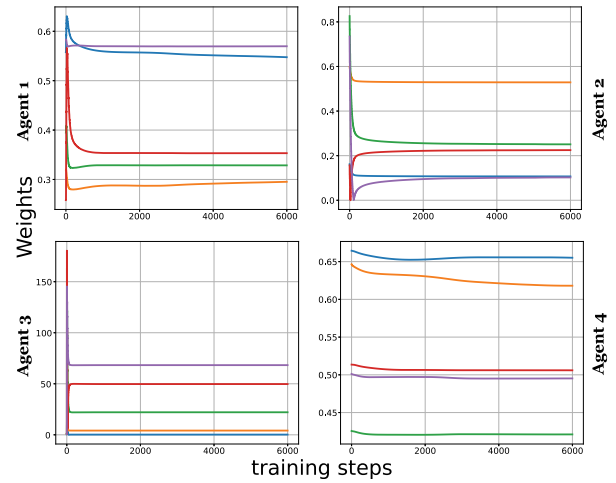


Fig. 6. Critic network weights.

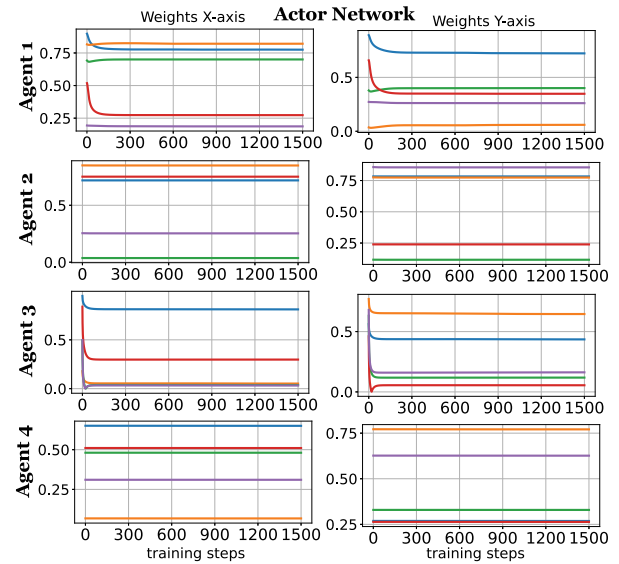


Fig. 7. Actor network weights.

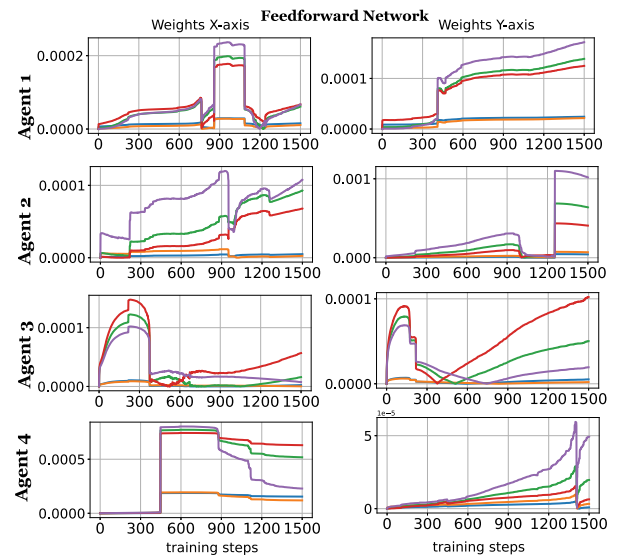


Fig. 8. Feed-forward network weights.

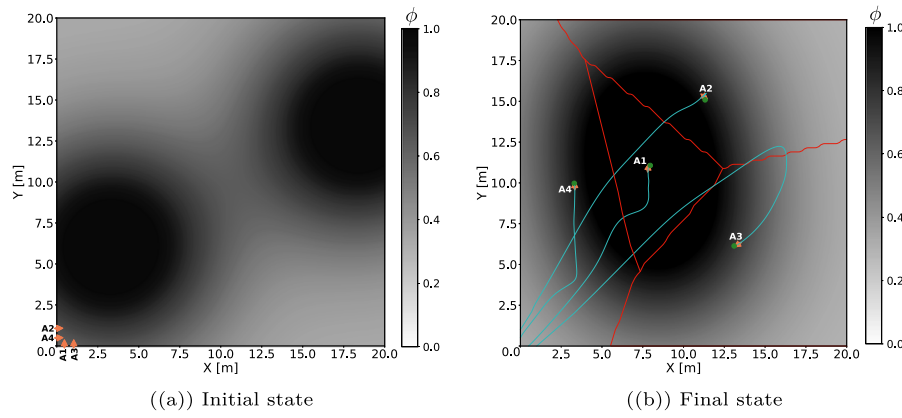


Fig. 9. Configurations of the agents at the initial and final times of the simulation with evolving (time varying) environment/risk. Polygonal Voronoi cells and risk density function are depicted as well.

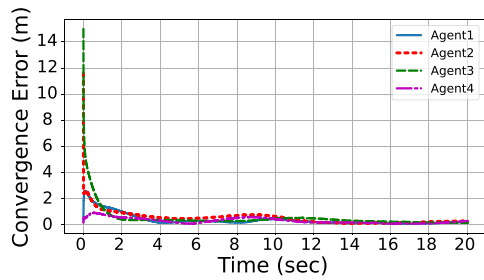


Fig. 10. Evolution of the errors and control inputs from the simulation of the non-autonomous area coverage induced by time varying risk (without feed-forward).

6.1.2. Actor-critic only: non-zero tracking error

To further illustrate the importance of the feed-forward network in the optimal tracking problem, we present the results from an actor-critic algorithm without the feed-forward network. This would be the approximated solution for the optimal regulator problem, appropriate in a static environment. The simulation is performed in an environment with the same evolving risk density distribution as in the previous set of results. This comparison illustrates the role of the feed-forward network in tracking time evolving Voronoi centroids, and therefore its crucial role in a model-free, data driven non-autonomous area coverage control as formulated in this paper. From Fig. 10, it is evident that the actor-critic structure alone may reduce the error, but it is not sufficient to track with zero error while the environment evolves. On the other hand, the proposed method with the additional neural network drives the error to zero as shown in Fig. 3(a), with the feed-forward network approximating the control action to track Voronoi centroids.

A comparison between Figs. 5 and 11 shows that the algorithm with feed-forward neural network results in a lower value function through the evolution, especially as visualized by a zooming window on simulation data pertaining to tracking of time varying centroids.

6.2. Autonomous area coverage: static environment

The performance of the proposed algorithm without feed-forward network, is compared to a well-established framework for area coverage control (Soleymani et al., 2022), namely, Lloyd's algorithm (Cortés et al., 2004) with time-invariant risk, which makes the feed-forward control signal identically zero. Four agents are initially placed at (9.022, 4.82), (4.42, 15.64), (7.38, 16.22), and (5.81, 8.015). The risk density is described by the Gaussian function from (49), about three randomly selected static locations at $s^{[1]} = (3.05, 8.1)$, $s^{[2]} = (5.1, 12.53)$, $s^{[3]} =$

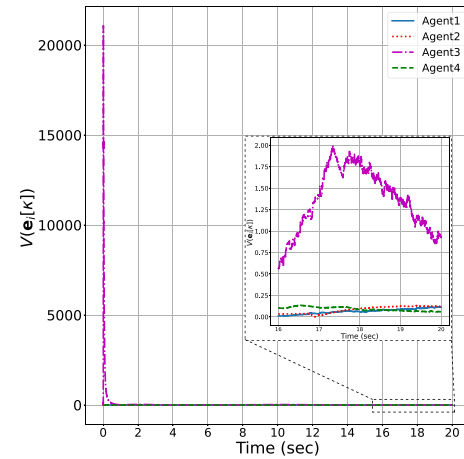


Fig. 11. Evolution of the approximated value function (without feed-forward).

Table 2

Hyper-parameters in static environment.

Hyper-Parameters	Size
Simulation time steps	2000
Inner-loop length K	4
Outer-loop length N_a	500
Critic learning rate α_c	5×10^{-3}
Actor learning rate α_a	5×10^{-4}
ϵ_v	5×10^{-2}
Time step T	0.01
Test duration t	20

(8.82, 8.3). The simulation is performed for $t = 20$ (see Table 2). Tuned R matrices used for this set of simulations are

$$R_1 = \begin{bmatrix} 155 & 0 \\ 0 & 155 \end{bmatrix}, \quad R_2 = \begin{bmatrix} 0.78 & 0 \\ 0 & 0.75 \end{bmatrix}$$

$$R_3 = \begin{bmatrix} 45.5 & 0 \\ 0 & 45.5 \end{bmatrix}, \quad R_4 = \begin{bmatrix} 2.55 & 0 \\ 0 & 2.45 \end{bmatrix}$$

As in the case of evolving environment, the data required for implementation is collected along the trajectories to sense the environment (function ϕ) and a set of neighbouring agents (position data can be acquired through direct communication). The error dynamics is illustrated in Fig. 12, which shows asymptotic convergence to zero of the errors generated by the proposed control policy and from Lloyd's algorithm. The proposed approach shows the expected slower response compared to Lloyd's algorithm, which is implemented in continuous time.

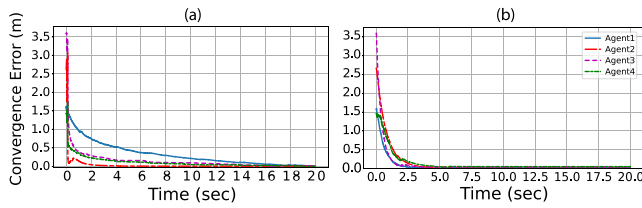


Fig. 12. Error dynamics (time-invariant risk). (a) Actor-critic algorithm, (b) Lloyd's algorithm.

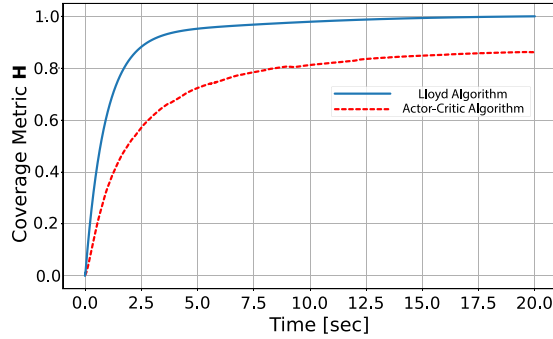


Fig. 13. A comparison of the obtained coverage metric H using actor-critic algorithm and Lloyd's algorithm. Both coverage metrics are normalized with respect to maximum coverage attained by Lloyd's algorithm in a time-invariant environment.

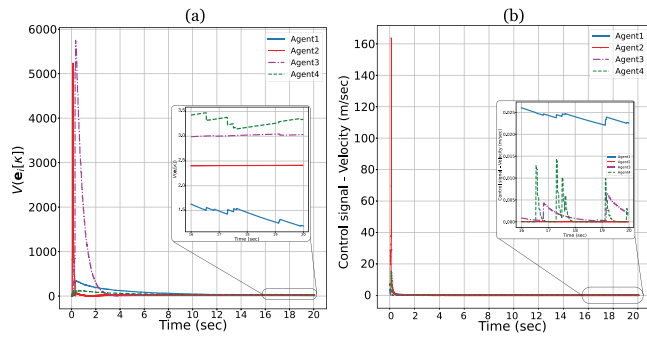


Fig. 14. (a) Evolution of the approximated value function, (b) Control policy.

A performance comparison of actor-critic and Lloyd's algorithm in the time-invariant environment is depicted in Fig. 13 by showing the evolution of the coverage metric with respect to the maximum coverage obtained by Lloyd's algorithm, where the faster convergence of Lloyd's algorithm appears again. The value function is monotonically decreasing, as illustrated in Fig. 14(a), with a control signal generated by the algorithm shown in Fig. 14(b). The evolution of critic and actor weights are illustrated in Figs. 15 and 16 respectively.

7. Summary and conclusion

We proposed an online adaptive control framework to address the area coverage control problem with multi-agent systems in an evolving environment. Bellman's principle of optimality is applied to a performance index based on the typical coverage metric that encodes the environment and the agents' sensing performance. The adaptive control policy aims at maximizing the coverage metric index, while driving the system error dynamics to zero, which can be achieved as the agents converge to their respective Voronoi centroid. The optimal solution is achieved by an adaptive model-free control policy based on reinforcement learning, with an actor-critic architecture and an additional feed-forward neural network for tracking. The actor neural network

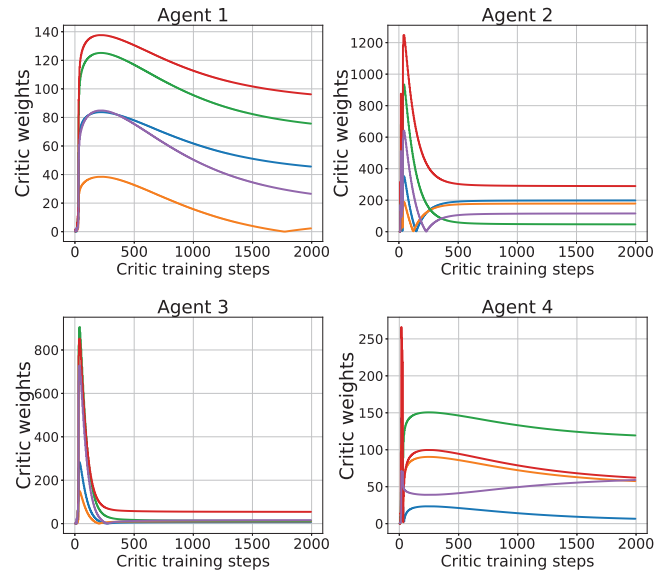


Fig. 15. Weights of the critic networks (time-invariant risk).

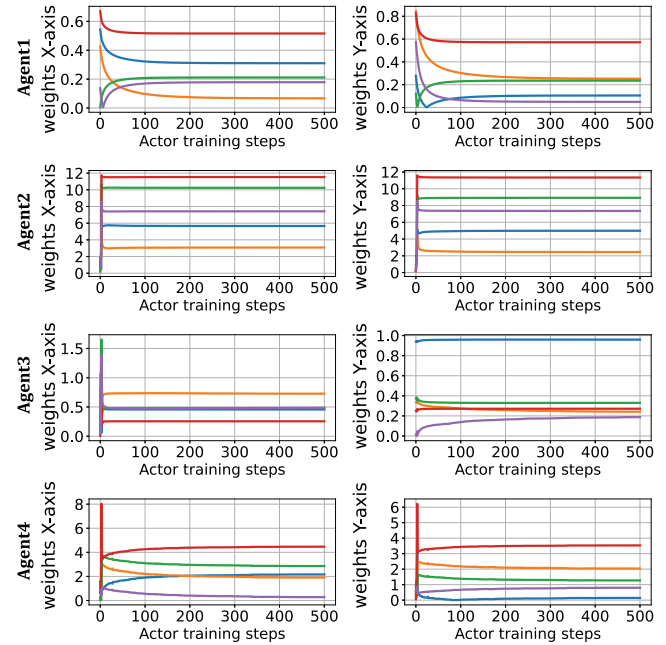


Fig. 16. Weights of the actor networks (time-invariant risk).

estimates the control policy that minimizes the error dynamics, while the feed-forward neural network determines the control signal that tracks evolving centroids. In a static environment, a regulator structure is compared to Lloyd's algorithm, illustrating that the approximate dynamic programming scheme in the present work converges to the same class of solutions of the area coverage control problem.

CCRediT authorship contribution statement

Farzan Soleymani: Methodology, Algorithm, Data curation, Visualization, Reviewing. **Md Suruz Miah:** Conceptualization, Supervision, Reviewing, Editing. **Davide Spinello:** Conceptualization, Formal analysis, Reviewing, Editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) via the grant RGPIN-2016-05783.

References

- Bai, Y., Wang, Y., Svinin, M., Magid, E., Sun, R., 2021. Adaptive multi-agent coverage control with obstacle avoidance. *IEEE Control Syst. Lett.*
- Baird, L.C., 1994. Reinforcement learning in continuous time: Advantage updating. In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, Vol. 4. IEEE, pp. 2448–2453.
- Basu, H., Yoon, S.Y., 2020. Cooperative output regulation of networked motors under switching communication and detectability constraints. *IEEE Trans. Control Syst. Technol.* 29 (3), 1296–1303.
- Beard, R.W., Saridis, G.N., Wen, J.T., 1997. Galerkin approximations of the generalized Hamilton-Jacobi-Bellman equation. *Automatica* 33 (12), 2159–2177.
- Bertsekas, D.P., 1995. *Dynamic Programming and Optimal Control*, Vol. 1. Athena scientific Belmont, MA.
- Bullo, F., Carli, R., Frasca, P., 2012. Gossip coverage control for robotic networks: Dynamical systems on the space of partitions. *SIAM J. Control Optim.* 50 (1), 419–447.
- Bullo, F., Cortes, J., Martinez, S., 2009. Distributed Control of Robotic Networks: a Mathematical Approach to Motion Coordination Algorithms. In: *Applied Mathematics Series*, Princeton University Press, <http://dx.doi.org/10.1515/9781400831470>.
- Cao, L., Yao, D., Li, H., Meng, W., Lu, R., 2022. Fuzzy-based dynamic event triggering formation control for nonstrict-feedback nonlinear MASs. *Fuzzy Sets and Systems*.
- Carron, A., Zeilinger, M.N., 2020. Model predictive coverage control. *IFAC-PapersOnLine* 53 (2), 6107–6112.
- Chu, B., Hong, D., Park, J., Chung, J.-H., 2008. Passive dynamic walker controller design employing an RLS-based natural actor-critic learning algorithm. *Eng. Appl. Artif. Intell.* 21 (7), 1027–1034. <http://dx.doi.org/10.1016/j.engappai.2007.11.001>.
- Cortes, J., Martinez, S., Karatas, T., Bullo, F., 2002. Coverage control for mobile sensing networks: Variations on a theme. In: *10th Mediterranean Conf. on Control and Automation*, Vol. 1.
- Cortés, J., Martínez, S., Karatas, T., Bullo, F., 2004. Coverage control for mobile sensing networks. *IEEE Trans. Robot. Autom.* 20 (2), 243–255.
- Diaz-Mercado, Y., Lee, S.G., Egerstedt, M., 2017. Human-swarm interactions via coverage of time-varying densities. In: *Trends in Control and Decision-Making for Human-Robot Collaboration Systems*. Springer, pp. 357–385.
- Dierks, T., Jagannathan, S., 2009. Optimal tracking control of affine nonlinear discrete-time systems with unknown internal dynamics. In: *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) Held Jointly with 2009 28th Chinese Control Conference*. pp. 6750–6755. <http://dx.doi.org/10.1109/CDC.2009.5399697>.
- Duca, R.N., Bugeja, M.K., 2020. Multi-robot energy-aware coverage control in the presence of Time-Varying Importance Regions. *IFAC-PapersOnLine* 53 (2), 9676–9681.
- Elmokadem, T., 2019. Distributed coverage control of quadrotor multi-UAV systems for precision agriculture. *IFAC-PapersOnLine* 52 (30), 251–256. <http://dx.doi.org/10.1016/j.ifacol.2019.12.530>.
- Hanselmann, T., Noakes, L., Zaknich, A., 2007. Continuous-time adaptive critics. *IEEE Trans. Neural Netw.* 18 (3), 631–647.
- Heydari, A., 2014. Revisiting approximate dynamic programming and its convergence. *IEEE Trans. Cybern.* 44 (12), 2733–2743.
- Hu, J., Xie, L., Lum, K.-Y., Xu, J., 2013. Multiagent information fusion and cooperative control in target search. *IEEE Trans. Control Syst. Technol.* 21 (4), 1223–1235.
- Kantaros, Y., Thanou, M., Tzes, A., 2015. Distributed coverage control for concave areas by a heterogeneous Robot-Swarm with visibility sensing constraints. *Automatica* 53, 195–207. <http://dx.doi.org/10.1016/j.automatica.2014.12.034>.
- Kawajiri, S., Hirashima, K., Shiraiishi, M., 2021. Coverage control under connectivity constraints. In: *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*. pp. 1554–1556.
- Kiumarsi, B., Lewis, F.L., 2015. Actor-critic-based optimal tracking for partially unknown nonlinear discrete-time systems. *IEEE Trans. Neural Netw. Learn. Syst.* 26 (1), 140–151. <http://dx.doi.org/10.1109/TNNLS.2014.2358227>.
- Lawhead, R.J., Gosavi, A., 2019. A bounded actor-critic reinforcement learning algorithm applied to airline revenue management. *Eng. Appl. Artif. Intell.* 82, 252–262. <http://dx.doi.org/10.1016/j.engappai.2019.04.008>.
- Lee, S., Diaz-Mercado, Y., Egerstedt, M., 2015. Multirobot control using time-varying density functions. *IEEE Trans. Robot.* 31 (2), 489–493.
- Lekien, F., Leonard, N.E., 2009. Nonuniform coverage and cartograms. *SIAM J. Control Optim.* 48 (1), 351–372.
- Lewis, F.L., Vrabie, D., Syrmos, V.L., 2012. *Optimal Control*. John Wiley & Sons.
- Liang, H., Chen, L., Pan, Y., Lam, H.-K., 2022. Fuzzy-based robust precision consensus tracking for uncertain networked systems with cooperative-antagonistic interactions. *IEEE Trans. Fuzzy Syst.*
- Liu, Y.-C., Lin, T.-C., Lin, M.-T., 2021. Indirect/direct learning coverage control for wireless sensor and mobile robot networks. *IEEE Trans. Control Syst. Technol.*
- Liu, D., Wei, Q., 2014. Policy iteration adaptive dynamic programming algorithm for discrete-time nonlinear systems. *IEEE Trans. Neural Netw. Learn. Syst.* 25 (3), 621–634. <http://dx.doi.org/10.1109/TNNLS.2013.2281663>.
- Lloyd, S., 1982. Least squares quantization in PCM. *IEEE Trans. Inform. Theory* 28 (2), 129–137. <http://dx.doi.org/10.1109/TIT.1982.1056489>.
- Meng, S., Kan, Z., 2021. Deep reinforcement learning based effective coverage control with connectivity constraints. *IEEE Control Syst. Lett.*
- Miah, M.S., Nguyen, B., Bourque, A., Spinello, D., 2014. Nonuniform deployment of autonomous agents in harbor-like environments. *Unmanned Syst.* 2 (4), 377–389.
- Miah, M.S., Nguyen, B., Bourque, A., Spinello, D., 2015. Nonuniform coverage control with stochastic intermittent communication. *IEEE Trans. Automat. Control* 60 (7), 1981–1986.
- Miah, S., Panah, A.Y., Fallah, M.M.H., Spinello, D., 2017. Generalized non-autonomous metric optimization for area coverage problems with mobile autonomous agents. *Automatica* 80, 295–299. <http://dx.doi.org/10.1016/j.automatica.2017.02.044>.
- Okabe, A., Boots, B., Sugihara, K., Chiu, S.N., 2000. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, second ed. John Wiley & Sons, LTM.
- Pane, Y.P., Nagesh Rao, S.P., Kober, J., Babuška, R., 2019. Reinforcement learning based compensation methods for robot manipulators. *Eng. Appl. Artif. Intell.* 78, 236–247. <http://dx.doi.org/10.1016/j.engappai.2018.11.006>.
- Papathodorou, S., Tzes, A., Giannousakis, K., Stergiopoulos, Y., 2018. Distributed area coverage control with imprecise robot localization: Simulation and experimental studies. *Int. J. Adv. Robot. Syst.* 15 (5), <http://dx.doi.org/10.1177/1729881418797494>, <https://doi.org/10.1177/1729881418797494>.
- Pimenta, L.C.A., Pereira, G.A.S., Goncalves, M.M., Michael, N., Turpin, M., Kumar, V., 2013. Decentralized controllers for perimeter surveillance with teams of aerial robots. *Adv. Robot.* 27 (9, SI), 697–709. <http://dx.doi.org/10.1080/01691864.2013.778942>.
- Purvis, M., Cranefield, S., Ward, R., Nowostawski, M., Carter, D., Bush, G., 2003. A multi-agent system for the integration of distributed environmental information. *Environ. Model. Softw.* 18 (6), 565–572. [http://dx.doi.org/10.1016/S1364-8152\(03\)00031-8](http://dx.doi.org/10.1016/S1364-8152(03)00031-8).
- Schwager, M., Vitus, M.P., Powers, S., Rus, D., Tomlin, C.J., 2015. Robust adaptive coverage control for robotic sensor networks. *IEEE Trans. Control Netw. Syst.* 4 (3), 462–476.
- Serrano, W., 2022. Deep reinforcement learning with the random neural network. *Eng. Appl. Artif. Intell.* 110, 104751. <http://dx.doi.org/10.1016/j.engappai.2022.104751>.
- Simetti, E., Cresta, M., 2007. Towards the use of a team of USVs for civilian harbour protection: The problem of intercepting detected menaces. In: *IEEE OCEANS. Sydney, NSW*, pp. 1–7.
- Soleymani, F., Miah, M.S., Spinello, D., 2019. Non-autonomous state-feedback to stabilize the error dynamics in time-varying area coverage control problems. In: *2019 IEEE International Symposium on Robot and Sensors Environments (ROSE)*. IEEE, pp. 1–5. <http://dx.doi.org/10.1109/ROSE.2019.8790405>.
- Soleymani, F., Miah, S., Spinello, D., 2022. Temporal difference learning of area coverage control with multi-agent systems. In: *2022 IEEE International Symposium on Robot and Sensors Environments (ROSE)*. IEEE, pp. 1–8.
- Soleymani, F., Paquet, E., 2020. Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—deepbreath. *Expert Systems with Applications* 156, 113456.
- Soleymani, F., Paquet, E., 2021. Deep graph convolutional reinforcement learning for financial portfolio management—deep-pocket. *Expert Systems with Applications* 182, 115127.
- Spinello, D., Stilwell, D.J., 2014. Distributed full-state observers with limited communication and application to cooperative target localization. *J. Dyn. Syst. Meas. Control - Trans. ASME* 136 (3).
- Sutton, R.S., 1988. Learning to predict by the methods of temporal differences. *Mach. Learn.* 3 (1), 9–44.
- Sutton, R.S., Barto, A.G., 2018. *Reinforcement Learning: An Introduction*, second ed. In: *Adaptive Computation and Machine Learning Series*, The MIT Press, Cambridge, Massachusetts.
- Tang, Y., Agrawal, S., 2019. Discretizing continuous action space for on-policy optimization. *arXiv preprint arXiv:1901.10500*.

- Vamvoudakis, K.G., Lewis, F.L., 2010. Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem. *Automatica* 46 (5), 878–888.
- Vrabie, D., Lewis, F.L., 2008. Adaptive optimal control algorithm for continuous-time nonlinear systems based on policy iteration. In: 2008 47th IEEE Conference on Decision and Control. IEEE, pp. 73–79.
- Vrabie, D., Pastravanu, O., Abu-Khalaf, M., Lewis, F.L., 2009a. Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica* 45 (2), 477–484.
- Vrabie, D., Vamvoudakis, K., Lewis, F., 2009b. Adaptive optimal controllers based on generalized policy iteration in a continuous-time framework. In: 2009 17th Mediterranean Conference on Control and Automation. IEEE, pp. 1402–1409.
- Werbos, P.J., 1989. Neural networks for control and system identification. In: Proceedings of the 28th IEEE Conference on Decision and Control. IEEE, pp. 260–265.
- Werbos, P., 1992. Approximate dynamic programming for real time control and neural modelling. *Handb. Intell. Control: Neural Fuzzy Adapt. Approaches* 493–525.
- Xiao, J., Wang, G., Zhang, Y., Cheng, L., 2020. A distributed multi-agent dynamic area coverage algorithm based on reinforcement learning. *IEEE Access* 8, 33511–33521. <http://dx.doi.org/10.1109/ACCESS.2020.2967225>.
- Xu, X., Diaz-Mercado, Y., 2020. Multi-agent control using coverage over time-varying domains. In: 2020 American Control Conference (ACC). IEEE, pp. 2030–2035.
- Zhang, G., Fricke, G.K., Garg, D.P., 2013. Spill detection and perimeter surveillance via distributed swarming agents. *IEEE-ASME Trans. Mechatronics* 18 (1), 121–129.
- Zhou, Y., Ho, H.W., 2022. Online robot guidance and navigation in non-stationary environment with hybrid hierarchical reinforcement learning. *Eng. Appl. Artif. Intell.* 114, 105152. <http://dx.doi.org/10.1016/j.engappai.2022.105152>.
- Zhu, P., Ren, W., 2020. Fully distributed joint localization and target tracking with mobile robot networks. *IEEE Trans. Control Syst. Technol.* 29 (4), 1519–1532.

Bibliography

- [1] H. Lodish and S. L. Zipursky, “Molecular cell biology,” *Biochem Mol Biol Educ*, vol. 29, pp. 126–133, 2001.
- [2] M. A. Clark and J. Choi, “[etextbook] biology-2e,” 2018.
- [3] F. Zhang, W. Shi, J. Zhang, M. Zeng, M. Li, and L. Kurgan, “Probselect: accurate prediction of protein-binding residues from proteins sequences via dynamic predictor selection,” *Bioinformatics*, vol. 36, pp. i735–i744, 12 2020.
- [4] A. Chatr-Aryamontri, A. Ceol, L. Licata, and G. Cesareni, “Protein interactions: integration leads to belief,” *Trends in Biochemical Sciences*, vol. 33, no. 6, pp. 241–241, 2008.
- [5] J. De Las Rivas and C. Fontanillo, “Protein–protein interactions essentials: key concepts to building and analyzing interactome networks,” *PLoS computational biology*, vol. 6, no. 6, p. e1000807, 2010.
- [6] S. Jones and J. M. Thornton, “Principles of protein-protein interactions,” *Proceedings of the National Academy of Sciences*, vol. 93, no. 1, pp. 13–20, 1996.
- [7] H.-X. Zhou and Y. Shan, “Prediction of protein interaction sites from sequence profile and residue neighbor list,” *Proteins: Structure, Function, and Bioinformatics*, vol. 44, no. 3, pp. 336–343, 2001.
- [8] L. Skrabanek, H. K. Saini, G. D. Bader, and A. J. Enright, “Computational prediction of protein–protein interactions,” *Molecular biotechnology*, vol. 38, no. 1, pp. 1–17, 2008.
- [9] Z.-H. You, L. Zhu, C.-H. Zheng, H.-J. Yu, S.-P. Deng, and Z. Ji, “Prediction of protein-protein interactions from amino acid sequences using a novel multi-scale continuous and discontinuous feature set,” in *BMC bioinformatics*, vol. 15, pp. 1–9, Springer, 2014.
- [10] S. Sandhya, R. Mudgal, G. Kumar, R. Sowdhamini, and N. Srinivasan, “Protein sequence design and its applications,” *Current opinion in structural biology*, vol. 37, pp. 71–80, 2016.
- [11] S. Nivedha and S. Bhavani, “A survey on prediction of protein-protein interactions,” in *Journal of Physics: Conference Series*, vol. 1937, p. 012011, IOP Publishing, 2021.
- [12] T. Sun, B. Zhou, L. Lai, and J. Pei, “Sequence-based prediction of protein protein interaction using a deep-learning algorithm,” *BMC bioinformatics*, vol. 18, no. 1, pp. 1–8, 2017.
- [13] J. P. Mackay, M. Sunde, J. A. Lowry, M. Crossley, and J. M. Matthews, “Protein interactions: is seeing believing?,” *Trends in Biochemical Sciences*, vol. 32, no. 12, pp. 530–531, 2007.
- [14] Z. Zhao and X. Gong, “Protein-protein interaction interface residue pair prediction based on deep learning architecture,” *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, pp. 1753–1759, 2017.
- [15] J. Zhao, Y. Cao, and L. Zhang, “Exploring the computational methods for protein-ligand binding site prediction,” *Computational and structural biotechnology journal*, vol. 18, pp. 417–426, 2020.
- [16] H. Lu, Q. Zhou, J. He, Z. Jiang, C. Peng, R. Tong, and J. Shi, “Recent advances in the development of protein–protein interactions modulators: mechanisms and clinical trials,” *Signal transduction and targeted therapy*, vol. 5, no. 1, pp. 1–23, 2020.

- [17] L. Liu, X. Zhu, Y. Ma, H. Piao, Y. Yang, X. Hao, Y. Fu, L. Wang, and J. Peng, "Combining sequence and network information to enhance protein-protein interaction prediction," *BMC bioinformatics*, vol. 21, no. 16, pp. 1–13, 2020.
- [18] Y. Wang, Z. You, L. Li, and Z. Chen, "A survey of current trends in computational predictions of protein-protein interactions," *Frontiers of Computer Science*, vol. 14, no. 4, pp. 1–12, 2020.
- [19] R. Jansen, H. Yu, D. Greenbaum, Y. Kluger, N. J. Krogan, S. Chung, A. Emili, M. Snyder, J. F. Greenblatt, and M. Gerstein, "A bayesian networks approach for predicting protein-protein interactions from genomic data," *science*, vol. 302, no. 5644, pp. 449–453, 2003.
- [20] J.-F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G. F. Berriz, F. D. Gibbons, M. Dreze, N. Ayivi-Guedehoussou, *et al.*, "Towards a proteome-scale map of the human protein-protein interaction network," *Nature*, vol. 437, no. 7062, pp. 1173–1178, 2005.
- [21] U. Stelzl, U. Worm, M. Lalowski, C. Haenig, F. H. Brembeck, H. Goehler, M. Stroedicke, M. Zenkner, A. Schoenherr, S. Koeppen, *et al.*, "A human protein-protein interaction network: a resource for annotating the proteome," *Cell*, vol. 122, no. 6, pp. 957–968, 2005.
- [22] S. Ferrari, F. Pellati, and M. Costi, "Disruption of protein-protein interfaces," 2013.
- [23] F. Browne, H. Zheng, H. Wang, and F. Azuaje, "From experimental approaches to computational techniques: a review on the prediction of protein-protein interactions.," *Advances in Artificial Intelligence (16877470)*, 2010.
- [24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [25] O. Jangmin, J. Lee, J. W. Lee, and B.-T. Zhang, "Adaptive stock trading with dynamic asset allocation using reinforcement learning," *Information Sciences*, vol. 176, no. 15, pp. 2121–2147, 2006.
- [26] J. Wu, E. Paquet, H. L. Viktor, and W. Michalowski, "Paying attention: Using a siamese pyramid network for the prediction of protein-protein interactions with folding and self-binding primary sequences," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2021.
- [27] P. Jiang, S. Huang, Z. Fu, Z. Sun, T. M. Lakowski, and P. Hu, "Deep graph embedding for prioritizing synergistic anticancer drug combinations," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 427–438, 2020.
- [28] X.-w. Chen and J. C. Jeong, "Sequence-based prediction of protein interaction sites with an integrative method," *Bioinformatics*, vol. 25, no. 5, pp. 585–591, 2009.
- [29] J. Zahiri, O. Yaghoubi, M. Mohammad-Noori, R. Ebrahimpour, and A. Masoudi-Nejad, "Ppievo: Protein-protein interaction prediction from pssm based evolutionary information," *Genomics*, vol. 102, no. 4, pp. 237–242, 2013.
- [30] Z.-H. You, K. C. Chan, and P. Hu, "Predicting protein-protein interactions from primary protein sequences using a novel multi-scale local feature representation scheme and the random forest," *PloS one*, vol. 10, no. 5, p. e0125811, 2015.
- [31] F. Soleymani, E. Paquet, H. Viktor, W. Michalowski, and D. Spinello, "Protein-protein interaction prediction with deep learning: A comprehensive review," *Computational and Structural Biotechnology Journal*, 2022.
- [32] H. Chen and H.-X. Zhou, "Prediction of interface residues in protein-protein complexes by a consensus neural network method: test against nmr data," *Proteins: Structure, Function, and Bioinformatics*, vol. 61, no. 1, pp. 21–35, 2005.
- [33] E. Rodgers-Melnick, M. Culp, and S. P. DiFazio, "Predicting whole genome protein interaction networks from primary sequence data in model and non-model organisms using ents," *BMC genomics*, vol. 14, no. 1, pp. 1–17, 2013.
- [34] A. Emamjomeh, B. Goliaei, A. Torkamani, R. Ebrahimpour, N. Mohammadi, and A. Parsian, "Protein-protein interaction prediction by combined analysis of genomic and conservation information," *Genes & genetic systems*, vol. 89, no. 6, pp. 259–272, 2014.

- [35] M. Kotlyar, C. Pastrello, F. Pivetta, A. Lo Sardo, C. Cumbaa, H. Li, T. Naranian, Y. Niu, Z. Ding, F. Vafaee, *et al.*, “In silico prediction of physical protein interactions and characterization of interactome orphans,” *Nature methods*, vol. 12, no. 1, pp. 79–84, 2015.
- [36] J.-W. Chang, Y.-Q. Zhou, M. T. Ul Qamar, L.-L. Chen, and Y.-D. Ding, “Prediction of protein–protein interactions by evidence combining methods,” *International journal of molecular sciences*, vol. 17, no. 11, p. 1946, 2016.
- [37] B.-Q. Li, K.-Y. Feng, L. Chen, T. Huang, and Y.-D. Cai, “Prediction of protein-protein interaction sites by random forest algorithm with mrmr and ifs,” *PloS one*, 2012.
- [38] X. Hu, C. Feng, T. Ling, and M. Chen, “Deep learning frameworks for protein-protein interaction prediction,” *Computational and Structural Biotechnology Journal*, 2022.
- [39] J. Jia, X. Li, W. Qiu, X. Xiao, and K.-C. Chou, “ippi-pseaac (cgr): Identify protein-protein interactions by incorporating chaos game representation into pseaac,” *Journal of theoretical biology*, vol. 460, pp. 195–203, 2019.
- [40] J. Pan, L.-P. Li, C.-Q. Yu, Z.-H. You, Y.-J. Guan, and Z.-H. Ren, “Sequence-based prediction of plant protein-protein interactions by combining discrete sine transformation with rotation forest,” *Evolutionary Bioinformatics*, vol. 17, p. 11769343211050067, 2021.
- [41] M. Dorn, M. B. e Silva, L. S. Buriol, and L. C. Lamb, “Three-dimensional protein structure prediction: Methods and computational strategies,” *Computational biology and chemistry*, vol. 53, pp. 251–276, 2014.
- [42] A. Sagheer and M. Kotb, “Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems,” *Scientific reports*, vol. 9, no. 1, pp. 1–16, 2019.
- [43] H. Wang, Y. Zhang, J. Liang, and L. Liu, “Dafa-bilstm: Deep autoregression feature augmented bidirectional lstm network for time series prediction,” *Neural Networks*, vol. 157, pp. 240–256, 2023.
- [44] K. Dandajena, I. M. Venter, M. Ghaziasgar, and R. Dodds, “Complex sequential data analysis: A systematic literature review of existing algorithms,” in *Conference of the South African Institute of Computer Scientists and Information Technologists 2020*, pp. 44–50, 2020.
- [45] M. AlQuraishi, “Proteinnet: a standardized data set for machine learning of protein structure,” *BMC bioinformatics*, vol. 20, no. 1, pp. 1–10, 2019.
- [46] D. H. Dorr and A. M. Denton, “Establishing relationships among patterns in stock market data,” *Data & Knowledge Engineering*, vol. 68, no. 3, pp. 318–337, 2009.
- [47] W. Bessler, H. Opfer, and D. Wolff, “Multi-asset portfolio optimization and out-of-sample performance: an evaluation of black–litterman, mean-variance, and naïve diversification approaches,” *The European Journal of Finance*, vol. 23, no. 1, pp. 1–30, 2017.
- [48] J. J. Ellis, F. P. Huard, C. M. Deane, S. Srivastava, and G. R. Wood, “Directionality in protein fold prediction,” *BMC bioinformatics*, vol. 11, no. 1, pp. 1–16, 2010.
- [49] J. Wang, B. Wicher, V. Maurizot, and I. Huc, “Directing the self-assembly of aromatic foldamer helices using acridine appendages and metal coordination,” *Chemistry–A European Journal*, 2022.
- [50] A. O. Idowu, A. M. Alashi, I. D. Nwachukwu, T. N. Fagbemi, and R. E. Aluko, “Functional properties of sesame (*sesamum indicum* linn) seed protein fractions,” *Food Production, Processing and Nutrition*, vol. 3, no. 1, pp. 1–16, 2021.
- [51] W. Bao, J. Yue, and Y. Rao, “A deep learning framework for financial time series using stacked autoencoders and long-short term memory,” *PloS one*, vol. 12, no. 7, p. e0180944, 2017.
- [52] S. Gopali, F. Abri, S. Siامي-Namini, and A. S. Namin, “A comparative study of detecting anomalies in time series data using lstm and tcn models,” *arXiv preprint arXiv:2112.09293*, 2021.
- [53] W. Gao, S. P. Mahajan, J. Sulam, and J. J. Gray, “Deep learning in protein structural modeling and design,” *Patterns*, p. 100142, 2020.

- [54] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [55] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:1803.01271*, 2018.
- [56] A. Zhang and J. Paisley, “Deep bayesian nonparametric tracking,” in *International conference on machine learning*, pp. 5833–5841, PMLR, 2018.
- [57] L. Zhu, C. Lu, I. Kamwa, and H. Zeng, “Spatial–temporal feature learning in smart grids: A case study on short-term voltage stability assessment,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1470–1482, 2018.
- [58] G. H. Zerze, F. H. Stillinger, and P. G. Debenedetti, “Computational investigation of retro-isomer equilibrium structures: Intrinsically disordered, foldable, and cyclic peptides,” *FEBS letters*, vol. 594, no. 1, pp. 104–113, 2020.
- [59] F. Soleymani and E. Paquet, “Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—deepbreath,” *Expert Systems with Applications*, vol. 156, p. 113456, 2020.
- [60] F. Soleymani and E. Paquet, “Deep graph convolutional reinforcement learning for financial portfolio management—deep-pocket,” *Expert Systems with Applications*, vol. 182, p. 115127, 2021.
- [61] P. Zhao, S. C. Hoi, J. Wang, and B. Li, “Online transfer learning,” *Artificial Intelligence*, vol. 216, pp. 76–102, 2014.
- [62] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A comprehensive survey on transfer learning,” *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.
- [63] M. Grandini, E. Bagli, and G. Visani, “Metrics for multi-class classification: an overview,” *arXiv preprint arXiv:2008.05756*, 2020.
- [64] F. Soleymani and E. Paquet, “Long-term financial predictions based on feynman–dirac path integrals, deep bayesian networks and temporal generative adversarial networks,” *Machine Learning with Applications*, vol. 7, p. 100255, 2022.
- [65] R. Wan, S. Mei, J. Wang, M. Liu, and F. Yang, “Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting,” *Electronics*, vol. 8, no. 8, p. 876, 2019.
- [66] X. Du, Y. Li, Y.-L. Xia, S.-M. Ai, J. Liang, P. Sang, X.-L. Ji, and S.-Q. Liu, “Insights into protein–ligand interactions: mechanisms, models, and methods,” *International journal of molecular sciences*, vol. 17, no. 2, p. 144, 2016.
- [67] K. T. B. Crosara, E. B. Moffa, Y. Xiao, and W. L. Siqueira, “Merging in-silico and in vitro salivary protein complex partners using the string database: a tutorial,” *Journal of proteomics*, vol. 171, pp. 87–94, 2018.
- [68] M. Nan, M. Trăscău, A. M. Florea, and C. C. Iacob, “Comparison between recurrent networks and temporal convolutional networks approaches for skeleton-based action recognition,” *Sensors*, vol. 21, no. 6, p. 2051, 2021.
- [69] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” *arXiv preprint arXiv:1609.03499*, 2016.
- [70] Y. Zhang, C. Wang, L. Gong, Y. Lu, F. Sun, C. Xu, X. Li, and X. Zhou, “A power-efficient accelerator based on fpgas for lstm network,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 629–630, IEEE, 2017.
- [71] Y.-w. Chen, K. Stanley, and W. Att, “Artificial intelligence in dentistry: current applications and future perspectives,” *Quintessence Int*, vol. 51, no. 3, pp. 248–257, 2020.
- [72] M. Zeng, F. Zhang, F.-X. Wu, Y. Li, J. Wang, and M. Li, “Protein–protein interaction site prediction through combining local and global features with deep neural networks,” *Bioinformatics*, vol. 36, pp. 1114–1120, 2020.

- [73] S. Lu, Y. Li, X. Nan, and S. Zhang, "Attention-based convolutional neural networks for protein-protein interaction site prediction," *bioRxiv*, 2021.
- [74] A. Banerjee and S. K. Pati, "Predicting antiviral drugs for covid-19 treatment using artificial intelligence based approach," in *Understanding COVID-19: The Role of Computational Intelligence*, pp. 245–269, Springer, 2022.
- [75] N. Ravikumar and G. Greenfield, "Glucose-6-phosphate dehydrogenase deficiency: a review," *International Journal of Medical Students*, vol. 8, no. 3, pp. 281–287, 2020.
- [76] M. C. Melo, J. R. Maasch, and C. de la Fuente-Nunez, "Accelerating antibiotic discovery through artificial intelligence," *Communications biology*, vol. 4, no. 1, pp. 1–13, 2021.
- [77] M. Helmy, D. Smith, and K. Selvarajoo, "Systems biology approaches integrated with artificial intelligence for optimized metabolic engineering," *Metabolic Engineering Communications*, vol. 11, p. e00149, 2020.
- [78] M. R. Diansyah, W. A. Kusuma, *et al.*, "Analysis of protein-protein interaction using skyline query on parkinson disease," in *2019 International Conference on Advanced Computer Science and information Systems (ICACSIS)*, pp. 175–180, IEEE, 2019.
- [79] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [80] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [81] K. Madi, E. Paquet, and H. Kheddouci, "New graph distance for deformable 3d objects recognition based on triangle-stars decomposition," *Pattern Recognition*, vol. 90, pp. 297–307, 2019.
- [82] J. Chen, S. Zheng, H. Zhao, and Y. Yang, "Structure-aware protein solubility prediction from sequence through graph convolutional network and predicted contact map," *Journal of cheminformatics*, vol. 13, no. 1, pp. 1–10, 2021.
- [83] J. Rao, X. Zhou, Y. Lu, H. Zhao, and Y. Yang, "Imputing single-cell rna-seq data by combining graph convolution and autoencoder neural networks," *Iscience*, vol. 24, no. 5, p. 102393, 2021.
- [84] Y. Song, S. Zheng, Z. Niu, Z.-H. Fu, Y. Lu, and Y. Yang, "Communicative representation learning on attributed molecular graphs.," in *IJCAI*, vol. 2020, pp. 2831–2838, 2020.
- [85] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," *Advances in neural information processing systems*, vol. 30, 2017.
- [86] M. Baranwal, A. Magner, J. Saldinger, E. S. Turali-Emre, S. Kozarekar, P. Elvati, J. S. VanEpps, N. A. Kotov, A. Violi, and A. O. Hero, "Struct2graph: A graph attention network for structure based predictions of protein-protein interactions," *bioRxiv*, 2020.
- [87] F. Soleymani, M. S. Miah, and D. Spinello, "Non-autonomous state-feedback to stabilize the error dynamics in time-varying area coverage control problems," in *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, pp. 1–5, IEEE, 2019.