Novel Deep Learning Models for Spatiotemporal Predictive Tasks

By

Author:
Quang Le

Supervisor:
Francois Chan
Claude D'Amours

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the
Master of Applied Science degree

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Spatiotemporal Predictive Learning (SPL) is an essential research topic involving many practical and real-world applications, e.g., motion detection, video generation, precipitation forecasting, and traffic flow prediction. The problems and challenges of this field come from numerous data characteristics in both time and space domains, and they vary depending on the specific task. For instance, spatial analysis refers to the study of spatial features, such as spatial location, latitude, elevation, longitude, the shape of objects, and other patterns. From the time domain perspective, the temporal analysis generally illustrates the time steps and time intervals of data points in the sequence, also known as interval recording or time sampling. Typically, there are two types of time sampling in temporal analysis: regular time sampling (i.e., the time interval is assumed to be fixed) and the irregular time sampling (i.e., the time interval is considered arbitrary) related closely to the continuous-time prediction task when data are in continuous space. Therefore, an efficient spatiotemporal predictive method has to model spatial features properly at the given time sampling types.

In this thesis, by taking advantage of Machine Learning (ML) and Deep Learning (DL) methods, which have achieved promising performance in many complicated computational tasks, we propose three DL-based models used for Spatiotemporal Sequence Prediction (SSP) with several types of time sampling. First, we design the Trajectory Gated Recurrent Unit Attention (TrajGRU-Attention) with novel attention mechanisms, namely Motion-based Attention (MA), to improve the performance of the standard Convolutional Recurrent Neural Networks (ConvRNNs) in the SSP tasks. In particular, the TrajGRU-Attention model can alleviate the impact of the vanishing gradient, which leads to the blurry effect in the long-term predictions and handle both regularly sampled and irregularly sampled time series. Consequently, this model can work effectively with different scenarios of spatiotemporal sequential data, especially in the case of time series with missing time steps. Second, by taking the idea of Neural Ordinary Differential Equations (NODEs), we propose Trajectory Gated Recurrent Unit integrating Ordinary Differential Equation techniques (TrajGRU-ODE) as a continuous time-series model. With Ordinary Differential Equation (ODE) techniques and the TrajGRU neural network, this model can perform continuous-time spatiotemporal prediction tasks and generate resulting output with high accuracy. Compared to TrajGRU-Attention, TrajGRU-ODE benefits from the development of efficient and accurate ODE solvers. Ultimately, we attempt to combine those two models to create TrajGRU-Attention-ODE. NODEs are still in their early stage of research, and recent ODE-based models were designed for many relatively simple tasks. In this thesis, we will train the models with several video datasets to verify the ability of the proposed models in practical applications.

To evaluate the performance of the proposed models, we select four available spatiotemporal datasets based on the complexity level, including the MovingMNIST, MovingMNIST++, and two real-life datasets: the weather radar HKO-7 and KTH Action. With each dataset, we train, validate, and test with distinct types of time sampling to justify the prediction ability of our models. In summary, the experimental results on the four datasets indicate the proposed models can generate predictions properly with high accuracy and sharpness. Significantly, the proposed models outperform state-of-the-art ODE-based approaches under SSP tasks with different circumstances of interval recording.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisors, Dr. Francois Chan, and Dr. Claud D'Amours, for giving me the opportunity to pursue my graduate study and research at the University of Ottawa. I also greatly appreciate the funding that the Directorate of Technical Airworthiness & Engineering Support has provided for this project.

Second, I would like to give my special thanks to Dr. Tuan Do-Hong, who gave me valuable advice on my master's courses.

I also want to extend my thanks to my friends The Le Van, Matthew Southcott, and Jana Rasras, for their valuable suggestions and encouragement.

Finally, I would like to thank my parents for encouraging and supporting me over these years. Their care and patience helped me finish my graduate study.

Table of Content

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Networks
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
ConvGRU	Convolutional Gated Recurrent Unit
ConvLSTM	Convolutional Long-Short Term Memory
ConvRNN	Convolutional Recurrent Neural Network
Conv-TT-LSTM	Convolutional Tensor-Train Long Short-Term Memory
DL	Deep Learning
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MA	Motion-based Attention
MAE	Mean Absolute Error
MDA	Motion-based Decoding Attention
MEA	Motion-based Encoding Attention
ML	Machine Learning
MSE	Mean Square Error
NLP	Natural Language Processing
NODEs	Neural Ordinary Differential Equations
ODE	Ordinary Differential Equation

PSNR	Peak-Signal-to-Noise Ratio
RNN	Recurrent Neural Network
SPL	Spatiotemporal Predictive Learning
SSIM	Structural Similarity Index Measure
SSP	Spatiotemporal Sequence Prediction
TrajGRU	Trajectory Gated Recurrent Unit
TrajGRU-Attention	Trajectory Gated Recurrent Unit Attention
TrajGRU-Attention-ODE	Trajectory Gated Recurrent Unit Attention integrating Ordinary Differential Equation
TrajGRU-ODE	Trajectory Gated Recurrent Unit integrating Ordinary Differential Equation

Chapter 1: Introduction

1.1 Motivation

Spatiotemporal Sequence Predictive (SSP) learning has been an essential research topic and plays a critical role in many real-world applications, attracting widespread attention and investigation in recent years. In real life, spatiotemporal data and phenomena often occur around us whenever we collect data across space and time. The common cases are video clips which we watch on social media daily and include many other complicated spatiotemporal matters related to the scientific and industrial fields, such as the traffic flow used in the transformation and diffusion of air pollutants in the environment, regional rainfall, radar echo, satellite images used in the weather forecasting field, etc. Overall, SSP learning refers to the approaches used for studying and processing these data for practical tasks ranging from many daily practical applications (e.g., motion prediction, physical scene understanding, traffic flow prediction) to many global-scale applications (e.g., precipitation nowcasting, weather hazard forecasting) [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Nowadays, we witness a tremendous and unstoppable growth of Artificial Intelligence (AI) technology as well as Deep Learning (DL) and Machine Learning (ML) techniques, which have been proven to be the potential solutions for complicated computational tasks. With SSP learning tasks, researchers, and enthusiasts alike, have studied and proposed various ML algorithms and advanced DL-based models with relatively promising performances. However, prior works showed that the performance of a specific model varies significantly depending on the different tasks and datasets, especially with the spatiotemporal sequences. This data type requires an analysis of both the space and time domains, making the job complicated and challenging [3, 4, 8, 11, 12, 13, 14]. However, we are still in the early stages of the SSP field of research. Therefore, there is still much room to modify and develop the methods by investigating the pros and cons of previous related works and learning from the practical experiments with spatiotemporal datasets.

The spatiotemporal data's complexity stems from its properties (i.e., spatial reference, time reference, auto-correlation, and heterogeneity) [15, 16]. From the perspective of the space domain, the spatial reference indicates many different features associated with the spatial location, latitude, elevation, longitude, and other terms like the distribution, intensity, and shape of objects in the case of the image data. On the other hand, the temporal reference illustrates the evolution of things over time. In this thesis, we are considered in the time interval between two adjacent sampled data points and whether this interval is fixed or varying. Typically, there are two types of time sampling in temporal analysis: regular time sampling (i.e., the time interval is assumed to be constant) and irregular time sampling (i.e., the time interval is considered arbitrary).

Although some sequence models usually apply a standard time sampling variable and use the concept of frame rate to sample the data, capturing samples at the same time interval is relatively

complicated and challenging. Moreover, various scientific spatiotemporal data, such as continuous-time physical modelling (i.e., the data are collected in the continuous domain), economics, and medical data analysis, do not follow the regular time sampling assumption. In addition, data collected in real-time frequently miss values at random timesteps because of hardware errors, noises, and other outside effects. Eventually, the spatial and temporal features incorporate, leading to autocorrelation (i.e., spatial, and temporal correlations between near data frames) and heterogeneity. Therefore, an efficient model must properly extract spatial and temporal features simultaneously and learn their dependencies for a specified time sampling assumption (regular or irregular).

Recent studies proposed approaches to solve complicated SSP tasks with fixed frame rates [1, 2, 3, 7, 11, 13, 17, 18]. Other works used the concept of irregular time sampling in continuous-time video generation and prediction [14, 19, 20, 21, 22]. However, those continuous-time models experimented with relatively simple datasets; thus, they are not really practical for real-world applications. Based on the observations, we attempt to build DL-based models for the overall spatiotemporal predictive learning problem, which fits different spatial datasets and time sampling assumptions.

1.2 Problem statement

With the given spatiotemporal sequence input, the main objective of SSP task is to forecast the sequence output at future timesteps. To do that, we build trainable DL sequence-to-sequence models consisting of two parts: the encoder and decoder, which we will present later. Simply put, the encoder receives, processes the input sequence, and sends practical information to the decoder, generating output predictions.

In our work, the spatiotemporal sequence is commonly represented by a 4-D tensor $\boldsymbol{X} \in \mathbb{R}^{S\times C\times H\times W}$; where S is the number of timesteps or the sequence length, C being the number of channels, H and W being the height and the width of a single frame, respectively. For the preprocessing, each sampled sequence is split into two subsequences, one used as the input data sequence of the encoder and the other used as the target (i.e., the ground truth of the output prediction sequence) to compare with the output prediction generated by the decoder. As such, the main SSP problem formulation can be expressed as

$$\begin{cases} \boldsymbol{X} = \left(\boldsymbol{X}_{input} | \boldsymbol{X}_{target}\right) = \left(\boldsymbol{X}(t_1) \quad \dots \quad \boldsymbol{X}(t_N) | \boldsymbol{X}(t_{N+1}) \quad \dots \quad \boldsymbol{X}(t_{N+K})\right) \\ \boldsymbol{X}_{prediction} = f_{decoder}\left(f_{encoder}\left(\boldsymbol{X}_{input}\right)\right) = f_{decoder}\left(f_{encoder}\left(\boldsymbol{X}(t_1) \quad \dots \quad \boldsymbol{X}(t_N)\right)\right) \\ = \left(\widetilde{\boldsymbol{X}}(t_{N+1}) \quad \dots \quad \widetilde{\boldsymbol{X}}(t_{N+K})\right) \\ \qquad min\, f_{loss}\left(\boldsymbol{X}_{prediction}, \boldsymbol{X}_{target}\right) \\ \qquad \Leftrightarrow min\, f_{loss}\left(\left(\widetilde{\boldsymbol{X}}(t_{N+1}) \quad \dots \quad \widetilde{\boldsymbol{X}}(t_{N+K})\right), \left(\boldsymbol{X}(t_{N+1}) \quad \dots \quad \boldsymbol{X}(t_{N+K})\right)\right) \end{cases}, \quad (1.1)$$

where $\boldsymbol{X} \in \mathbb{R}^{S\times C\times H\times W}$ indicates the examined sequence, consisting of S data frames at time steps $(t_1 \quad t_2 \quad \cdots \quad t_S)$; $\boldsymbol{X}_{input} \in \mathbb{R}^{N\times C\times H\times W}$, $\boldsymbol{X}_{target} \in \mathbb{R}^{K\times C\times H\times W}$ are the input and target sequences, respectively (i.e., $S = N + K$); $\boldsymbol{X}_{prediction} \in \mathbb{R}^{K\times C\times H\times W}$ presents the output prediction sequence of the model; $f_{encoder}$, $f_{decoder}$ represent for the encoder and decoder of the model, respectively; f_{loss} indicates the loss function guiding the training process.

In particular, the time interval $\Delta t = t_{i+1} - t_i$ between two adjacent frames is fixed for the regular time sampling. In contrast, the time interval $\Delta t = t_{i+1} - t_i$ varies and depends on the given time steps in the irregular sampling assumption. To solve the overall SSP problem, we will examine our models with two sampling types, regular time sampling and irregular time sampling. Furthermore, we apply our models with different spatiotemporal datasets ranging from simple spatial features to complicated ones (i.e., different levels of spatial, temporal references, correlation, and heterogeneity). In addition, the problem statement formula (1.1) contains the concepts of extrapolation and interpolation tasks. To perform a specific task, we can define the timestamps of the target frames.

1.3 Overview of our system

Overall, we build the system based on the following steps

The first step is the model designing process. The proposed DL models are essentially based on conventional DL sequence-to-sequence models. To design the models, we are concerned with three main elements: the model structure, model architecture, and building blocks. The overview of this step is shown in Figure 1.1

- Model structure: As mentioned earlier in the primary problem formulation, the models follow the encoding-decoding system with two main parts: the encoder and the decoder. Each component is constructed by multiple layers, which are essentially represented by specified types of artificial neural networks. In terms of functioning, the encoder obtains the input, processes it, then transfers essential information from the observations to the decoder generating the output predictions.

- Model architecture: This element illustrates the connection between the encoder and the decoder, different layers, and small neural network cells. Therefore, the information flow running forward and backward through the entire system, can be visualized. Since the model architecture impacts the system's performance directly, we attempt to design a suitable architecture for the SSP task.

- Building blocks: These blocks represent the neural network type of each layer. We classify the building blocks into two classes: the central block and the connection block. As such, the central blocks have the function of learning and storing the empirical knowledge of the

observation, while the connection blocks process and transport the information through the system.

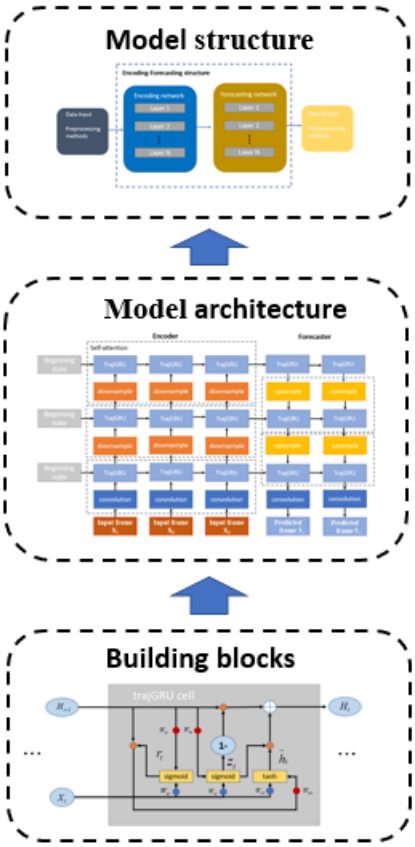


Figure 1.1: Overview of the model designing process. Three main elements are the model structure, model architecture, and building blocks.

The second step is the data selecting and sampling. Four spatiotemporal datasets are used, including the MovingMNIST, MovingMNIST++, radar weather HKO-7, and the KTH Action. We choose those datasets according to the order of increasing complexity in spatial features. As demonstrated in the below section, our overall SSP task covers constant time-interval, irregularly sampled time-series prediction, and even continuous-time video generation (i.e., the output data are in the continuous domain). With each dataset, we apply different time-sampling methods to evaluate the performance of models in this task. For the encoding-decoding structure, a single examined sequence is split into two subsequences: the input data subsequence and the target subsequence. Consequently, we can consider four sampling methods in the time domain, as follows, and as illustrated in Figure 1.2

- Regular time sampling is applied to both input and target subsequences.
- Irregular time sampling is applied to the input subsequence and regular time sampling is applied on the target subsequence.

- Regular time sampling is applied to the input subsequence and irregular time sampling on the target subsequence.
- Irregular time sampling is applied to both input and target subsequences.

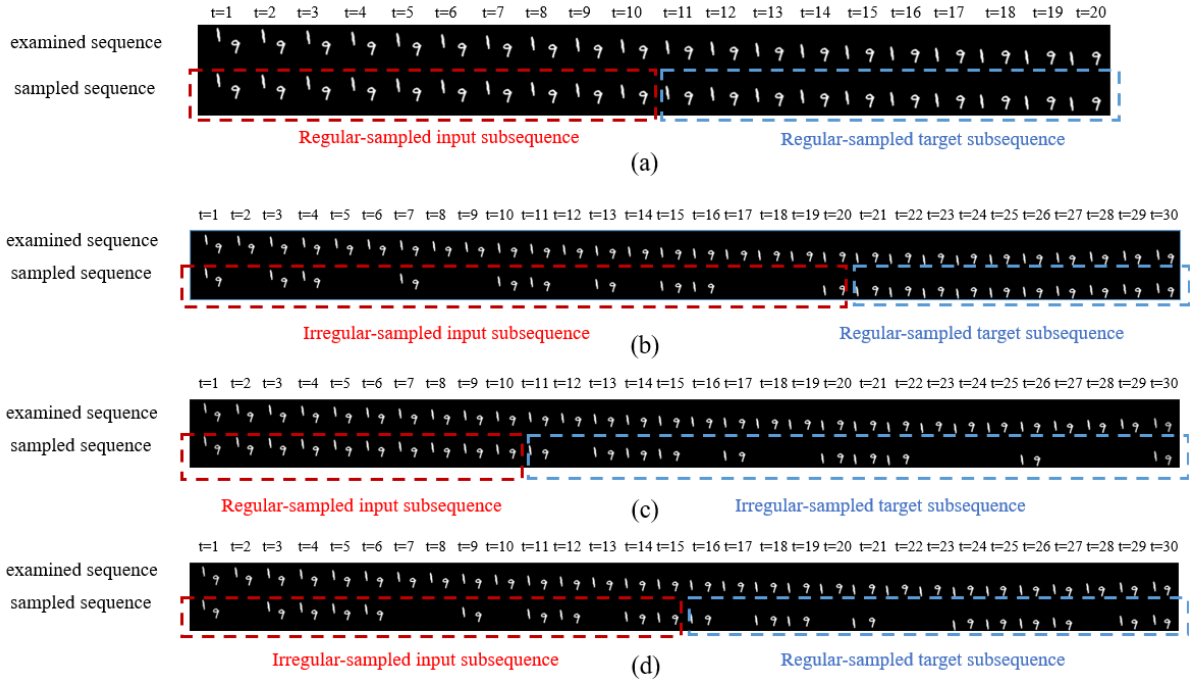


Figure 1.2: Overview of the time sampling methods. There are four cases: a) The regular time sampling is applied to both input and target subsequences; b) The irregular time sampling is applied to the input subsequence and the regular time sampling is applied to the target subsequence; c) The regular time sampling is applied to the input subsequence and the irregular time sampling to the target subsequence; d) The irregular time sampling is applied to both input and target subsequences.

Based on the first observations, these scenarios are sorted in ascending order of difficulty. We suppose these four scenarios can represent the overall SSP task. Then, we investigate the performance of the proposed models with the expectation that they can work efficiently in all four cases with different spatiotemporal datasets.

The third step indicates the training and validating processes. In this step, we define the loss function and optimizing method guiding the training process. Besides, we employ the learning rate updating mechanism and the early stopping method for generalization.

The final step indicates the testing process. To test models' performance, we calculate some famous evaluation metrics, especially image quality metrics: the Mean Square Error (MSE), Mean Absolute Error (MAE), Structural Similarity Index Measure (SSIM), and Peak Signal to Noise Ratio (PSNR). Finally, we present some observations and conclusions of our models based on the results.

1.4 Contributions

The major contributions in this thesis can be summarized as follows

- The overall SSP task working with different temporal scenarios is proposed, as illustrated below. Therefore, the models can handle spatiotemporal datasets with varying levels of complexity and sampling types (i.e., the temporal, spatial references, auto-correlation, heterogeneity).
- We propose three novel DL-based sequence-to-sequence models which can work effectively with the overall SSP task.
    - To alleviate the impact of the vanishing gradient issue occurring quite often in traditional Recurrent Neural Networks (RNNs), we investigate and design a novel attention mechanism called Motion-based Attention (MA) based on the motion characteristic analysis. Moreover, this attention mechanism has a side effect allowing the system to learn and remember the spatial features at arbitrary time steps. Then, the MA is integrated to the TrajGRU model to create a novel Trajectory Gated Recurrent Unit Attention (TrajGRU-Attention). As a result, the TrajGRU-Attention model is the improved version of the standard TrajGRU for the overall SSP task.
    - By taking the idea of Neural Ordinary Differential Equations (NODEs), Trajectory Gated Recurrent Unit integrating Ordinary Differential Equation techniques (TrajGRU-ODE) is designed as a continuous time-series model. This model can perform continuous-time spatiotemporal prediction tasks with complex data and generate results with high accuracy. Compared to TrajGRU-Attention, TrajGRU-ODE benefits from the development of efficient and accurate ODE solvers.
    - To further investigate the capacity of learning continuous-time and spatiotemporal dynamics, the TrajGRU-ODE model is combined with the Motion-Attention to design the Trajectory Gated Recurrent Unit integrating the Attention and Ordinary Differential Equation (TrajGRU-Attention-ODE).
- We propose and design new model architecture, combining the conventional stacked style and the zigzag style; this architecture enables the models to learn the extracted features at different scaling levels and increases the memory ability of the whole system. Therefore, it helps weaken the effect of the vanishing gradient issue.

- For the training process, we use the combination of two widely used errors (the Mean Square Error (MSE) and Mean Absolute Error (MAE)) and one image quality metric (Structural Similarity Index Measure (SSIM)) as the loss function.

1.5 Outline

The upcoming chapters are organized as follows

Chapter 2 “Literature review”: This chapter starts by introducing the fundamental knowledge of spatiotemporal data, including temporal and spatial analysis. Then, comprehensive review of Deep Learning will be provided, including widely used neural networks and related components to build sequence models.

Chapter 3 “Related Work”: This chapter reviews other research efforts related to this work, including Convolutional Recurrent Neural Networks (ConvRNNs), ODE-based models, Generative Adversarial Networks (GANs), and transformers.

Chapter 4 “Proposed Method”: This chapter introduces three proposed models, TrajGRU-Attention, TrajGRU-ODE, and TrajGRU-Attention-ODE. The model structure, architecture, and the building blocks of each model are explained in detail.

Chapter 5 “Experiments”: This chapter starts by introducing four spatiotemporal datasets, including the MovingMNIST, MovingMNIST++, HKO-7 and the KTH Action. Then, we conduct experiments and analyze the results from our proposed models for the overall SSP task. Finally, we compare them with other recent related models to demonstrate our models' ability.

Chapter 6 “Conclusion and Future Work”: This chapter summarizes the proposed models for the overall SSP task and presents the potential improvement that could be considered in future research.

Chapter 2: Literature review

In this chapter, we first present the fundamental knowledge of sequence learning. Then, some deep learning terms will be reviewed that we will employ to build the models in this work.

2.1 Sequence learning

There are various data types in the data science field, each of which fits a particular circumstance and application. Based on the statistical data distributions, we can classify the data into two groups: independent, identically distributed (i.i.d.) data and the group of data that do not follow this assumption. For example, in signal processing, most prior algorithms are investigated and designed for i.i.d. distributed data which share the same probability distribution and are independent events. However, many applications are not interested in this type of data, especially when it comes to sequential data. We can list several sequential data we meet daily, such as text streams, audio clips, video clips, and different time-series data found in economics, medicine, and physical sciences. These data are often analyzed in sequence as the practical knowledge lies in the dependencies between data points within a specific series. For example, we need to watch a video clip or read a text document sequentially to understand the content instead of looking at a single data point. Therefore, sequence learning has been getting much more attention in recent years because of the essential role of sequential data in real-world applications.

Sequence learning is the study of data science algorithms designed for applications requiring sequential data [23, 24]. Many models that input and output sequential data are known as the sequence or sequence-to-sequence models [1, 2, 3, 14, 25, 26, 27, 28]. According to the development of artificial intelligence, various advanced sequence models have been designed to meet the high demand of industries.

2.1.1 Temporal analysis

Essentially, temporal analysis (time series analysis) is the study of the statistical characteristics of the time series, which is a particular type of sequential data and is collected at different points in time [3, 15, 16, 20, 29, 30, 31, 32, 33]. By obtaining the practical information of observed data and the dependencies within a time series, we can perform the time series forecasting task, one of the most common use cases in the sequence learning field. For instance, predicting the motions of an object in the future requires knowledge of its movements in a particular evolution process. From the perspective of temporal analysis problems, continuous-time and irregularly sampled time-series predictions are novel and challenging time-series forecasting tasks. Moreover, besides the time-series data following the regular timestep assumption, there is also the time series with non-uniform intervals in the discrete-time domain and sequential data in the continuous-time field. These data types often occur in real life. However, they are much more challenging to handle as

the models must capture the state dynamics at arbitrary time steps (i.e., complex temporal reference directly affects the other spatiotemporal properties). In this thesis, several spatiotemporal datasets will be used and investigated to perform the overall SSP task defined below. Furthermore, the SSP task covers the concept of the irregularly sampled time series and continuous-time prediction. Generally, spatiotemporal data is a special type of sequential data characterized by spatial, temporal, and spatiotemporal correlations.

2.1.2 Spatial analysis

The spatial analysis includes approaches which study and capture the knowledge from the topological, geometric, spatial properties and the spatial dependency of spatial data [3, 15, 16, 22, 31, 32, 33]. Recently, there have been a variety of techniques using different spatial data analytic algorithms and applied in various research fields, such as astronomy, biology, ecology, economics, and meteorology. For instance, earth observation data, such as satellite imagery and weather radar data, are increasingly being used to monitor and forecast atmospheric phenomena and severe weather events. To effectively analyze these data, we must learn and understand their spatial information; thus, spatial analysis has become a fundamental topic getting widespread attention. However, the spatial analysis task is challenging due to many factors like spatial dependency, heterogeneity, and transformation effects (e.g., scaling, distortion, illusion). Therefore, we need advanced models equipped with their learning process so that they can analyze and capture detailed knowledge.

2.2 Deep learning

Deep Learning (DL) and Machine Learning (ML) are essential science research fields of Artificial Intelligence (AI) which allow machines to mimic human thinking and learning processes [34]. Furthermore, these technologies can perform specific tasks even better than humans, especially in significantly complicated applications. Essentially, DL algorithms refer to Artificial Neural Networks (ANNs) inspired by the neural operation of the human brain. Each ANN is constructed by composing multiple layers, and each layer refers to a differentiable parametric function. Recently, there have been various types of ANNs designed for specific problems.

2.2.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is one of the most popular Deep Learning algorithms for processing spatial data (e.g., 2-Dimension and 3-Dimension data), especially image data, which has a grid pattern [34, 35, 36]. A CNN typically consists of different neural layers, and the convolutional layer is the most important one.

A convolutional layer can take input data and assign trainable weights and biases by a set of filters based on the convolution operator in signal processing methods. Generally, each filter is represented by a specific kernel size directly relating to its number of learnable parameters (i.e.,

weights and biases). Compared to the fully connected layers, also known as linear layers, a convolution layer can generate better performance with the image dataset because the reusability of kernel weights following the slicing window process allows convolution layers to learn the complicated spatial features of the data. Moreover, reducing the number of parameters can somewhat address the overfitting issue.

In the simple case: the input $\boldsymbol{I}$ has the size of (C_i, H_i, W_i) and the set of convolutional filters has the size of (C_k, H_k, W_k), the output $\mathbf{O}$ can be precisely described as

$$\boldsymbol{O}(j, H_o, W_o) = \boldsymbol{B}(j) + \sum_{q=0}^{C_i-1} \boldsymbol{W}(j, q, \dots) * \boldsymbol{I}(q, \dots), \tag{2.1}$$

where, "$*$" denotes the convolution operation; $\boldsymbol{W}$ refers the learnable weights and has the size of (C_i, C_k, H_k, W_k), $\boldsymbol{B}$ refers to the biases. As the result, the output size (C_o, H_o, W_o) can be expressed as

$$\begin{aligned} H_o &= \frac{H_i + 2H_p - H_d(H_k - 1) - 1}{H_s} + 1 \\ W_o &= \frac{W_i + 2W_p - W_d(W_k - 1) - 1}{W_s} + 1 \\ C_o &= C_k, \end{aligned} \tag{2.2}$$

where, the padding $p(H_p, W_p)$, the dilation $d(H_d, W_d)$, and the stride $s(H_s, W_s)$ present the hyperparameters of the convolutional layer besides the kernel size $k(H_k, W_k)$. Figure 2.1 illustrates the computational process of a common convolutional layer.

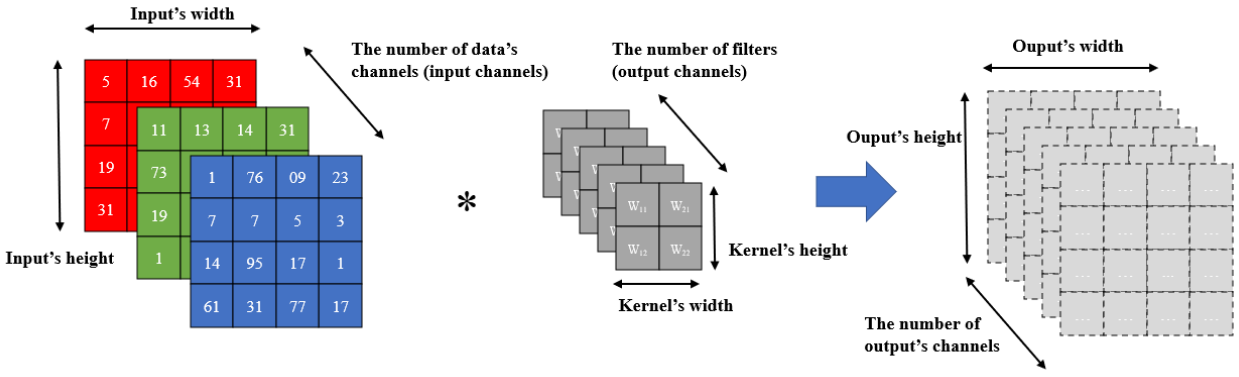


Figure 2.1: The computational process of a convolutional layer.

In addition, we can also apply the convolution layer to process the sequential data with a size of (N, C_i, H_i, W_i). In that case, the output data can be computed as

$$\boldsymbol{O}(n, j, H_o, W_o) = Bias(j) + \sum_{q=0}^{C_i-1} \boldsymbol{W}(j, q, \dots) * \boldsymbol{I}(n, q, \dots), \quad (2.3)$$

In this work, we are also concerned with the transposed convolutional layer or deconvolutional layer, which can be seen as the gradient of the layer below [37]. Typically, encoding-decoding systems combine convolutional and deconvolutional layers for feature extraction.

2.2.2 Recurrent Neural Network

A recurrent neural network (RNN) is a particular type of artificial neural network designed with the ability to process temporal information; thus, it can handle sequential data like time-series data [38, 39]. As mentioned earlier, typical sequential data includes several data points; for example, a time series consists of data points at different time steps that are dependent and related. In terms of the structure, each RNN is commonly equipped with an internal mechanism to learn those dependencies, which stores the input information as hidden states. With each pair of data input at a specific time step and a hidden state at the previous time step, the network will process to generate the new form, containing the information of prior inputs. This recurrent computation works as an internal loop for sequence learning, extended over all observed timestamps of a particular sequence. However, a simple RNN cannot remember the information about the inputs seen in many time steps before, especially when it comes to long-range series. Consequently, this leads to the vanishing gradient problem decreasing the system's overall performance.

Recently, there have been various variants of RNNs proposed for different purposes, and they are generally distinguished based on the internal mechanism. For example, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are two widely used RNNs and the base techniques of many advanced recent sequence models. The core idea of LSTM and GRU is the concept of memory in DL that allows the networks to learn long-term dependencies and store the realistic representations of the input. Essentially, they use gated operations to regulate, understand and forget the information of sequence data.

The LSTM algorithm was first proposed by Hochreiter and Schmidhuber (1997) to address the gradient vanishing issue [40]. Overall, an LSTM neural cell uses three gates, including the forget gate, the input gate, the output gate, and the cell state, as shown in Figure 2.2.

It can be seen in Figure 2.2 that at a specific timestep *t*, an input $\boldsymbol{X}_t$, a previous hidden state $\boldsymbol{H}_{t-1}$ and a previous cell state $\boldsymbol{C}_{t-1}$ are fed into the input gate $\boldsymbol{I}_t$ performing the representation learning task, the forget gate $\boldsymbol{F}_t$ removing unnecessary information of the previous cell state, and the output gate $\boldsymbol{O}_t$ keeps useful information of previous inputs. Then, the updated cell state $\boldsymbol{C}_t$ storing the knowledge of the recent input is computed based on the obtained values at timestep *t*. Finally, the current hidden state $\boldsymbol{H}_t$ is calculated based on the values of the output gate and the updated cell state. Hence, the common LSTM algorithms can be formulated as follows

$$
\begin{aligned}
\boldsymbol{I}_t &= \sigma(\boldsymbol{W}_{xi}\boldsymbol{X}_t + \boldsymbol{W}_{hi}\boldsymbol{H}_{t-1} + \boldsymbol{W}_{ci} \circ \boldsymbol{C}_{t-1} + \boldsymbol{B}_i) \\
\boldsymbol{F}_t &= \sigma\big(\boldsymbol{W}_{xf}\boldsymbol{X}_t + \boldsymbol{W}_{hf}\boldsymbol{H}_{t-1} + \boldsymbol{W}_{cf} \circ \boldsymbol{C}_{t-1} + \boldsymbol{B}_f\big) \\
\boldsymbol{C}_t &= \boldsymbol{F}_t \circ \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \circ tanh(\boldsymbol{W}_{xc}\boldsymbol{X}_t + \boldsymbol{W}_{hc}\boldsymbol{H}_{t-1} + \boldsymbol{B}_c) \\
\boldsymbol{O}_t &= \sigma(\boldsymbol{W}_{xo}\boldsymbol{X}_t + \boldsymbol{W}_{ho}\boldsymbol{H}_{t-1} + \boldsymbol{W}_{co} \circ \boldsymbol{C}_{t-1} + \boldsymbol{B}_o) \\
\boldsymbol{H}_t &= \boldsymbol{O}_t \circ tanh(\boldsymbol{C}_t),
\end{aligned} \tag{2.4}
$$

where "∘" is Hadamard product; $(\boldsymbol{W}_{xi}, \boldsymbol{W}_{xf}, \boldsymbol{W}_{xc}, \boldsymbol{W}_{xo}), (\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{hc}, \boldsymbol{W}_{ho})$ and $\boldsymbol{W}_{co}$ are learnable weights; $(\boldsymbol{B}_i, \boldsymbol{B}_f, \boldsymbol{B}_c, \boldsymbol{B}_o)$ are the biases. Typically, these parameters are assigned by applying neural network layers such as fully connected layers.

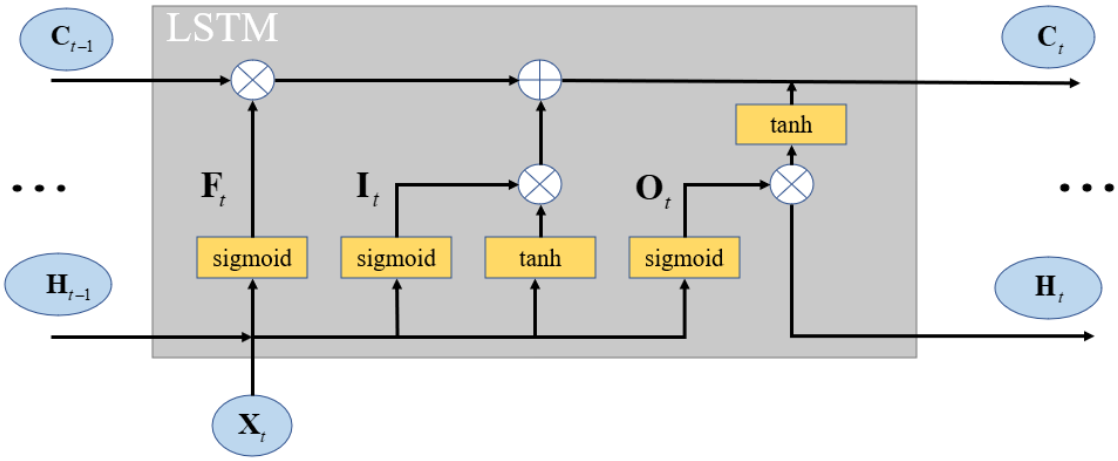


Figure 2.2: LSTM cell.

A slightly more dramatic variation on the RNNs is the Gated Recurrent Unit (GRU), introduced by Cho et al. (2014) [41]. The GRU cell combined the forget and input gates into a single gate, called the update gate. Moreover, GRU networks cross off the concept of the cell state. As a result, the GRU model is more straightforward than the LSTM model, and they have found that the GRU-based models can generate better results than LSTMs for specific tasks and datasets [42, 43]. Figure 2.3 illustrates the internal mechanism of a GRU unit.

Based on Figure 2.3, the main formulas of a standard GRU cell can be expressed as

$$
\begin{aligned}
\boldsymbol{Z}_t &= \sigma(\boldsymbol{W}_{xz}\boldsymbol{X}_t + \boldsymbol{W}_{hz}\boldsymbol{H}_{t-1} + \boldsymbol{B}_z) \\
\boldsymbol{R}_t &= \sigma(\boldsymbol{W}_{xr}\boldsymbol{X}_t + \boldsymbol{W}_{hr}\boldsymbol{H}_{t-1} + \boldsymbol{B}_r) \\
\widetilde{\boldsymbol{H}}_t &= tanh(\boldsymbol{W}_{xh}\boldsymbol{X}_t + \boldsymbol{R}_t \circ (\boldsymbol{W}_{hh}\boldsymbol{H}_{t-1}) + \boldsymbol{B}_h) \\
\boldsymbol{H}_t &= (1 - \boldsymbol{Z}_t) \circ \widetilde{\boldsymbol{H}}_t + \boldsymbol{Z}_t \circ \boldsymbol{H}_{t-1},
\end{aligned} \tag{2.5}
$$

where $\boldsymbol{Z}_t$, $\boldsymbol{R}_t$ represent the update gate, reset gate, respectively, regulating the information flows; $\widetilde{\boldsymbol{H}}_t$ indicates the memory state storing the new information of the recent input data; "∘" denotes the Hadamard product; $(\boldsymbol{W}_{xz}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh})$, $(\boldsymbol{W}_{hz}, \boldsymbol{W}_{hr}, \boldsymbol{W}_{hh})$ are learnable weights;

$(\boldsymbol{B}_i, \boldsymbol{B}_f, \boldsymbol{B}_c, \boldsymbol{B}_o)$ are the biases; "tanh" refers to the Tanh activation function, "σ" refers to the Sigmoid function. Typically, these parameters are assigned by applying neural network layers such as fully connected layers.

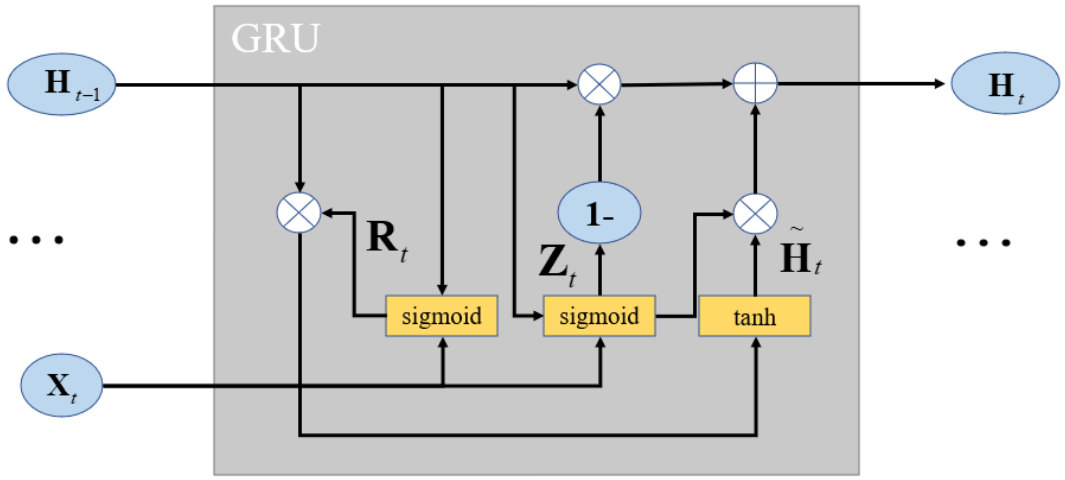


Figure 2.3: GRU cell.

2.2.3 Ordinary Differential Equation Neural Network

In mathematics, an Ordinary Differential Equation (ODE) is a differential equation containing an independent variable and one or more of its derivatives with respect to that variable. ODEs occur quite often in many real-life contexts related to sciences, physics, meteorology, and economics since they are mathematical descriptions of those events' changing and evolution processes [44, 45]. For example, we can derive a simple ODE function demonstrating the rate of change as

$$\frac{dy}{dx} = f(y, x). \tag{2.6}$$

Inspired by ODEs, Chen et al. proposed the Neural Ordinary Differential Equation network (NODE), a new class of deep learning techniques applied to time series data, especially with non-uniform intervals and continuous-time data [19]. Regarding the component structure, NODEs can be seen as continuous variants of residual networks [19, 46]. For the state updating, the popular residual connection is commonly described as

$$\boldsymbol{H}_i = \boldsymbol{H}_{i-1} + f_{\theta_i}(\boldsymbol{H}_{i-1}, \theta_i), \tag{2.7}$$

where f_{θ_i} indicates a specific building block (i.e., the composition of multiple neural layers) containing trainable parameters θ_i. Typically, a particular residual network consists of numerous discrete building blocks.

Instead of defining discrete blocks in residual networks, NODEs use only a single approximation continuous function f_θ represented by a specific neural network. For temporal analysis, this function indicates the derivative of a particular state at a specific time step

$$\frac{d\boldsymbol{H}(t)}{dt} = f_\theta(\boldsymbol{H}(t), \theta, t), \tag{2.8}$$

where θ presents the set of learnable parameters of the function f_θ, also called the ODE network. Solving this ODE, we obtain the following result

$$\boldsymbol{H}(t_i) = \boldsymbol{H}(t_{i-1}) + \int_{t=t_{i-1}}^{t_i} f_\theta(\boldsymbol{H}(t), \theta, t)dt. \tag{2.9}$$

Finally, the integral of function f_θ can be solve by numerical methods that approximate a solution at various discrete evaluation points [45]. Runge-Kutte (RK) and Adam-Bashforth methods are two popular groups of ODE numerical methods [47]. Thanks to the evolution of mathematic programming systems, there are various ODE solvers using these ODE methods, which are implemented in different programming languages and frameworks. Consequently, we can rewrite (2.9) as

$$\boldsymbol{H}(t_i) = ODESolver(f_\theta(\boldsymbol{H}(t), \theta, t), \boldsymbol{H}(t_{i-1}), t_{i-1}, t_i, ODE_{method}), \tag{2.10}$$

Equation (2.10) can be analyzed as an implicit layer defined to satisfy some joint condition of input and output [48]. In this thesis, we will implement one of the most straightforward RK method - Euler's method. Basically, this numerical method operates as the iterative method used to solve first-order differential equations. For instance, we will analyze (2.6). The Euler's method approximate values (x_i, y_i) when the slope $\frac{dy}{dx}$, an initial condition (x_0, y_0), and the time interval Δx_i are given. Then, we can find the values (x_i, y_i) by applying the following recursive formulas

$$\begin{cases} x_i = x_{i-1} + \Delta x_i \\ y_i = y_{i-1} + \frac{dy_{i-1}}{dx_{i-1}} \Delta x_i \end{cases} \tag{2.11}$$

Based on some initial observations, NODEs are more efficient in terms of parameters compared to residual networks because they apply the same weight matrix at each time step. Furthermore, NODEs benefit from the long-term historical development of numerical ODE methods, which allow them to model continuous state dynamics. Therefore, NODEs open a new approach for learning the representations in the continuous space of neural networks and are applied to continuous-time prediction tasks. They are also called ODE-based models in DL techniques. Figure 2.4 compares the process of residual networks and NODEs.

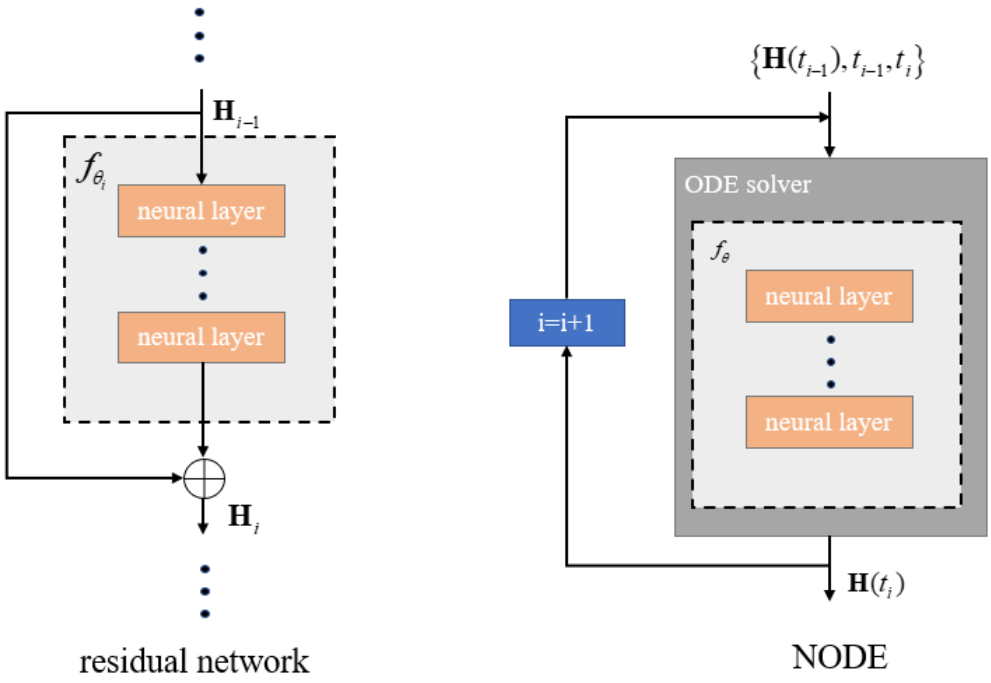


Figure 2.4: Overall structure of residential network and NODE.

2.2.4 Backpropagation through time

Backpropagation Through Time (BPTT) is the weight updating algorithm used in the training process of sequence models. The objective of backpropagation methods is to adjust the trainable weights of models to optimize the cost function (i.e., minimize the error). To do that, they mathematically apply the derivative chain rule and compute the gradients [43, 49].

2.2.4.1 Backpropagation through time for RNNs

To describe a common BPTT algorithm for RNNs, we examine a simple model as

$$\begin{aligned} \boldsymbol{H}_t &= f_{RNN}(\boldsymbol{H}_{t-1}, \boldsymbol{X}_t, \boldsymbol{W}_{RNN}) \\ \boldsymbol{P}_t &= f_{decoder}(\boldsymbol{H}_t, \boldsymbol{W}_{decoder}) \\ L &= f_{loss}\left(\{\boldsymbol{P}_t\}_{t=1,\dots,T}, \{\boldsymbol{X}_t\}_{=1,\dots,T}\right) = \frac{1}{T}\sum_{t=1}^{T} f_{loss}(\boldsymbol{P}_t, \boldsymbol{X}_t), \end{aligned} \tag{2.12}$$

where f_{RNN}, $f_{decoder}$, f_{loss} present the RNN with learnable weights $\boldsymbol{W}_{RNN}$, decoder network with learnable weights $\boldsymbol{W}_{decoder}$, and loss function, respectively; $\boldsymbol{H}_t$, $\boldsymbol{X}_t$ indicate the hidden state and input data at timestep t ; $\mathbf{P}_t$ indicates the final output prediction at timestep t; the loss function value L is calculated using the set of predictions $\{\boldsymbol{P}_t\}_{t=1,\dots,T}$ and inputs $\{\boldsymbol{X}_t\}_{=1,\dots,T}$. As such, the entire step-by-step process of a typical BPTT algorithm for RNNs is illustrated as follows

First, the prediction at each timestep is computed straightforwardly using the input and hidden state. As introduced earlier, the internal mechanism of RNNs works as a loop across time, known as the rolling RNNs. Therefore, the networks need to be unrolled to run backward, as shown in Figure 2.5.

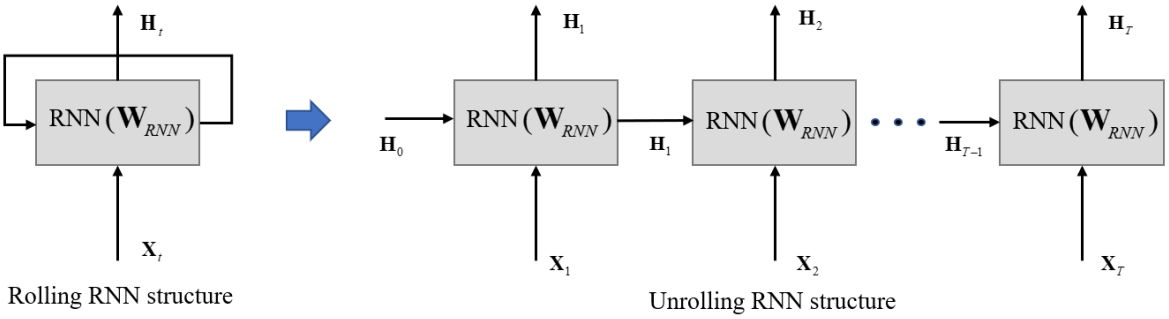


Figure 2.5: Rolling RNN and unrolling RNN structures

To optimize the loss function, we employ the chain rule to compute the gradients regarding the trainable weights of the loss function. With respect to the weights $\boldsymbol{W}_{decoder}$, the derivative of the loss function can be simply calculated using the single-variable chain rule (i.e., the differentiable function is dependent on a single variable)

$$\begin{aligned}\frac{dL}{d\boldsymbol{W}_{decoder}} &= \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{d\boldsymbol{W}_{decoder}} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{d\boldsymbol{P}_t}\frac{d\boldsymbol{P}_t}{d\boldsymbol{W}_{decoder}} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{df_{decoder}(\boldsymbol{H}_t,\boldsymbol{W}_{decoder})}\frac{df_{decoder}(\boldsymbol{H}_t,\boldsymbol{W}_{decoder})}{d\boldsymbol{W}_{decoder}}\end{aligned} \tag{2.13}$$

Based on (2.13), we run the network backward and calculate each gradient separately, then combine them to get the derivative of the loss function. Similarly, we compute the derivative of the loss function regarding the weights $\boldsymbol{W}_{RNN}$, as illustrated below

$$\begin{aligned}\frac{dL}{d\boldsymbol{W}_{RNN}} &= \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{d\boldsymbol{W}_{RNN}} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{d\boldsymbol{P}_t}\frac{d\boldsymbol{P}_t}{d\boldsymbol{W}_{RNN}} = \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{d\boldsymbol{P}_t}\frac{d\boldsymbol{P}_t}{d\boldsymbol{H}_t}\frac{d\boldsymbol{H}_t}{d\boldsymbol{W}_{RNN}} \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{df_{loss}(\boldsymbol{P}_t,\boldsymbol{X}_t)}{d\boldsymbol{P}_t}\frac{d\boldsymbol{P}_t}{d\boldsymbol{H}_t}\frac{df_{RNN}(\boldsymbol{H}_{t-1},\boldsymbol{X}_t,\boldsymbol{W}_{RNN})}{d\boldsymbol{W}_{RNN}}\end{aligned} \tag{2.14}$$

Because of the recurrent computation, the hidden state $\boldsymbol{H}_t$ depends on both the previous state $\boldsymbol{H}_{t-1}$ and the weights $\boldsymbol{W}_{RNN}$. Therefore, we use multiple-variable chain rule to expand the total derivative of f_{RNN} with respect to $\boldsymbol{W}_{RNN}$ as

$$\frac{df_{RNN}(\boldsymbol{H}_{t-1}, \boldsymbol{X}_t, \boldsymbol{W}_{RNN})}{d\boldsymbol{W}_{RNN}}$$

$$= \frac{\delta f_{RNN}(\boldsymbol{H}_{t-1}, \boldsymbol{X}_t, \boldsymbol{W}_{RNN})}{\delta \boldsymbol{H}_{t-1}} \frac{d\boldsymbol{H}_{t-1}}{d\boldsymbol{W}_{RNN}} + \frac{\delta f_{RNN}(\boldsymbol{H}_{t-1}, \boldsymbol{X}_t, \boldsymbol{W}_{RNN})}{\delta \boldsymbol{W}_{RNN}} \frac{d\boldsymbol{W}_{RNN}}{d\boldsymbol{W}_{RNN}}$$

$$= \frac{\delta f_{RNN}(\boldsymbol{H}_{t-1}, \boldsymbol{X}_t, \boldsymbol{W}_{RNN})}{\delta \boldsymbol{H}_{t-1}} \frac{d\boldsymbol{H}_{t-1}}{d\boldsymbol{W}_{RNN}} + \frac{\delta f_{RNN}(\boldsymbol{H}_{t-1}, \boldsymbol{X}_t, \boldsymbol{W}_{RNN})}{\delta \boldsymbol{W}_{RNN}} \tag{2.15}$$

Based on the above equations, we can find all the gradients. Then, we apply a specified optimization algorithm (i.e., an optimization algorithm for differentiable functions) to update the parameters to minimize the loss function. Recently, several optimization algorithms have been used in the DL training process that we will introduce later in this section.

2.2.4.2 Backpropagation through time for NODEs

Unlike the backpropagation algorithm of RNNs running across the hidden states with the discrete sampling time, NODEs require an algorithm for computing in the continuous domain. In particular, the gradients are computed using the implicit function theorem for continuous-time backpropagation.

First, we examine a simple ODE problem given an initial value as

$$\begin{cases} \frac{d\boldsymbol{H}(t)}{dt} = f(\boldsymbol{H}(t), t, \boldsymbol{W}) \\ \boldsymbol{H}(t_0) = \boldsymbol{H}_0 \end{cases}, \tag{2.16}$$

where f indicates the ODE network with the collection of learnable parameters $\boldsymbol{W}$. For the forward computation, the hidden states $\boldsymbol{H}(t)$ at arbitrary timestep can be solved by different integration methods (i.e., ODE methods) given the initial state $\boldsymbol{H}(t_0)$ and the ODE network f, as illustrated in (2.9) and (2.10). For the continuous-time modelling, we consider a scalar-valued loss function as the following

$$min_{\boldsymbol{W}}\, L\left(\boldsymbol{H}(t_1)\right) = min_{\boldsymbol{W}}\, L\left(\boldsymbol{H}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{H}(t), t, \boldsymbol{W})\, dt\right)$$

$$s.t.$$

$$\begin{cases} \frac{d\boldsymbol{H}(t)}{dt} = f(\boldsymbol{H}(t), t, \boldsymbol{W}) \\ \boldsymbol{H}(t_0) = \boldsymbol{H}_0 \end{cases} \tag{2.17}$$

The above problem formulation is referred to as the constrained optimization problem. To solve this problem, we need to compute the gradient of the loss function L with respect to the parameters $\boldsymbol{W}$ as

$$\frac{dL}{d\boldsymbol{W}} = \int_{t_0}^{t_1} \frac{df(\boldsymbol{H}(t),t,\boldsymbol{W})}{d\boldsymbol{W}} dt \tag{2.18}$$

To compute the loss gradient, we can apply two methods, the forward and adjoint methods. With the forward approach, we can directly use the total derivative to express the right-hand side of (2.18) as

$$\int_{t_0}^{t_1} \frac{df(\boldsymbol{H}(t),t,\boldsymbol{W})}{d\boldsymbol{W}} dt = \int_{t_0}^{t_1} \frac{\delta f(\boldsymbol{H}(t),t,\boldsymbol{W})}{\delta \boldsymbol{W}} + \frac{\delta f(\boldsymbol{H}(t),t,\boldsymbol{W})}{\delta \boldsymbol{H}(t)} \frac{d\boldsymbol{H}(t)}{d\boldsymbol{W}} dt \tag{2.19}$$

In this equation, we are concerned about the quantity $\frac{d\boldsymbol{H}(t)}{d\boldsymbol{W}}$, which is complex and challenging to compute as it depends on the size of the collection of parameters $\boldsymbol{W}$. Typically, we need to split this collection into small equal pieces and then implicitly derive the ODE problem with respect to each element of $\boldsymbol{W}$. As a result, the problem turns into solving a new system of various ODEs and takes considerable computation efforts.

In [19], Chen et al. proposed a reverse-mode differentiation method for training continuous-time models based on the Adjoint Sensitivity method that addresses the computational complexity issue [50]. By observing that problem (2.17) is an equality constraint optimization, we can solve it using the Lagrange method. With the adjoint method, an adjoint state playing the same role as the Lagrange multiplier is defined as

$$\boldsymbol{a}(t) = \frac{\delta L}{\delta \boldsymbol{H}(t)} \tag{2.20}$$

In other words, we can choose the Lagrange multiplier formula that helps solve the Lagrange problem effectively. According to this adjoint state (i.e., the Lagrange Multiplier of the Lagrange function), the loss derivative is computed as

$$\frac{dL}{d\boldsymbol{W}} = \int_{t_0}^{t_1} \boldsymbol{a}(t)^T \frac{\delta f(\boldsymbol{H}(t),t,\boldsymbol{W})}{\delta \boldsymbol{W}} dt \tag{2.21}$$

Based on the definition the chain rule, the total derivative of the adjoint state with respect to time is derived as

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)^T \frac{\delta f}{\delta \boldsymbol{H}(t)} \tag{2.22}$$

Consequently, the backward process of the adjoint state can be illustrated as

$$\boldsymbol{a}(t_0) = \boldsymbol{a}(t_1) + \int_{t_1}^{t_0} \boldsymbol{a}(t)^T \frac{\delta f(\boldsymbol{H}(t),t,\boldsymbol{W})}{\delta \boldsymbol{H}(t)} dt \tag{2.23}$$

To solve (2.21) and (2.23), we can use another numerical integrator (i.e., an ODE solver) to get the required gradients for the optimization task.

2.2.5 Attention mechanism

As mentioned earlier about the concept of the memory ability of sequence models, we will introduce the attention mechanism, another component boosting the memorization and recognition ability of the system, especially with long-term sequence prediction tasks.

Attention is a concept widely used in sequence models, especially sequence-to-sequence models. In particular, attention mechanisms have been researched and implemented in two famous sequence-to-sequence models, including RNNs [51, 52, 53] and transformers [54, 55, 56, 57]. In DL, the attention mechanism is typically implemented by a composition of artificial neural layers helping the system focus and pay greater attention to certain parts when processing the sequential data across time. Therefore, the system can acknowledge more effectively the heterogeneity and correlation in the long-range series. Consequently, it plays a vital role in supporting the memory mechanism of the model.

In [54, 58, 59], attention mechanisms have been proven effective in translation and generation tasks and used in state-of-the-art models, especially with Natural Language Processing (NLP) applications. Generally, there are two main attention types in sequence-to-sequence models: the general attention and self-attention modules. These two types are classified based on how they are integrated into the network's architecture and what information they will manage. For instance, the general attention component controls and quantifies the interdependence between the input and the target sequence (i.e., the goal is to create the connection between each output data point at the decoder and an observed series at the encoder). In contrast, the self-attention component helps each part of the system (i.e., the encoder and the decoder) learn the correlation and heterogeneity between each data point with the rest in the given sequence. In RNN-based models, these two attention mechanisms are illustrated in Figures 2.6 and 2.7.

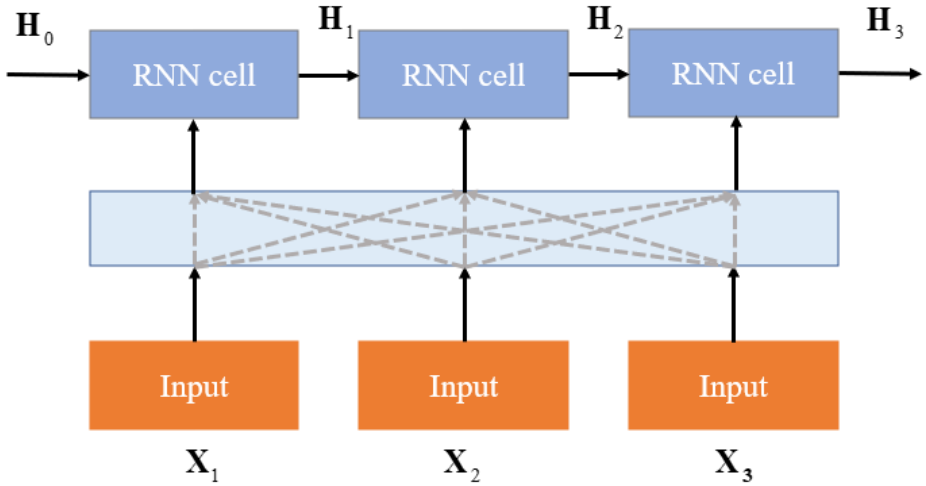


Figure 2.6: Self-attention mechanism used in RNN models.

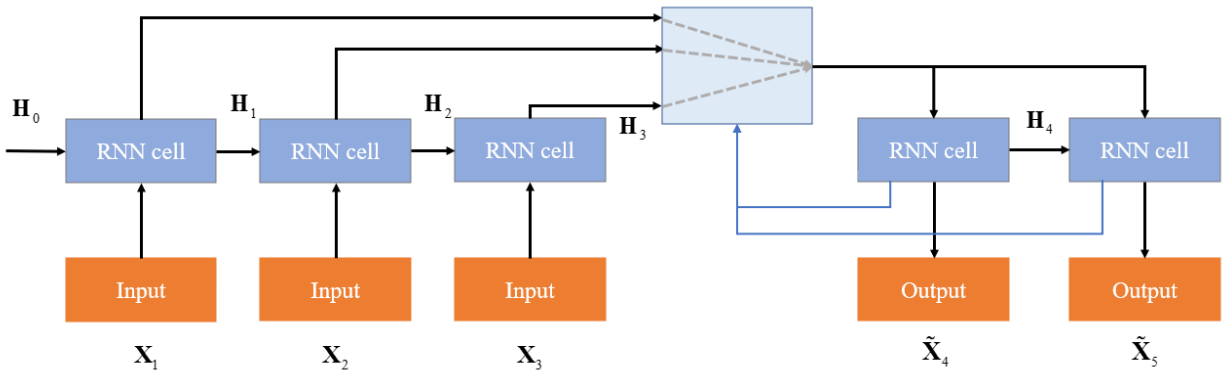


Figure 2.7: General attention used in RNN models.

In the internal operation, a standard attention layer usually employs three components: the queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$, and values $\boldsymbol{V}$. Essentially, the values $\boldsymbol{V}$ represent the hidden states generated by the encoder, the keys $\boldsymbol{K}$ indicate some significant characteristics of these states, and the queries $\boldsymbol{Q}$ relate closely to the concerned knowledge (i.e., they can be the output hidden states generated by the decoder). For example, we can assume this process as reading a multi-volume story. Until now, we have read the first four chapters and attempted to predict the content of the upcoming chapter while waiting for the next chapter. In this scenario, the contents of the first four chapters indicate the values $\boldsymbol{V}$. Then, we can analyze those contents and acknowledge them by different perspectives. For example, four popular critical lenses are used in literary criticism: a Marxist lens, a Deconstructionist lens, a Feminist lens, and a Psychoanalytical lens [60]. Those four lenses represent the keys $\boldsymbol{K}$ allow us to obtain the contents with different meanings. Ultimately, the queries $\boldsymbol{Q}$ illustrate the expected content of the next chapter according to a pair of $\boldsymbol{V}$ and $\boldsymbol{K}$.

Mathematically, the common attention mechanism uses three main steps. The first step is to analyze the relationship $\boldsymbol{S}$ between the queries and the keys by using similarity calculation methods or neural network layers with learnable weights $f_{\boldsymbol{W}}$, as shown below

$$\boldsymbol{S} = f_{\boldsymbol{W}}(\boldsymbol{Q}, \boldsymbol{K}) \tag{2.24}$$

Hence, we create a bond between the expected information and the key characteristics of observed information through a set of learnable parameters. For the normalization purpose, the relationship value $\boldsymbol{S}$ then goes through a specified activation function, usually the SoftMax function.

$$\boldsymbol{S}_n = SoftMax(\boldsymbol{S}) \tag{2.25}$$

The value $\boldsymbol{S}_n$ is also called as the alignment score indicating the attention weights of each observed states regarding to the expected states $\boldsymbol{Q}$. Finally, we assign these weights according to the values $\boldsymbol{V}$ to compute the context vector representing the attention information

$$\boldsymbol{C} = f(\boldsymbol{S}_n, \boldsymbol{V}) \tag{2.26}$$

where f indicates the product between $\boldsymbol{S}_n$ and $\boldsymbol{V}$, and can be represented by a neural network layer with learnable parameters. Consequently, this common attention mechanism can be integrated into any sequence-to-sequence model to improve the overall performance.

2.2.6 Activation layer

In ANNs, an activation function is added to the end of nodes and layers to help the network learn nonlinear information by mapping the linear transformation of previous computations to a high-dimensional nonlinear space [61]. Therefore, the models can extract more complicated features. Furthermore, most activation functions are derivative, so they can be easily dropped in the DL models. As a side effect, it could alleviate the gradient vanishing problem in the training process when it involves to many deep ANNs.

Until now, there have been a variety of activation functions with different mathematical equations, and they are employed, depending on the specific purposes of the model. In this work, we applied five activation functions, including the Sigmoid function, Tanh function, Softmax function, RELU function, and Leaky RELU function. A detailed comparison of these activation functions can be found in [61].

2.2.7 Image quality assessment

Image quality assessment is one of the essential fields of digital image processing [62, 63]. The main objective is to design and propose mathematical formulas for image quality metrics. For instance, the peak-signal-to-noise ratio (PSNR) and the structural similarity index measure (SSIM) are two famous image metrics illustrating the visual quality of images. In the DL field, those metrics are considered as the loss functions guiding the training process as well as the evaluation metrics to qualify the model's performance. In our work, we will apply SSIM as a component of our loss function. The SSIM metric can be expressed as

$$\begin{aligned} SSIM(\boldsymbol{X},\boldsymbol{Y}) &= l(\boldsymbol{X},\boldsymbol{Y}) \cdot c(\boldsymbol{X},\boldsymbol{Y}) \cdot s(\boldsymbol{X},\boldsymbol{Y}) \\ &= \left(\frac{2\mu_{\boldsymbol{X}}\mu_{\boldsymbol{Y}}+C_1}{\mu_{\boldsymbol{X}}^2+\mu_{\boldsymbol{Y}}^2+C_1}\right) \cdot \left(\frac{2\sigma_{\boldsymbol{X}}\sigma_{\boldsymbol{Y}}+C_2}{\sigma_{\boldsymbol{X}}^2+\sigma_{\boldsymbol{Y}}^2+C_2}\right) \cdot \left(\frac{\sigma_{\boldsymbol{XY}}+C_3}{\sigma_{\boldsymbol{X}}\sigma_{\boldsymbol{Y}}+C_3}\right), \end{aligned} \tag{2.27}$$

where $l(\boldsymbol{X},\boldsymbol{Y})$ denotes the luminance comparison function which measures the closeness of the two images luminance; $\mu_{\boldsymbol{X}},\mu_{\boldsymbol{Y}}$ present the mean values of two analyzed images, respectively; $c(\boldsymbol{X},\boldsymbol{Y})$ is the contrast comparison function which measures the closeness of the contrast of the two images based on their standard deviation $(\sigma_{\boldsymbol{X}},\sigma_{\boldsymbol{Y}})$; $s(\boldsymbol{X},\boldsymbol{Y})$ is the structure comparison function which measures the correlation coefficient between the two images; (C_1, C_2, C_3) are the positive constant values used to avoid a null denominator. The values of the SSIM are in $[0,1]$ and the higher the SSIM value is, the higher the image quality of the results is.

The formulation of the PSNR metric can be expressed as

$$PSNR(\boldsymbol{X},\boldsymbol{Y}) = 10\, log_{10}\left(\frac{{MAX_I}^2}{MSE(\boldsymbol{X},\boldsymbol{Y})}\right), \tag{2.28}$$

where MAX_I presents the pixel intensity's maximum value, $MSE(\boldsymbol{X}, \boldsymbol{Y})$ refers to the Mean Square Error between $\boldsymbol{X}$ and $\boldsymbol{Y}$. A higher PSNR value provides a higher image quality. A detailed comparison of these metrics can be found in Reference [64].

2.2.8 Loss function

To guide the training process and evaluate the model's performance, we need to initially define a loss function, also known as a cost function. Typically, the loss function refers to a differentiable function whose gradient is computed to investigate the rate of change at a specified point. Depending on the DL task, a loss function is proposed to guide the whole training process to find the optimal collection of model parameters. Overall, we need the ground truth label to compare with the resulting output of the model, especially with a supervised learning problem. For the classification task, the Cross-entropy loss function is commonly used [65]. For prediction and regression tasks, the two popular cost functions are Mean Square Error (MSE) and Mean Absolute Error (MAE). They are also called L2 loss and L1 loss. For many image processing tasks, image quality assessment is also considered to be a potential loss function.

During the training process, the loss function of DL models usually has various local optima; therefore, minimizing the training loss function does not guarantee that we can find the globally optimal collection of parameters. Moreover, we must also choose a suitable optimization algorithm for the defined loss function to update the parameters appropriately.

2.2.9 Optimization

Generally, we train the DL model to find the collection of parameters which show the optimal results in the defined loss function. Therefore, the model's parameters represent the entire system. During the training process, the model will apply a specific optimization algorithm to update its learnable parameters in order to optimize the loss function.

Recently, there have been many different types of optimization algorithms [66]. Based on the property of the loss function, we can classify the optimization algorithm into two groups: algorithms that use derivative information (e.g., bracketing, local descent, first-order, and second-order algorithms) and algorithms that do not use derivative information (e.g., direct algorithm, stochastic algorithms, population algorithms). In our thesis, we focus on the group using derivative information as we apply the backpropagation algorithms in our DL models. For training DL models, First-order algorithms are the most commonly used, including Gradient Descent [67], Momentum [68], Adagrad [69], RMSProp [70], and Adam algorithm [71]. A detailed comparison of these optimization algorithms can be found in [67]. The Adam optimizer is the most popular algorithm because of its stability and adaptive features. Thus, we will apply the Adam optimizer to train all the models.

Chapter 3: Related works

Recently, there have been several DL-based models showing promising performances in SSP tasks. In this thesis, sequence-to-sequence models, which input and output sequential data, are investigated. To the best of our knowledge, they can be divided into main groups for the SSP task: Convolutional Recurrent Networks (ConvRNNs), Ordinary Differential Equation-based (ODE-based) models or Neural Ordinary Differential Equations (NODEs), Generative Adversarial Networks (GANs), and Transformers.

3.1 Convolutional recurrent networks (ConvRNNs)

Essentially, ConvRNNs are proposed and designed based on two popular types of DL methods: RNNs and CNNs. The main difference between these two types is the ability to process temporal and spatial features of data. For instance, CNNs can effectively handle spatial data, such as images, using the convolution operation of their filters [72]. However, when the data comes in sequence like time series, CNNs cannot learn the dependencies between sampled points because the features of each data frame are extracted separately by the same collection of filters. In that scenario, RNNs turn out to be a powerful solution because they can learn the dependencies and correlation within the given series as the standard RNN mechanisms work as a loop in which an output hidden state at a particular time step will be an input of the system at the next time step. Generally, traditional RNNs employ fully connected layers for the feature extraction task; this leads to an explosive increase in the number of trainable weights, especially when it comes to a sequence of complex spatial data (e.g., high-resolution videos). Furthermore, the fully connected layers (i.e., linear layers) apply the matrix multiplication in the transitions between states and inputs so that spatial information is not encoded effectively [1, 2, 10]. For the SSP tasks, the spatiotemporal phenomena will be sampled as sequences of multiple spatial frames at specific time steps. Therefore, an efficient spatiotemporal predictive model must adequately deal with spatial and temporal features. Based on the observations of the distinct abilities of RNNs and CNNs, Shi et al. proposed the Convolutional Long-Short Term Memory (ConvLSTM) [1]. They used this technique to build a sequence-to-sequence model for the precipitation nowcasting problem, one of the fundamental applications in the SPL field.

3.1.1 Convolutional Long-short Term Memory (ConvLSTM) and Convolutional Gated Recurrent Unit (ConvGRU) networks

In the ConvRNNs group, Convolutional Long-short Term Memory (ConvLSTM) and Convolutional Gated Recurrent Unit (ConvGRU) are two prior variants. Essentially, ConvLSTM [1] and ConvGRU [73, 74] networks are extensions of LSTM and GRU, as illustrated below. Besides the recurrent operation, they are equipped with convolutional structures by integrating the convolutional neural layers in the input-to-state and state-to-state transitions. Therefore, they can

handle spatiotemporal data based on the built-in spatiotemporal memory mechanism. Furthermore, instead of using matrix multiplications, the convolution operator helps the network work with high-dimension information (e.g., 3-Dimension, 4-Dimension tensors) without fattening it and deals with the redundancy problem of the fully connected layers. The principal formulas of a ConvLSTM cell can be expressed as [1]

$$
\begin{aligned}
\boldsymbol{I}_t &= \sigma(\boldsymbol{W}_{xi} * \boldsymbol{X}_t + \boldsymbol{W}_{hi} * \boldsymbol{H}_{t-1} + \boldsymbol{W}_{ci} \circ \boldsymbol{C}_{t-1} + \boldsymbol{B}_i) \\
\boldsymbol{F}_t &= \sigma(\boldsymbol{W}_{xf} * \boldsymbol{X}_t + \boldsymbol{W}_{hf} * \boldsymbol{H}_{t-1} + \boldsymbol{W}_{cf} \circ \boldsymbol{C}_{t-1} + \boldsymbol{B}_f) \\
\boldsymbol{C}_t &= \boldsymbol{F}_t \circ \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \circ tanh(\boldsymbol{W}_{xc} * \boldsymbol{X}_t + \boldsymbol{W}_{hc} * \boldsymbol{H}_{t-1} + \boldsymbol{B}_c) \\
\boldsymbol{O}_t &= \sigma(\boldsymbol{W}_{xo} * \boldsymbol{X}_t + \boldsymbol{W}_{ho} * \boldsymbol{H}_{t-1} + \boldsymbol{W}_{co} \circ \boldsymbol{C}_{t-1} + \boldsymbol{B}_o) \\
\boldsymbol{H}_t &= \boldsymbol{O}_t \circ tanh(\boldsymbol{C}_t)
\end{aligned}
\tag{3.1}
$$

Basically, similar to an LSTM cell, a ConvLSTM cell uses four components to regulate the information flows: the input gate $\boldsymbol{I}_t$ encodes the input data $\boldsymbol{X}_t$ at the current time step, and the hidden state $\boldsymbol{H}_{t-1}$, cell state $\boldsymbol{C}_{t-1}$ at the previous time step, to generate the representation form; the forget gate $\boldsymbol{F}_t$ helps the network remove unnecessary information; the cell state $\boldsymbol{C}_t$ containing information of both previous and current decoded features is calculated based on the outputs of the gates; the new state $\boldsymbol{H}_t$ is updated by letting the cell state go through the output gate $\boldsymbol{O}_t$. In (3.1), "$*$" denotes the convolution operator; "$\circ$" denotes the Hadamard product; "σ" refers to the Sigmoid function; $(\boldsymbol{W}_{xi}, \boldsymbol{W}_{xf}, \boldsymbol{W}_{xc}, \boldsymbol{W}_{xo})$, $(\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{hc}, \boldsymbol{W}_{ho})$, $\boldsymbol{W}_{co}$, and $(\boldsymbol{B}_i, \boldsymbol{B}_f, \boldsymbol{B}_c, \boldsymbol{B}_o)$ are the trainable weights and biases of convolutional layers. Figure 3.1 shows the operation of a ConvLSTM cell

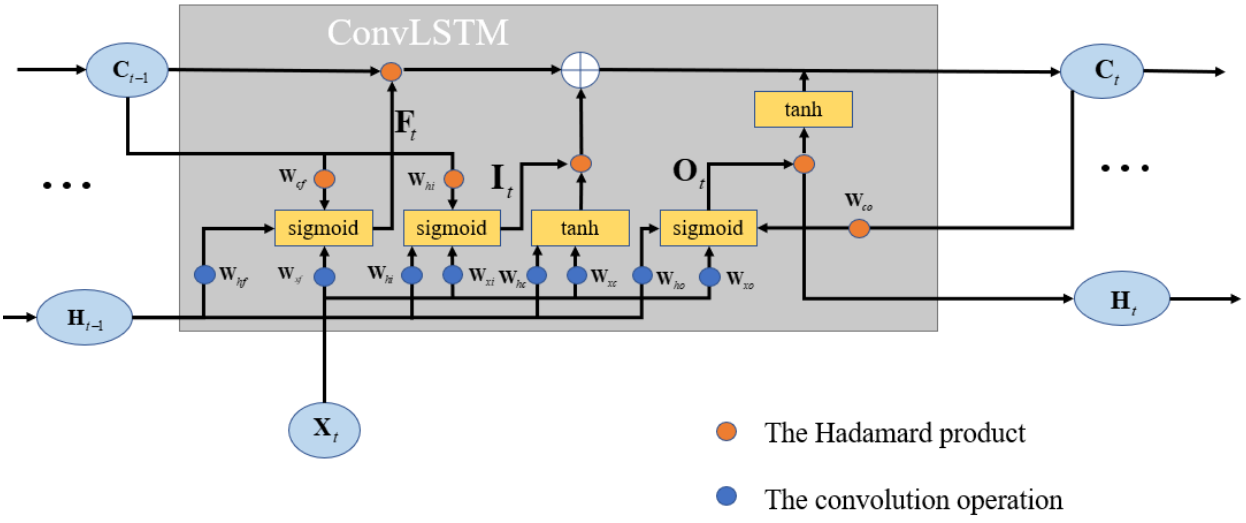


Figure 3.1: ConvLSTM cell.

The main formulas of the ConvGRU cell can be expressed as

$$\begin{aligned}
\boldsymbol{Z}_t &= \sigma(\boldsymbol{W}_{xz} * \boldsymbol{X}_t + \boldsymbol{W}_{hz} * \boldsymbol{H}_{t-1} + \boldsymbol{B}_z) \\
\boldsymbol{R}_t &= \sigma(\boldsymbol{W}_{xr} * \boldsymbol{X}_t + \boldsymbol{W}_{hr} * \boldsymbol{H}_{t-1} + \boldsymbol{B}_r) \\
\widetilde{\boldsymbol{H}}_t &= f_{activation}(\boldsymbol{W}_{xh} * \boldsymbol{X}_t + \boldsymbol{R}_t \circ (\boldsymbol{W}_{hh} * \boldsymbol{H}_{t-1}) + \boldsymbol{B}_h) \\
\boldsymbol{H}_t &= (1 - \boldsymbol{Z}_t) \circ \widetilde{\boldsymbol{H}}_t + \boldsymbol{Z}_t \circ \boldsymbol{H}_{t-1}
\end{aligned} \tag{3.2}$$

In (3.2), "$*$" indicates convolution operator and "$\circ$" denotes Hadamard product; "σ" refers to the Sigmoid activation function; $(\boldsymbol{W}_{xz}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh})$, $(\boldsymbol{W}_{hz}, \boldsymbol{W}_{hr}, \boldsymbol{W}_{hh})$, and $(\boldsymbol{B}_i, \boldsymbol{B}_f, \boldsymbol{B}_c, \boldsymbol{B}_o)$ present the learnable weights and biases, respectively of convolutional layers; $\boldsymbol{Z}_t$, $\boldsymbol{R}_t$ represent the update gate and reset gate, respectively, regulating the information flows; $\widetilde{\boldsymbol{H}}_t$ presents the memory state storing new information of the recent input data. Figure 3.2 shows the operation of a ConvGRU cell

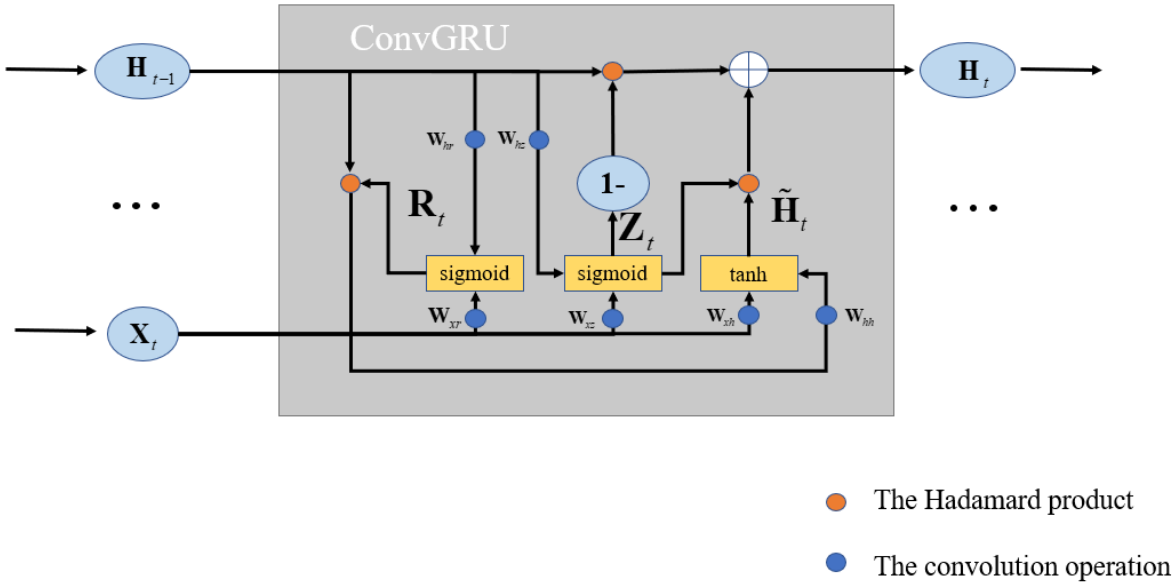


Figure 3.2: ConvGRU cell.

Generally, a DL model needs the main structure and architecture so that some small modules, such as neural layers (e.g., convolutional, recurrent, and activation layers), can be built up accordingly. For example, sequence-to-sequence systems usually use the encoding-decoding structure, including two main parts: the encoder and decoder [1, 2, 3, 13]. Recently, various types of model architecture have been analyzed, illustrating how neural modules communicate with each other [75, 76]. From the architectural perspective, we can investigate the depth of a system based on the number of neural layers and their connection styles. For example, the ConvLSTM model in [1] applied the encoding-decoding structure and stacked architecture with multiple neural layers, as illustrated in Figure 3.3.

Although the ConvLSTM and the ConvGRU models can learn spatiotemporal information and apply them in general-purpose sequence modelling, their performance is relatively limited due to their simple structure. From the perspective of the time domain, these two models follow the regular time interval sampling assumption because the transition learning only focuses on the dependencies between every two adjacent frames and ignores the time variable of each frame. Furthermore, at each iteration of the recurrent computation process, their memory mechanism uses only one previous hidden state. As a result, the systems might forget the representation states seen in many time stamps before, especially with long-term sequences. Therefore, many researchers consider ConvLSTM and ConvGRU as the baseline ConvRNNs to modify and design new approaches.

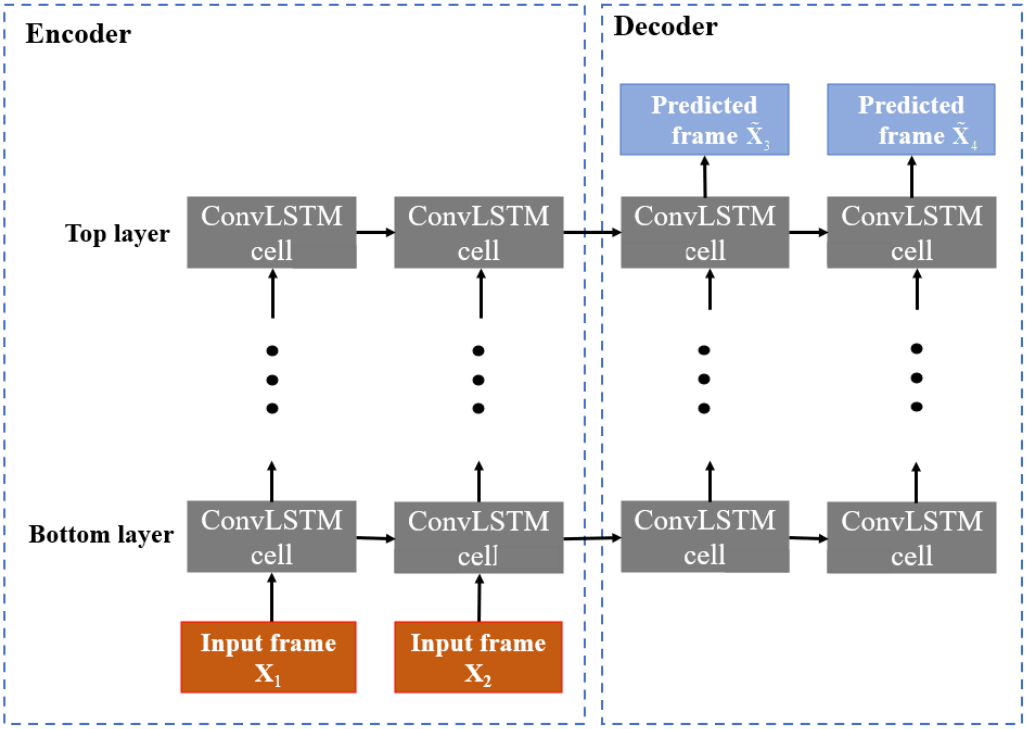


Figure 3.3: Encoding-decoding model structure with the stacked architecture.

3.1.2 Trajectory Gated Recurrent Unit (TrajGRU) networks

In terms of structured recurrent connection for spatiotemporal modelling, most motion patterns' spatial locations are changeable over time. In particular, some complicated motions present the data's heterogeneity property. For example, we can assume that a small motion pattern is represented by a fixed collection of image pixels at specific location points in the grid pattern. When this pattern rotates, it changes the location points and neighbourhood set, referring to the nearby pixels. If we apply a collection of vectors to illustrate the movement in the object's pixels, the rotation will change the direction of each vector over time. This observation demonstrates the deficiency of conventional ConvRNNs like ConvLSTM and ConvGRU since the convolutional

layers apply location-invariant filters to the analyzed sequence [2, 77]. When defining the convolutional layers with fixed parameters like kernel size, dilation, and padding, these layers use their filters, slicing horizontally and vertically each frame of the given sequence so that the connection structure and weights are unchangeable for all the locations. As a result, the standard ConvLSTM and ConvGRU cannot effectively handle sequences with chaotic motions and many transformations like rotation, scaling, or more generic warping. Thus, a possibly temporal solution to address this issue is data manipulation to generate new data with those transformation effects. However, we have to consider the data size because massive training sets will require many resources.

In [78], Max Jaderberg et al. demonstrated that we could do better with the spatial transformer, which explicitly allows the spatial manipulation of data within the network by inserting this learnable module into the model. Based on the fundamental idea of the learnable spatial transformer module, Shi et al. proposed the Trajectory Gated Recurrent Unit (TrajGRU) model that can learn the location-variant structure for the recurrent connection and capture the spatiotemporal corrections effectively [2]. To visualize the connection structure of the rotation or scaling, we can use ML optical flow algorithms to analyze the flow fields illustrated by a set of vectors with specified magnitudes and directions. Moreover, we can also imagine and predict the motions based on changing the flow fields over time [79, 80]. Therefore, the TrajGRU employed a spatial transformer network to produce the continuous optical flows representing the location indices at each time step. Specifically, this network is constructed by combining a set of convolutional and activation layers. Then, it combines the data frame at a specified time step and the previous hidden state as the input data. Hence, the main formulations of a TrajGRU cell can be expressed as

$$
\begin{aligned}
&\boldsymbol{U}_t, \boldsymbol{V}_t = \gamma(\boldsymbol{X}_t, \boldsymbol{H}_{t-1}) \\
&\boldsymbol{Z}_t = \sigma\left(\boldsymbol{W}_{xz} * \boldsymbol{X}_t + \sum_{l=1}^{L} \boldsymbol{W}_{hz,l} * warp(\boldsymbol{H}_{t-1}, \boldsymbol{U}_{t,l}, \boldsymbol{V}_{t,l}) + \boldsymbol{B}_z\right) \\
&\boldsymbol{R}_t = \sigma\left(\boldsymbol{W}_{xr} * \boldsymbol{X}_t + \sum_{l=1}^{L} \boldsymbol{W}_{hr,l} * warp(\boldsymbol{H}_{t-1}, \boldsymbol{U}_{t,l}, \boldsymbol{V}_{t,l}) + \boldsymbol{B}_r\right) \\
&\widetilde{\boldsymbol{H}}_t = f_{activation}\left(\boldsymbol{W}_{xh} * \boldsymbol{X}_t + \boldsymbol{R}_t \circ \left(\sum_{l=1}^{L} \boldsymbol{W}_{hh,l} * warp\left(\boldsymbol{H}_{t-1}, \boldsymbol{U}_{t,l}, \boldsymbol{V}_{t,l}\right)\right) + \boldsymbol{B}_h\right) \\
&\boldsymbol{H}_t = (1 - \boldsymbol{Z}_t) \circ \widetilde{\boldsymbol{H}}_t + \boldsymbol{Z}_t \circ \boldsymbol{H}_{t-1} \ , \qquad (3.3)
\end{aligned}
$$

where $\boldsymbol{X}_t$, $\boldsymbol{H}_{t-1}$ indicate the input data at time step t and the previous hidden state at time step $t-1$, respectively; γ presents the spatial transformer network; $\boldsymbol{U}_t, \boldsymbol{V}_t$ denote the continuous flow fields and each of them is divided into L small sets presenting L different connection links in the transition; $warp$ is a function used to combine the sets of flows and the hidden state to generate an intermediate representation with approximated location information. From the perspective of the model structure, TrajGRU is relatively similar to ConvGRU: the information flows at each

time step are controlled by the update gate $\boldsymbol{Z}_t$ and reset gate $\boldsymbol{R}_t$; $\tilde{\boldsymbol{H}}_t$ represents the memory state, which is modified according to the reset gate and information flows at each time step; eventually, the new state $\boldsymbol{H}_t$ is generated based on the memory state, the update gate output and the previous hidden state. In terms of the learnable parameters, $(\boldsymbol{W}_{xz}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh})$ denote the input-to-state transition weights; $(\boldsymbol{W}_{hz,l}, \boldsymbol{W}_{hr,l}, \boldsymbol{W}_{hh,l})$ denote the state-to-state transition weights at each local link; $(\boldsymbol{B}_z, \boldsymbol{B}_r, \boldsymbol{B}_h)$ denote the biases; the structure generating network γ also have a collection of learnable parameters which help the model learn the connection topology. Finally, "$*$" denotes the convolution operator; "$\circ$" denotes the Hadamard product; σ refers to the Sigmoid function; $f_{activation}(\ldots)$ refers to the activation to get the memory state, which can be the Tanh function in most of the cases. Figure 3.4 and 3.5 show the spatial transformer module (the structure-generating network) and the operation of a TrajGRU cell, respectively.

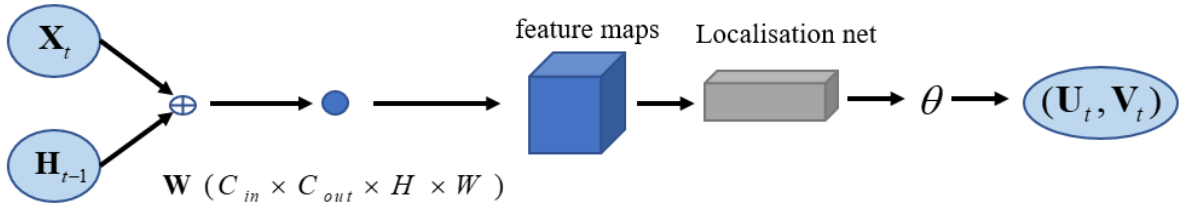


Figure 3.4: Spatial transformer module.

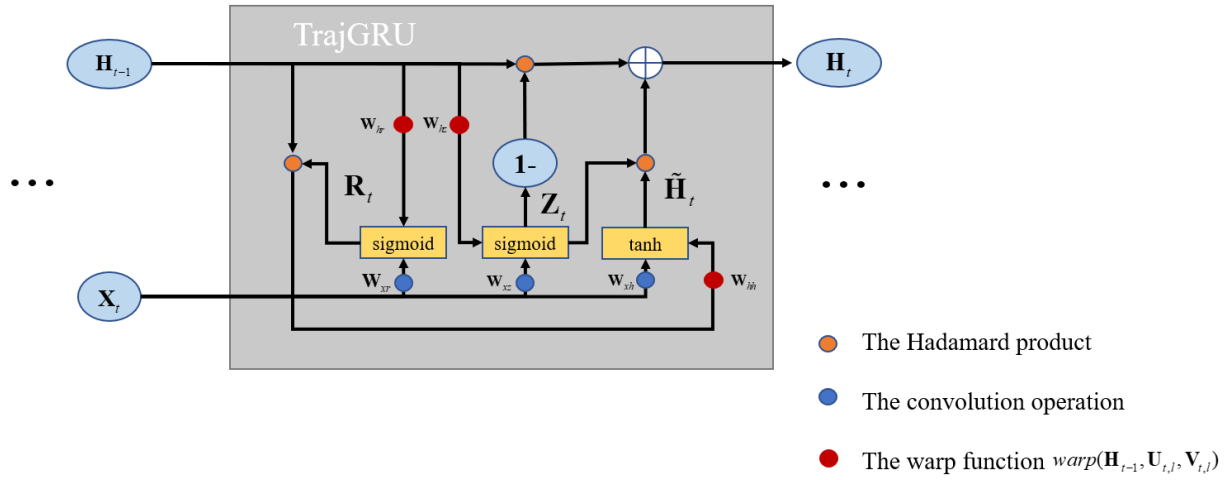


Figure 3.5: TrajGRU cell.

In [2], the TrajGRU model used the encoding-decoding structure and stacked style. For learning the spatiotemporal references at different levels (i.e., the high-level states illustrate the local information, the low-level states present the global representation), this model inserts resampling blocks (e.g., down-sampling and up-sampling layers) between adjacent TrajGRU

layers. In particular, these resampling modules are implemented by the convolutional and deconvolutional layers. Furthermore, the decoder allows reversed layers compared to the encoder. As such, the information flows are processed from the global spatiotemporal representations to the local features at the encoder and vice versa at the decoder. Eventually, the lowest level neural layer at the decoder will generate the final predictions. The representation states at this layer are constructed by their higher-level features so that they can contain both local and global information about the target sequence. As a result, the output predictions can maintain high sharpness compared to the target. Therefore, this model architecture can make more sense than the conventional stacked architecture of the ConvLSTM illustrated in Figure 3.3. Figure 3.6 shows the detailed information of the TrajGRU model, including the overall structure, architecture, and building blocks.

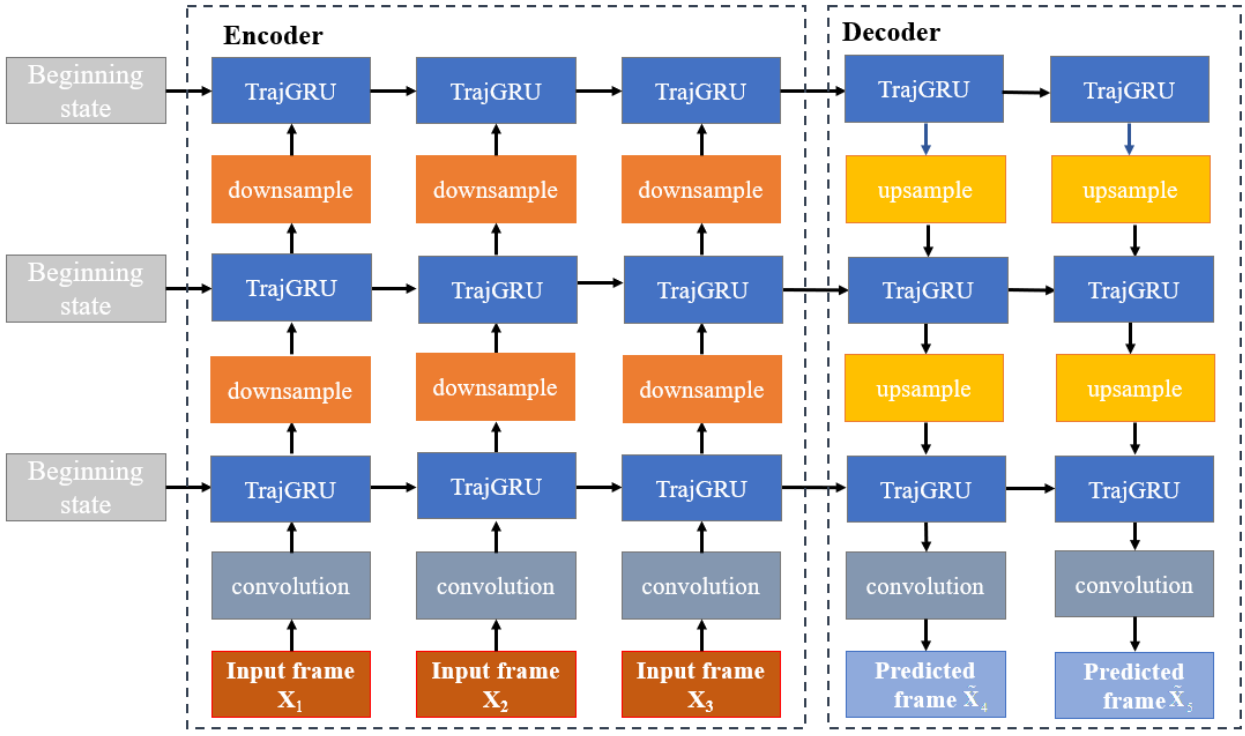


Figure 3.6: TrajGRU model: the encoding-decoding structure, the stacked architecture with reversed decoding layers.

3.1.3 Convolutional Recurrent Networks with advanced memory mechanisms

The spatiotemporal memory mechanism is the fundamental component of a specified ConvRNN, illustrating the ability to process data as time passes. Based on the reported observations, the internal mechanism of a neural cell and the model architecture significantly

impact the memory ability of the model. With conventional ConvRNNs, their internal operations usually use only one cell state to store the knowledge of previous input information (e.g., the cell state $\boldsymbol{C}_t$ in the ConvLSTM layer, $\widetilde{\boldsymbol{H}}_t$ in the ConvGRU layer). In the case of long sequences, this memory cell state needs to handle both short-term dependencies (i.e., the correlation between two adjacent frames) and long-term dependencies (i.e., the correlation between the current frame and frames at many timesteps before). This leads to the information overload and vanishing gradient issues. Therefore, a potential solution to improve the performance of the standard ConvRNN models is adjusting the memory mechanisms according to different SSP tasks. From the perspective of the model architecture, many prior studies proved that increasing the depth of the neural networks can exponentially benefit feature extraction and sequence modelling [3, 81, 82, 83]. For the sequence models, the concept of neural network depth is analyzed in the time (i.e., the number of representations between two given timesteps are captured) and space domains (i.e., the number of extracted spatial hidden states at a specified timestamp). Therefore, in the case of the stacked architecture shown in Figure 3.6, we can increase the number of neural layers to generate more states at each time step. However, this will cause a tremendous increase in the number of trainable parameters.

In [3, 13, 81], Wang and al. proposed PredRNN, PredRNN++, and PredRNN-v2 models with novel ConvRNN layers for the SSP task. Essentially, the PredRNN cell is the extended version of the ConvLSTM unit, and its operation also replaces the matrix multiplication with the convolutional operators. In particular, its internal mechanism contains dual memories: the temporal and spatial memory states, instead of a single cell state in the ConvLSTM structure shown in Figure 3.1. As a result, the recurrent layer can store more knowledge of the observed series, and the system can alleviate the issue when the gradients quickly vanish in a single memory cell. Moreover, they designed a spatiotemporal memory flow architecture allowing the spatial memory cell state to traverse all model cells in a zigzag path from the top layer to the bottom layer repeatedly across time, as shown in Figure 3.7. Therefore, this new architecture style lengthens the networking depth and creates an adaptive connection between neural cells and layers. Eventually, the group of PredRNN models can perform effectively with long-term SSP tasks based on the concept of deeper-in-time [3, 83]. However, the dual memory mechanism's usage can complicate neural cell structure as these two memory states are updated separately. Thus, this might lead to considerable computational complexity with a massive number of trainable parameters, especially when we want to increase the number of layers.

In addition, the standard ConvLSTM and ConvGRU models are considered first-order Markovian models because their memory cell takes only one previous hidden state at a time to generate the spatiotemporal representation state [84]. However, this causes an intrinsic difficulty in learning spatiotemporal correlations in the long-term forecasting task. Therefore, Su et al. extended the standard ConvLSTM to a higher-orders network that can combine several previous states to feed into the memory mechanism at a time step, namely Convolutional Tensor-Train Long Short-Term Memory (Conv-TT-LSTM) [18]. Based on the idea of the tensor train decomposition

[85], this model can not only reduce the blurry effect in the long-term prediction task but also keep the computational complexity at an acceptable level. However, the tensor decomposition algorithms might be hard and unstable to implement into the model as they are basically approximation methods.

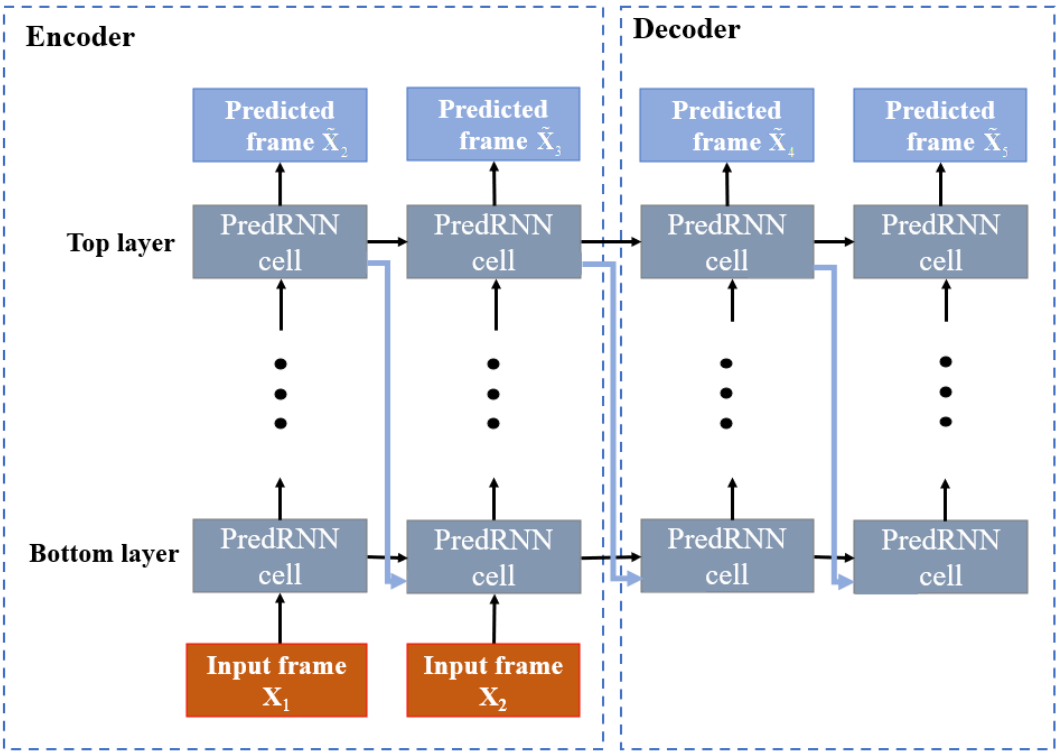


Figure 3.7: Spatiotemporal memory flow architecture of the PredRNN models.

3.2 ODE-based models

For video prediction task, most of the recent ConvRNN models, such as the previous models, follow the regular time sampling assumption (i.e., the time interval or time recording between two adjacent frames is constant). Therefore, the learning ability of these models is limited to specific datasets with a fixed time gap between adjacent data frames. As a result, these models cannot perform irregularly sampled or continuous-time prediction tasks. A temporary solution to this problem is to preprocess the specified time-series datasets. In other words, we remove some intermediate data points from the series with constant time intervals. However, ignoring frames means decreasing the number of data and removing some helpful knowledge, especially with sparse datasets. Moreover, this will significantly impact the training and testing processes of the whole system. Consequently, the preprocessing step is not optimal for the given task. In that

situation, NODEs or ODE-based models are proposed as promising methods for irregularly sampled and continuous-time prediction problems [19, 46].

The group of ODE-based models is a new family of deep neural network models [19]. In the SPL field, we can present the evolution of a phenomenon by differential equations and solve these equations by mathematical methods (i.e., analytic, and numerical methods). Based on this observation, ODE-based models have been designed and ANNs have been proposed as the differential equations describing the changing processes of representation states in the latent field. Moreover, the numerical methods for solving ODE (i.e., the ODE solvers) have been implemented in various commonly used frameworks, so we can straightforwardly insert them into the DL models. For instance, Chen et al. proposed a generative latent function continuous-time model (Latent-ODE), considered as one of the first NODEs implemented for time series prediction tasks [19, 20]. This ODE-based model also uses the encoding-decoding structure: the encoder is implemented by the RNNs technique, and the decoder is implemented by an ODE solver, as shown in Figure 3.8. First, the encoder processes the data series at the given time step to generate the statistical characteristics (e.g., the mean, variance, and other moments). After that, the decoder takes these statistical characteristics and makes predictions at the arbitrary time steps for the extrapolation (i.e., making predictions arbitrarily forwards in time) and interpolation (i.e., making predictions arbitrarily backwards in time) tasks. However, the input series at the encoder needs to be sampled at constant time intervals when this model uses conventional RNNs.

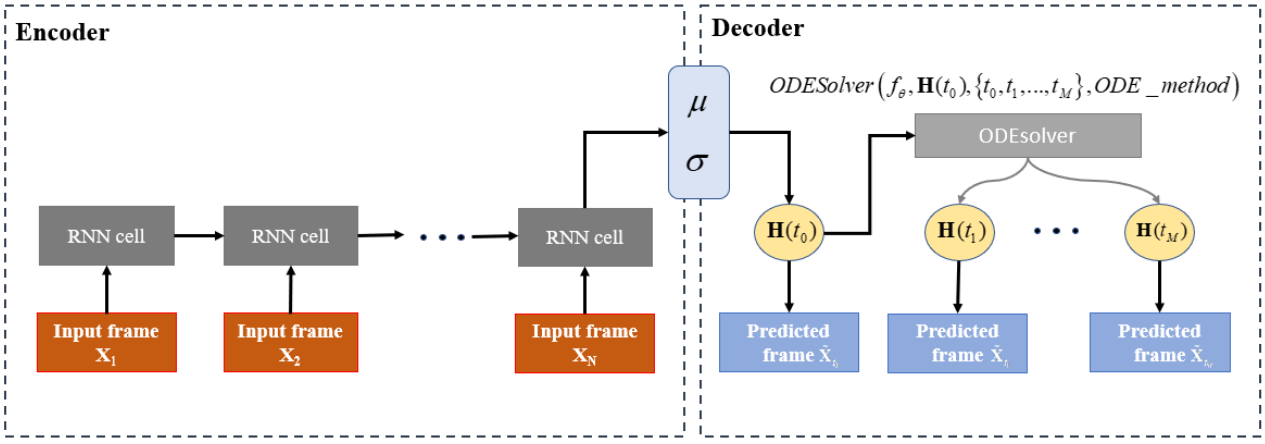


Figure 3.8: Latent-ODE model.

Based on the previous observation, Tidiz et al. proposed Deep generative second-order ODEs with Bayesian neural networks (ODE2VAE) for continuous-time modelling [21]. In particular, this model used the auto-encoder structure for the representation learning [86], so the RNN-based time series modelling at the encoder of the Lalent-ODE illustrated below is unnecessary. For the

feature extraction, the dual encoder consists of the position and velocity encoder modules, representing the set of ODE networks. Furthermore, these second-order ODE modules allow the system to generate more high-order continuous dynamics that benefit the model's feature learning. Therefore, the ODE2VAE can handle input time series thoroughly at arbitrary timestamps. In addition, with the auto-encoder structure, we do not need to split a sequence into two subsequences (i.e., the input and the target sequences); the model will take the examined series and reconstruct it. For the extrapolation task, ODE2VAE can continue to generate future unseen frames based on the reconstructed output. However, the resulting extrapolated results might not be generated effectively since no explicit memory mechanism is defined to help the system look back at previous information, especially when we want to predict a long-range sequence. In addition, the lack of spatiotemporal dependency investigation leads to a critical degradation in the performances of sequence models for the SSP task. Figure 3.9 shows the schematic illustration of the ODE2VAE model.

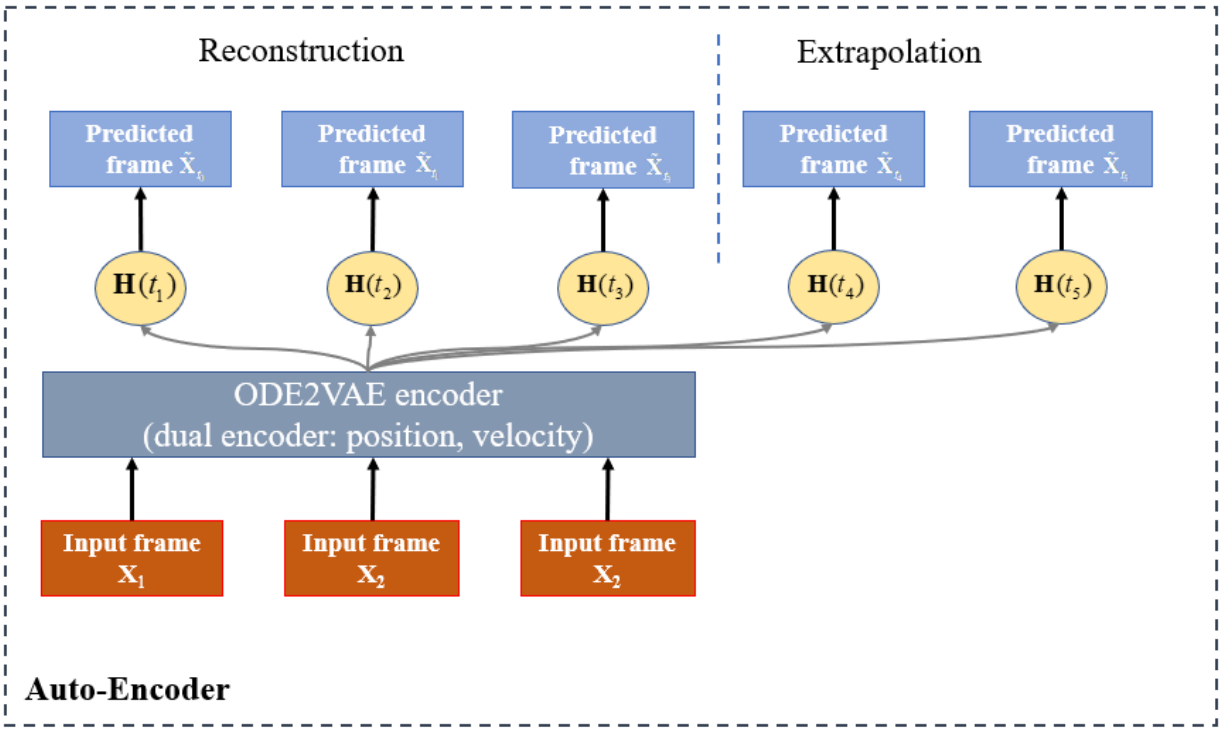


Figure 3.9: ODE2VAE model.

Although the ODE-based models above can handle irregularly sampled data and learn the continuous-time state dynamics, their performances were investigated with relatively simple datasets, such as image rotation and bouncing ball [20, 21, 87]. Thus, they are not ready for many practical applications with detailed spatiotemporal datasets. In other words, a single ODE neural network is not sufficiently powerful to extract complex spatial and temporal features

simultaneously. In [14], Park et al. proposed a continuous-time video generation model using the ODE technique, namely Vid-ODE, which shows some promising performances with real-world videos. Vid-ODE essentially combined the standard ConvGRU network, NODE techniques, and the pixel-level composition technique. From the structure perspective, this model used an encoding-decoding structure for the video generation task. The encoder is constructed by the ConvGRU layer and a specified ODE neural network which helps capture the irregular sampling intervals; the decoder is constructed by an ODE solver and the pixel-level composition technique, which helps maintain the sharpness of the resulting predictions. As such, this model can handle the input series at random timestamps. However, although the composition masking can help the resulting output maintain high sharpness, this technique might not work effectively with spatiotemporal datasets with complicated motions. Figure 3.10 shows the schematic illustration of the Vid-ODE model.

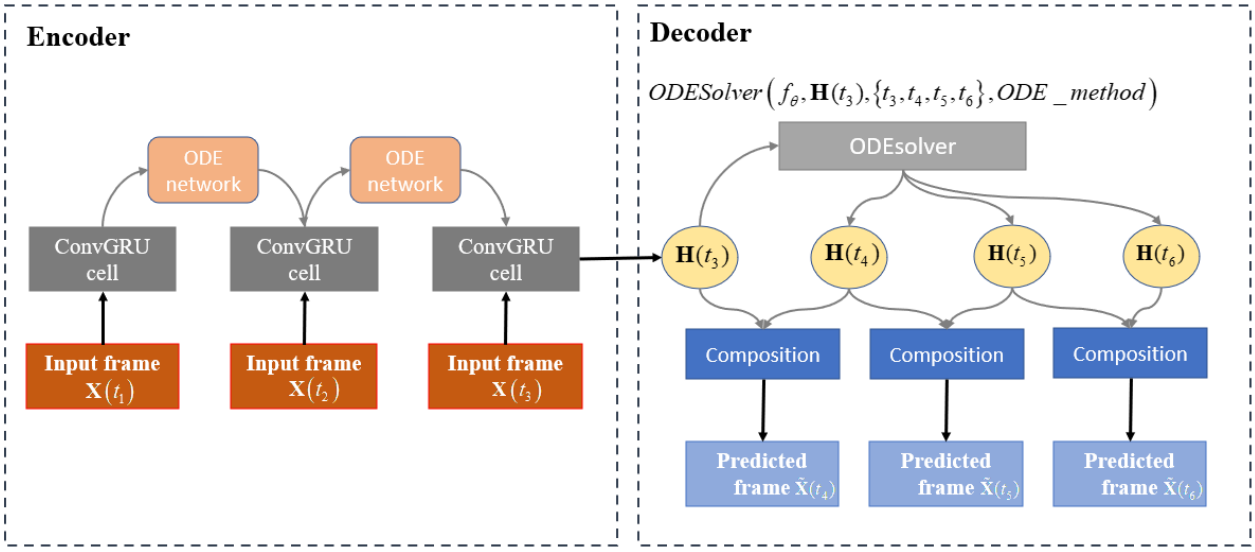


Figure 3.10: Vid-ODE model.

3.3 Transformer

The transformer is a new DL architecture for sequence modelling and transduction problems (e.g., natural language processing and machine translation) which have been getting more awareness recently. Traditional RNNs, such as LSTM and GRU networks, apply recurrent computation with their internal mechanism that precludes parallelization within training sequences and reduces computational efficiency. Furthermore, their simple memory mechanism was not powerful enough to capture long-range information. Therefore, in 2017, the Google team designed and proposed their transformer model for machine translation tasks [54]. In particular, this transformer removed the recurrent computation and used only attention mechanisms to obtain dependencies between input and output. In addition, RNN-based models can also integrate the attention module to boost the overall performance, but their operation relies heavily on recurrent

computation and does not benefit from the parallelization computation. For the sequence modelling, this transformer applies the encoding-decoding structure including two main parts (e.g., the encoder and the decoder) and the stacked architecture constructed by multi-head self-attention, point-wise operator, and other neural layers (e.g., linear layers, activation layers), as illustrated in Figure 3.11. Based on the general observation, the attention component played a crucial role in the whole system as this mechanism allows the transformer to learn long-range dependencies. Although this model can work effectively with translation tasks using sequential data, its structure requires numerous linear layers, which is unsuitable for processing spatial data in the SSP tasks. Furthermore, some complicated spatiotemporal datasets require the dependency extraction between each pair of adjacent input states so that ConvRNN-based models are still the baseline methods for the SSP tasks.

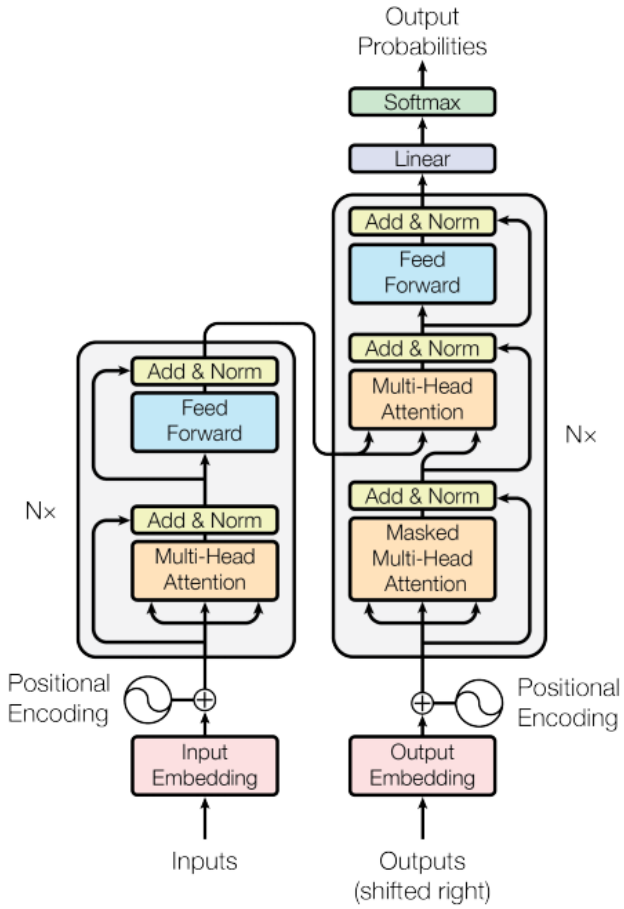


Figure 3.11: Transformer – model architecture. Image obtained from paper [54].

Recently, there have been various transformers designed for the SPL tasks. For instance, Yan et al. proposed the Learning Spatio-Temporal transformer for visual tracking [88]. Zha et al. proposed the Shifted Chunk transformer as a clip encoder to model long-term temporal

dependencies [89]. Overall, the main concept of transformers is the combination between the attention mechanism and specified deep neural networks.

3.4 GAN

Generative Adversarial Networks (GANs) have been the most popular generative model since it was first released in 2014, and GAN models have shown their superior abilities in the image, video generation tasks and other complicated problems in the computer vision area [90]. In essence, a GAN consists of two main components: the generator and discriminator. In this structure, the generator learns the statistical distribution of the latent variables (i.e., the hidden states) and generates the realistic output. Then, the discriminator takes this output and estimates the probability of this data coming from the actual data space. As such, GANs can capture the stochastic behaviour of data that deterministic models like conventional RNNs, and transformers cannot perform effectively. Therefore, GANs have gained considerable attention in recent years. In addition, some studies have used DL-based approaches with adversarial networks for spatiotemporal-based applications such as trajectory prediction, time-series data generation and imputation [91].

For the SSP task, the authors of [92] proposed a model that combines the Conditional GAN and RNN structures to optimize both the regression loss and GAN loss. To be more specific, optimizing the regression loss aims to compute the errors between the resulting prediction and the ground truth, while the GAN loss is used to estimate the data distribution to deal with the uncertainty to produce non-blurry predictions; thus, the model can provide better visualizations. As such, the ConvRNN architecture will be used as predictive learning to create predictions which can be blurry, while conditional GAN will learn a mapping function to map those blurry predictions to the original non-blurry targets. Although the results showed that the sharpness of the forecast is significantly improved, the overall accuracy is decreased, especially with long-term sequence generation. Moreover, GANs are not easy to design and train properly since they require more computing resources and developing processes (e.g., the generator and discriminator).

Chapter 4: Proposed method

In this chapter, we will present the proposed DL-based models designed for the overall SSP task and the drawbacks of conventional methods that we need to improve. First, in terms of system design, main components will be illustrated separately to construct a model: the structure, architecture, and building blocks. Then, we will combine these components to build the entire model.

4.1 Drawbacks of conventional approaches

Before presenting the proposed methods, some fundamental drawbacks of the conventional DL-based approaches for the SSP applications will be analyzed, which many prior studies have reported. Then, we attempt to address those issues in our models.

In this work, we focus on analyzing and addressing two compelling challenges of sequence-to-sequence models: the vanishing gradient problem and the ability to handle irregularly sampled time series. Overall, the vanishing gradient causes a massive downside in the performance of DL models, especially with long-range sequence forecasting problems. Furthermore, irregularly sampled time series contain complicated temporal references that require unique mechanisms to capture those properties.

4.1.1 Vanishing gradient

Most DL-based systems apply the backpropagation and optimization algorithms requiring the gradient computation to update the model parameters, as introduced below. In deep ANNs with multiple layers, the gradients run backward through the networks and adjust after each node following the chain rules. Therefore, such models may quickly meet the vanishing gradient issue when the computed gradients get smaller over the depth of the network.

In essence, the vanishing gradient is an issue in which the gradient values become extremely small and may approach zeros. Hence, the learning ability of some network nodes or layers will be easily saturated, and the system can not update learnable parameters properly. As a result, this phenomenon significantly reduces the model's overall performance. The vanishing gradient problem occurs relatively often in DL-based sequence-to-sequence models such as RNNs [93]. Although some variants of RNNs like LSTM and GRU applied memory mechanisms represented by the gating techniques, they can not effectively handle long-range sequential data due to the natural structure of RNNs (i.e., the rolling and unrolling form). Based on the cell structure, the current and cell memory states are updated using only one hidden representation state at the previous time step, so this knowledge is insufficient to handle long-range sequences. From the perspective of mathematics, equations (2.13) and (2.14) demonstrate that the total derivative of the loss function is computed based on the multiplication product of all the derivatives of states of the

entire input series. Therefore, the model can quickly encounter the vanishing gradient issue if the training sequences are comprised of hundreds of time steps. As a side effect, this problem significantly slows down the learning process since the trainable weights are updated at a slow pace. For the long-term video prediction application, we can experience blurry effects (i.e., the decrease and distortion in the detail of an image) in some frames at the end of the series. In other words, the resulting predictions worsen over time when the gradients get smaller.

In addition, the vanishing gradient issue closely involves the model architecture. For instance, in the stacked encoding-decoding architecture of the TrajGRU model shown in Figure 3.6 [2], the final output of each neural layer of the encoder is represented by a single hidden representation, and the recurrent computation essentially generates this state. As a result, this state cannot contain all the critical features of the input sequence (e.g., short-term, and long-term features); thus, the decoder will not receive enough information and perform effectively. Moreover, the gradients running backward from the decoder to the encoder can be represented by equations (2.13) and (2.14) since the interested states are computed using just a single previous form. Eventually, we meet the vanishing gradient issue one more time.

4.1.2 Irregularly sampled time series prediction

In the SPL field, most recent works follow the regular time sampling rule for the SSP tasks when preprocessing the data to generate training, testing, and validation sequences. For instance, RNN-based systems (e.g., ConvLSTM [1], ConvGRU [73, 74], TrajGRU [2], PredRNN [3, 13, 81], Conv-TT-LSTM [18]) employ the same collection of learnable parameters to compute the hidden states at different timestamps based on the general rolling structure. Consequently, they intrinsically assume the time gap (i.e., the time interval) between adjacent data points is fixed and unchanged over time.

Although the constant time interval sampling can be accepted as the standard assumption in the temporal analysis for simplicity purposes, there are numerous scenarios in which this assumption is not practical. First, not all spatiotemporal datasets follow this sampling type; some research areas are interested in data recorded at arbitrary time steps, such as medical, economics, and climate modelling [94, 95]. Besides discrete series, some spatiotemporal data in continuous space are applied for continuous-time physical models [96], biology applications [97, 98] and telecommunications [99]. Second, the fixed time interval sampling might be challenging to implement in real-world applications because of hardware errors, noise, and outside effects on machines. As such, an observed time series can lose data points at some time, which data analysists can capture based on the recording logs.

Furthermore, even if a sequential dataset is obtained with proper regular time sampling, we may not want to use the complete information of a given series in some situations. For example, a simple case will be analyzed using a traditional DL-based sequence-to-sequence model, which follows the regular time sampling for an SSP task. At first, this model is assumed to have two parts: the encoder and the decoder. With the overall SSP problem, our mission is to predict the

future K frames based on the given input sequence with N frames. In the encoding part, let the input sequence length is ten ($N = 10$), and each data point be sampled regularly at time steps $t_i = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. In terms of the decoding part, if the output sequence length or the prediction length be three ($K = 3$), the predicted frames in the output sequence are supposed to represent the information at $t_o = \{11, 12, 13\}$. However, if we want to obtain the three future data frames at time steps $t_o = \{14, 17, 19\}$, we have to predict an entire sequence with nine data frames at $t_o = \{11, 12, 13, 14, 15, 16, 17, 18, 19\}$. As a result, we will waste time and resources on computing unnecessary information. As a side effect, the system may easily get the gradient vanishing problem when the total number of data points increases. In this situation, we can do better by directly predicting three frames at time steps $t_o = \{14, 17, 19\}$ using the ten previous frames. Furthermore, we can even do this task better by not using all ten previous frames (e.g., five frames at time steps $t_i = \{1, 3, 5, 9, 10\}$) to make the prediction. Consequently, we can not only save time and resources but also handle time series recorded at arbitrary timestamps. Figure 4.1 illustrates these scenarios

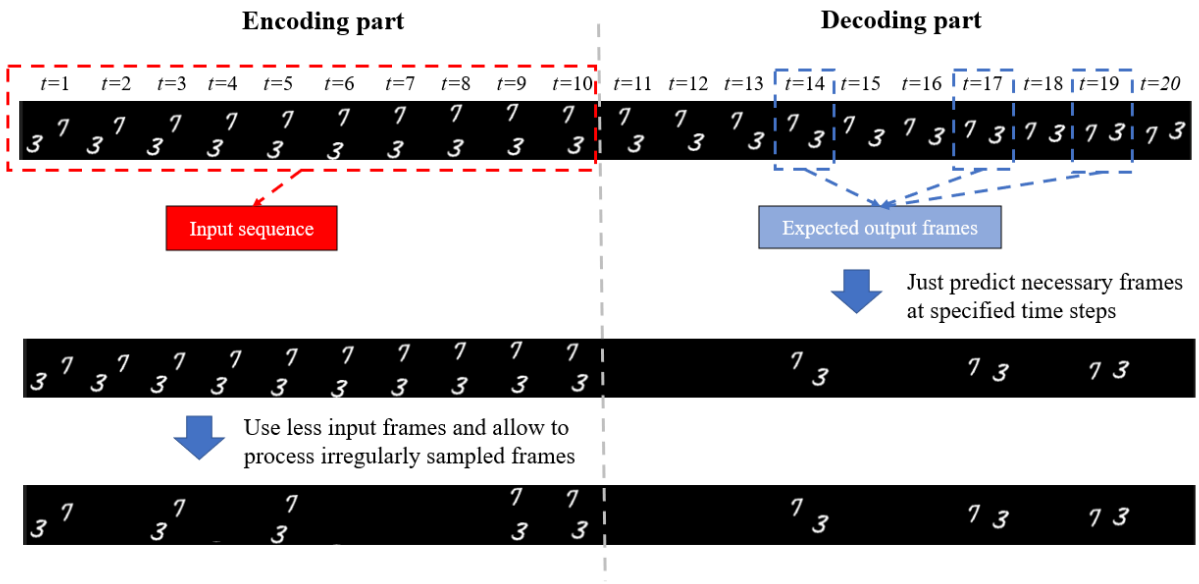


Figure 4.1: Overview illustration of the scenario using the irregular sampled time series for the SSP task.

However, predicting future spatial data at arbitrary timestamps is highly challenging because of the complex temporal reference. The system must correctly capture the irregularly sampled data points and reconstruct the output with acceptable sharpness. Once we can capture this temporal reference property, we can flexibly adjust the time gaps according to different scenarios. Therefore, handling irregularly sampled sequences in the discrete domain involves the ability to process data in a continuous-time field. For instance, we can interpolate data frames at unseen time steps based on the given time series and extrapolate future frames with different frame rates or

arbitrary future timestamps. Eventually, this is the fundamental idea of our overall SSP task and the motivation for us to design the DL-based sequence-to-sequence models.

4.2 Proposed system for the overall Spatiotemporal Sequence Predictive task

This section will present the proposed system for the overall SSP task. Specifically, fundamental elements will be introduced for designing the models: the model structure, architecture, and building blocks. Then, those elements are used to build three novel DL-based sequence-to-sequence models for the overall SSP task: the TrajGRU-Attention, TrajGRU-ODE, and the TrajGRU-Attention-ODE.

4.2.1 Model structure

In the first step of the design process, the encoding-decoding structure is chosen for our sequence-to-sequence models as it is the most commonly used structure for the SSP tasks. This structure has two main parts: the encoder and decoder; each part takes distinct functions and supports the other part by a particular communication approach represented by the model architecture.

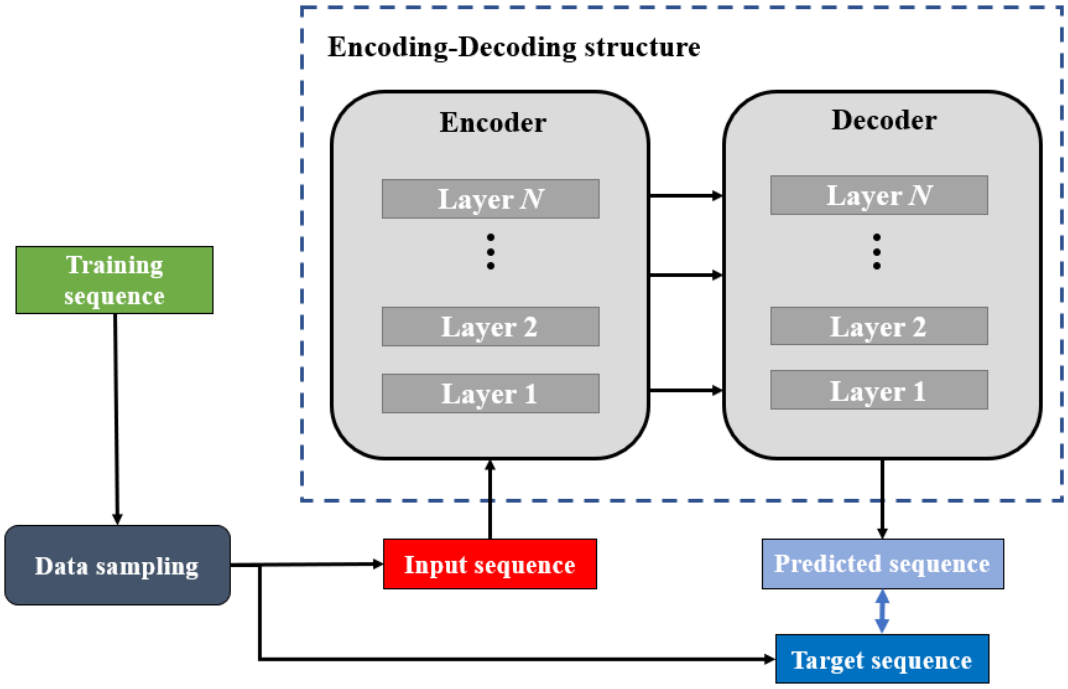


Figure 4.2: Encoding-decoding model structure.

The encoding part takes the input sequence to extract spatiotemporal features, then generates the representation state to transfer to the decoder. Hence, the decoder can obtain the knowledge of

observation data and then make predictions. Eventually, multiple neural layers are combined to construct the encoder and decoder according to a specified architecture. In this work, the building blocks indicate artificial neural layers, and numerous central blocks (i.e., main layers) contain different levels of spatiotemporal features. Figure 4.2 shows the overall view of our encoding-decoding structure.

For the training process, the loss function value is calculated using the prediction (i.e., the output of the decoder) and the target sequence (i.e., the subsequence of the examined sequence). After that, the gradients running backward from the decoder's main layers to the encoder's main layers are computed using the BPTT algorithms, as described in section 2.2.4. Eventually, the parameters of each part are updated using a specified optimization algorithm, as shown in section 2.2.8. In the next section, the novel model architecture will be introduced to analyze further the information flows running forward and the gradient flows running back through the entire system.

4.2.2 Model architecture

The next step in building the model is to design its architecture. In particular, the model architecture describes the connection between distinct components: the encoder and the decoder, building blocks (i.e., the neural layers), and neural cells (i.e., the small units of recurrent layers). Therefore, researchers can visualize how they communicate with others and how the information and gradient flow through the system. As a result, this element affects the model's parameter updating and optimization processes, so we must consider its design carefully.

There have been several architectures for the encoding-decoding structure of sequence-to-sequence models, such as the conventional stacked and the deep-transition used in the ConvLSTM model [1, 3], gradient highway used in the PredRNN++ [81], etc. First, the stacked architecture will be examined, one of the most common styles, as shown in Figure 3.3. In the stacked ConvLSTM architecture, during the recurrent computation of each layer, the output hidden state of each neural cell is duplicated and steered in two distinct paths following the horizontal and vertical directions. As mentioned in section 3.1, the ConvLSTM uses the input data and previous state to update the current hidden state. In this case, the output following the horizontal direction will be the previous state of the following recurrent computation in the current neural layer. Meanwhile, the production following the vertical direction will play a role in the current input data of the higher-level layer. Therefore, the system can perform hierarchical feature extraction for efficient representation modelling. However, there are some drawbacks when implementing this architecture to the model. First, the neural unit of the top layer at a specific time step might forget some practical information coming from the neural cell of the bottom layer at the previous time step when the system uses a high number of stacked layers. Second, the final representation state of each recurrent layer in the encoder cannot contain sufficient knowledge of a long-range series; thus, the decoder cannot learn the spatiotemporal features effectively. Consequently, these issues significantly reduce the system's performance.

Based on the above observations, we design a novel architecture combining the stacked style, zigzag connection, and reverse layers at the decoder, as illustrated in Figure 4.3. Although the stacked architecture can lead to the vanishing gradient issue, it allows the system to extract more features with different sizes and dimensions by stacking multiple neural layers (i.e., the hierarchical feature extraction). To alleviate the negative effect of this style, we take the idea of the spatiotemporal memory flow of PredRNN++ [81] to design a zigzag connection, which creates a shortcut to communicate between the top and bottom recurrent layers. In particular, this zigzag shortcut can boost the model's memory ability by increasing the network's depth (i.e., the connection between units of top and bottom layers allows the model to generate more representations between two timestamps). Furthermore, this architecture enables the model to use more stacked intermediate layers. As a result, it intrinsically improves the system's overall performance. In Figure 4.3, at the encoder, the operations of neural cells when applying the zigzag connection can be formulated as

$$\begin{aligned}
\boldsymbol{H}_{zz}(t_i) &= leakyRELU\left(\boldsymbol{W}_{up_sampling} * \boldsymbol{H}_{top_layer}(t_i)\right) \\
\boldsymbol{Z}_{zz}(t_i) &= \sigma\left(\boldsymbol{W}_h * \boldsymbol{H}_{bottom_layer}(t_i) + \boldsymbol{W}_{zz} * \boldsymbol{H}_{zz}(t_i)\right) \\
\boldsymbol{H}^{new}_{bottom_layer}(t_i) &= \boldsymbol{H}_{bottom_layer}(t_i) + \boldsymbol{Z}_{zz}(t_i) \circ \boldsymbol{H}_{zz}(t_i)
\end{aligned} \tag{4.1}$$

And vice versa at the decoder, so that we have the according equations for the zigzag shortcut as

$$\begin{aligned}
\boldsymbol{H}_{zz}(t_i) &= leakyRELU\left(\boldsymbol{W}_{down_sampling} * \boldsymbol{H}_{bottom_layer}(t_i)\right) \\
\boldsymbol{Z}_{zz}(t_i) &= \sigma\left(\boldsymbol{W}_h * \boldsymbol{H}_{top_layer}(t_i) + \boldsymbol{W}_{zz} * \boldsymbol{H}_{zz}(t_i)\right) \\
\boldsymbol{H}^{new}_{top_layer}(t_i) &= \boldsymbol{H}_{top_layer}(t_i) + \boldsymbol{Z}_{zz}(t_i)\boldsymbol{H}_{zz}(t_i)
\end{aligned} \tag{4.2}$$

In (4.1) and (4.2), $\boldsymbol{Z}_{zz}(t_i)$ indicates the update gate at the time step t_i, which decides how much information of the zigzag state $\boldsymbol{H}_{zz}(t_i)$ will be obtained; $\boldsymbol{W}_{up\_sampling}$, $\boldsymbol{W}_{down\_sampling}$ refer to the learnable parameters of the up-sampling , down-sampling layers, respectively; $\boldsymbol{W}_h$, $\boldsymbol{W}_{zz}$ refer to the learnable parameters of the convolutional layers applied to extract the features of the hidden states. Consequently, the new state $\boldsymbol{H}^{new}_{bottom\_layer}(t_i)$ at the encoder and $\boldsymbol{H}^{new}_{top\_layer}(t_i)$ at the decoder can be obtained.

In addition, we apply the reversed decoding layers like the TrajGRU model [2], as shown in Figure 3.6, because we want the information flows transferred straightforwardly from the encoder to the decoder. To be more specific, we will present the operation of each primary part of this model. First, the bottom central building block in the encoder refers to the lowest-level layer that receives the original input sequences and generates the global representations to feed into the next layer. Then, the higher-level layers will process and extract features with smaller state sizes and higher numbers of channels, represented for local spatiotemporal states. Eventually, the top layer will contain the local representations with the smallest size and the highest number of channels. In addition, a down-sampling block is inserted between each pair of adjacent recurrent layers to

increase the feature's level. Overall, the encoder follows the bottom-up approach to extract the local features from the global information. In contrast, the decoder applies the top-down approach, and the higher-level layers employ the local features to construct the global representations. The lowest-level states at the bottom directly influence the final outputs of the whole system.

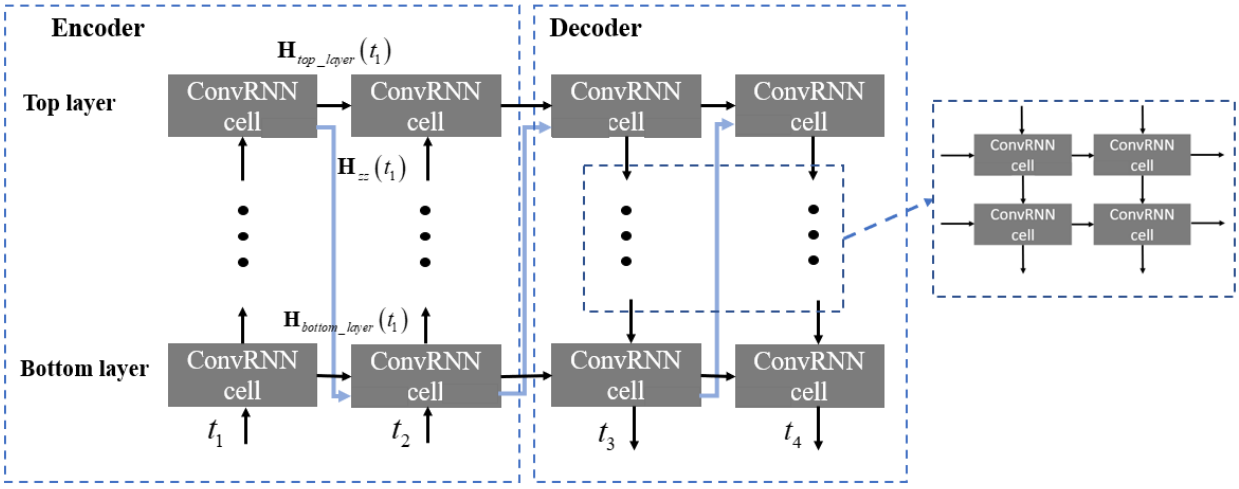


Figure 4.3: Proposed model architecture.

Consequently, the resulting outcomes can be reconstructed with highly detailed information because the other higher-level features generate the final representation states. In addition, we implement the up-sampling block between two adjacent central building blocks to divide the decoding layers into different levels. As a side benefit, this architecture can help the model save the number of recurrent layers since each component (i.e., the encoder or decoder) uses a single resampling block type: the encoder applies the down-sampling blocks, and the decoder applies the up-sampling blocks. In the next section, the information on building blocks to build the model will be presented.

4.2.3 Building blocks

In the proposed design, building blocks are the minor components used to construct the model structure based on the proposed architecture. Basically, they indicate the neural network layers (e.g., convolutional, deconvolutional layers, ConvRNN layers) and other DL-based modules. In this thesis, we divide the model building blocks into two groups: the central and the connection building blocks. In the group of central blocks, we proposed a novel ConvRNN, namely Trajectory Gated Recurrent Unit, integrating the Ordinary Differential Equation technique (TrajGRU-ODE). Furthermore, we implement the ConvGRU and TrajGRU networks to build our models and the baseline. In the group of connection blocks, besides up-sampling and down-sampling blocks, we proposed a novel attention module, namely Motion-based Attention, which can be flexibly dropped into another layer to boost the memory ability of the system and alleviate the gradient vanishing issue.

4.2.3.1 Trajectory Gated Recurrent Unit integrating Ordinary Differential Equation technique (TrajGRU-ODE)

In mathematics, the main goal of solving ODEs is to describe the rate of change and evolution of a phenomenon over a certain period with respect to the initial conditions. Based on this observation, prior ODE-based models have been proposed and designed for continuous-time prediction and generation tasks (i.e., these models attempt to predict future or unseen data points at random timestamps in the continuous domain regarding the initial conditions). Therefore, the essential requirements of these models are the initial conditions. The initial condition refers to the representation states in the SPL field at the beginning. There are numerous ways to generate the initial state in DL-based models. Generally, the encoder will develop the initial state using the given input information in the encoding-decoding structure. Then, the decoder will apply a defined ODE solver to generate the future continuous hidden forms in the latent space. For instance, the Latent-ODE model uses the RNN to encode the statistical characteristics as the initial conditions, as illustrated in Figure 3.8. With the proposed system, the ConvRNNs and Motion-based Attention (MA) modules are implemented to construct the encoder, generating the initial representation state for solving the ODEs defined in the decoder.

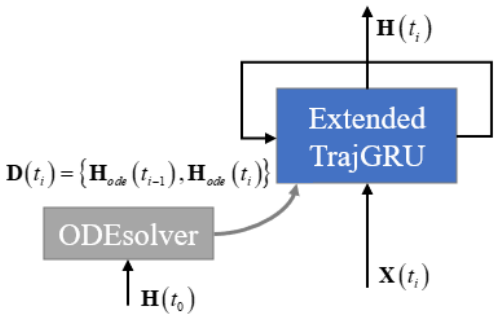
In this section, the Trajectory Gated Recurrent Unit integrating the Ordinary Differential Equation technique (TrajGRU-ODE) will be introduced as the central building block. After that, in the next section we will present the entire TrajGRU-ODE model, including all terms: the building blocks, model architecture and structure. From the structure perspective, there are two fundamental components in the TrajGRU-ODE block: the ODE solver module and the extended TrajGRU unit equipped with additional gates. To clarify its operation, we start with the rolling RNN structure presenting the internal loop mechanism of the neural layer. Then, we will roll it out to dive deeper into the detailed computations of each neural cell at each timestamp, as shown in Figure 4.4.

The TrajGRU-ODE block implements an internal loop in the rolling structure to perform the recurrent computation like many conventional RNNs. However, instead of just using the pair of the input data $\boldsymbol{X}(t_i)$ at the time step t_i and the previous hidden state $\boldsymbol{H}(t_{i-1})$, a new intermediate state $\boldsymbol{D}(t_i)$ is defined, namely the difference representation; then, we insert this state into the recurrent computation alongside this pair. In particular, the state $\boldsymbol{D}(t_i)$ is computed using the couple of latent continuous-time states $\{\boldsymbol{H}_{ode}(t_{i-1}), \boldsymbol{H}_{ode}(t_i)\}$ which is generated by the ODE solver component. Hence, this state can emphasize the distinctness and the evolution of data from time step t_{i-1} to t_i. Eventually, the formulation to compute $\boldsymbol{D}(t_i)$ can be rewritten as the following

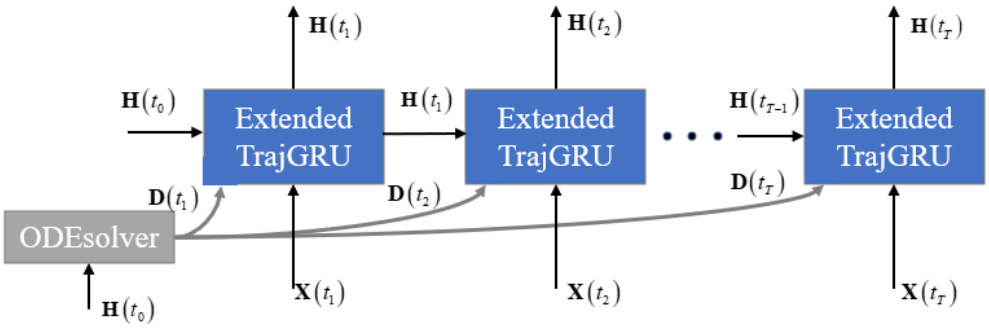
$$\boldsymbol{D}(t_i) = W_{combine}^{ode} * \{\boldsymbol{H}_{ode}(t_{i-1}), \boldsymbol{H}_{ode}(t_i)\}, \tag{4.3}$$

where $W_{combine}^{ode}$ denotes the convolutional layer with a kernel size of (1×1), resizing the concatenation of two latent states $\{\boldsymbol{H}_{ode}(t_{i-1}), \boldsymbol{H}_{ode}(t_i)\}$. More specifically, we will examine a simple task to generate T unseen data frames at T random timestamps regarding the given initial

knowledge. First, we define the integrated ODE solver module by designing an ANN that played a role of the ODE function and a numerical integration method, as illustrated in Figure 2.4. In this thesis, the ODE function is represented by a sequence of convolutional and tanh activation layers, as shown in Figure 4.5. Second, this ODE solver will take the initial state to generate the set of continuous-time hidden states $\{\boldsymbol{H}_{ode}(t_0), \ldots, \boldsymbol{H}_{ode}(t_T)\}$ in the latent space at the expected timestamps from t_0 to t_T.



a) The rolling structure



b) The unrolling structure

Figure 4.4: Two types of TrajGRU-ODE structures: a) The rolling structure. b) The unrolling structure.

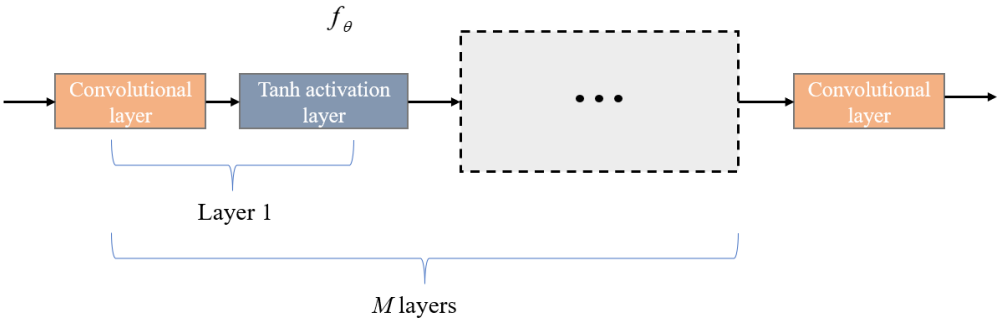


Figure 4.5: The ODE function.

Although those states obtain the continuous-time dynamics, the learnable ODE function of the ODE solver module is not sufficient to capture practical features and reconstruct the resulting outcomes with high sharpness simultaneously, especially when the sequence data contain different complicated spatiotemporal properties. Therefore, we incorporate the latent continuous-time states generated by the ODE solver module and the hidden states generated by the extended TrajGRU layer to get the work done more effectively. As mentioned earlier, the intermediate state $\boldsymbol{D}(t_i)$ is computed by combining the pair of two adjacent latent states $\{\boldsymbol{H}_{ode}(t_{i-1}), \boldsymbol{H}_{ode}(t_i)\}$, as shown in (4.3). As such, the state $\boldsymbol{D}(t_i)$ at time step t_i, allows the system to foresee how the examined state can evolve and what may happen at this time step. After that, the extended TrajGRU will combine this state with other knowledge processed by the gated mechanism to reconstruct the resulting output. Therefore, the TrajGRU-ODE block can learn the sequence's continuous-time and spatiotemporal dynamics simultaneously. As a result, this technique can be applied to handle the irregularly sampled time series and perform continuous-time prediction tasks. The detailed operation formulations of the TrajGRU-ODE unit can be written as follows

$$
\begin{aligned}
&\boldsymbol{U}(t_i), \boldsymbol{V}(t_i) = \gamma\big(\boldsymbol{X}(t_i), \boldsymbol{H}(t_{i-1})\big) \\
&\boldsymbol{Z}(t_i) = \sigma\left(\boldsymbol{W}_{xz} * \boldsymbol{X}(t_i) + \sum_{l=1}^{L} \boldsymbol{W}_{hz,l} * warp\big(\boldsymbol{H}(t_{i-1}), \boldsymbol{U}(t_i, l), \boldsymbol{V}(t_i, l)\big) + \boldsymbol{B}_z\right) \\
&\boldsymbol{R}(t_i) = \sigma\left(\boldsymbol{W}_{xr} * \boldsymbol{X}(t_i) + \sum_{l=1}^{L} \boldsymbol{W}_{hr,l} * warp\big(\boldsymbol{H}(t_{i-1}), \boldsymbol{U}(t_i, l), \boldsymbol{V}(t_i, l)\big) + \boldsymbol{B}_r\right) \\
&\widetilde{\boldsymbol{H}}(t_i) = tanh\left(\boldsymbol{W}_{xh} * \boldsymbol{X}(t_i) + \boldsymbol{R}(t_i) \circ \left(\sum_{l=1}^{L} \boldsymbol{W}_{hh,l} * warp\big(\boldsymbol{H}(t_{i-1}), \boldsymbol{U}(t_i, l), \boldsymbol{V}(t_i, l)\big)\right) + \boldsymbol{B}_h\right) \\
&\boldsymbol{H}_{TrajGRU}(t_i) = \boldsymbol{Z}(t_i) \circ \boldsymbol{H}(t_{i-1}) + \big(1 - \boldsymbol{Z}(t_i)\big) \circ \widetilde{\boldsymbol{H}}(t_i) \\
&\boldsymbol{Z}_{ode}(t_i) = \sigma\big(\boldsymbol{W}_{hzode} * \boldsymbol{H}_{TrajGRU}(t_i) + \boldsymbol{W}_{dzode} * \boldsymbol{D}(t_i) + \boldsymbol{B}_{zode}\big) \\
&\boldsymbol{R}_{ode}(t_i) = \sigma\big(\boldsymbol{W}_{hrode} * \boldsymbol{H}_{TrajGRU}(t_i) + \boldsymbol{W}_{drode} * \boldsymbol{D}(t_i) + \boldsymbol{B}_{rode}\big) \\
&\widetilde{\boldsymbol{H}}_{ode}(t_i) = tanh\big(\boldsymbol{W}_{hhode} * \boldsymbol{H}_{TrajGRU}(t_i) + \boldsymbol{R}_{ode}(t_i) \circ \big(\boldsymbol{W}_{dhode} * \boldsymbol{D}(t_i)\big) + \boldsymbol{B}_{hode}\big) \\
&\boldsymbol{H}(t_i) = \boldsymbol{Z}_{ode}(t_i) \circ \boldsymbol{H}_{mid}(t_i) + \big(1 - \boldsymbol{Z}_{ode}(t_i)\big) \circ \widetilde{\boldsymbol{H}}_{ode}(t_i)
\end{aligned}
\tag{4.4}
$$

In (4.4), the equations are divided into two separate parts: the standard TrajGRU unit and the extended gates equipped to capture the continuous-time dynamic states. From the perspective of the TrajGRU operation, $\boldsymbol{X}(t_i), \boldsymbol{H}(t_{i-1})$ indicate the input data at the time step t_i and the previous hidden state, respectively; γ presents the neural network generating the continuous flow fields $\boldsymbol{U}(t_i), \boldsymbol{V}(t_i)$ with L different connection links in the transition; $warp$ presents a function using to combine the sets of flow fields and the hidden state to generate an intermediate state with approximated location information; $\boldsymbol{Z}(t_i)$, $\boldsymbol{R}(t_i)$ indicate the update, reset gates, respectively; $\widetilde{\boldsymbol{H}}(t_i)$ represents the memory state, which is modified according to the reset gate and information flows at each time step; $\boldsymbol{H}_{TrajGRU}(t_i)$ represents the output hidden state of the standard TrajGRU

unit. In terms of the extended gates, the states $\boldsymbol{H}_{TrajGRU}(t_i)$, $\boldsymbol{D}(t_i)$ denote the input data, the hidden state, respectively; $\boldsymbol{Z}_{ode}(t_i)$, $\boldsymbol{R}_{ode}(t_i)$ denote the additional update, reset gates; $\widetilde{\boldsymbol{H}}_{ode}(t_i)$ represents the new memory state; eventually, the final output state $\boldsymbol{H}(t_i)$ contains both continuous-time and spatiotemporal features. For the optimizing process, the system has the collections of learnable and trainable parameters: $(\boldsymbol{W}_{xz}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh})$, $(\boldsymbol{W}_{hz,l}, \boldsymbol{W}_{hr,l}, \boldsymbol{W}_{hh,l})$, $(\boldsymbol{B}_z, \boldsymbol{B}_r, \boldsymbol{B}_h)$, $(\boldsymbol{B}_{zode}, \boldsymbol{B}_{rode}, \boldsymbol{B}_{hode})$, and $(\boldsymbol{W}_{hzode}, \boldsymbol{W}_{dzode}, \boldsymbol{W}_{hrode}, \boldsymbol{W}_{hrode}, \boldsymbol{W}_{hhode}, \boldsymbol{W}_{dhode})$. Finally, "$*$", as previously, denotes the convolution operator; "$\circ$" denotes the Hadamard product; σ refers to the sigmoid function; $tanh$ refers to the tanh function. Generally, the sigmoid function, whose values are from 0 to 1, allows the reset and update gates to regulate the information flows; the tanh function, whose values are from -1 to 1, can represent the new information of the current input data. Figure 4.6 illustrates the operation of the TrajGRU-ODE cell.

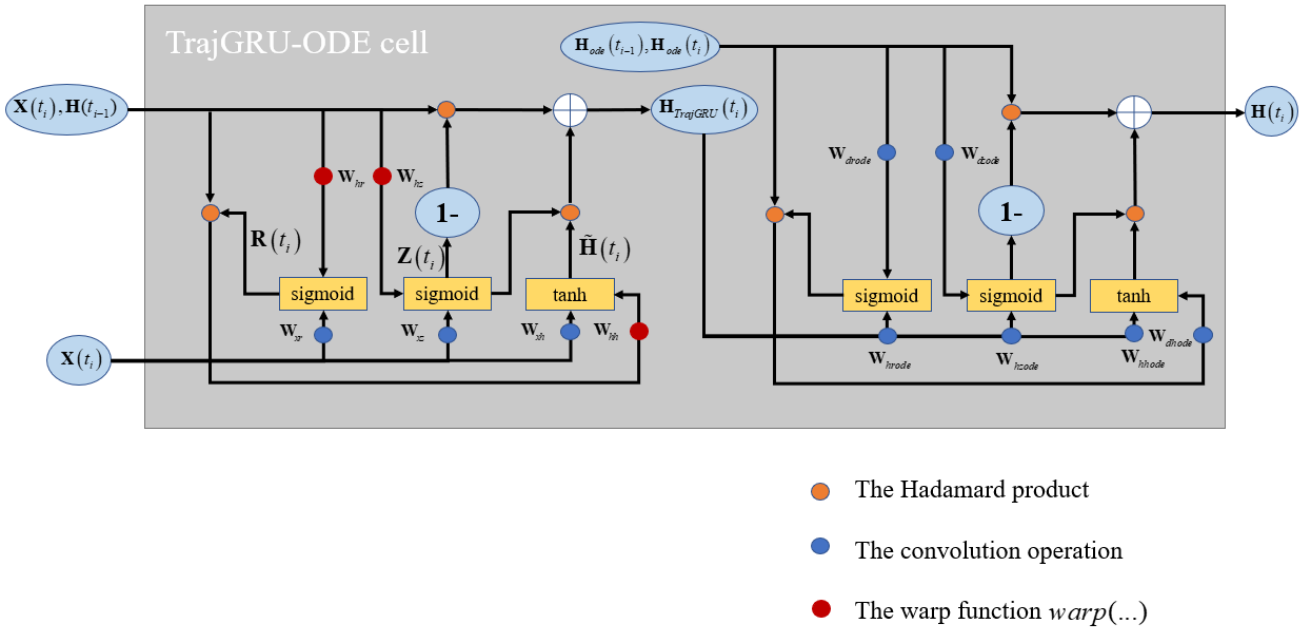


Figure 4.6: TrajGRU-ODE cell.

4.2.3.2 Motion-based Attention module

Generally, the attention mechanism employs three fundamental components: the keys, values, and queries, as introduced previously. In the SPL research field, the evolution and changing processes over time of the spatiotemporal phenomena intrinsically involve the motion characteristics, which are mathematically described in terms of position, velocity, acceleration, and time variables. Based on this observation, we leverage the characteristic motion analysis as the key idea to build our novel attention mechanisms, namely Motion-based Attention (MA). The main objective of this mechanism is to help the system look back on the prior knowledge and pay more attention to valuable parts based on the motion characteristics of the observed hidden states and input frames. Moreover, by learning the representation states of the motions over time, the

system is capable of handling data points sampled at arbitrary given timestamps. Consequently, it can be applied to the overall SSP task.

In the thesis, the MA module is built as a connection building block that we can flexibly insert into different recurrent network layers. According to the encoding-decoding model structure used in prediction tasks, there are two separate parts with distinct functions, and each part focuses on analyzing specified data. Thus, we design two attention mechanisms (i.e., the encoding and decoding attention). The encoding attention mechanism is also called self-attention, which explores the interdependencies of data points within the input sequence. In decoding attention, its operation concerns the correlation between the current analyzed state information with a certain number of previous forms, including the states at both the encoder and decoder sides. In other words, we will build the encoding attention module, which takes the input sequence as the information of interest, and the decoding attention, which helps the model look back on the previous hidden states. In addition, these mechanisms follow the left-right approach that creates the parallelization computation inside the recurrent operation of central layers.

- **The Motion-based Encoding Attention (MEA)**

From the side of the encoder, the Motion-based Encoding Attention (MEA) mechanism is proposed for performing the parallelization computations on the input sequence. Typically, at each timestamp of the input sequence, the current representation state can be computed using the current data frame and previous hidden state when using the traditional recurrent computation of ConvRNNs (e.g., ConvLSTM, ConvGRU). As introduced earlier, this current state will spread out in the stacked architecture with multiple layers in the vertical and horizontal directions, as illustrated in figure 3.3. The information flow running through the vertical direction refers to the previous hidden state of the following recurrent computation step. The state in the horizon direction refers to the input data of the neural cell in the adjacent recurrent layer. As such, in the forward propagation, the higher-level layer takes the output of the lower-level one as the input knowledge and captures the valuable features by analyzing the dependencies. However, since the updating computation at each neural cell takes into account the short-term dependencies of two adjacent data frames, the long-term dependencies between the current input and all previous series are not effectively captured. As a result, this issue leads to the downside in the reconstruction task, especially when it comes to time series with high correlation. Therefore, we need an operation combining multiple input frames to update the hidden state at the encoding part. In particular, we will implement a parallelization computation in the input series following the vertical direction to allow the model to capture long-term interdependencies productively. Eventually, the entire step-by-step of applying the MEA is described as follows

First, at the time step t_Q of the input sequence, we derive the formulas used to compute additional states which represent the motion characteristics. By assuming the set of input frames $\{\boldsymbol{X}(t_i)\}_{i\leq Q, i\in N^*}$ (i.e., the set length is Q) as the position information, we can define the set of velocity states as follows

$$\{\boldsymbol{V}(t_i)\}_{i\leq Q, i\in N^*}: \begin{cases} \boldsymbol{V}(t_1) = \boldsymbol{X}(t_1) \\ \boldsymbol{V}(t_i) = f_V\left(\frac{\boldsymbol{X}(t_i)-\boldsymbol{X}(t_{i-1})}{t_i-t_{i-1}}\right) \end{cases}. \tag{4.5}$$

Then, according to the set of velocity states, the set of acceleration states can be defined as

$$\{\boldsymbol{A}(t_i)\}_{i\leq q, i\in N^*}: \begin{cases} \boldsymbol{A}(t_1) = \boldsymbol{V}(t_1) \\ \boldsymbol{A}(t_i) = f_A\left(\frac{\boldsymbol{V}(t_i)-\boldsymbol{V}(t_{i-1})}{t_i-t_{i-1}}\right) \end{cases}. \tag{4.6}$$

In (4.5) and (4.6), f_V, f_A indicate distinct ANNs: each includes a convolutional layer with the kernel size of (1×1) and an average pooling layer. Thus, sets $\{\boldsymbol{V}(t_i)\}_{i\leq q, i\in N^*}$, $\{\boldsymbol{A}(t_i)\}_{i\leq q, i\in N^*}$ can be represented by tensors $\boldsymbol{V}_{t_Q}$ and $\boldsymbol{A}_{t_Q}$, respectively, and they are reshaped into the same shape of $(Q \times B \times C \times 1)$ (i.e., Q denotes the set length, B denotes the batch size, C denotes the number of channels of the state). In addition, the set of time steps $\boldsymbol{T}_{t_Q}$ is reshaped from $(Q \times B \times 1)$ to $(Q \times B \times C \times 1)$ by replicating the values because we want all keys of the attention mechanism to have a common shape.

Based on the computations of Luong's attention in the RNNs [52], the formulation computing the correlation between the current input data $\boldsymbol{X}(t_q)$ with its previous frames $\{\boldsymbol{X}(t_i)\}_{i\leq q, i\in N^*}$, as

$$\boldsymbol{C}_{t_Q} = Avg\_pooling(tanh\left(\boldsymbol{W}_{a1} * \boldsymbol{X}(t_Q) + \boldsymbol{W}_{a2} * \left\{\boldsymbol{X}(t_Q)\right\}_{i\leq Q, i\in N^*}\right) \ , \tag{4.7}$$

where $\boldsymbol{W}_{a1}$, $\boldsymbol{W}_{a2}$ present the learnable weights of convolutional layers, $Avg\_pooling$ denotes the average pooling layer. In (4.7), the term $\boldsymbol{W}_{a1} * \boldsymbol{X}(t_Q)$ is added with each element in the set of tensors $\boldsymbol{W}_{a2} * \{\boldsymbol{X}(t_i)\}_{i\leq Q, i\in N^*}$; then they are stacked together before passing through the tanh function and average pooling. Therefore, the final resulting tensor of this computation can have the size of $(Q \times B \times C \times 1)$. Eventually, four tensors $\boldsymbol{V}_{t_Q}$, $\boldsymbol{A}_{t_Q}$, $\boldsymbol{T}_{t_Q}$, $\boldsymbol{C}_{t_Q}$ are obtained with the same shape of $(Q \times B \times C \times 1)$, presenting four keys of our attention. In particular, we expect that those states can help the model make the decision about how much attention to pay to each data input frame. Furthermore, since those states contain the information of the changing rates (i.e., the evolution) regardless of regular or irregular sampling assumptions, the model can learn and recognize spatiotemporal representations at specified time steps. Figure 4.7 shows the overview of this step.

In the next step, based on those keys, we compute the alignment scores, which indicate the collection of weights to assign to each element in the set of input frames $\{\boldsymbol{X}(t_i)\}_{i\leq Q, i\in N^*}$

$$\boldsymbol{S}_{t_Q} = Soft\,Max\left(f_{combine}\left(\boldsymbol{V}_{t_Q}, \boldsymbol{A}_{t_Q}, \boldsymbol{T}_{t_Q}, \boldsymbol{C}_{t_Q}\right)\right), \tag{4.8}$$

where $f_{combine}$ represents the linear layer with the number of input channels being $(4 \times C)$ and the number of output channels being (C); $SoftMax$ denotes the SoftMax function; thus, the

alignment score $\boldsymbol{S}_{t_Q}$ has a size of $(Q \times B \times C \times 1)$. Then, we assign this score $\boldsymbol{S}_{t_Q}$ to the set of input frames $\{\boldsymbol{X}(t_i)\}_{i<Q, i\in N^*}$ to generate the attention state at time step t_Q, as

$$\boldsymbol{H}^a\left(t_Q\right) = \boldsymbol{S}_{t_Q} \circ \{\boldsymbol{X}(t_i)\}_{i<Q, i\in N^*}, \tag{4.9}$$

where "$\circ$" denotes the Hadamard product. Eventually, we combine the current input $\boldsymbol{X}(t_Q)$ with this attention state $\boldsymbol{H}^a\left(t_Q\right)$ to generate the new input data, as

$$\boldsymbol{X}^{new}(t_Q) = \boldsymbol{W}_{1\times1} * \{\boldsymbol{X}(t_Q), \boldsymbol{H}^a\left(t_Q\right)\}, \tag{4.10}$$

where $\boldsymbol{W}_{1\times1}$ represents the learnable weights of the convolutional layers with a kernel size of (1×1). Consequently, the new input intrinsically obtains the knowledge of data frames stretching from the beginning timestamp t_1 to the current instant t_Q. Therefore, the MEA of the encoder possesses a long-term memory and an ability to learn the irregular time sampled series.

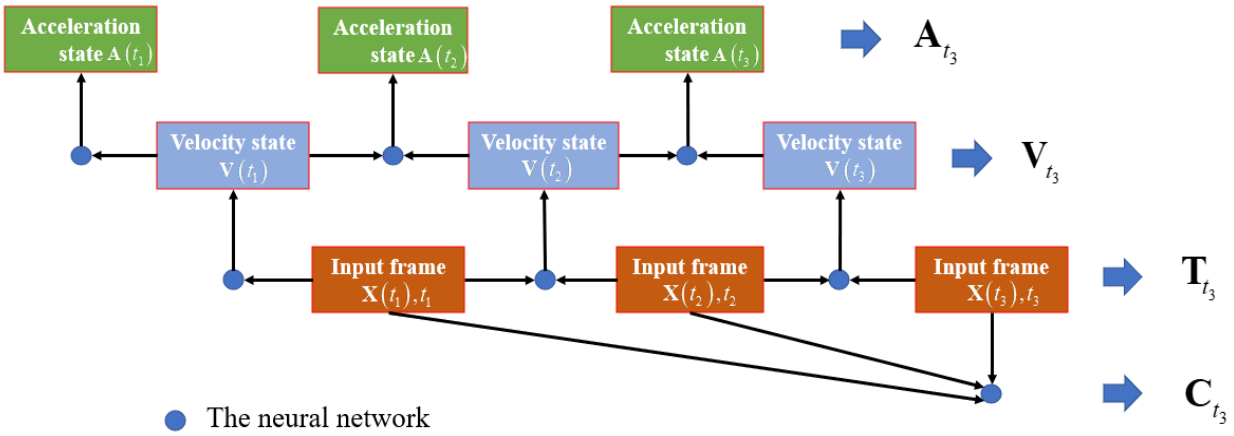


Figure 4.7: The motion characteristics extraction.

- **The Motion-based Decoding Attention (MDA)**

From the perspective of the decoder, the Motion-based Decoding Attention (MDA) mechanism is proposed for making parallel computations on the collection of hidden states instead of the input sequence in the case of the MEA. As described previously, the encoder's recurrent layers take the attention states generated by implementing the MEA on the input series to create the output hidden form for each timestamp. Since the decoder uses the representation states from the encoder to make predictions, the attention mechanism is applied in the set of hidden states. Therefore, the entire attention operations in the encoding-decoding structure are straightforward and follow a common left-right approach. Eventually, the whole step-by-step of applying the MEA is described as follows

Similar to the MEA, the first step is to compute the four keys $\boldsymbol{V}_{t_Q}$, $\boldsymbol{A}_{t_Q}$, $\boldsymbol{T}_{t_Q}$, $\boldsymbol{C}_{t_Q}$ at a specified time step t_Q in the target sequence based on the set of hidden states $\{\boldsymbol{H}(t_i)\}_{i\in\{Q-T,\dots,Q\},i\in N^*}$. In this case, we do not use all available hidden states stretching from the initial time step t_1 to the current moment t_Q at both encoding and decoding parts, we set the fixed set length of T states and change the elements within it using a simple sliding window approach. The reason is that we want to keep the balance between the short-term and the long-term dependencies. Moreover, the computational complexity of the decoder can be maintained at an acceptable level when it comes to long-range sequences.

In the next step, the current hidden state $\boldsymbol{H}(t_Q)$ is first computed using its previous form and the current input data. After that, we can also compute the alignment score $\boldsymbol{S}_{t_Q}$ using (4.9) and the attention state $\boldsymbol{H}^a(t_Q)$ using (4.10). Finally, we get the new form at the time step t_Q by applying (4.11).

In conclusion, the encoder-decoder-based models can use the MEA and MDE modules to strengthen memory ability and handle irregular sampling time series for the SSP task. As a result, the model can be applied with varying spatiotemporal datasets of different applications and longer-range prediction tasks. In addition, we consider inserting these attention mechanisms into particular layers to optimize their performance in multiple-layer systems. Therefore, in this thesis, the models will be tested with different configurations to choose the best built-in attention structure. The following section will introduce the other connection building blocks used to resample the features.

4.2.3.3 Resampling blocks

For the hierarchical feature extraction, the input data is typically processed and transformed into multiple representation states with different levels represented by their shape. In particular, compared to the original input, the higher-level states containing the input's global information usually have smaller channels and larger frame sizes (i.e., the height and width of the state). In contrast, the lower-level states containing the local information will have more channels and small frame sizes. Once the model studies enough features, including local and global knowledge, it can effectively perform the reconstruction and prediction tasks. Consequently, the resulting output will have high accuracy and sharpness.

We insert a resampling block between two adjacent central blocks (i.e., central neural layers) for the SSP task to help the system capture spatial features with different levels. In proposed models, we build the up-sampling block using a pair of convolutional and leakyRELU activation layers. In contrast, the down-sampling block is constructed by a pair of deconvolutional and leakyRELU activation layers. As such, the shape of the output states can be easily adjusted by defining the hyper-parameters of the convolutional and deconvolutional layers (e.g., the kernel size, number of input and output channels, padding, stride, dilation).

4.2.4 Trajectory Gated Recurrent Unit Attention (TrajGRU-Attention)

Based on the fundamental elements of the system design process illustrated above (i.e., the model structure, architecture, and building blocks), the entire model will be constructed with the primary goal of solving the overall SSP task.

First, the TrajGRU-Attention model is designed as a sequence-to-sequence model. To be more specific, we will present the major to minor elements. In terms of the structure, we implement two main parts: the encoder and the decoder, as illustrated in Figure 4.2. Overall, the encoder takes the input sequence to produce the representation states. Then, the decoder uses these states to make the prediction.

Second, the novel architecture is applied by combining the stacked style with the zigzag shortcut, as shown in Figure 4.3. In this architecture, the encoder and decoder can be established by combining multiple neural layers (i.e., building blocks). Moreover, they have the same number of central layers (i.e., central building blocks) because a pair of layers at the same level will directly exchange the states of the same size together. With the zigzag shortcuts connecting the bottom layer's units with the top layer's ones, the proposed system can address the gradient vanishing issue to some degree.

Finally, we select and assemble the encoder and decoder building blocks regarding the model architecture. As mentioned earlier, there are two main groups: the central and connection blocks. This model uses the TrajGRU layers as the primary building blocks to capture and regulate state information. For hierarchical feature extraction, the encoder uses the down-sampling blocks, and the decoder uses the up-sampling blocks. This design makes sense as the decoder layer orders are reversed. As such, the representation states are computed in two directions in the forward propagation: the recurrent computation follows the horizontal direction, and the hierarchical feature extraction follows the vertical direction. In the steep path in the encoding part, the input data is down sampled before going through each central layer, allowing the system to see from the big picture (i.e., the global information) to the detailed features (i.e., the local information). On the decoding side, the top layer uses this local knowledge to construct the lower-level states. As a result, the final predictions are generated using the lowest-level forms in the decoder's bottom layer. Similarly, the backpropagation will also go in two directions. Then, the MEA is used at the encoder and the MDE is used at the decoder to handle the irregularly sampled time series. Moreover, the model can deal with the long-range sequence prediction with these attention mechanisms. However, if the MEA and MDE modules are integrated into all central blocks of the model, there will be a considerable computational complexity with a massive set of trainable parameters. Therefore, we just insert those attention modules on specific layers to balance the overall performance and the complexity. Consequently, we expect the TrajGRU-Attention model can effectively handle the long-term and irregularly sampled spatiotemporal data to complete the overall SSP task. Figure 4.8 illustrates the entire TrajGRU-Attention model.

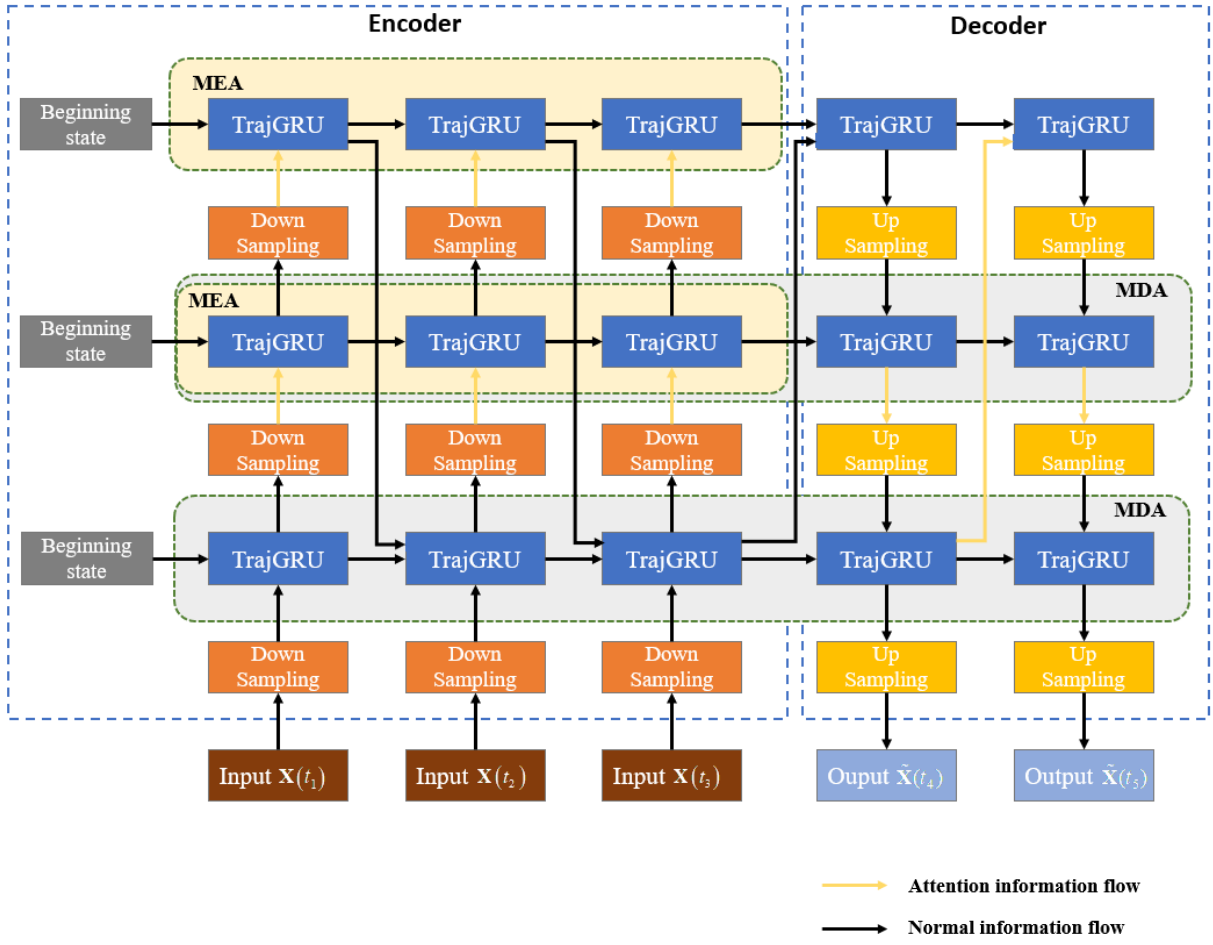


Figure 4.8: Illustration of TrajGRU-Attention model. In the figure, the encoder and the decoder use three central and six resampling blocks (layers). In the encoder, MEA modules are inserted into layer 2 and layer 3. In decoder, while MDA modules are inserted into layer 1 and layer 2.

In addition, the attention states generated by the MEA and the MDE will go in the vertical direction; thus, they will be the input series of the next layer. In particular, the attention states generated in the decoder's bottom layer will be transferred through the zigzag shortcuts. As a result, central block units can communicate with each other effectively as their distances are significantly reduced by the zigzag shortcuts and the attention mechanisms.

4.2.5 Trajectory Gated Recurrent Unit Ordinary Differential Equation (TrajGRU-ODE)

Although the TrajGRU-Attention can handle the irregularly sampled sequence data, employing multiple MEA and MDA modules might lead to considerable computational complexity, especially when we build the model with a high number of stacked layers. Therefore, the TrajGRU-ODE model is proposed with the main objective of processing irregularly sampled time series and predicting outcomes in the continuous domain.

From the model structure and architecture perspective, the TrajGRU-ODE is designed the same way as the TrajGRU-Attention. The main difference between these two models is that the TrajGRU-ODE blocks replace the combination of TrajGRU blocks and MDE modules as the center blocks in the decoder. Since the TrajGRU-ODE blocks themselves can generate continuous hidden states and reconstruct the outcomes at the expected time steps, the MEA modules are implemented at the encoder to generate the initial representation states. Figure 4.9 illustrates the entire TrajGRU-ODE model.

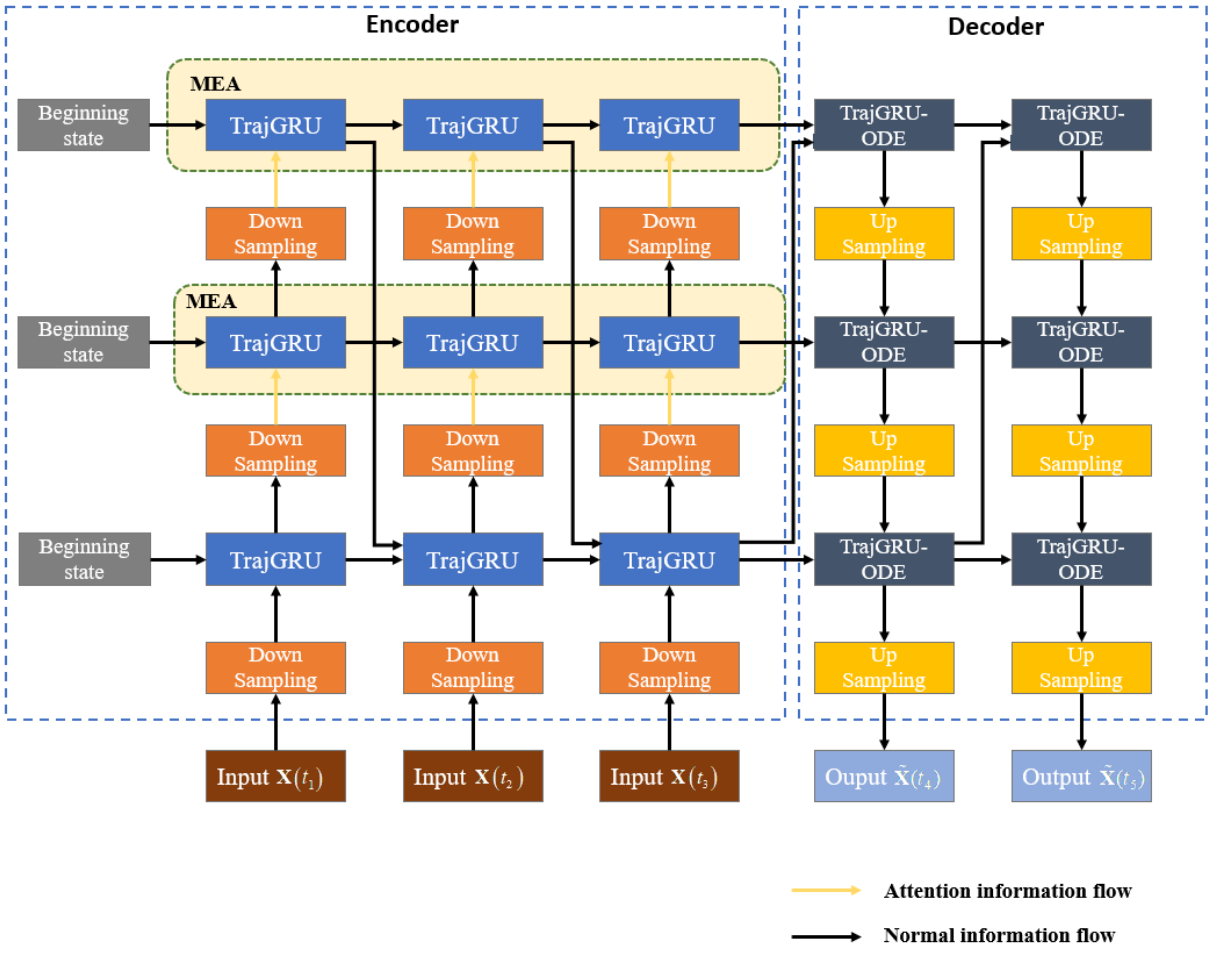


Figure 4.9: Illustration of TrajGRU-ODE model. In the figure, the encoder and the decoder use three central and six resampling blocks (layers). In the encoder, MEA modules are inserted into layer 2 and layer 3.

4.2.6 Trajectory Gated Recurrent Unit Attention Ordinary Differential Equation (TrajGRU-Attention-ODE)

Theoretically, the TrajGRU-ODE model can generate continuous-time sequences; however, this model may not perform properly when it comes to long-term prediction tasks. Thus, we further combine the TrajGRU-Attention and TrajGRU-ODE models to design the TrajGRU-Attention-ODE. In terms of the structure and architecture, we construct this model in the same way as the TrajGRU-ODE. For the long-term prediction problem, the TrajGRU-Attention-ODE's decoder is also equipped with the MDA modules like the TrajGRU-Attention. Figure 4.10 illustrates the entire TrajGRU-Attention-ODE model.

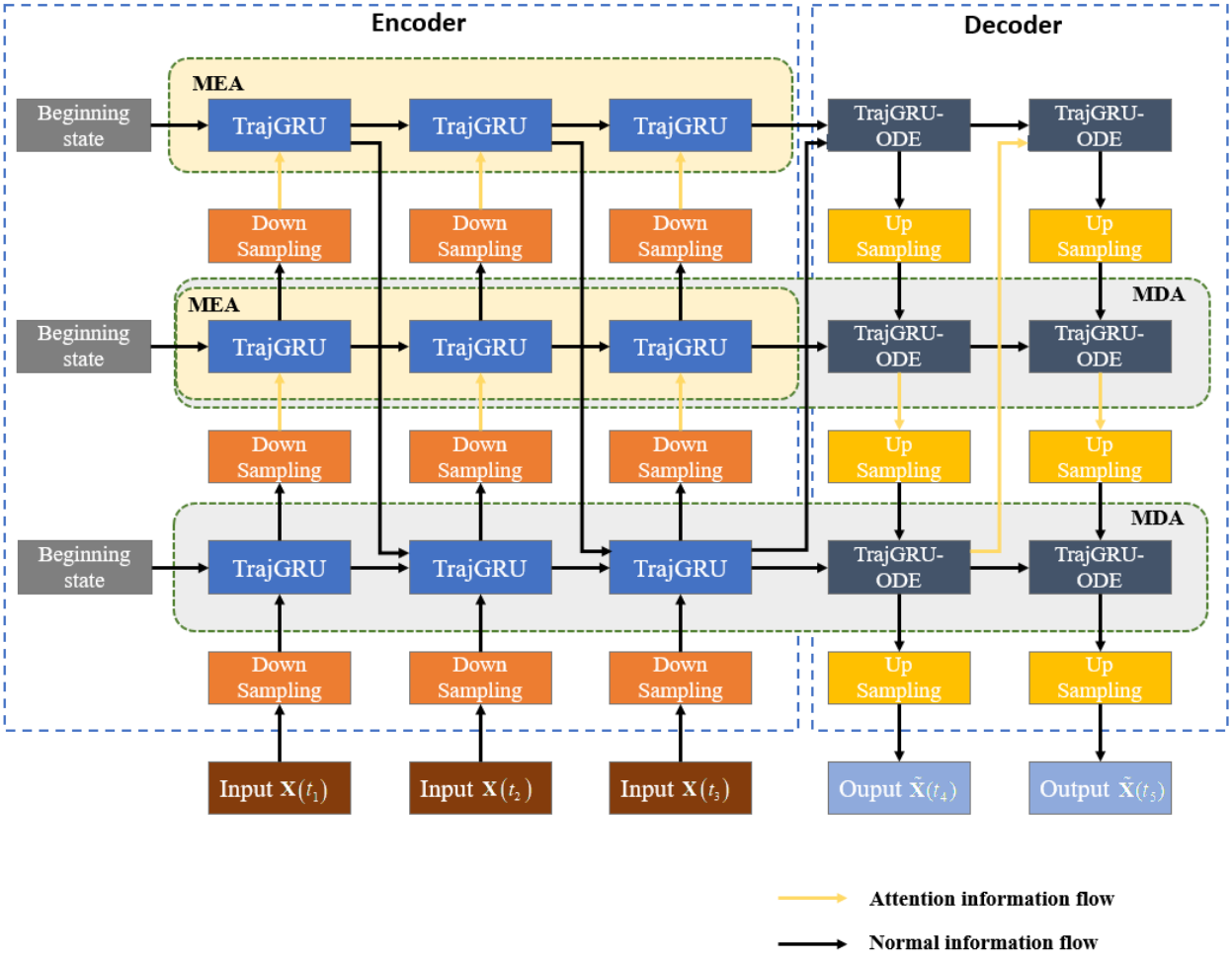


Figure 4.10: Illustration of TrajGRU-Attention-ODE model. In the figure, the encoder and decoder use three central and six resampling blocks (layers). In the encoder, MEA modules are inserted into layer 2, 3. In decoder, MDA modules are inserted into layer 1, 2.

Chapter 5: Experiments

In this chapter, the experiments of proposed models with four spatiotemporal datasets: the MovingMNIST, MovingMNIST++, HKO-7 and KTH Action, will be presented. The models are implemented in Python with the help of the Pytorch framework. In addition, we run all the simulations on a computer with two GPUs, NVIDIA GeForce RTX 3090.

First, the baseline methods used in this work to compare with the proposed models, will be introduced. Then, we will present the model configuration including the hyperparameters and loss function. Finally, the models will be trained, validated and tested using the given configuration.

5.1 The baseline

In this thesis, three models are implemented as the baseline models: the ConvGRU, the TrajGRU, and Vid-ODE. As mentioned in the introduction chapter, both regular and irregular time-sampling assumptions will be examined on the datasets; thus, ConvGRU and TrajGRU models cannot use in some cases. For a fair comparison, we use some common hyperparameters of the ConvGRU, TrajGRU and the proposed models. With the Vid-ODE, we implement the model with a relatively similar number of learnable parameters compared with the other two baselines. In addition, the Vid-ODE model parameters are described in [14], and the configurations of the ConvGRU, TrajGRU are illustrated in Appendix A.

5.2 The proposed method

5.2.1 Model configuration

In this thesis, we will consider two groups of configurations: the first group implemented with the MovingMNIST and MovingMNIST++ and the second one, with the real-world datasets, the HKO-7 and KTH Action. The main reason lies in each dataset's general spatiotemporal properties, which we will present later in this section. Tables 5.1 and 5.2 describe the three proposed models' overall configurations and important hyperparameters.

As mentioned earlier, we do not want to employ attention mechanisms for all the central layers of the proposed models because of the computational complexity issue. In tables 5.1 and 5.2, the proposed systems use the sets of three central layers (i.e., the central blocks) at the encoder and decoder. In the encoding side of all three models, two MEA modules are implemented at layers 2 and 3. In the decoding side of the TrajGRU-Attention and TrajGRU-Attention-ODE models, two MDE modules are implemented at layers 2 and 1. To choose these attention mechanism settings, the TrajGRU-Attention model is tuned using the MovingMNIST data in the scenario where the numbers of attention modules vary and the other hyperparameters are constant. The detailed information and result of this tuning process are described in Appendix B. In addition, the training processes of all models used the early stopping mechanism to obtain the final results.

Table 5.1 Model configurations of the proposed methods used with the MovingMNIST and MovingMNIST++

<table>
<tr><th colspan="3">Model</th></tr>
<tr><th>TrajGRU-Attention</th><th>TrajGRU – Attention - ODE</th><th>TrajGRU – ODE</th></tr>
<tr><th colspan="3">Common hyperparameters</th></tr>
<tr><td colspan="3">Common central building blocks
3 TrajGRU blocks at the encoder: the learnable parameters are the same as the baseline TrajGRU model, as illustrated in Table A.1
Common connection building blocks
3 down-sampling blocks at the encoder and 4 up-sampling blocks at the decoder:
- Learnable weights of down-sampling blocks of the encoder:
$\big((1 \times 64 \times 3 \times 3), (64 \times 96 \times 3 \times 3), (96 \times 128 \times 3 \times 3)\big)$
- Learnable weights of up-sampling blocks of the decoder:
$\big((128 \times 96 \times 4 \times 4), (96 \times 64 \times 4 \times 4), (64 \times 32 \times 3 \times 3), (32 \times 1 \times 1 \times 1)\big)$
2 MEA blocks inserted at central layer 2 and 3 at the encoder:
- To compute the motion characteristics: all convolutional layers have a kernel size of (1×1) (4 convolutional layers for each module) and the numbers of input, output filters: layer 2 $(96, 96)$, layer 3 $(128, 128)$); and numbers of input, output filters for the linear layers (1 linear layer for each module): layer 2 $(384, 96)$, layer 3 $(512, 128)$).
Zigzag connection shortcut
3 convolutional layers in each part (i.e., the encoder or decoder):
- At the encoder: $\boldsymbol{W}_{upsampling}(128 \times 64 \times 4 \times 4)$, $\boldsymbol{W}_{h}(64 \times 64 \times 3 \times 3)$,$\boldsymbol{W}_{zz}(64 \times 64 \times 3 \times 3)$
- At the decoder: $\boldsymbol{W}_{downsampling}(64 \times 128 \times 4 \times 4)$, $\boldsymbol{W}_{h}(128 \times 128 \times 3 \times 3)$,$\boldsymbol{W}_{zz}(128 \times 128 \times 3 \times 3)$
Common hyperparameters for the training process
- Batch size: 4
- Maximum number of epochs: 25
- Number of iterations per epoch: 8000
- Optimizer: Adam with learning rate = 10^{-4} and momentum = 0.9</td></tr>
<tr><th colspan="3">Central building blocks</th></tr>
<tr><td>3 TrajGRU blocks at the decoder: the learnable parameters are the same as the baseline TrajGRU model, as illustrated in Table 1</td><td colspan="2">3 TrajGRU-ODE blocks at the decoder. In the block structure:
- The standard TrajGRU gates ($\boldsymbol{W}_{x}$, $\boldsymbol{W}_{h}$): the learnable parameters are the same as the baseline TrajGRU model.
- The extended TrajGRU gates ($\boldsymbol{W}_{hode}$, $\boldsymbol{W}_{dode}$):
• layer 1 $\big((64 \times 192 \times 1 \times 1), (64 \times 192 \times 3 \times 3)\big)$,
• layer 2 $\big((96 \times 288 \times 1 \times 1), (96 \times 288 \times 3 \times 3)\big)$,
• layer 3 $\big((128 \times 384 \times 1 \times 1), (128 \times 384 \times 3 \times 3)\big)$.</td></tr>
</table>

	- The ODE solver: 3 layers (i.e., 3 tanh activation layers and 4 convolutional layers with the common kernel size of (3×3) and the numbers of input and output filters are depended on the layer level). The ODE method is Euler.	
Connection building blocks		
2 MDA blocks inserted at central layer 2 and 1 of the decoder: - To compute the motion characteristics: all convolutional layers have the kernel size of (1×1) and the numbers of input, output filters: layer 2 $(96, 96)$, layer 1 $(64, 64)$); the numbers of input, output filters for the linear layers: layer 2 $(384, 96)$, layer 3 $(256, 64)$).		With the TrajGRU-ODE, the MDA blocks are not used in the decoder.
Total number of parameters		
7258509	8844685	9857837

Table 5.2 Model configurations of the proposed methods used with the HKO-7 and KTH Action

Model		
TrajGRU-Attention	**TrajGRU – Attention - ODE**	**TrajGRU – ODE**
Common hyperparameters		
Common central building blocks 3 TrajGRU blocks at the encoder: the learnable parameters are the same as the baseline TrajGRU model, as illustrated in Table A.2 **Common connection building blocks** 3 down-sampling blocks at the encoder and 4 up-sampling blocks at the decoder: - Learnable weights of down-sampling blocks of the encoder: $\big((1 \times 96 \times 3 \times 3), (96 \times 96 \times 3 \times 3), (96 \times 128 \times 3 \times 3)\big)$ - Learnable weights of up-sampling blocks of the decoder: $\big((128 \times 96 \times 4 \times 4), (96 \times 96 \times 4 \times 4), (96 \times 32 \times 3 \times 3), (32 \times 1 \times 1 \times 1)\big)$ 2 MEA blocks inserted at central layer 2 and 3 at the encoder: - For computing the motion characteristics: all convolutional layers have a kernel size of (1×1) (4 convolutional layers for each module) and the numbers of input, output filters: layer 2 $(96, 96)$, layer 3 $(128, 128)$); and numbers of input, output filters for the linear layers (1 linear layer for each module): layer 2 $(384, 96)$, layer 3 $(512, 128)$). **Zigzag connection shortcut** 3 convolutional layers in each part (i.e., the encoder or decoder): - At the encoder: $\boldsymbol{W}_{upsampling}(128 \times 96 \times 4 \times 4)$, $\boldsymbol{W}_{h}(96 \times 96 \times 3 \times 3)$,$\boldsymbol{W}_{zz}(96 \times 96 \times 3 \times 3)$ - At the decoder: $\boldsymbol{W}_{downsampling}(96 \times 128 \times 4 \times 4)$, $\boldsymbol{W}_{h}(128 \times 128 \times 3 \times 3)$,$\boldsymbol{W}_{zz}(128 \times 128 \times 3 \times 3)$ **Common hyperparameters for the training process** - Batch size: 4		

<table>
<tr><td colspan="3">- Maximum number of epochs: 25
- Number of iterations per epoch: 8000
- Optimizer: Adam with learning rate = 10^{-4} and momentum = 0.5</td></tr>
<tr><td colspan="3">Central building blocks</td></tr>
<tr><td>3 TrajGRU blocks at the decoder: the learnable parameters are the same as the baseline TrajGRU model, as illustrated in Table 1</td><td colspan="2">3 TrajGRU-ODE blocks at the decoder. In the block structure:
- The standard TrajGRU gates ($\boldsymbol{W}_x, \boldsymbol{W}_h$): the learnable parameters are the same as the baseline TrajGRU model.
- The extended TrajGRU gates ($\boldsymbol{W}_{hode}, \boldsymbol{W}_{dode}$):
• layer 1 $\big((96 \times 288 \times 1 \times 1), (96 \times 288 \times 3 \times 3)\big)$,
• layer 2 $\big((96 \times 288 \times 1 \times 1), (96 \times 288 \times 3 \times 3)\big)$,
• layer 3 $\big((128 \times 384 \times 1 \times 1), (128 \times 384 \times 3 \times 3)\big)$.
- The ODE solver: 3 layers (i.e., 3 tanh activation layers and 4 convolutional layers with the common kernel size of (3×3) and the numbers of input and output filters are depended on the layer level). The ODE method is Euler.</td></tr>
<tr><td colspan="3">Connection building blocks</td></tr>
<tr><td colspan="2">2 MDA blocks inserted at central layer 2 and 1 of the decoder:
- For computing the motion characteristics: all convolutional layers have a kernel size of (1×1) and the numbers of input, output filters are: layer 2 $(96, 96)$, layer 1 $(96, 96)$); and numbers of input, output filters for linear layers: layer 2 $(384, 96)$, layer 1 $(384, 96)$).</td><td>With the TrajGRU-ODE, the MDA blocks are not used in the decoder.</td></tr>
<tr><td colspan="3">Total number of parameters</td></tr>
<tr><td>9336677</td><td>11181413</td><td>12583973</td></tr>
</table>

5.2.2 The loss function

Generally, the MAE and the MSE are the commonly used loss functions for sequence prediction tasks. However, many prior studies have mentioned the drawbacks when using those error functions for the training process. Since they compute the global errors between the output and ground truth during the process, the system might ignore some local errors. Consequently, the sharpness of the resulting images decreased significantly, especially with long-term sequence prediction problems. Therefore, some related research proposed other loss functions instead of using only a simple MAE or MSE [100]. Therefore, we propose the combination of the MAE, MSE, and SSIM as the loss function to guide the training process. With the SSIM metric, we expect the system to take into account the similarity between two sequences by modelling the image distortion; thus, we can somewhat alleviate the blurry effect. The main formula of proposed loss function can be derived as

$$\begin{cases} L_{MSE} = \frac{\sum_{i=1}^{K}(\boldsymbol{X}_i - \widetilde{\boldsymbol{X}}_i)^2}{K} \\ L_{MAE} = \frac{\sum_{i=1}^{K}|\boldsymbol{X}_i - \widetilde{\boldsymbol{X}}_i|}{K} \\ L_{SSIM} = 1 - SSIM \\ L = aL_{MSE} + bL_{MAE} + cL_{SSIM} \end{cases}, \tag{5.1}$$

where $\boldsymbol{X}_i \in \mathbb{R}^{C\times H\times W}, \widetilde{\boldsymbol{X}}_i \in \mathbb{R}^{C\times H\times W}$ represent the target and output frames, respectively; $a = 1, b = 1, c = 0.05$ denote the weights of the three elements of the loss function. The weights of the MSE and MAE are equal and more significant than the SSIM because MSE and MAE are supported to be the main elements manipulating the global errors, and the SSIM is the supportive factor impacting the image quality. In addition, the weight c is selected by using the "trial and error" method (i.e., it is set from 0.01 to 0.1). The comparison between different loss functions is shown in Table B.2

5.3 Experiments with the MovingMNIST

5.3.1 Dataset

First, we will start with a relatively simple video dataset, the MovingMNIST. Essentially, this dataset illustrates the movements of the handwriting MNIST digits in 64×64 frames.

For data preprocessing, we generate sequences consisting of two random MNIST digits. For the overall SSP task, we divide each series into two subsequences (i.e., the input and the target) and then apply the sampling methods in each subsequence. As shown in Figure 1.2, we define four collections of timestamps representing four different sampling assumptions in Table 5.3. As a result, the input is ten frames long, and the target is ten frames long. When subsequences are regularly sampled, the training set contains 10000 sequences, the validation set contains 2000 sequences, and the testing set contains 5000 sequences. Otherwise, the training set contains 32000 sequences, the validation set contains 2000 sequences, and the testing set contains 5000 sequences.

Table 5.3 The configuration of four sampling methods

Sampling method	**Regular-sampled input and target**	**Irregular-sampled input, regular-sampled target**	**Regular-sampled input, irregular-sampled target**	**Irregular-sampled input and target**
Set size	1	100	100	100
Input interval	[1, 10]	[1, 20]	[1, 10]	[1, 15]
Target interval	[11, 20]	[21, 30]	[11, 30]	[16, 30]

In Table 5.3, the set size presents the number of distinct collections of time steps, and the input and target time intervals denote the minimum and maximum time steps for each sampling method.

5.3.2 Results

In this section, we will implement DL sequence-to-sequence models using the MovingMNIST dataset for four sampling assumptions. In particular, the ConvGRU, the TrajGRU, the Vid-ODE

and our three proposed models are implemented with the first assumption (i.e., regular-sampled input and target). For the second sampling assumption (i.e., irregular-sampled input and regular-sampled target), we implement the TrajGRU and Vid-ODE to compare with our proposed model. Finally, we implement the Vid-ODE and our three models in the other two time-sampling scenarios (i.e., regular-sampled input and irregular-sampled target; irregular-sampled input and target).

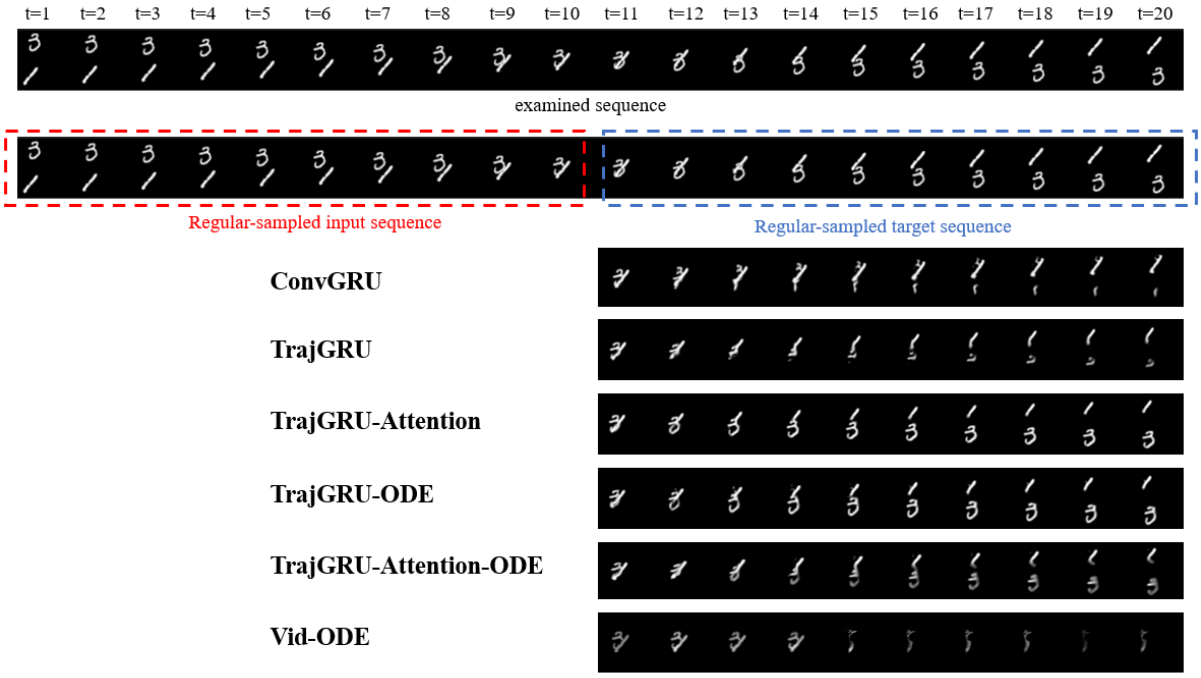


Figure 5.1. Prediction results using the MovingMNIST with regular-sampled input and target sequences.

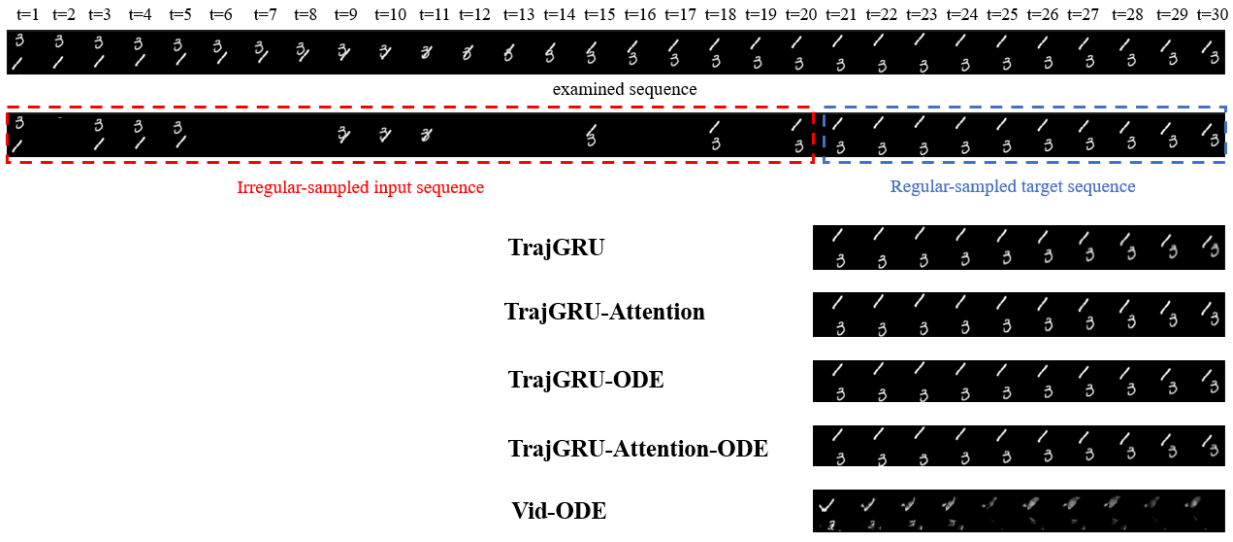


Figure 5.2. Prediction results using the MovingMNIST with irregular-sampled input and regular-sampled target sequences.

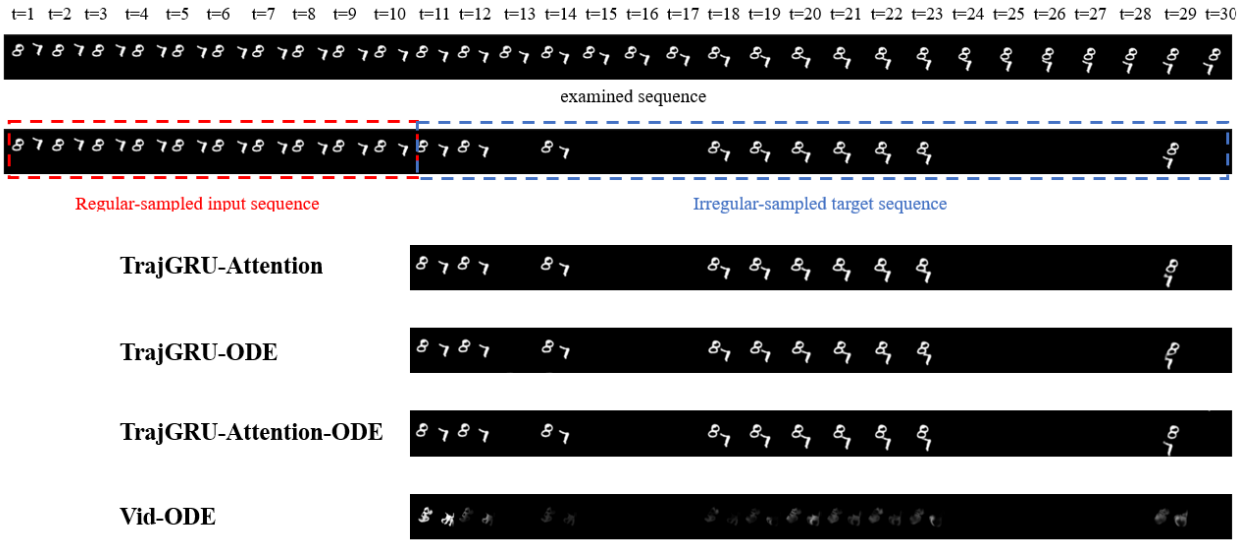


Figure 5.3. Prediction results using the MovingMNIST with regular-sampled input and irregular-sampled target sequences.

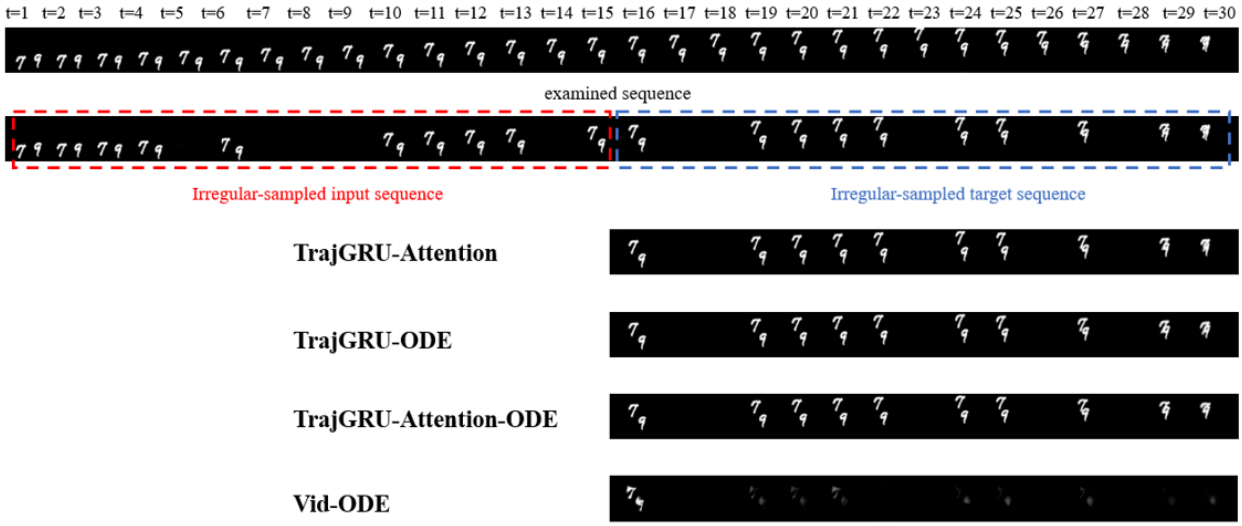


Figure 5.4. Prediction results using the MovingMNIST with irregular-sampled input and target sequences.

Some prediction results for the four sampling methods are shown in Figures 5.1, 5.2, 5.3, and 5.4, respectively. To test the models' performance, we use four evaluation metrics: the MSE, MAE, SSIM and PSNR. Table 5.4 presents their values for the examined models using the MovingMNIST dataset.

Table 5.4 Values of evaluation metrics for the models with the MovingMNIST dataset

Sampling assumption: regular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
ConvGRU	80.724	178.22	92.383	21.352
TrajGRU	58. 407	141. 57	94.321	22.943
Vid-ODE	319.31	507.19	70.789	15.014
TrajGRU-Attention	39.621	110.03	96.059	24.955
TrajGRU-ODE	41.552	115.33	95.778	24.614
TrajGRU-Attention-ODE	56.675	141.50	94.263	23.041
Sampling assumption: irregular-sampled input and regular-sampled target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
TrajGRU	97.155	214.89	90.003	19.617
Vid-ODE	344.38	512.66	68.432	14.571
TrajGRU-Attention	56.274	143.18	94.202	23.117
TrajGRU-ODE	60.283	153.94	93.715	22.767
TrajGRU-Attention-ODE	64.399	162.67	93.137	22.436
Sampling assumption: regular-sampled input and irregular-sampled target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
Vid-ODE	397.76	523.86	67.671	14.045
TrajGRU-Attention	89.525	201.29	90.763	20.821
TrajGRU-ODE	101.93	220.15	89.536	20.235
TrajGRU-Attention-ODE	86.513	194.27	91.121	21.007
Sampling assumption: irregular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
Vid-ODE	356.27	503.87	69.672	14.526
TrajGRU-Attention	77.288	181.88	91.893	21.541
TrajGRU-ODE	91.014	203.01	90.583	20.744
TrajGRU-Attention-ODE	76.276	180.12	91.952	21.618

5.3.3 Comments

Overall, the results show that all three proposed models provided better performance than the ones of the baselines when applied with the MovingMNIST dataset. In particular, the TrajGRU-Attention achieved the best performance in the first (i.e., regular-sampled input and target) and second sampling assumptions (i.e., irregular-sampled input and regular-sampled target). With the other time-sampling methods, the TrajGRU-Attention-ODE generated the predictions with the best performance.

5.4 Experiments with the MovingMNIST++

5.4.1 Dataset

Compared to the MovingMNIST dataset, the MovingMNIST++ dataset added more complicated motion patterns to the MNIST digits, such as random rotations, scale changes, and illumination changes [1, 2]. Similarly to previous case, we use the same preprocess and sampling methods to generate the input and target sequences for the training, validation, and testing sets. Table 5.5 compares the MovingMNIST and MovingMNIST++ used in the thesis.

Table 5.5 Comparison between the MovingMNIST and MovingMNIST++

Dataset	**MovingMNIST**	**MovingMNIST++**
The frame size	64×64	64×64
The number of digits	2	3
The input sequence length	10	10
The target sequence length	10	10
Motion patterns	Motions with constant speed	Motions with constant speed, rotation, scaling, illumination.
Angle of rotation	0°	$[-30^{\circ}, 30^{\circ}]$
Scaling range	[1.0,1.0]	[0.9,1.1]
Illumination range	[1.0,1.0]	[0.6,1.0]

From the above table, performing the SSP with the MovingMIST++ seems to be more challenging than with the MovingMNIST.

5.4.2 Results

Similar to the case of the MovingMNIST, the same settings are applied with the MovingMNIST++. Consequently, some prediction results of the models for the four sampling methods are shown in Figures 5.5, 5.6, 5.7, and 5.8, respectively. Table 5.6 provides the metric values of the examined models with the MovingMNIST++ dataset.

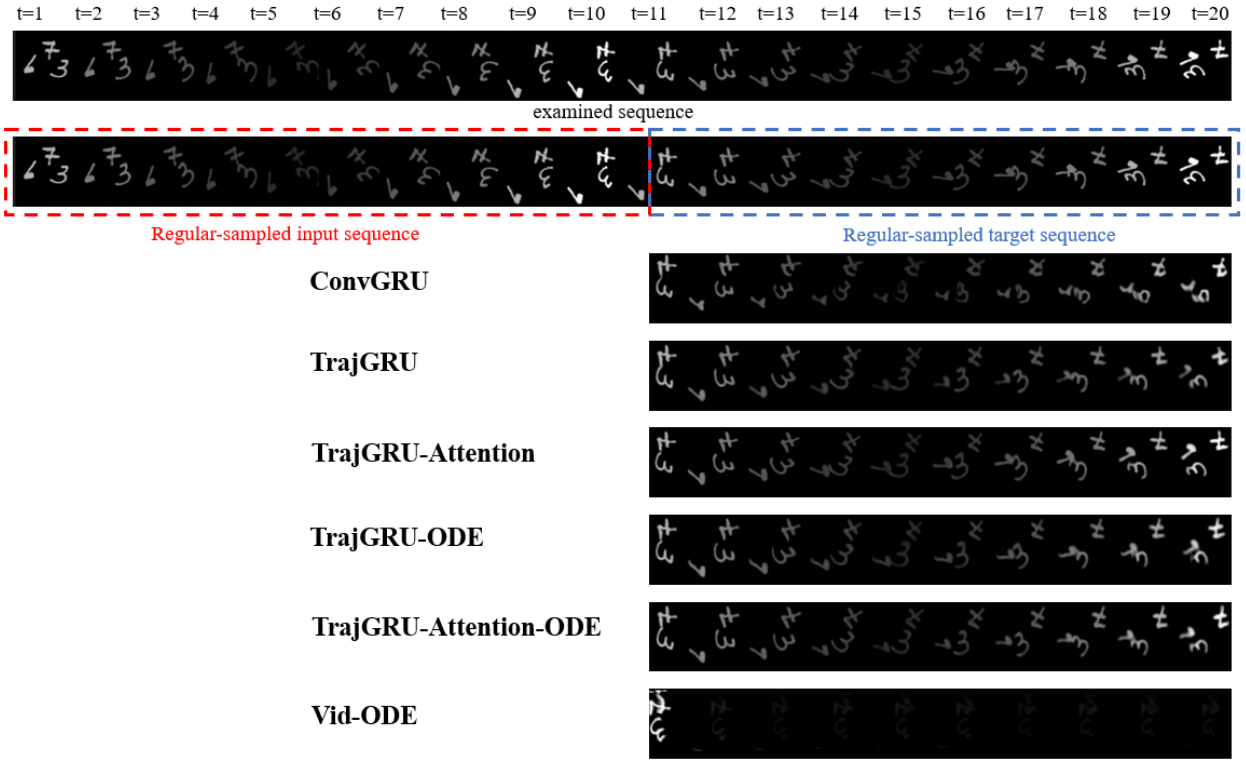


Figure 5.5. Prediction results using the MovingMNIST++ with regular-sampled input and target sequences.

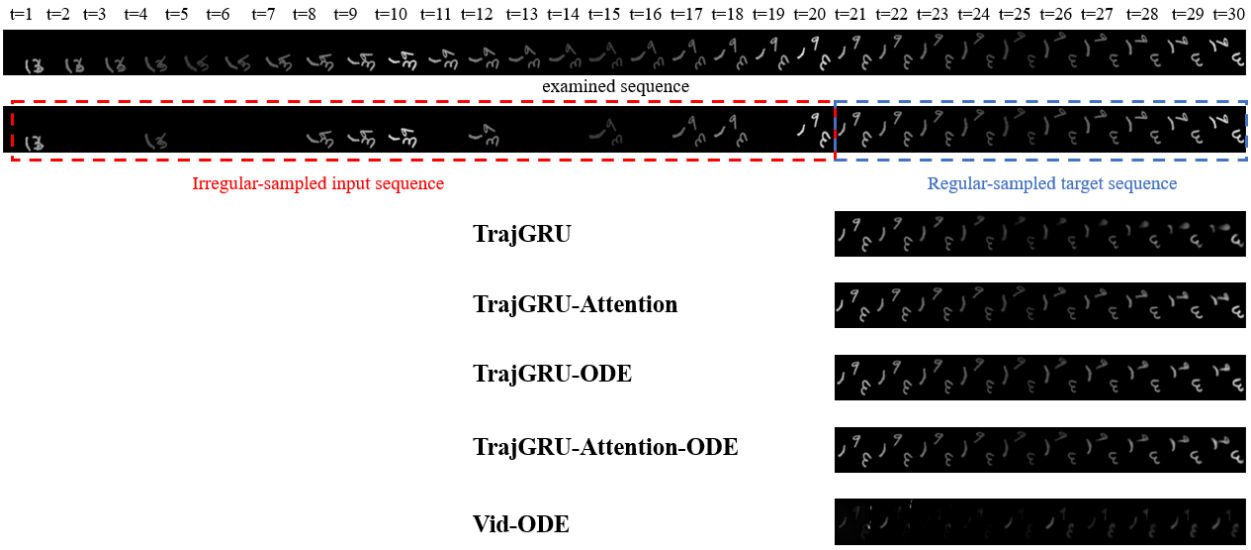


Figure 5.6. Prediction results using the MovingMNIST++ with irregular-sampled inputs and regular-sampled target sequences.

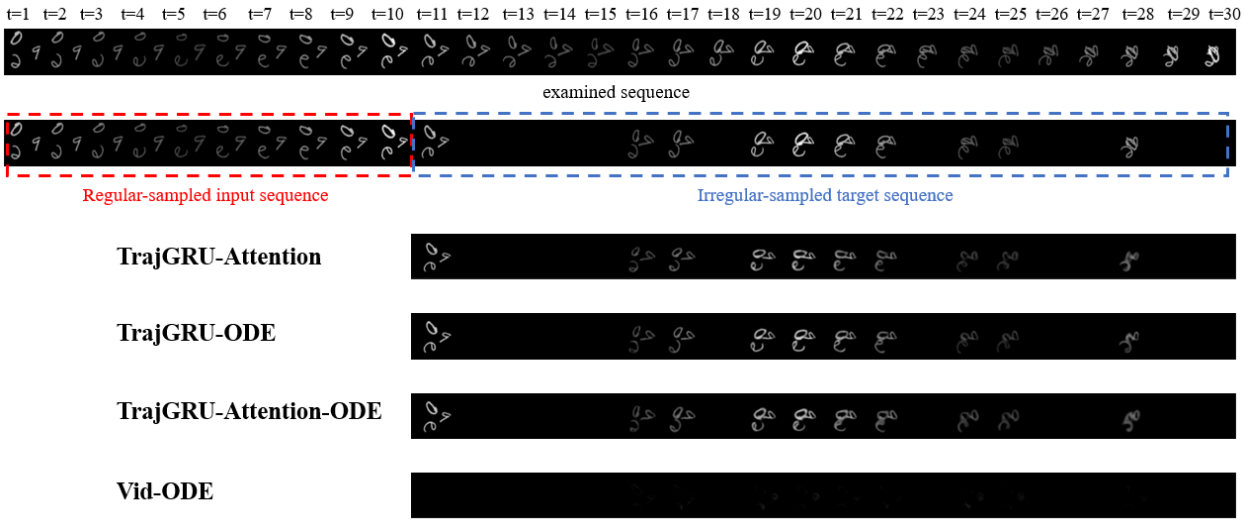


Figure 5.7. Prediction results using the MovingMNIST++ with regular-sampled input and irregular-sampled target sequences.

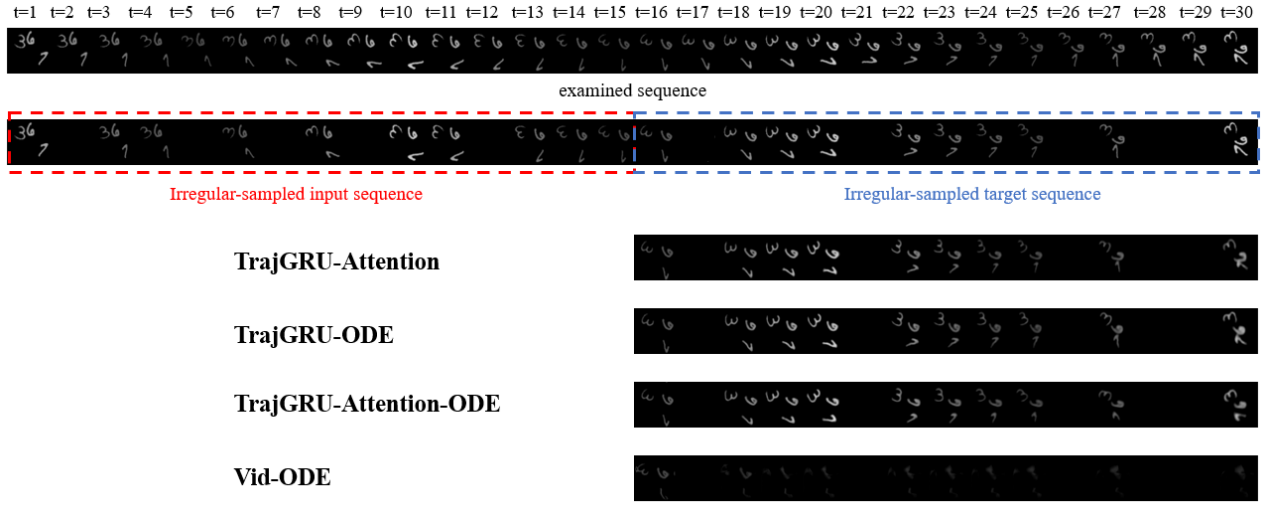


Figure 5.8. Prediction results using the MovingMNIST++ with irregular-sampled input and target sequences.

Table 5.6 Values of evaluation metrics for the models with the MovingMNIST++ dataset

Sampling assumption: regular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
ConvGRU	107.88	298.11	75.444	19.836
TrajGRU	92.409	262.75	78.251	20.135
Vid-ODE	369.32	494.41	72.123	14.366
TrajGRU-Attention	78.356	226.17	82.346	21.143
TrajGRU-ODE	90.739	260.53	78.507	20.529
TrajGRU-Attention-ODE	87.851	251.06	80.107	20.749
Sampling assumption: irregular-sampled input and regular-sampled target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
TrajGRU	113.10	307.85	73.162	19.889
Vid-ODE	367.12	497.18	63.123	14.394
TrajGRU-Attention	86.991	255.45	79.562	20.832
TrajGRU-ODE	93.767	274.64	77.573	20.468
TrajGRU-Attention-ODE	100.34	281.01	76.299	20.166
Sampling assumption: regular-sampled input and irregular-sampled target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
Vid-ODE	368.53	487.75	63.034	14.377
TrajGRU-Attention	143.27	355.28	74.501	18.886
TrajGRU-ODE	142.26	351.41	74.759	18.975
TrajGRU-Attention-ODE	153.35	373.01	71.528	18.453
Sampling assumption: irregular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
Vid-ODE	378.31	500.56	61.934	14.263
TrajGRU-Attention	112.15	307.61	75.718	19.727
TrajGRU-ODE	111.33	305.14	76.014	19.941
TrajGRU-Attention-ODE	131.65	336.89	72.751	19.014

5.4.3 Comments

Overall, the results show that all three proposed models provided better performance than the ones of baseline models. In particular, the TrajGRU-Attention achieved the best performance in all sampling methods, and the Vid-ODE could not complete the overall SSP task with the MovingMNIST++. The main reasons leading to the limitations of Vid-ODE's performance are the time sampling assumptions and the properties of datasets (e.g., MovingMNIST and MovingMNIST++). In the paper of the Vid-ODE model, the authors used the concepts of frame rate that can be fixed at the encoding and decoding parts (e.g., the encoder's frame rate is 1, and the decoder's frame rate is 0.5). In this work, the changing rate of the MovingMNIST, MovingMNIST++ is relatively higher, and the time intervals between adjacent frames are arbitrary (i.e., sequences in training, validation, and testing sets are irregularly sampled).

5.5 Experiments with the HKO-7

5.5.1 Dataset

The HKO-7 is a real-world weather radar dataset. In particular, it contains radar echo data from 2009 to 2015 collected by HKO. The radar reflectivity images, which have a resolution of 480×480 pixels, are taken from an altitude of 2 km and cover a 512 km × 515 km area centered in Hong Kong. The data are recorded every 6 minutes (the minimum time gap is 6-minute), and hence there are 240 frames per day [1, 2]. In this work, we downsize the HKO-7 dataset to a resolution of 80×80 pixels because of the hardware memory limitations.

For data preprocessing, we choose some rainy days based on the rain barrel information to generate the training, validation, and testing sets. To be more specific, 812 days are used for training, 50 days are used for validation, and 131 days are used for testing [1, 2]. Similar to the previous datasets, we define the input length as ten frames long and the target length as ten frames long. For the SSP task, two collections of timesteps representing two main sampling assumptions: the regular-sampled input and target sequences, irregular-sampled input, and target sequences, are used, as shown in Table 5.3.

5.5.2 Results

In this section, we will implement DL sequence-to-sequence models using the HKO-7 dataset for two sampling assumptions. In particular, we implement the TrajGRU, Vid-ODE and our three proposed models for the first sampling assumption (i.e., regular sampled input and target). We implement the Vid-ODE and our three models to compare the performance for the other sampling assumption (i.e., irregular-sampled input and target). Some prediction results for the two sampling methods are shown in Figures 5.9 and 5.10, respectively.

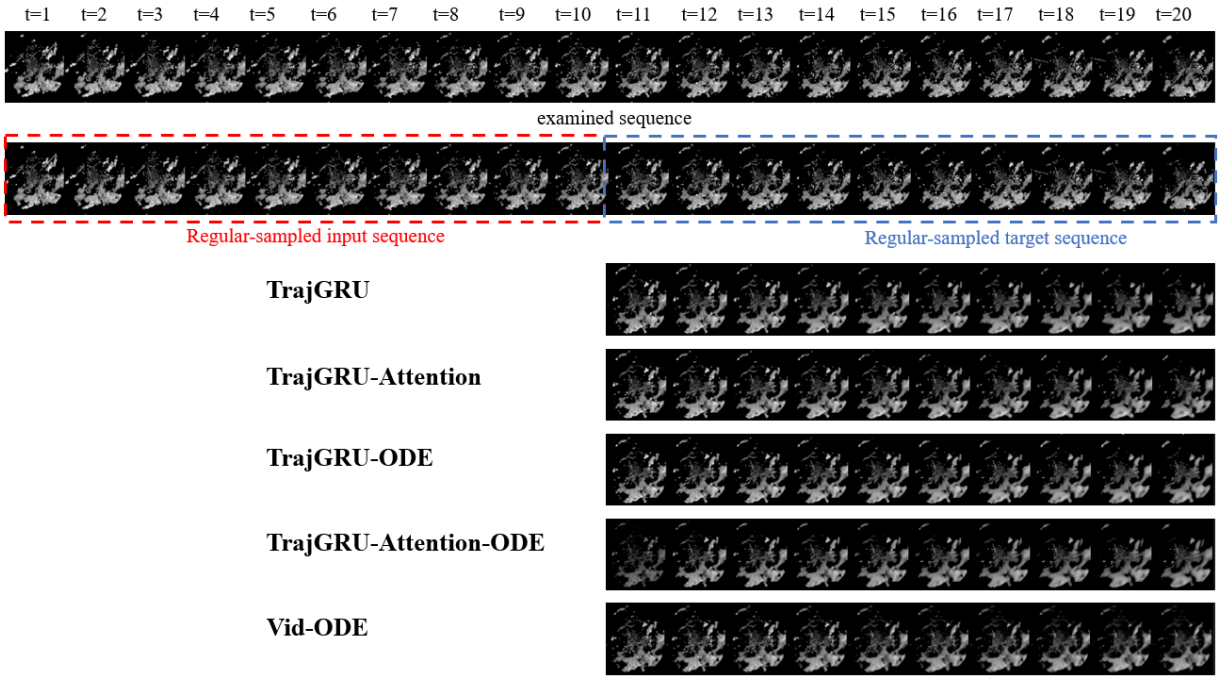


Figure 5.9. Prediction results using the HKO-7 with regular-sampled input and target sequences.

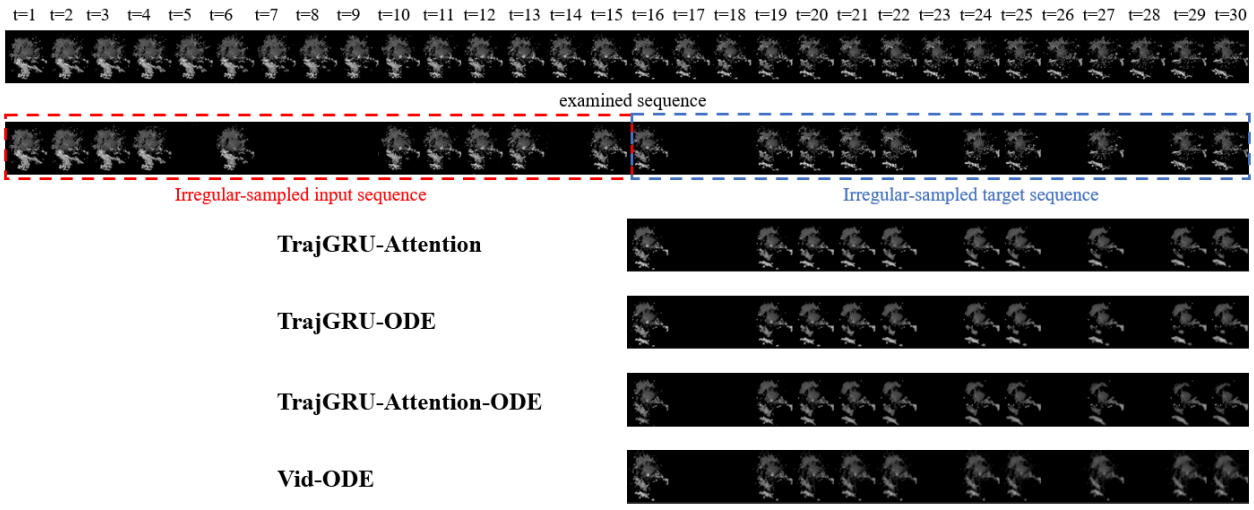


Figure 5.10. Prediction results using the HKO-7 with irregular-sampled input and target sequences.

Table 5.7 Values of evaluation metrics for the models with the HKO-7 dataset

Sampling assumption: regular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
TrajGRU	122.91	390.19	63.341	19.826
Vid-ODE	145.71	493.88	57.517	19.111
TrajGRU-Attention	120.61	386.02	65.853	19.892
TrajGRU-ODE	113.70	375.21	65.901	20.112
TrajGRU-Attention-ODE	125.40	405.78	62.571	19.682
Sampling assumption: irregular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
Vid-ODE	199.61	609.74	50.025	17.899
TrajGRU-Attention	146.58	440.95	61.962	19.151
TrajGRU-ODE	148.05	441.47	60.611	19.121
TrajGRU-Attention-ODE	150.52	459.71	59.685	19.055

5.5.3 Comments

Overall, the proposed models show some acceptable results compared to the baseline models. In the case of the irregular-sampled assumption, our proposed models can provide better performance than the ones of Vid-ODE model. However, the improvements are not clear, and the sharpness of some predictions is not high, especially with the TrajGRU-Attention-ODE model.

5.6 Experiments with the KTH Action

5.6.1 Dataset

KTH Action is the video database describing six types of human actions: walking, jogging, running, boxing, hand waving, and clapping [101]. For data preprocessing, we use 255 videos for training and 144 videos for validating and testing. Similarly to the case of the HKO-7, the input and target lengths are ten, and the two time-sampling methods are also used for the SSP task.

5.6.2 Results

In this section, DL sequence-to-sequence models using the KTH Action dataset will be implemented in two sampling assumptions. In particular, we implement the TrajGRU, Vid-ODE and our three proposed models with the first sampling assumption (i.e., regular-sampled input and target). We implement the Vid-ODE and our three models to compare the performance for the other sampling assumption (i.e., irregular-sampled input and target). Some prediction results of the four sampling methods are shown in Figures 5.11 and 5.12, respectively.

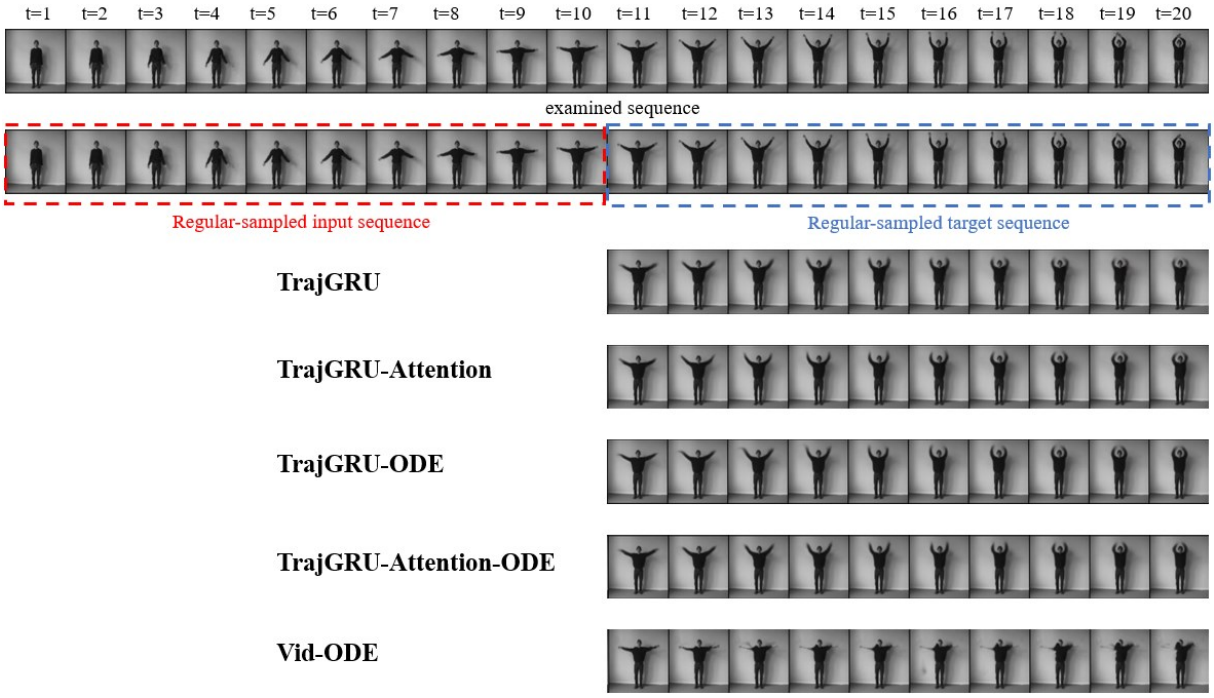


Figure 5.11. Prediction results using the KTH Action with regular sampled inputs and targets.

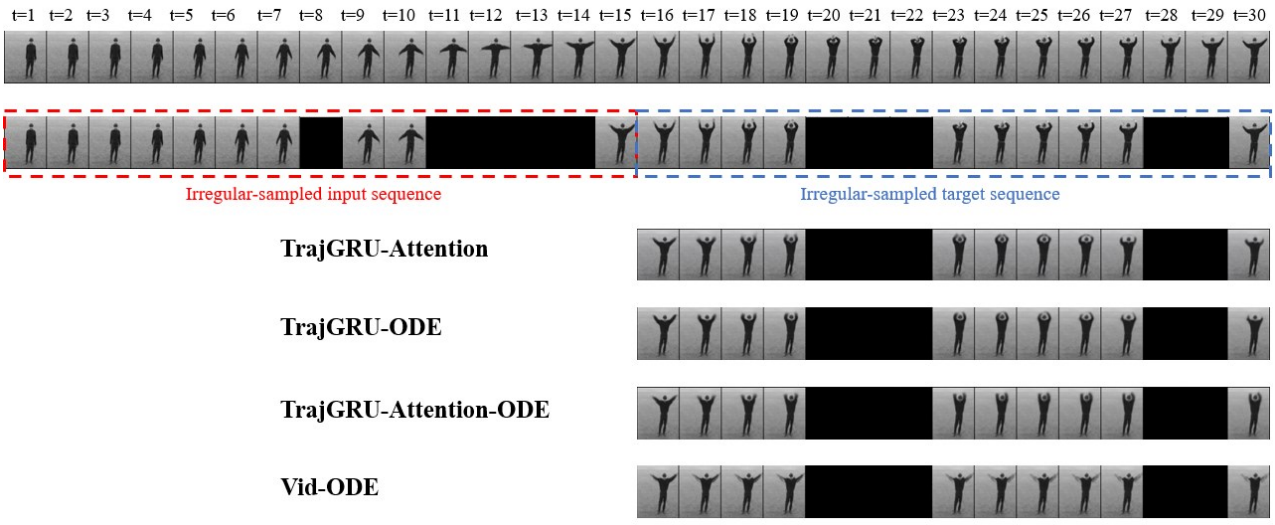


Figure 5.12. Prediction results using the KTH Action with irregular sampled inputs and targets.

Table 5.8 Values of evaluation metrics for the models with the KTH Action dataset

Sampling assumption: regular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
TrajGRU	22.737	183.33	87.981	27.296
Vid-ODE	35.601	224.27	84.675	25.230
TrajGRU-Attention	20.843	175.74	89.432	27.595
TrajGRU-ODE	21.057	174.49	89.759	27.668
TrajGRU-Attention-ODE	20.732	174.19	90.007	27.735
Sampling assumption: irregular-sampled input and target				
Model	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
Vid-ODE	44.902	263.63	83.719	24.388
TrajGRU-Attention	29.598	210.27	87.058	26.043
TrajGRU-ODE	28.645	209.35	87.269	26.156
TrajGRU-Attention-ODE	31.457	224.31	86.623	25.781

5.6.3 Comments

Overall, all three proposed models generate better results than the baseline models when applied to the KTH Action dataset. Proposed models significantly outperform the Vid-ODE in the irregular-sampled assumption.

Chapter 6: Conclusion and Future Work

In this section, we provide some conclusions for the proposed sequence-to-sequence models for the overall SSP task and suggest future works.

6.1 Discussion and Conclusion

In this thesis, we examined the overall SSP task whose key idea is the time-sampling assumption of spatiotemporal datasets. This task can cover regular-sampled and irregular-sampled spatiotemporal prediction in the discrete domain and even the continuous-time modelling in the continuous field. Therefore, we designed and proposed three DL-based sequence-to-sequence models to solve this problem.

In the model designing process, we build our specified ANNs as different blocks to flexibly modify and put them together regarding our model architecture. Eventually, we can create the encoder and decoder of the sequence-to-sequence model based on the building blocks and model architecture. We design the MA (i.e., MEA and MDA) and TrajGRU-ODE blocks to build the ability to extract features at arbitrary timestamps. In particular, the MA blocks help the system pays more attention to certain parts of the previous states and capture elements at the given time steps based on the motion characteristics. The TrajGRU-ODE employs the numerical integrator to generate continuous hidden forms representing features in the continuous space.

In the encoding-decoding structure, we can insert the MA blocks into both parts (i.e., the encoder and decoder) so that the encoder and decoder can handle the irregular-sampled sequences. In contrast, the TrajGRU-ODE cannot be inserted in the encoder because it requires the initial state. Therefore, the TrajGRU-ODE can replace the MDE in the decoder but not the MEA in the encoder.

To evaluate the performance of the proposed models, we use four spatiotemporal datasets: the MovingMNIST, MovingMNIST++, HKO-7, and KTH Action. The MovingMNIST is one of the most widely used video datasets for the SSP tasks. Compared to the MovingMNIST, the MovingMNIST++ has more complicated spatiotemporal properties due to some transformation effects. Besides those two datasets, the HKO-7 and KTH Action are real-world datasets. The HKO-7 contains the weather radar reflectivity images, and KTH Action includes videos presenting different human actions. In addition, the background of all four datasets is static over time. When the background is chaotic, the preprocessing and denoising methods might be required to maintain the overall performance of the models.

All three models can perform the overall SSP task with all four datasets. Moreover, our models can achieve a promising performance in specific scenarios compared to the baseline models (i.e., the ConvGRU, TrajGRU, Vid-ODE). Based on the experiments, we can summarize some essential advantages as follows.

1) First, our proposed models (i.e., TrajGRU-Attention, TrajGRU-ODE, TrajGRU-Attention-ODE) made better predictions than the baseline models (i.e., ConvGRU, TrajGRU, Vid-ODE) when using the MovingMNIST, MovingMNIST++, and KTH Action for all time-sampling assumptions. Our models significantly outperform the Vid-ODE with the irregular time-sampling assumption. To further demonstrate the models' performance, we show the learning curves during the training and validating processes with the MovingMNIST and KTH Action in Figures C.1, C.2, C.3, and C.4
2) In the experiments with the MovingMNIST, the TrajGRU-Attention shows the best performance when the target sequence is regularly sampled, and the TrajGRU-Attention-ODE shows the best performance in the case where the target sequence is irregularly sampled. However, in the experiments with the MovingMNIST++, while TrajGRU-Attention still offers the best performance when the target sequence is regularly sampled, the TrajGRU-ODE yields the best performance in the case where the target sequence is irregularly sampled but not TrajGRU-Attention-ODE.
3) In the experiments with the HKO-7, the TrajGRU-ODE shows the best performance when the sequences are regularly sampled, and the TrajGRU-Attention shows the best performance when the sequences are irregularly sampled.
4) In the experiments with the KTH Action, the TrajGRU-Attention shows the best performance when the sequences are regularly sampled, and the TrajGRU-ODE shows the best performance when the sequences are irregularly sampled.

Besides those advantages, we can conclude some main observations and drawbacks of our models as follows

1) In the experiments with the HKO-7 and KTH Action, when the sequences are regularly sampled, the proposed models' improvements are not clear.
2) By comparing advantages 2, 3, and 4, we can conclude that the NODE techniques are not stable to some degree and rely heavily on the data change rates.
3) The TrajGRU-Attention-ODE model does not meet the expectations in some scenarios where it does not perform the SSP task as effectively as the other two models.
4) Finally, applying the attention mechanisms and NODE in the RNN models will increase the number of trainable parameters, leading to a considerable computational complexity.

6.2 Future work

Despite some promising results of our models, there is still room for further improvement in our system and some issues were also identified during the experiments. First, the NODE technique is still in the preliminary stages of development and investigation, and we can further optimize its ability for the SSP task. Second, the attention mechanism has been getting more attention in recent years, and there are several types of mechanisms that we can try and adjust regarding the requirements of the problem. Furthermore, the combination of attention mechanisms and NODE is very promising since our models, TrajGRU-ODE, and TrajGRU-Attention-ODE, can perform the SSP task effectively in some instances.

Appendix A The baseline model configuration

Table A.1 Model configurations of the ConvGRU and TrajGRU used with the MovingMNIST and MovingMNIST++

<table>
<tr><th colspan="2">Models</th></tr>
<tr><th>ConvGRU</th><th>TrajGRU</th></tr>
<tr><td colspan="2">Common hyperparameters</td></tr>
<tr><td colspan="2">Common connection building blocks
- 3 down-sampling blocks at the encoder and 4 up-sampling blocks at the decoder
- Learnable weights of down-sampling blocks:
$\big((1 \times 64 \times 3 \times 3), (64 \times 96 \times 3 \times 3), (96 \times 128 \times 3 \times 3)\big)$
- Learnable weights of up-sampling blocks:
$\big((128 \times 96 \times 4 \times 4), (96 \times 64 \times 4 \times 4), (64 \times 32 \times 3 \times 3), (32 \times 1 \times 1 \times 1)\big)$
Common hyperparameters for the training process
- Batch size: 4
- Baximum number of iterations: 200000
- Optimizer: Adam with learning rate = 10^{-4} and momentum = 0.5</td></tr>
<tr><td colspan="2">Central building blocks</td></tr>
<tr><td>3 ConvGRU blocks at the encoder and 3 ConvGRU blocks at the decoder:
- Learnable input-to-state weight $\boldsymbol{W}_x$: layer 1 $(64 \times 192 \times 3 \times 3)$; layer 2 $(96 \times 288 \times 3 \times 3)$; layer 3 $(128 \times 384 \times 3 \times 3)$
(Note that $C_i = 3 * C_o$ because there are three distinct state flows ($\boldsymbol{W}_{xz}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh}$))
- Learnable state-to-state weight $\boldsymbol{W}_h$: layer 1 $(64 \times 192 \times 3 \times 3)$; layer 2 $(96 \times 288 \times 3 \times 3)$; layer 3 $(128 \times 384 \times 3 \times 3)$</td><td>3 TrajGRU blocks at the encoder and 3 TrajGRU blocks at the decoder:
- Learnable input-to-state weight $\boldsymbol{W}_x$: layer 1 $(64 \times 192 \times 3 \times 3)$; layer 2 $(96 \times 288 \times 3 \times 3)$; layer 3 $(128 \times 384 \times 3 \times 3)$
- Learnable state-to-state weight $\boldsymbol{W}_h$: layer 1 $(64 \times 192 \times L_1 \times L_1)$; layer 2 $(96 \times 288 \times L_2 \times L_2)$; layer 3 $(128 \times 384 \times L_3 \times L_3)$; with $L_1 = L_2 = L_3 = 9$
- Learnable weights of the spatial transformer modules: each module contains three convolutional layers with a common kernel size of (3×3) and the numbers of input and output filters are: layer 1 $(64,64)$; layer 2 $(96,96)$; layer 3 $(128,128)$</td></tr>
<tr><td colspan="2">Total number of parameters</td></tr>
<tr><td>3695553</td><td>4859181</td></tr>
</table>

Table A.2 Model configurations of the ConvGRU and TrajGRU used with the HKO-7 and KTH

<table>
<tr><th colspan="2">Models</th></tr>
<tr><th>ConvGRU</th><th>TrajGRU</th></tr>
<tr><th colspan="2">Common hyperparameters</th></tr>
<tr><td colspan="2">Common connection building blocks
- 3 down-sampling blocks at the encoder and 4 up-sampling blocks at the decoder
- Learnable weights of down-sampling blocks:
$\big((1 \times 96 \times 3 \times 3), (96 \times 96 \times 3 \times 3), (96 \times 128 \times 3 \times 3)\big)$
- Learnable weights of up-sampling blocks:
$\big((128 \times 96 \times 4 \times 4), (96 \times 96 \times 4 \times 4), (96 \times 32 \times 3 \times 3), (32 \times 1 \times 1 \times 1)\big)$
Common hyperparameters for the training process
- Batch size: 4
- Maximum number of epochs: 25
- Number of iterations per epoch: 8000
- Optimizer: Adam with learning rate = 10^{-4} and momentum = 0.9</td></tr>
<tr><th colspan="2">Central building blocks</th></tr>
<tr><td>3 ConvGRU blocks at the encoder and 3 ConvGRU blocks at the decoder:
- Learnable input-to-state weight $\boldsymbol{W}_x$: layer 1 $(96 \times 288 \times 3 \times 3)$; layer 2 $(96 \times 288 \times 3 \times 3)$; layer 3 $(128 \times 384 \times 3 \times 3)$
(Note that $C_i = 3 * C_o$ because there are three distinct state flows $(\boldsymbol{W}_{xz}, \boldsymbol{W}_{xr}, \boldsymbol{W}_{xh})$)
- Learnable state-to-state weight $\boldsymbol{W}_h$: layer 1 $(96 \times 288 \times 3 \times 3)$; layer 2 $(96 \times 288 \times 3 \times 3)$; layer 3 $(128 \times 384 \times 3 \times 3)$</td><td>3 TrajGRU blocks at the encoder and 3 TrajGRU blocks at the decoder:
- Learnable input-to-state weight $\boldsymbol{W}_x$: layer 1 $(96 \times 288 \times 3 \times 3)$; layer 2 $(96 \times 288 \times 3 \times 3)$; layer 3 $(128 \times 384 \times 3 \times 3)$
- Learnable state-to-state weight $\boldsymbol{W}_h$: layer 1 $(96 \times 288 \times L_1 \times L_1)$; layer 2 $(96 \times 288 \times L_2 \times L_2)$; layer 3 $(128 \times 384 \times L_3 \times L_3)$; with $L_1 = L_2 = L_3 = 11$
- Learnable weights of the spatial transformer modules: each module contains three convolutional layers with a common kernel size of (3×3) and the numbers of input and output filters are: layer 1 $(96,96)$; layer 2 $(96,96)$; layer 3 $(128,128)$</td></tr>
<tr><th colspan="2">Total number of parameters</th></tr>
<tr><td>4367553</td><td>6166853</td></tr>
</table>

Appendix B Additional Information of the proposed models

Table B.1 illustrates the resulting evaluation metrics of the TrajGRU-Attention model with different attention settings when training and evaluating with the MovingMNIST dataset. For instance, when the encoder uses the attention module at its central layer 3 and the decoder uses the attention module at its central layer 3, we abbreviate this setting as "en_a3_de_a3".

Table B.1 Comparison between different settings of the TrajGRU-Attention model when using the MovingMNIST

Setting	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
en_a3_de_a3	61.812	151.23	93.782	22.592
en_a2_de_a2	62.112	156.12	93.712	22.553
en_a1_de_a1	58.423	150.23	93.961	22.854
en_a23_de_a32	53.712	136.34	94.661	23.335
en_a23_de_a21	49.632	131.32	94.941	23.733
en_a123_de_a321	60.832	145.23	93.921	22.922

Based on Table B.1, the setting "en_a23_de_a21" shows the best results in all four metrics so that we choose this setting to design the proposed models.

Table B.2 illustrates the resulting evaluation metrics of the TrajGRU model with different loss functions when training and evaluating with the MovingMNIST dataset.

Table B.2 Comparison between different loss functions of the TrajGRU model when using the MovingMNIST

Loss function	**MSE**$\times 10^4(\downarrow)$	**MAE**$\times 10^4(\downarrow)$	**SSIM**$\times 10^2(\uparrow)$	**PSNR**$(\uparrow)$
TrajGRU (MSE)	61.212	176.23	92.013	22.353
TrajGRU (MSE + MAE)	69.223	167.43	92.032	22.064
TrajGRU (our loss function)	61.932	153.34	93.894	22.734

Based on Table B.2, our proposed loss function helps the model generate the results with the best evaluation metrics, so we choose this loss function as the standard function to train all the models (i.e., the proposed and baseline models).

Appendix C Additional information for the comparison of models

For the training process, the number of epochs is set to 25 for all algorithms, and the initial learning rate of the Adam optimizer is set to 10^{-4}. The early stopping mechanism is applied to obtain the final collection of trainable parameters so that the system attempts to stop training when the validation loss continues to increase during two epochs in a row. Although this setting allowed the models to generate results with relatively high accuracy, it is not optimal. First, it is not a reasonable comparison because not all learning curves of the proposed models are converged. For instance, in the case of the MovingMNIST dataset, Figures C.1 and C.2 show that the learning curves of the proposed models (i.e., the TrajGRU-Attention, TrajGRU-ODE, and TrajGRU-Attention-ODE) have not yet converged after 25 training epochs. Therefore, we can improve the performance extending the number of training epochs. Second, the initial learning rate needs to be tuned to optimize the training process of a specific model.

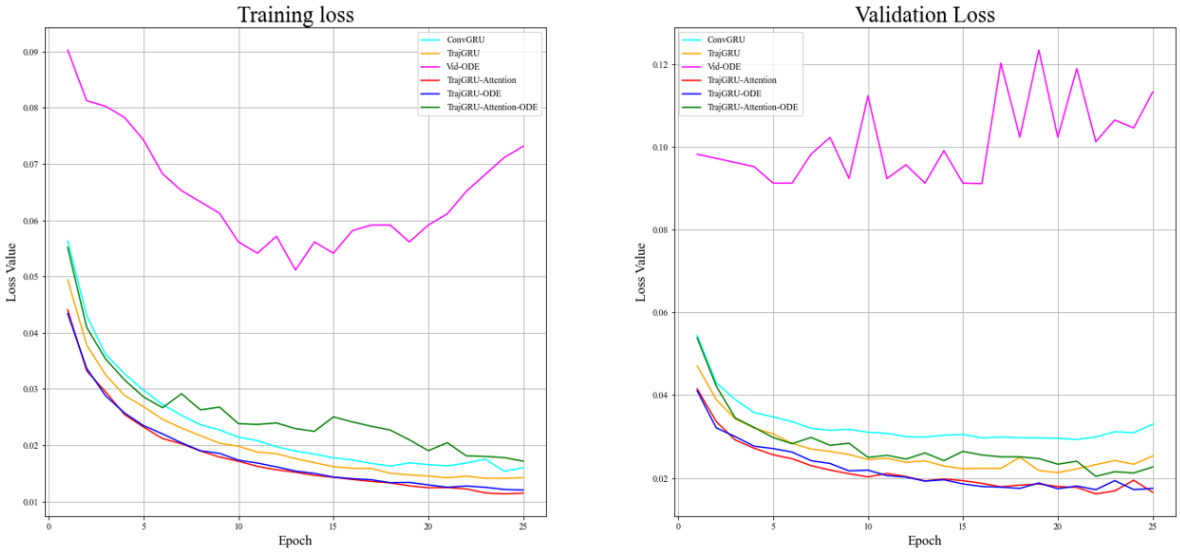


Figure C.1 Learning curves of models using the MovingMNIST with regular sampling assumption at both the encoder and decoder: (a) Training learning curve, (b) validation learning curve.

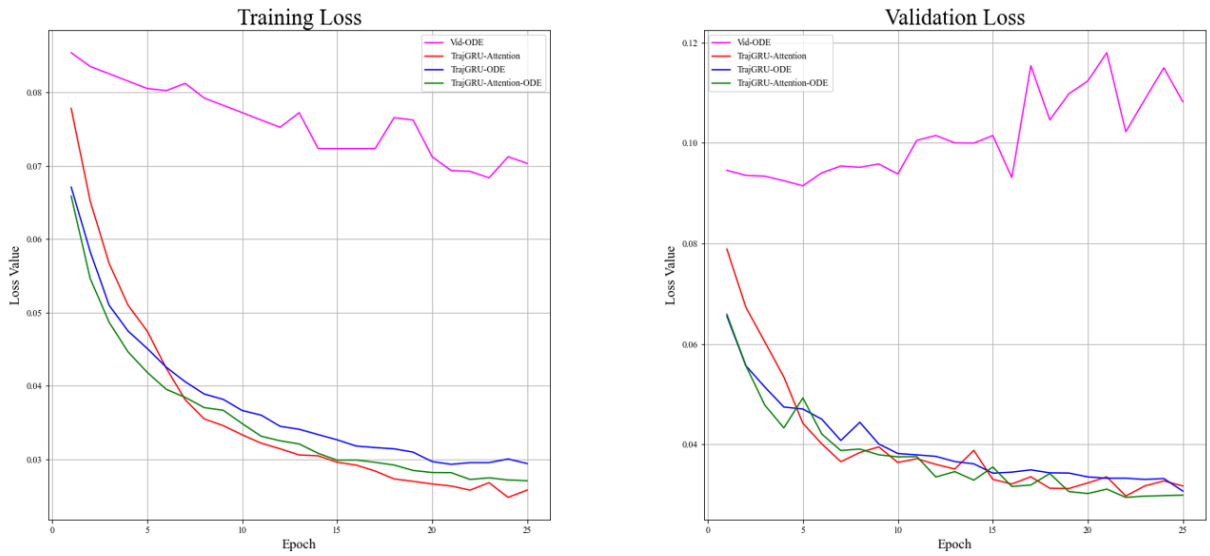


Figure C.2 learning curves of models using the MovingMNIST with irregular sampling assumption at both the encoder and decoder: (a) training learning curve, (b) validation learning curve.

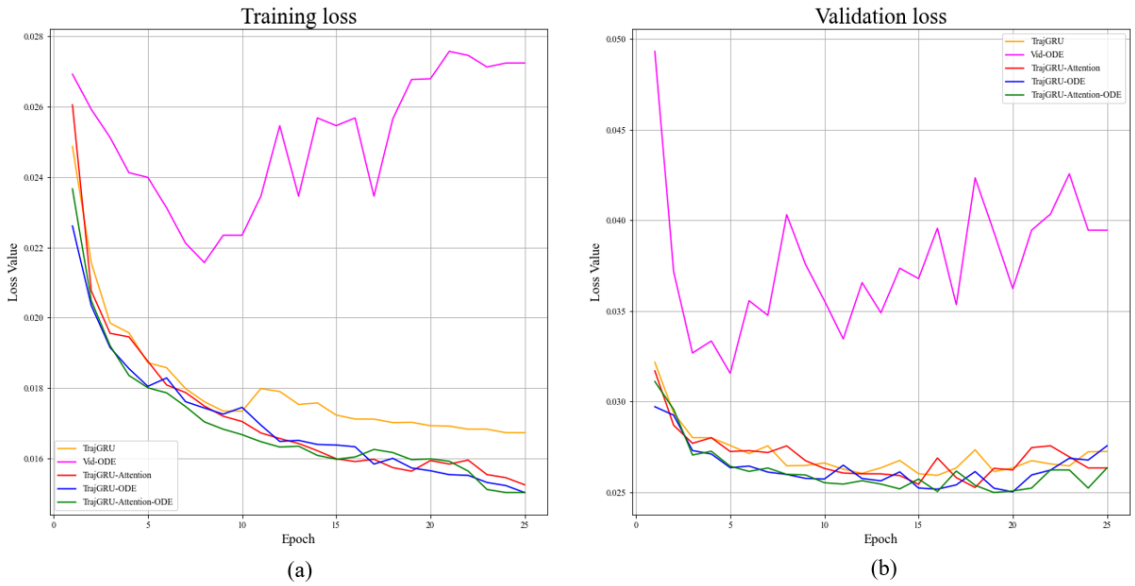


Figure C.3 Learning curves of models using the KTH Action with regular sampling assumption at both the encoder and decoder: (a) training learning curve, (b) validation learning curve.

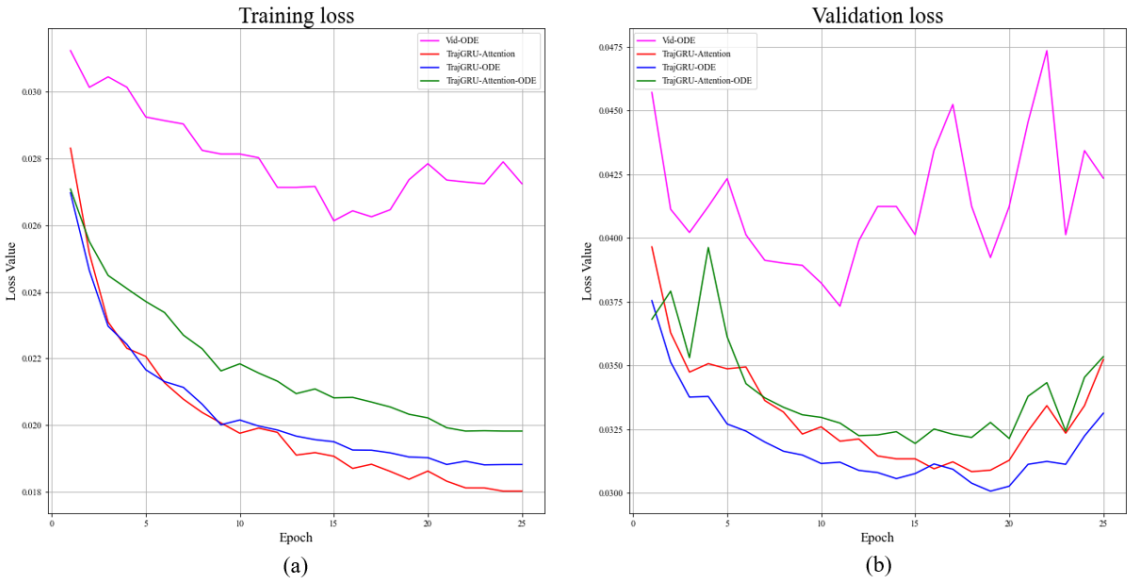


Figure C.4 Learning curves of models using the KTH Action with irregular sampling assumption at both the encoder and decoder: (a) training learning curve, (b) validation learning curve.

References

[1] Shi, X. Chen, Z. Wang, H. Yeung, D.-Y. Wong, W.-K. Woo and Wang-chun, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems,* vol. 28, 2015.

[2] Shi, X. Gao, Z. Lausen, L. Wang, H. Yeung, D.-Y. Wong, W.-k. Woo and Wang-chun, "Deep learning for precipitation nowcasting: A benchmark and a new model," *Advances in neural information processing systems,* vol. 30, 2017.

[3] Wang, Y. Long, M. Wang, J. Gao, Z. Yu and P. S, "Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms," *Advances in neural information processing systems,* vol. 30, 2017.

[4] Fang, W. Chen, Y. Xue and Qiongying, "Survey on research of RNN-based Spatio-temporal sequence prediction algorithms," *Journal on Big Data,* vol. 3, no. 3, p. 97, 2021.

[5] Xu, Z. a. Wang, Y. a. Long, M. a. Wang, J. a. KLiss and M, "PredCNN: Predictive Learning with Cascade Convolutions.," in *IJCAI*, 2018, pp. 2940--2947.

[6] Wu, J. a. Lu, E. a. Kohli, P. a. Freeman, B. a. Tenenbaum and Josh, "Learning to see physics via visual de-animation," *Advances in Neural Information Processing Systems,* vol. 30, 2017.

[7] Wang, Y. a. Jiang, L. a. Yang, M.-H. a. Li, L.-J. a. Long, M. a. Fei-Fei and Li, "Eidetic 3D LSTM: A model for video prediction and beyond," in *International conference on learning representations*, 2018.

[8] Wang, Y. Zhang, J. Zhu, H. Long, M. Wang, J. Yu and P. S, "{Memory in memory: A predictive neural network for learning higher-order non-stationarity from spatiotemporal dynamics," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9154--9162.

[9] V. Steenkiste, S. Chang, M. Greff, K. Schmidhuber and J, "Relational neural expectation maximization: Unsupervised discovery of objects and their interactions," *arXiv preprint arXiv:1802.10353,* 2018.

[10] Wang, S. a. Cao, J. a. Yu and Philip, "Deep learning for spatio-temporal data mining: A survey," *IEEE transactions on knowledge and data engineering,* 2020.

[11] Trebing, K. Stańczyk, T. Mehrkanoon and Siamak, "SmaAt-UNet: Precipitation nowcasting using a small attention-UNet architecture," *Pattern Recognition Letters,* vol. 145, pp. 178--186, 2021.

[12] Tran, D. Bourdev, L. Fergus, R. Torresani, L. Paluri and Manohar, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489--4497.

[13] Wang, Y. Wu, H. Zhang, J. Gao, Z. Wang, J. Yu, P. Long and Mingsheng, "Predrnn: A recurrent neural network for spatiotemporal predictive learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2022.

[14] Park, S. Kim, K. Lee, J. Choo, J. Lee, J. Kim, S. Choi and Edward, "Vid-ode: Continuous-time video generation with neural ordinary differential equation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 2412--2422.

[15] Atluri, G. Karpatne, A. Kumar and Vipin, "Spatio-temporal data mining: A survey of problems and methods," *ACM Computing Surveys (CSUR),* vol. 51, pp. 1--41, 2018.

[16] Kisilevich, S. Mansmann, F. Nanni, M. Rinzivillo and Salvatore, "Spatio-temporal clustering," in *Data mining and knowledge discovery handbook*, Springer, 2009, pp. 855--874.

[17] Yu, W. Lu, Y. Easterbrook, S. Fidler and Sanja, "Efficient and information-preserving future frame prediction and beyond," 2020.

[18] Su, J. Byeon, W. Kossaifi, J. Huang, F. Kautz, J. Anandkumar and Anima, "Convolutional tensor-train lstm for spatio-temporal learning," *Advances in Neural Information Processing Systems,* vol. 33, pp. 13714--13726, 2020.

[19] Chen, R. T. Rubanova, Y. Bettencourt, J. Duvenaud and D. K, "Neural ordinary differential equations," *Advances in neural information processing systems,* vol. 31, 2018.

[20] Rubanova, Y. Chen, R. T. Duvenaud and D. K, "Latent ordinary differential equations for irregularly-sampled time series," *Advances in neural information processing systems,* vol. 32, 2019.

[21] Yildiz, C. a. Heinonen, M. a. Lahdesmaki and Harri, "ODE2VAE: Deep generative second order ODEs with Bayesian neural networks," *Advances in Neural Information Processing Systems,* vol. 32, 2019.

[22] D. Brouwer, E. Simm, J. Arany, A. Moreau and Yves, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," *Advances in neural information processing systems,* vol. 32, 2019.

[23] Bakeman, R. Gottman and J. M, Observing interaction: An introduction to sequential analysis, Cambridge university press, 1997.

[24] Minar, M. R. Naher and Jibon, "Recent advances in deep learning: An overview," *arXiv preprint arXiv:1807.08169,* 2018.

[25] Du, S. Li, T. Horng and Shi-Jinn, "Time series forecasting using sequence-to-sequence deep learning framework," in *2018 9th international symposium on parallel architectures, algorithms and programming (PAAP)*, IEEE, 2018, pp. 171--176.

[26] Baccouche, M. Mamalet, F. Wolf, C. Garcia, C. Baskurt and Atilla, "Sequential deep learning for human action recognition," in *International workshop on human behavior understanding*, 2011, Springe, pp. 29--39.

[27] Venugopalan, S. Rohrbach, M. Donahue, J. Mooney, R. Darrell, T. Saenko and Kate, "Sequence to sequence-video to text," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4534--4542.

[28] Chiu, C.-C. Sainath, T. N. Wu, Y. Prabhavalkar, R. Nguyen, P. Chen, Z. Kannan, A. Weiss, R. J. Rao, K. Gonina and E. others, "State-of-the-art speech recognition with sequence-to-sequence models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp. 4774--4778.

[29] Lin, W. Orgun, M. A. Williams and G. J. others, "An Overview Of Temporal Data Mining.," in *AusDM*, 2002, pp. 83--90.

[30] Ren, W. Singh, S. Singh, M. Zhu and Y. S, "State-of-the-art on spatio-temporal information-based video retrieval," *Pattern recognition,* vol. 42, pp. 267--282, 2009.

[31] Craig, C. A. Feng and Song, "A temporal and spatial analysis of climate change, weather events, and tourism businesses," *Tourism Management,* vol. 67, pp. 351--361.

[32] Delmelle, E. Kim, C. Xiao, N. Chen and Wei, "Methods for space-time analysis and modeling: An overview," *International Journal of Applied Geospatial Research (IJAGR),* vol. 4, pp. 1--18, 2013.

[33] S. Roy and Shouraseni, "A spatial analysis of extreme hourly precipitation patterns in India," *International Journal of Climatology: A Journal of the Royal Meteorological Society,* vol. 29, pp. 345--355, 2009.

[34] LeCun, Y. Bengio, Y. Hinton and Geoffrey, "Deep learning," *nature,* vol. 521, pp. 436--444, 2015.

[35] O'Shea, K. Nash and Ryan, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458,* 2015.

[36] Gu, J. Wang, Z. Kuen, J. Ma, L. Shahroudy, A. Shuai, B. Liu, T. Wang, X. Wang, G. Cai and J. others, "Recent advances in convolutional neural networks," *Pattern recognition,* vol. 77, pp. 354--377, 2018.

[37] Zeiler, M. D. Krishnan, D. Taylor, G. W. Fergus and Rob, "Deconvolutional networks," in *2010 IEEE Computer Society Conference on computer vision and pattern recognition*, IEEE, 2010, pp. 2528--2535.

[38] Tsoi and A. Chung, "Recurrent neural network architectures: an overview," *International School on Neural Networks, Initiated by IIASS and EMFCSC,* pp. 1--26, 1997.

[39] Salehinejad, H. Sankar, S. Barfett, J. Colak, E. Valaee and Shahrokh, "Recent advances in recurrent neural networks," *arXiv preprint arXiv:1801.01078,* 2017.

[40] Hochreiter, S. Schmidhuber and J, "Long short-term memory," *Neural computation,* vol. 9, no. 8, pp. 1735--1780, 1997.

[41] Chung, J. Gulcehre, C. Cho, K. Bengio and Yoshua, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555,* 2014.

[42] Staudemeyer, R. C. Morris and E. Rothstein, "Understanding LSTM--a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586,* 2019.

[43] Yamak, P. T. Yujian, L. Gadosey and P. K, "A comparison between arima, lstm, and gru for time series forecasting," in *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, 2019, pp. 49--55.

[44] Coddington, E. A. Levinson and Norman, Theory of ordinary differential equations, Tata McGraw-Hill Education, 1955.

[45] Browder, A. Wolfgang, W. Walter, W. Walter and W. Ludwig, Ordinary differential equations, vol. 182, Springer Science & Business Media, 1998.

[46] Chen, R. TQ, Amos, Brandon, Nickel and Maximilian, "Neural spatio-temporal point processes," *arXiv preprint arXiv:2011.04583,* 2020.

[47] Butcher and J. Charles, Numerical methods for ordinary differential equations, John Wiley & Sons, 2016.

[48] Bai, S. Kolter, J. Z. Koltun and Vladlen, "Deep equilibrium models," *Advances in Neural Information Processing Systems,* vol. {32, 2019.

[49] P. J. Werbos, "Backpropagation through time: what it does and how to do it," vol. 78, no. 10, pp. 1550-1560, 1990.

[50] Pontryagin and L. Semenovich, Mathematical theory of optimal processes, CRC press, 1987.

[51] Wang, F. Tax and D. MJ, "Survey on the attention based RNN model and its applications in computer vision," *arXiv preprint arXiv:1601.06823,* 2016.

[52] Luong, M.-T. Pham, H. Manning and C. D, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025,* 2015.

[53] Bahdanau, D. Cho, K. Bengio and Yoshua, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473,* 2014.

[54] A. Vaswani, N. Shazeer, N. Parmar, Uszkoreit, J. Jones, L. Gomez, A. N. Kaiser, L. Polosukhin and Illia, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[55] Vig, J. Belinkov and Yonatan, "Analyzing the structure of attention in a transformer language model," *arXiv preprint arXiv:1906.04284,* 2019.

[56] Vig and Jesse, "A multiscale visualization of attention in the transformer model," *arXiv preprint arXiv:1906.05714,* 2019.

[57] Galassi, A. Lippi, M. Torroni and Paolo, "Attention in natural language processing," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 32, no. 10, pp. 4291--4308, 2020.

[58] Kumar, A. Irsoy, O. Ondruska, P. Iyyer, M. Bradbury, J. Gulrajani, I. a. Zhong, V. a. Paulus, R. a. Socher and Richard, "Ask me anything: Dynamic memory networks for natural language processing," in *International conference on machine learning*, PMLR, 2016, pp. 1378--1387.

[59] S. Sharma, S. S. Athaiya and Anidhya, "Activation functions in neural networks," 2020.

[60] Tyson and Lois, "Critical theory today: A user-friendly guide," 2014.

[61] Nwankpa, Chigozie, Ijomah, Winifred, Gachagan, Anthony, Marshall and Stephen, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378,* 2018.

[62] Wang, Zhou, Bovik and A. C, "Modern image quality assessment," *Synthesis Lectures on Image, Video, and Multimedia Processing,* vol. 2, no. 1, pp. 1--156, 2006.

[63] Hore, Alain, Ziou and Djeme, "Image quality metrics: PSNR vs. SSIM," in *2010 20th international conference on pattern recognition*, IEEE, 2010, pp. 2366--2369.

[64] Kotevski, Zoran, Mitrevski and Pece, "Experimental comparison of psnr and ssim metrics for video quality estimation," in *International Conference on ICT Innovations*, Springer, 2009, pp. 357--366.

[65] Janocha, Katarzyna, Czarnecki and W. Marian, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659,* 2017.

[66] Kochenderfer, M. J, Wheeler and T. A, "Algorithms for optimization," 2019.

[67] Ruder and Sebastian, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747,* 2016.

[68] Sutskever, Ilya, Martens, James, Dahl, George, Hinton and Geoffrey, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, PMLR, 2013, pp. 1139--1147.

[69] Duchi, John, Hazan, Elad, Singer and Yoram, "Adaptive subgradient methods for online learning and stochastic optimization.," *Journal of machine learning research,* vol. 12, no. 7, 2011.

[70] Hinton, Geoffrey, Srivastava, Nitish, Swersky and Kevin, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on,* vol. 14, no. 8, p. 2, 2012.

[71] Kingma, D. P, Ba and Jimmy, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980,* 2014.

[72] LeCun, Yann, Bengio and Y. a. others, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks,* vol. 3361, no. 10, p. 1995, 1995.

[73] Ballas, Nicolas, Yao, Li, Pal, Chris, Courville and Aaron, "Delving deeper into convolutional networks for learning video representations," *arXiv preprint arXiv:1511.06432,* 2015.

[74] Siam, Mennatullah, Valipour, Sepehr, Jagersand, Martin, Ray and Nilanjan, "Convolutional gated recurrent networks for video segmentation," in *2017 IEEE international conference on image processing (ICIP)*, IEEE, 2017, pp. 3090--3094.

[75] Shrestha, Ajay, Mahmood and Ausif, "Review of deep learning algorithms and architectures," *IEEE access,* vol. 7, no. IEEE, pp. 53040--53065.

[76] Ahn and SungMahn, "Deep learning architectures and applications," *Journal of Intelligence and Information Systems,* vol. 22, no. 2, pp. 127--142, 2016}.

[77] Ayzel, G, Heistermann, M, Sorokin, A, Nikitin, O, Lukyanova and O, "All convolutional neural networks for radar-based precipitation nowcasting," *Procedia Computer Science,* vol. 150, no. Elsevier, pp. 186--192, 2019.

[78] Jaderberg, Max, Simonyan, Karen, Zisserman and A. a. others, "Spatial transformer networks," *{Advances in neural information processing systems,* vol. 28, 2015.

[79] Beauchemin, S. S., Barron and J. L., "The computation of optical flow," *ACM computing surveys (CSUR),* vol. 27, no. 3, pp. 433--466, 1995.

[80] Sun, Deqing, Roth, Stefan, Lewis, J. P, Black and M. J, "Learning optical flow," in *European Conference on Computer Vision*, Springer, 2008, pp. 83--97.

[81] Wang, Yunbo, Gao, Zhifeng, Long, Mingsheng, Wang, Jianmin, Philip and S. Yu, "Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning," in *International Conference on Machine Learning*, PMLR, 2018, pp. 5123--5132.

[82] Bianchini, Monica, Scarselli and Franco, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE transactions on neural networks and learning systems,* vol. 25, no. 8, pp. 1553--1565, 2014.

[83] Pascanu, Razvan, Mikolov, Tomas, Bengio and Yoshua, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, PMLR, 2013, pp. 1310--1318.

[84] Soltani, Rohollah, Jiang and Hui, "Higher order recurrent neural networks," *arXiv preprint arXiv:1605.00064,* 2016.

[85] Oseledets and I. V, "Tensor-train decomposition," *SIAM Journal on Scientific Computing,* vol. 33, no. 5, pp. 2295--2317, âđăă.

[86] Tschannen, Michael, Bachem, Olivier, Lucic and Mario, "Recent advances in autoencoder-based representation learning," *arXiv preprint arXiv:1812.05069,* âđắ.

[87] D. Brouwer, Edward, Simm, Jaak, Arany, Adam, Moreau and Yves, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," *Advances in neural information processing systems,* vol. 32, 2019.

[88] Yan, Bin, Peng, Houwen, Fu, Jianlong, Wang, Dong, Lu and Huchuan, "Learning spatio-temporal transformer for visual tracking," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10448--10457.

[89] Zha, Xuefan, Zhu, Wentao, Xun, Lv, Yang, Sen, Liu and Ji, "Shifted chunk transformer for spatio-temporal representational learning," *Advances in Neural Information Processing Systems,* vol. 34, pp. 11384--11396, 2021.

[90] I. a. P.-A. J. a. M. M. G. Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, Bengio and Yoshua, "Generative adversarial nets," *Advances in neural information processing systems,* vol. 27, 2014.

[91] Saxena, Divya, Cao and Jiannong, "D-GAN: Deep generative adversarial nets for spatio-temporal prediction," *arXiv preprint arXiv:1907.08556,* 2019.

[92] Liu, Hong-Bin, Lee and Ickjai, "MPL-GAN: Toward realistic meteorological predictive learning using conditional GAN," *IEEE Access,* vol. 8, pp. 93179--93186, 2020.

[93] Hochreiter and Sepp, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* vol. 6, no. 02, pp. 107-116, 1998.

[94] Foley and AM, "Uncertainty in regional climate modelling: A review," *Progress in Physical Geography,* vol. 34, no. 5, pp. 647-670, 2010.

[95] Kanakidou, Maria, Seinfeld, JH, Pandis, SN, Barnes, Ian, Dentener, F. Johannes, Facchini, M. Cristina, V. Dingenen, Rita, Ervens, Barbara, Nenes, ANCJSE, Nielsen and C. a. others, "Organic aerosol and global climate modelling: a review," *Atmospheric Chemistry and Physics,* vol. 5, no. 4, pp. 1053-1123, 2005.

[96] Zhong, Y. Desmond, Dey, Biswadip, Chakraborty and Amit, "Symplectic ode-net: Learning hamiltonian dynamics with control," *arXiv preprint arXiv:1909.12077,* 2019.

[97] Tong, Alexander, Huang, Jessie, Wolf, Guy, V. Dijk, David, Krishnaswamy and Smita, "Trajectorynet: A dynamic optimal transport network for modeling cellular dynamics," in *International conference on machine learning*, PMLR, 2020, pp. 9526-9536.

[98] Pinckaers, Hans, Litjens and Geert, "Neural ordinary differential equations for semantic segmentation of individual colon glands," *arXiv preprint arXiv:1910.10470,* 2019.

[99] Huai, Jianzhu, Zhuang, Yuan, Lin, Yukai, Jozkow, Grzegorz, Yuan, Qicheng, Chen and Dong, "Continuous-time spatiotemporal calibration of a rolling shutter camera-IMU system," *IEEE Sensors Journal,* vol. 22, no. 8, pp. 7920-7930, 2022.

[100] T. Quang-Khai and S. Sa-kwang, "Computer vision in precipitation nowcasting: Applying image quality assessment metrics for training deep neural networks," *Atmosphere,* vol. 10, no. 5, p. 244, 2019.

[101] Schuldt, C. Laptev, Ivan, Caputo and Barbara, "Recognizing human actions: a local SVM approach," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3, 2004, pp. 32--36.

[102] He, K. Zhang, X. Ren, S. Sun and Jian, "Deep residual learning for image recognitio," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770--778.