Automated Selection of ML/DL Techniques for Time Series Data

Yi Chen

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

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

Abstract

Time-series data is widely used in a lot of domains, such as finance, and the analysis of it positively affects the development of society. It provides information on how data changes over time. However, analyzing and forecasting time-series data is challenging since it always comes with specific characteristics which may have an impact on the performance of models and researchers do not know before implementing models. At the same time, various models make it difficult to choose a suitable one for the datasets. In this thesis, a novel Automated Model Selection technique is proposed. We collect a wide range of models and time-series datasets and choose some of them to conduct a series of experiments to explore how different elements affect the performances of models. We make a thorough quantitative and qualitative analysis of the experimental results and based on this analysis we formulate several outcomes. We then develop the proposed technique based on these outcomes. This Automated Model Selection technique achieves the goal of selecting a suitable model for the input datasets automatically.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. Verena Kantere who guides and supports me on my journey. She not only gives me guidance on the research and thesis but encourages me when I am not confident. She has walked me through all the stages of the writing of my thesis, and her advice and guidance make this challenging work more achievable. Because of her affirmation and encouragement, I try to express myself bravely and overcome shyness.

My appreciation also goes to Paraskevas Kerasiotis who gives me a lot of suggestions for my research. He made efforts to help me when I encountered problems while conducting experiments. His passion for academics inspires and encourages me as well to explore more fields in my future life.

Meanwhile, I am extremely grateful to my family for their unfailing love. I stand on the shoulders of my parents, and they lift me to touch and get involved in a larger world.

Finally, many thanks go to my boyfriend. He always appeases me when I am in the rough and discusses with me how to find a solution.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The general introduction of this thesis is given in this chapter. Section 1.1 mentions the motivation of the research, while Section 1.2 describes the problem. Then, Section 1.3 proposes a solution for that, and Section 1.4 is the methodology we employed to solve the problem. Finally, the outline of the thesis is given in Section 1.5.

1.1 Motivation

Nowadays, we are in the data era and surrounded by all kinds of information in our daily lives. Data is collected and recorded over time in a wide range of domains, such as finance, engineering, medicine, and weather forecasting. These observations lead to a collection of organized statistics called time-series data, which is a set of data points ordered in time [1] [2]. There are 2 types of time-series data: regular time series and irregular time series. Regular time series is the most common type of time series where observations come at regular time intervals [3]. This kind of data is recorded at different intervals, such as year, month, day, hour, or even minute, to measure how elements change over time. Another one is irregular time-series data, where data is not observed in regular time intervals. An example of this is the collection of data related to patient tests. We can obtain the data only if the patient heads to a clinic and carries out the test, and it may not happen in regular time intervals [3].

Time-series data can be decomposed into 3 components: the trend, the seasonal component, and the residuals. The trend represents slow-moving changes in a time series, which helps

discover whether the data values gradually decrease or increase over time [2]. Seasonal components refer to the seasonality of the series. Seasonal variation occurs repeatedly over a period of time, such as a year or 4 quarters, showing the seasonality of the data [2]. Finally, residuals encompass the behaviour of time series that cannot be explained by the 2 components mentioned [2]. These components can be discovered visually by the decomposition of the dataset. Such a data decomposition example is shown in Figure 1:

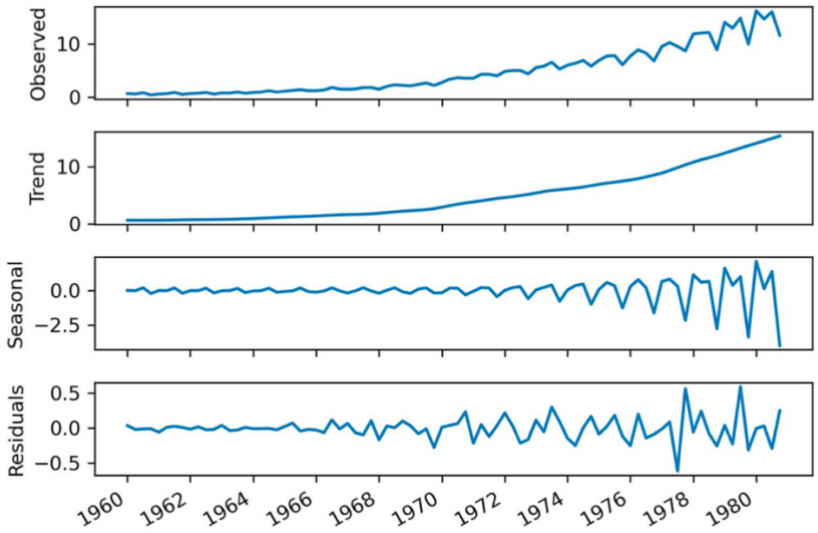


Figure 1 Decomposition Example [2]

The first graph in Figure 1 is the dataset that represents the quarterly earnings per share of Johnson & Johnson stock from 1960 to 1980. The data changed over time but gradually increased in the observed period. This fact is represented by the trend component, which shows that earnings showed a positive trend in these 20 years. Moreover, the seasonal component shows that specific cycles occurred repeatedly during this period. We can see that over a year, the earnings start low, increase, and decrease again at the end of the year [2]. This component can help Johnson & Johnson stock employees capture some rules of their profits. Finally, residuals reveal what is not explained by the trend and the seasonal component. In other words,

we can add the Trend and Seasonal graphs together and compare the output to the actual data in the first overall graph [2]. For instance, residuals are zero in some time steps, which means the values of those points are the same in both the observed overall graph and the output of the addition of the trend and the seasonal component graphs. Residuals are usually related to random errors in the datasets.

It is widely recognized that time-series data is essential in real life. Understanding the principle of how that data changes and the decomposition of the data is helpful for a large number of fields. For example, department stores can adjust their stock if they grasp information about which periods are busy, airlines are able to make a plan of flights based on the number of passengers at different times; factories can know about the consumption of energy in different seasons to ensure regular operation, etc. Time-series data carries a lot of information that we can leverage for prediction and decision-making if we can extract it from the data [4].

However, analyzing time series is not as easy as other types of data. First, most time series exhibit seasonality or cyclical patterns, which may be complicated. Second, time-series data is often affected by external factors that should also be considered. Moreover, the forecasting of time-series data usually relies on previous time points, so it is sensitive to the variation in time. For these reasons, the analysis and forecasting of time-series data have become vital but challenging tasks. At the same time, several methods for time-series data analysis have been proposed, such as ARIMA [7], Prophet [8], and some Deep Learning (DL) models. Each model has a structure that may be suitable for a specific type of data, thereby making it inadequate for others. Due to the deep knowledge required for each model and, as well the limited information about the characteristics of the input dataset that is provided, it is challenging to decide on the

proper one among these models for a specific dataset.

1.2 Problem Definition

Time-series data analysis and forecasting are significant for many applications in business and industry such as the stock market and exchange, weather forecasting, electricity management, fuel consumption, etc. [4]. Therefore, researchers have been exploring the creation of Machine Learning (ML) models to predict time-series data. A large number of models have been proposed to process time-series data forecasting. For example, Facebook released the source of Prophet—a forecasting tool used in Python and R languages in 2017 [5]. It is developed to meet the demand of the business level [5]. Moreover, Deep Learning models are also used to do time series predicting such as Long Short-Term Memory (LSTM) [8] and Gated Recurrent Unit (GRU) [9].

In order to perform forecasting with ML models, users need to implement a type of model appropriate for the input datasets. Then the models train with this data and generate predictions. Therefore, to conduct analysis and forecasting on a time-series dataset, we need to select the most suitable model. However, it is challenging since there is a variety of ML models to choose from and, furthermore, each model comes with a specific structure which may be suitable for a particular type of dataset. On the other hand, users may not know the characteristics of the particular time-series dataset that have an impact on the models' performance. Thus, the selection of the appropriate model can be time-consuming and inaccurate.

1.3 Proposed Solution

We propose creating an Automated Model Selection technique for time-series data forecasting

to solve the model selection problem. The envisioned technique should achieve the goal of selecting a suitable model for an input time-series dataset. Towards this end, we have designed a method that considers various models used for time-series data forecasting and selects the most appropriate one based on the characteristics of the input datasets. Our proposed Automated Model Selection Technique can help analyze time-series data more accurately and efficiently. Moreover, it not only provides a tool for model selection for time-series forecasting for non-expert users but also assists even experienced analysts that deal with complex time-series data to understand their characteristics. The proposed technique performs the model selection automatically.

1.4 Methodology

The methodology that we followed proceeded in the following steps. First, we reviewed a variety of models and time-series datasets. Second, we collected different types of time-series datasets and assessed whether they were suitable for our research. Third, we chose several kinds of models and 2 datasets to conduct experiments. Fourth, we meticulously designed the experimental study and performed experiments. Fifth, we analyzed the experimental results in depth and determined clear outcomes, which can be used to create a technique for automated model selection. Finally, based on the outcomes, we designed our proposed Automated Model Selection technique as a decision tree.

In the following section, we give details about steps 1, 2, and 3 of our methodology. Steps 4 and 6 are presented in Chapters 3 to 5 of the thesis.

Step 1. Reviewing of models. Various models can be used for time-series data forecasting. The

Autoregressive and Moving Average (ARMA) model brought forward by Yule, Slutsky, Walker and Yaglom is an important way to study time series [6]. Based on this, one of the most popular algorithms in time-series data prediction was proposed. In 1975, G. E. P. BOX et al. [7] proposed the Autoregressive Integrated Moving Average (ARIMA) due to the attention on how to prove the series is changed like what expected and get the magnitude of that. Moreover, a time-series forecasting tool called Prophet, which was developed to meet the demand of the business level data was released by Facebook [5]. Prophet took not only holidays or break intervals but also trends, outlier detection, and missing data into consideration [5]. One traditional method, Exponential Smoothing, which was created by Brown [10] in the 1950s to develop a tracking model for fire control information on the location of submarines, can also be used for time-series data. Besides, Deep Learning models are available for time-series data forecasting as well. Deep learning is a subset of machine learning that focuses on building models on the neural network architecture [2]. Some deep learning models are used in time-series areas such as LSTM, GRU, and Convolutional Neural Network (CNN). LSTM and GRU are 2 kinds of Recurrent Neural Network (RNN) sub-types. The unique characteristics of RNN is that it can keep information from past elements of a sequence and use it to process the next element by calculating a hidden state [2]. Unlike RNN, CNN applies convolutional operation, allowing it to create a reduced set of features [2]. CNN is the leading architecture behind some algorithms for image classification and segmentation, and it can also be used for time-series data. There are also some modifications of these models, like Bidirectional LSTM, Bidirectional GRU, and CNN_LSTM, which are the upgrade of traditional ones.

Step 2. Collection and reviewing of datasets. Beyond the models, we took into consideration

the possible characteristics of time-series datasets. Besides characteristics that also occur in other types of datasets, such as linearity, time series have some special characteristics. First, when we process a time-series dataset, it is necessary to check if it is stationary. Some models like ARIMA can only be used for stationary datasets. Also, another important feature of time-series datasets is that they may exhibit seasonality.

In the following, we collected different kinds of datasets. After a thorough search, we obtained 6 complete time-series datasets: AEP_hourly [11], Air_Passengers [2], Steel_industry_data [12], Canadian_climate_history [13], microdata [2], and DailyDelhiClimate [14]. AEP_hourly dataset is hourly power consumption data from PJM Interconnection LLC's website, and the data was recorded from the middle of 1998 to the end of 2001. Air_Passengers reflects the number of passengers in each month from December 1948 to December 1960. Steel_industry_data is the amount of energy consumption of a company which produces coils, steel plates, and iron plates. Related information on energy consumption is stored on the website of the Korea Electric Power Corporation. Canadian_climate_history and DailyDelhiClimate are both climate information in Canada and Delhi. The last one microdata is the actual gross domestic product of the United States from 1959 to 2009 [2].

Step 3. Choosing models and datasets. The next step of our methodology was to choose some models and datasets to conduct experiments in this research after collecting enough materials. Concerning models, we decided several kinds of deep learning models in the end. First, there are some limitations of other specific models. For example, ARIMA can only process stationary data. Deep learning models can be applied more generally, and they can not only tackle time-series data but also it is suitable for some other types of data which can help us enlarge the

functions of this technique in the future. At the same time, compared with other models, deep learning has the advantage that it tends to perform better as more data is available, so it shines when we have large complex datasets [2]. Among deep learning models, we divided 3 categories based on the architecture: regular deep learning networks, bidirectional networks, and hybrid networks. These 3 categories involve different types of neural networks and help us determine whether the architecture of networks affects the performances of tacking time-series data. In typical deep learning networks, we selected LSTM and GRU. LSTM is a powerful tool for solving sequence data and is particularly good at tasks with long-term dependencies, which makes it suitable for time-series data. GRU is a modification of LSTM but with a more straightforward structure. We can discover if the GRU can work well as LSTM in some tasks. Based on these 2 models, their corresponding bidirectional networks were chosen to compare with their performances. In the end, CNN-LSTM represents the hybrid neural networks category. CNN-LSTM is one of the most popular hybrid networks since it combines the capabilities of CNN for spatial feature extraction and LSTM for processing time-series data. Besides models, we also selected datasets to do experiments later among the 6 datasets we collected before. Since we focus on time-series datasets, we need the datasets to exhibit the primary and most essential characteristics of time-series datasets, which are stationarity and seasonality, which are adopted. After conducting the ADF Test (Test of stationarity, see Chapter 3) and decomposition on all datasets, we chose DailyDelhaClimate and Steel_industry_data. Beyond exhibiting the aforementioned characteristics, these 2 datasets were appropriate for our research because they involve data about climate and energy consumption, which may be affected by external factors. Thus, we can also explore if training with more features related to

external factors is favourable to the performances of models.

1.5 Outline of the Thesis

The outline of the thesis is as follows: Related work on automated selection techniques and the models we focus on, as well as metrics for performance, are discussed in Chapter 2. The design of the experimental study is discussed in Chapter 3. Chapter 4 discusses the implementation of experiments, including the methods and techniques we used in the training and predicting process. The results of the experiments are presented in Chapter 5, while Chapter 6 describes the outcomes and the design of the Automated Model Selection technique we proposed. Finally, Chapter 7 presents the conclusions and discusses future work.

Chapter 2

Related Work and Background

In this Chapter, related work and the background of the thesis are introduced. Section 2.1 mentions how other researchers applied the Automated Model Selection technique in ML models. Then, several algorithms that can be used for time-series data forecasting are introduced in Section 2.2. Finally, ML-related metrics that aim to evaluate the models' performance are shown in Section 2.3.

2.1 Related Work of Automated Model Selection

Automated model selection is a topic in ML research that has attracted a lot of interest since it promises to provide both convenience and efficiency in using ML models. In 1994, Yumi Iwasaki and Alon Y. Levy proposed an algorithm for selecting model fragments automatically for simulation [15]. They designed the algorithm based on relevance reasoning, which is used to determine which phenomenon can affect the query [15]. The model formulation algorithm consists of two steps: deciding which assumption classes should be represented first and then which composite model fragment should be included out of each [15]. The backward chaining achieves the first step, while the second is done by reasoning about the modeling assumptions necessary to answer the query. This algorithm provides an efficient way to generate an adequate and most straightforward model for simulation [15]. In addition, an R package glmulti, which aimed at selecting generalized linear models automatically, was designed and implemented by Vincent Calcagno in 2010 [16]. Glmulti can build all possible unique models based on a list of explanatory variables, what is more, it allows users to have a complete view of all candidate

models instead of providing the best model directly [16]. The basic part of this package is an enumerator, which returns all possible non-redundant formulas and helps specify some constraints. In 2016, Gustavo Malkomes et al. [17] employed Bayesian optimization for automated model selection. They constructed a novel kernel between models to explain a given dataset, which helps them find a model for the given dataset without human assistance [17]. They explored a space of infinite candidate kernels and quickly selected a promising model [17]. A novel "Kernel Kernel" is employed to capture the similarity in prior explanations that two models ascribe to a given dataset [17]. It is proved that modelling the evidence with a Guassian process in model space can predict the evidence value of unseen models with enough fidelity to effectively explore model space via Bayesian optimization [17]. Moreover, Lars Kotthoff et al. [18] released the source of Auto-WEKA, which is the addition of automatic selection technology to the original platform. They also used the Bayesian optimization method to help users identify the best approach for their particular datasets. The system treats all of WEKA as a single, highly parametric machine learning framework and uses Bayesian optimization to find a strong instantiation for a given dataset [18]. Moreover, the space of WEKA's algorithms space and hyperparameter spaces are combined and considered to minimize cross-validation loss. In recent years, Abdelhak Bentaleb et al. [19] proposed a kind of Automated Model for Prediction that is used to predict network bandwidth. They first learn and collect statistical and computational intelligence techniques to implement a suite of bandwidth prediction models that can work accurately under a broad range of network conditions [19]. In 2020, Yuanxiangyin et al. presented an automated model selection framework to find the most suitable model for time series anomaly detection [20]. They

achieved this by invoking a pre-trained model selector and a parameter estimator [20]. In 2022, Chunnan Wang et al. proposed an algorithm, AutoTS, which is used for designing the suitable forecasting model for the given time-series dataset [21]. Firstly, they constructed a search space by decomposing time-series forcasting models into 7 modules. Then they employed a two-stage purning and a knowledge graph analysis method to prune the search space and understand each component. In 2023, a framework for Automated model selection in natural language processing was created by Shehan Saleem and Sapna Kumarapathirage [22]. Shehan Saleem et al. collected datasets first to build a corpus od datasets to build a metadata set. Then, they conducted trials on 2 models (BOWRF and FastText) to select the best-performing models and evaluated the performance by F1 marco and time [22]. They also extracted several meta-features that are crucial for modelling and filtered these features by selecting those that had a strong correlation with the target class [22]. They conducted 1000 trials on machine learning models to select the best one for binary model selection and got results.

Totally, the related work of automated model selection can be summarized in Table 1:

Table 1 Summary of Related Work

Time & Author	**Method**
Yumi Iwasaki and Alon Y. Levy 1994	An algorithm for selecting model fragments automatically for simulation based on relevance reasoning
Calcagno V., & de Mazancourt C 2010	An R package glmulti which aimed at selecting generalized linear models automatically
Gustavo Malkomes et al. 2016	Bayesian optimization is implemented for automated model selection
Lars Kotthoff et al. 2017	Auto-WEKA, which is the addition of automatic selection technology to the original platform
Abdelhak Bentaleb et al. 2020	A kind of Automated Model for Prediction used to predict network bandwidth
Yuanxiangyin et al. 2020	An automated model selection framework to find the most suitable model for time series anomaly detection by invoking a pre-trained model selector and a parameter estimator
Chunnan Wang et al. 2022	AutoTS, which is used for designing the suitable forecasting model for the given time-series dataset
Shehan Saleem et al. 2023	AutoNLP, a framework for Automated model selection in natural language processing

It is recognized that the Automated Model Selection technique has been developed in various areas. However, there are a few techniques that target time-series data. The complexity of time-series data makes it challenging to develop an Automated Model selection for it. At the same time, the characteristics of the time-series datasets, such as seasonality, are not considered in its forecasting, and models are usually trained without this information. In this thesis, we propose a solution to address these issues.

2.2 Introduction of Algorithms

2.2.1 LSTM

Description. Long Short-Term Memory was first known by the public in 1997 when Sepp Hochreiter and Jurgen Schmidhuber published a paper called *Long Short-Term Memory* in Neural Comput. To solve the problem that using recurrent backpropagation to store information over long time intervals takes a long time, they analyzed the problem and proposed this efficient and gradient-based technology [8].

Long Short-Term Memory is a subtype of recurrent neural network. Recurrent neural networks can store information on recent input events in the form of activations using feedback connections, called short-term memory [8]. The architecture of the recurrent neural network is shown in Figure 2:

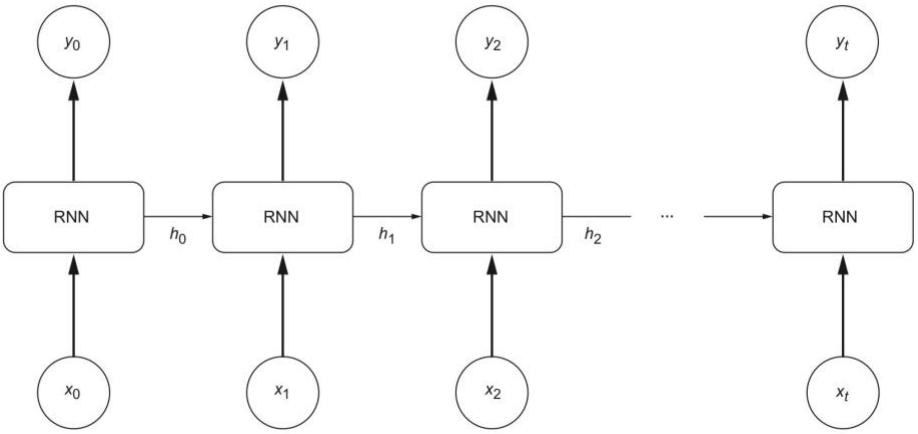


Figure 2 Architecture of RNN [2]

A recurrent neural network is a deep learning architecture especially adapted to processing data sequences [2]. From Figure 2, it is evident that each element has an input parameter and an output parameter. Once it gets the input, the network computes a hidden state and then feeds it to the next element [2]. In this way, the hidden state acts as a memory, helping the network compute the output using the previous information [2].

Different from recurrent neural networks, there is a new element in the Long Short-Term Memory architecture, and it is shown in Figure 3:

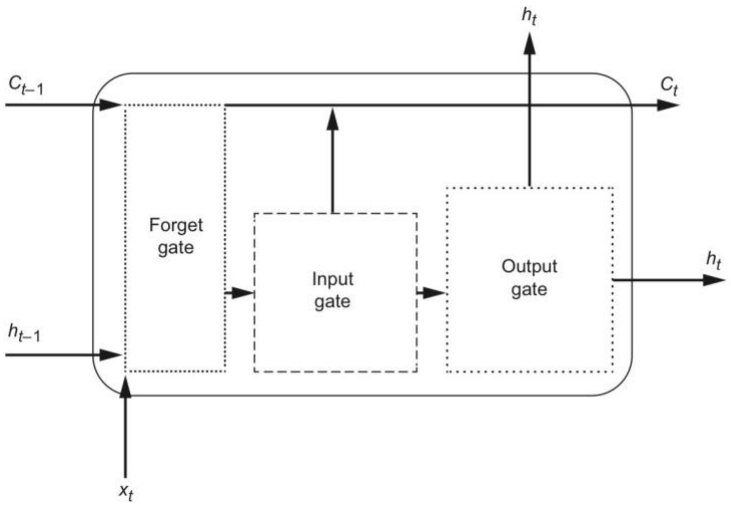


Figure 3 Architecture of LSTM [2]

In Figure 3, it is apparent that a new element is noted, which is C. The existence of C makes the difference between LSTM and traditional RNN. It is a cell state that is used to store past information for a longer time, which is why it is called long short-term memory. In LSTM, both the cell state and hidden state will be passed on to the next element in the sequence [2]. At the same time, there are 3 gates in the architecture of LSTM: forget gate, input gate, and output gate. The forget gate (shown in Figure 4) is used to determine whether the sequence information should be forgotten or kept in the network with the sigmoid function [2].

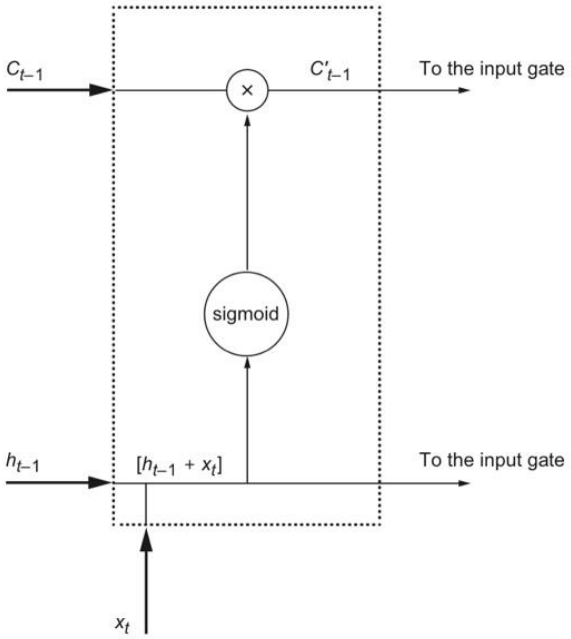


Figure 4 Architecture of Forget Gate [2]

The past hidden state h_{t-1} combined with the current value x_t will be duplicated. At the beginning. Then, one is passed to the input gate while another is sent to the sigmoid function. The definition of the sigmoid function is shown in (1) below:

$$f(x) = \frac{1}{1-e^{-x}} \quad (1)$$

The output of a sigmoid function is shown in Figure 5:

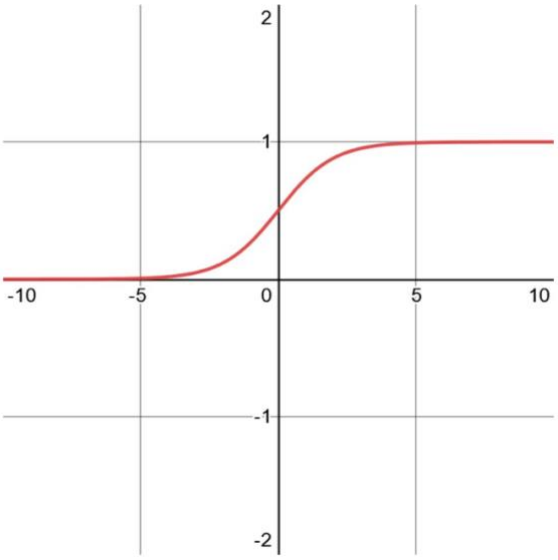


Figure 5 Sigmoid Function Definition [2]

From the vertical axis in the figure, it is clear that the sigmoid function outputs values between 0 and 1. Referring to the name of the forget gate, values is forgotten or kept in this gate according to certain rules. If the output of the sigmoid function is close to 0, the original value is forgotten. Otherwise, the value is kept. Then, the result of the function is combined with the previous cell state and the updated one is passed to the gate. The second gate is the input gate whose architecture is shown in Figure 6:

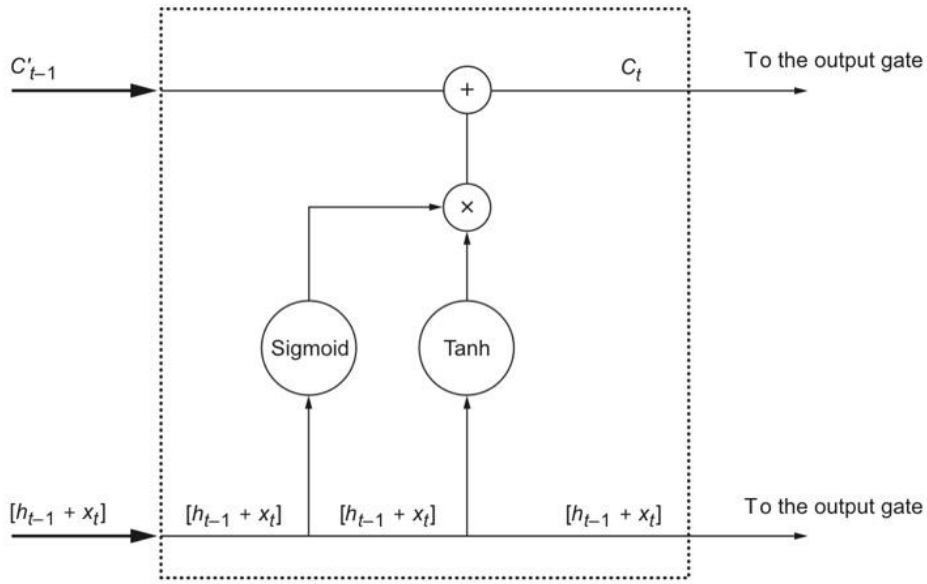


Figure 6 Architecture of Input Gate [5]

The purpose of the input gate is to determine whether the information is relevant to the current element of the sequence. Upon getting the 2 values from the forget gate, it creates 3 copies: one for the Sigmoid function, one for the Tanh function, and one for the output gate directly. Using the sigmoid function here also determines if the data is kept or not. The Tanh function is used to regulate the network, making sure that values do not get very large and ensuring that computation remains efficient [23]. The definition of the Tanh function is shown in Figure 7:

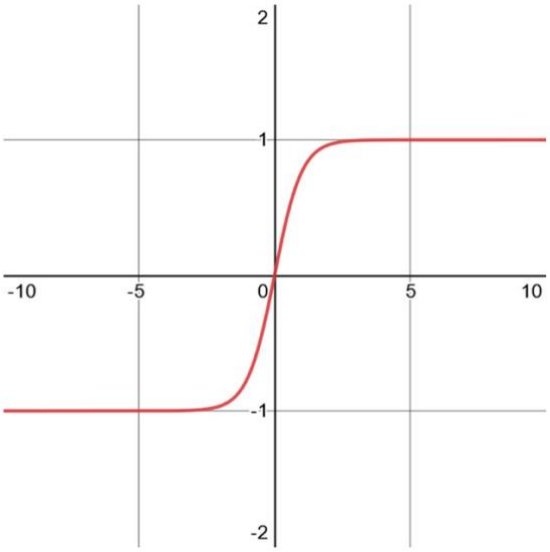


Figure 7 Definition of Tanh Function [20]

Figure 7 shows the output of the Tanh function between -1 and 1. It maps the production values into this interval so that the data can be normalized and the learning process is more stable. The result of the sigmoid function and Tanh function is multiplied by pointwise multiplication, and the result is used to update the cell state from the last time step. The last gate referred to is the output gate, which is shown in Figure 8:

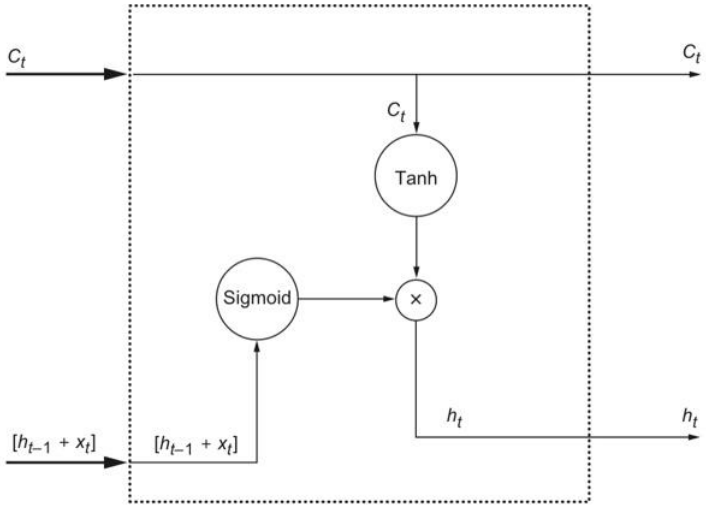


Figure 8 Architecture of Output Gate [2]

The output gate is the part where the past information in the cell state is used to process the current element. In this gate, the past hidden state and current element are sent to the Sigmoid function for the same purposes before, while the cell state is passed to the Tanh function. Then, these 2 results are combined, which is the step that pass information is used in the current element. After that, a new hidden state is generated based on the result after the combination. It proceeds to the next neuron or the output layer directly. At the same time, the cell state is also one part of the output.

In conclusion, there are a total of 3 gates, a hidden state, and a cell state in the architecture of LSTM. The forget gate determines whether the value is kept or discarded. The input gate aims to reserve the information related to the current element. The last output gate is the part where previous information acts on the current element of the sequence. The addition of cell state achieves the goal of long short-term memory which is the name of this type of architecture.

Discussion. One of the main advantages of LSTM is that it is long short-term memory which is helpful in keeping past information in memory for a long time [2]. At the same time, LSTM can effectively preserve the characteristics of historical information in long text sequences [24].

2.2.2 GRU

Description. Gated Recurrent Unit is also a modified Recurrent neural network model, which is much simpler than LSTM. In 2014, Kyunghyun Cho et al. [9] proposed a novel neural network model called RNN Encoder–Decoder which is consisted of 2 recurrent neural networks [9]. One is used to encode a sequence into a fixed-length vector representation, while another is for decoding the representation into another sequence [9]. GRU also uses some gates through

cells to tackle information, but there is no long short-term memory in it, and it just uses the hidden state to propagate the information [3].

On the one hand, GRU is similar to LSTM since it also uses a series of gates to regulate the information. However, it discards the function of long-term memory of LSTM to speed up the training process. So, there is only a hidden state in GRU that propagates information. Another difference with LSTM is that GRU consists of only 2 gates: the reset gate and the update gate. The architecture of GRU is shown in Figure 9:

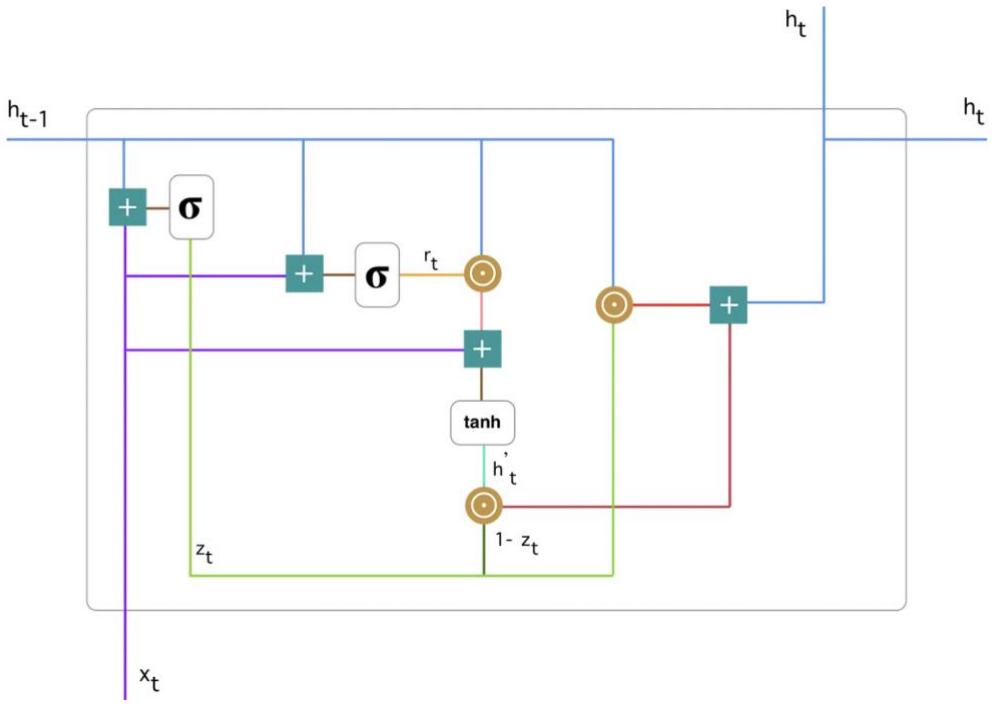


Figure 9 Architecture of GRU [25]

There are 2 parts in GRU. The update gate is responsible for determining whether the previous hidden state should be carried forward or not and whether the current potential hidden state should be written into a hidden state or not [3]. The equation of the reset gate is shown in (2):

$$z_t = \sigma(W^z x_t + U^z h_{t-1}) \quad (2)$$

The current element of sequence x_t is multiplied by its weight $W^z$for the network. At the

same time, the previous hidden state h_{t-1} is operated as the same: it is also multiplied by its weight U^z. After that, both of these results are fed to a Sigmoid function, which aims at controlling the output between 0 and 1. The update gate determines how much previous information is kept and passed to the future. That is powerful because the model can copy all the information from the past and eliminate the risk of the vanishing gradient problem [25]. After knowing that, we can find the update gate in Figure 10:

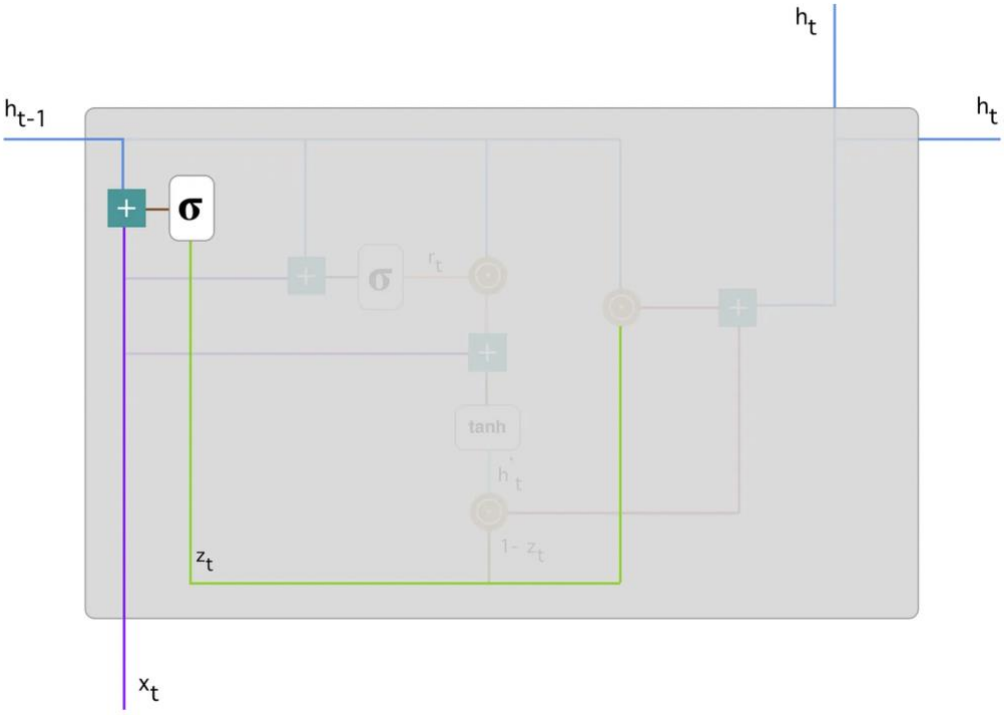


Figure 10 Update Gate in GRU [25]

Then, it comes to the reset gate. The reset gate is used for determining the amount of the previous hidden state that is considered as the potential hidden state of the current timestep, and it uses the Sigmoid function to achieve this [3]. The formula for the reset gate is shown in (3):

$$r_t = \sigma(W^r x_t + U^r h_{t-1}) \tag{3}$$

The function of the reset gate is the same as the formula of the update gate. The difference between them can be discovered in Figure 11:

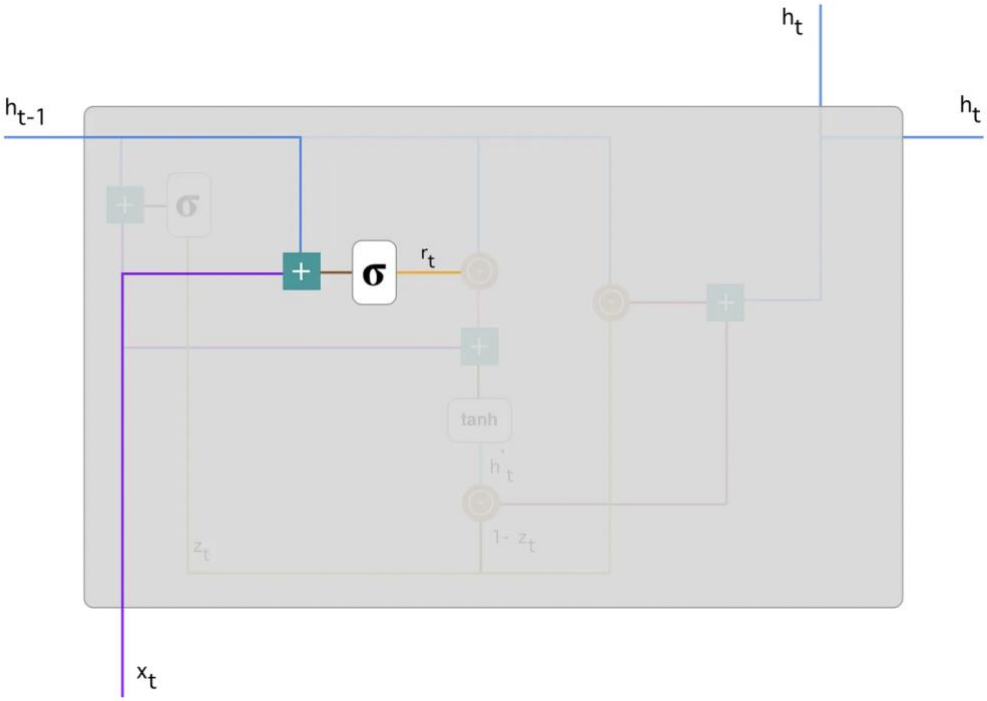


Figure 11 Reset Gate in GRU [25]

From Figure 11, it is evident that the current element x_t and previous hidden state h_{t-1} is duplicated at first. One is sent to the update gate, while the other is fed to the reset gate. They both obtain results by applying the Sigmoid function. After that, the current memory content h' is calculated by (4) below:

$$h'_t = tanh(Wx_t + r_i \odot Uh_{t-1}) \tag{4}$$

First, the current element x_t is multiplied by its weight W while the previous hidden state h_{t-1} also did the same operation. After that, the Hadamard product function is applied between the reset gate r_t and Uh_{t-1}. This step can help the model determine what needs to be removed from previous memory. Then, the sum of these 2 is fed to the Tanh function, which is the active function of this step. In the end, the hidden state of the current time step h_t is calculated and may be used in the next neuron. The function of h_t is shown in (5):

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{5}$$

The definition of h_t is the sum of Hadamard operation of update gate z_t and previous hidden state h_{t-1}, and the result of this operation for $(1 - z_t)$ and h'_t.

Discussion. The main strength of GRU comes from its structure since it only consists of 2 gates while there are 3 gates in LSTM. Therefore, GRU is more time-saving than LSTM in some instances. On the other hand, GRU is not as accurate as LSTM in tackling long sequences due to its structure. Meanwhile, GRU solves the vanishing gradient problem by saving related information instead of clearing the new input [25].

2.2.3 Bidirectional Networks

Description. Another category of models we have considered in this thesis is bidirectional networks. In this kind of model, the network analyzes the data in 2 directions: forward and backward, which can help the model understand the data in 2 directions. BiLSTM and BiGRU are 2 types of Bidirectional RNN. In 1997, Mike Schuster et al. [8] extended a regular recurrent network into a bidirectional recurrent network. The bidirectional RNN can be trained without the limitation of using input information just up to a preset future frame. It achieves training data simultaneously in positive and negative time directions. The general architecture of a bidirectional RNN is shown in Figure 12:

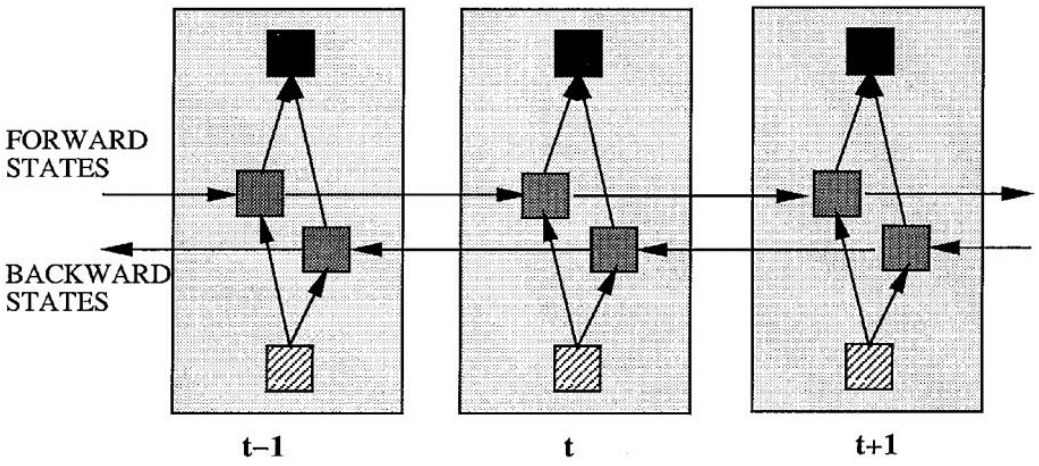


Figure 12 General Architecture of Bidirectional RNN [8]

Mike Schuster et al. split the normal state neuron in RNN into 2 parts: one is responsible for positive time direction while another part is in charge of negative time direction. At the same time, the output from each direction does not affect the other. In this way, the model can process the data in 2 directions simultaneously, which can analyze and learn more information and patterns of data.

Based on the creation of Bidirectional RNN, a series of bidirectional networks based on it were proposed, such as Bidirectional LSTM (BiLSTM) and Bidirectional GRU (BiGRU). In 2005, Alex Graves and Jurgen Schmidhuber first referred to and applied BiLSTM [26]. Before, recurrent neural networks were designed to analyze data in one direction only — the past [26]. An obvious solution is to put it in 2 directions: from past and future. Graves, A. and Schmidhuber, J proposed BiLSTM inspired by bidirectional networks created by Schuster and Baldi and applied it to the sequence processing tasks [26].

We first present unidirectional LSTM. The architecture of it, which is shown in Figure 13, is just like a recurrent neural network with LSTM cells [27].

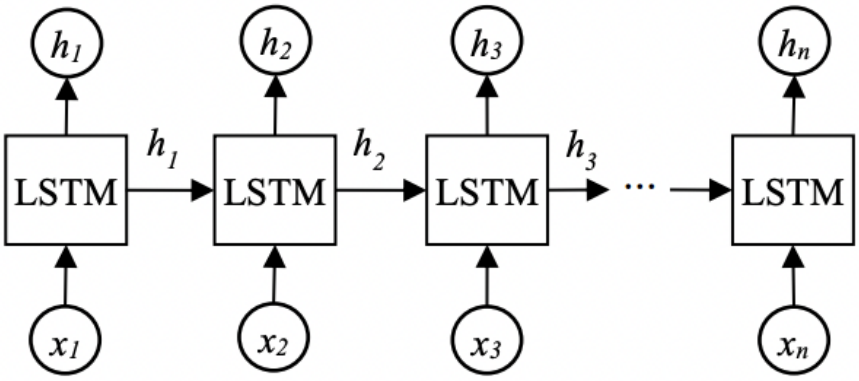


Figure 13 Architecture of Undirectional_LSTM [27]

It is absolutely true that the hidden state of an LSTM cell is passed to the next cell as input, which is just in one direction. In this way, the network can analyze previous data and get an output. The architecture of BiLSTM is shown in Figure 14:

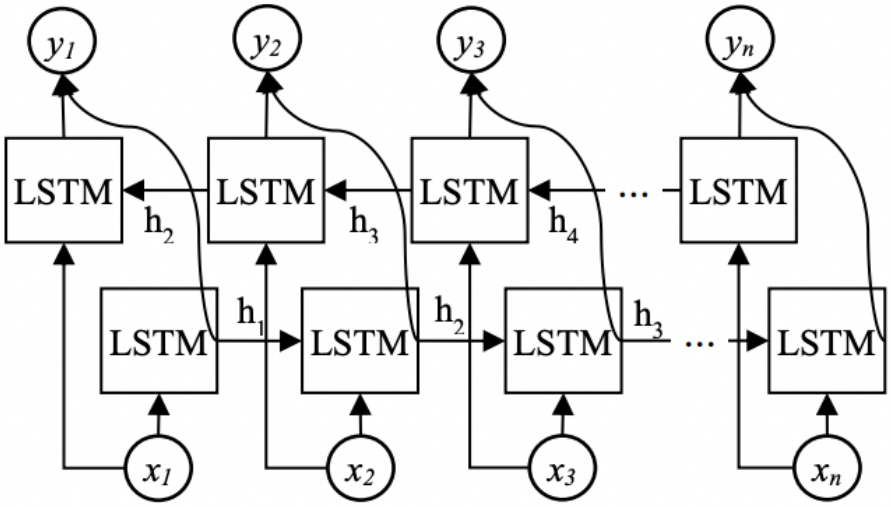


Figure 14 Architecture of BiLSTM [27]

In Figure 14, there are 2 layers of LSTM networks: one forward layer and another backward layer. The forward layer is used for a positive time direction, while the back forward one is responsible for a positive time direction [27]. The result y, which expresses predictions from 2 directions, is the concatenation of 2 layers' results.

Bidirectional_GRU (BiGRU) is also a type of bidirectional RNN like BiLSTM. It is a bidirectional network that consists of GRU units. BiGRU can help model data not only in the

forward direction but also in the backward direction. The characteristics of GRU make it more efficient for some instances of data analysis. BiGRU is widely used in different kinds of areas. Rui Lu et al. [28] proposed a multi-label BiGRU model in 2017, which is used to identify boundaries of sound events from acoustic recordings. In 2019, a type of BiGRU was applied for human identification from electrocardiogram-based biometrics by HTET MYET LYNN et al. [29].

Discussion. The straightforward advantage of both BiLSTM and BiGRU is that they are bidirectional networks, which means each component of the input sequence simultaneously has both the past and future information. In this way, the results will be more meaningful and accurate. However, this bidirectional setup makes training more time-consuming, which may result in the network being inefficient.

2.2.4 CNN-LSTM

Description. CNN-LSTM is a type of hybrid network that connects both Convolution Neural Networks (CNN) and LSTM. First, we introduce the CNN and then the CNN-LSTM network. CNN is a type of network which processes data in the form of a grid. This grid can be 2D, such as an image, 1D, such as a time series, 3D, such as data from LIDAR sensors, and so on [3]. In 1998, Yann Lecun et al. summarized some kinds of methods for handwritten character recognition. Then, they proposed a new method for document recognition area based on the gradient learning technique, which is a convolutional neural network (CNN) in the after-work [30]. With inspiration from the earlier works of Japanese computer scientist Kunihiko Fukushima, CNN was modelled after the brain's visual cortex, which is used to handle sight [31]. The network can produce a reduced set of features by convolutional operations, so it is an excellent way to prevent overfitting, regularize the network, etc. [2]. The main idea behind

CNN is convolution. There are 2 vital elements in the convolutional operation in CNN: inputs and kernel. Kernel is a matrix placed on top of the feature matrix [2]. Its size is much smaller than the input, and it can be slid into all positions of input. The way the kernel moving is shown in Figure 15:

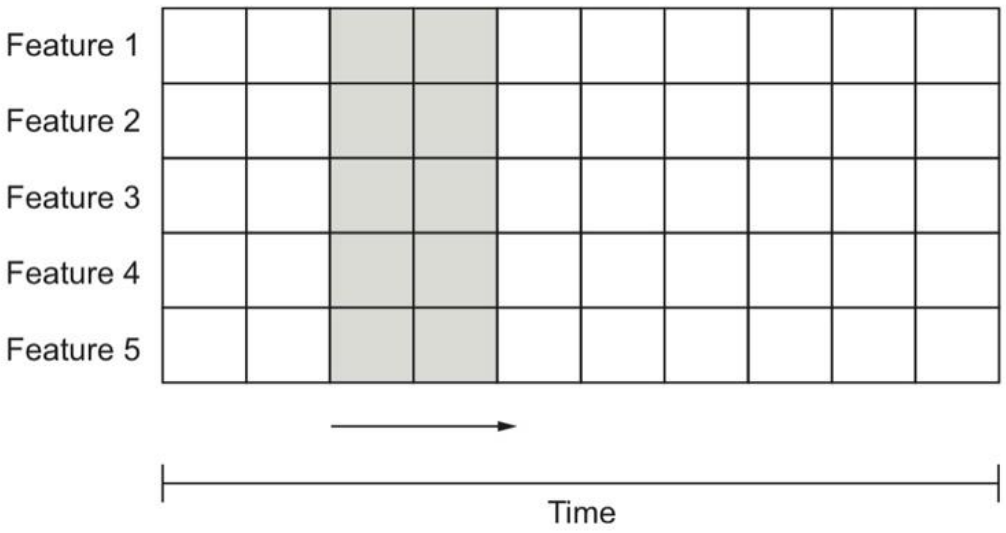


Figure 15 Kernel and Feature Map [2]

The kernel size is defined first, then it slips on the input features along the time axis. The results are a set of inputs in the datasets covered by the kernel. In this figure, the kernel moves in just one direction, and this operation is more common in time series prediction, which is called 1D convolution [2]. Besides the working principle of the kernel is shown in Figure 16:

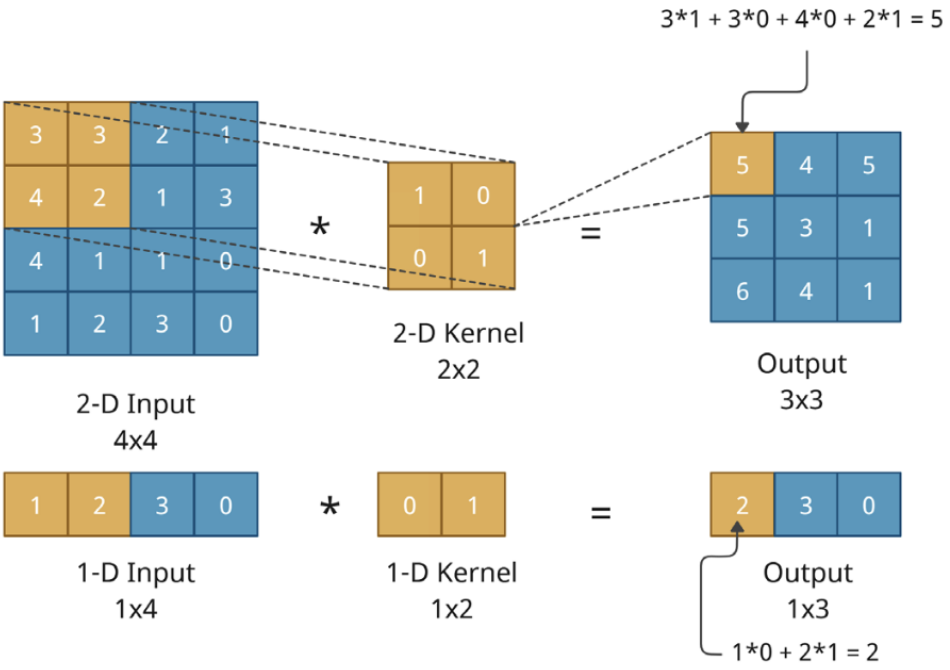


Figure 16 Calculation of Kernel [3]

In Figure 16, there are 2 kinds of kernels: 1-dimension kernel and 2-dimension one. It starts sliding at the edge of the input, then the element-wise multiplication is performed between the subset obtained by the kernel and the kernel itself. This step is repeated until the kernel processes all the positions of input.

Convolutional Neural Network architecture has been widely used in the computer vision area, especially in some algorithms for image classification and image segmentation [2]. At the same time, CNN is noise-resistant, so it is suitable for time-series data to filter out noise [2]. Moreover, its processing time is shorter than that of LSTM since its operation can be parallelized [2]. On the other hand, a lot of labelled training samples are needed for weight parameters learning in CNN, and also a powerful GPU is required in this process [32].

In recent research, more and more researchers use the combination of Convolutional Neural Networks and Long Short-Term Networks for prediction. In 2016, Jin Wang et al. analyzed dimensional sentiment by using a CNN-LSTM model [33]. In 2017, Amin Ullah et al. proposed

a model which combined CNN and Bidirectional-LSTM to recognize human action in video sequences [34][35]. CNN-LSTM has been widely used in many areas, especially in computer vision. It is well known that CNN is outstanding in feature engineering since it pays more attention to the most noticeable features in the line of sight [36]. At the same time, LSTM is used widely in time-series predicting since it will expand based on the sequence of time. So, it is simple to recognize that it will be excellent for predicting time series if combined. The architecture of CNN-LSTM is shown in Figure 17:

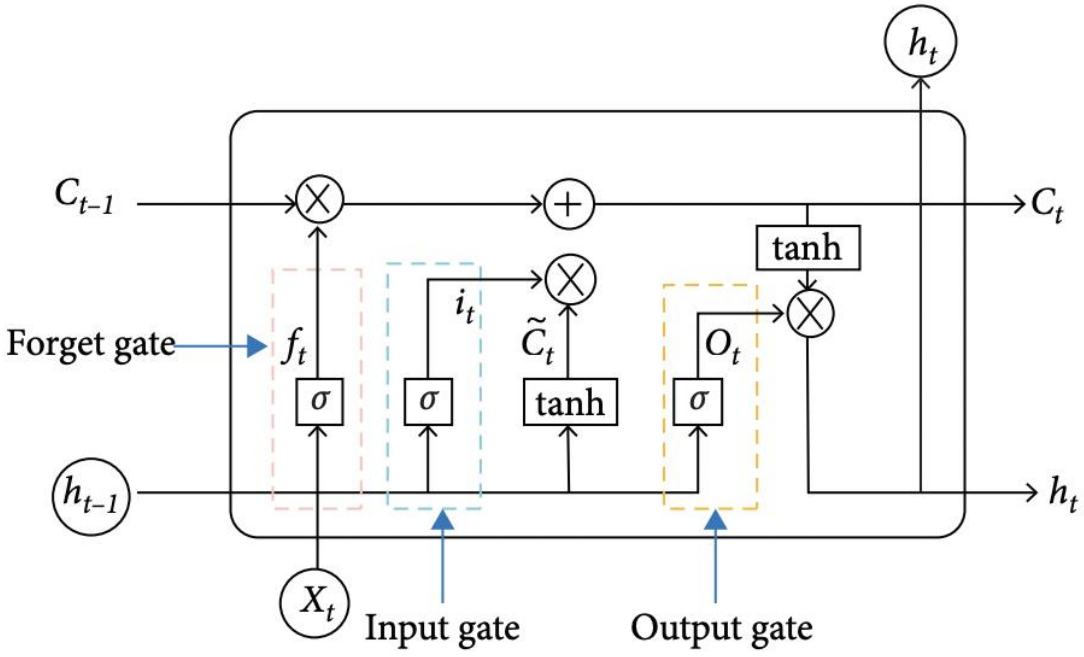


Figure 17 Architecture of CNN-LSTM [36]

The CNN-LSTM architecture combines a Convolutional Neural Network to extract features on input data and LSTM for prediction. The 2 algorithms of CNN-LSTM play different roles: CNN is responsible for feature engineering, while LSTM takes charge of forecasts. In this way, CNN-LSTM accumulates the advantages of both.

Discussion. CNN-LSTM model combines the advantages of both CNN and LSTM, which makes it able to perform timing analysis while extracting abstract features [37].

2.2.5 ARIMA

Description. Autoregressive Integrated Moving Average is one of the most popular algorithms in time-series data predicting, especially for linear data. In 1975, G. E. P. BOX and G. C. TIAO proposed the general function of ARIMA model due to the attention of how to prove the series is changed like what was expected and get the magnitude of that [7]. Autoregressive and Moving Average (ARMA) model proposed by Yule, Slutsky, Walker and Yaglom is a crucial way to study time series [6]. ARIMA was created based on ARMA, and the difference is that ARIMA differentiates the non-stationary data at first and transfers it into stationary data as data preparation.

ARIMA predicts future values based on previous values. There are 3 components in the definition of ARIMA: autoregressive process AR(p), integration I(d), and moving average MA(q). In AR(p), the output variable depends linearly on its previous values, which is shown in (6):

$$y_t = C + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \ldots \Phi_p y_{t-p} + \epsilon_t \quad (6)$$

C is a constant, ϵ_t is white noise, and the number of previous values that are taken into consideration will be dependent on p. In MA(q), the current value is linearly dependent on the current and past error terms (just like white noise), and it is shown in (7):

$$y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} \quad (7)$$

μ is the mean of the series. Based on these, the definition of ARIMA is shown in (8):

$$yt = C + \Phi_1 y_{t-1} + \Phi_2 y_{t-2} + \ldots \Phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_{t.} \quad (8)$$

At the same time, Integration I(d) is equal to the number of time series that have been differenced to become stationary.

Discussion. ARIMA is a linear regression-based forecasting approach that is more suitable for forecasting one-step out-of-sample forecast [38]. On the other hand, ARIMA is not embedded within any structural relations or theoretical model [39]. Moreover, it is 'backward-looking',

which means it is not good at predicting turning points [39].

2.2.6 Prophet

Description. In 2017, Facebook released the source of Prophet—a forecasting tool used in Python and R languages [5]. It was developed to meet the demands of the business level [5]. Prophet took not only holidays or break intervals but also trends, outlier detection, and missing data into consideration [5]. Now, it is widely used in time-series forecasting.

Different from ARIMA, Prophet focuses on change points other than old data. There are also 3 components in the Prophet: trend, seasonality, and holidays. Prophet is appropriate for datasets with seasonality. The formula of the Prophet is shown in (9):

$$y_t = g_t + s_t + h_t + \epsilon_t \tag{9}$$

Gt represents the overall trend of this dataset, and as mentioned above, Prophet concentrates on change points (the data which change directions). There are 3 kinds of growth functions: linear growth, logistic growth, and flat. The seasonality function is simply a function of time in Fourier Series. Moreover, Prophet detects the suitable number in the series automatically. As for holiday function, it considers a list of important holidays. Therefore, Prophet allows us to predict future data with the influence of some holidays.

Discussion. Prophet is based on the curve fitting technique in the Bayesian model [5]. Its parameters are easy to understand and determine, and it doesn't need a large amount of data for predicting [5]. All these points make it simple to implement. However, Facebook Prophet also has some limitations. It can only be applied in time-series predictions and only supports Python and R languages

2.3 Introduction of metrics

In order to evaluate the accuracy and efficiency of the models presented above, we need to use appropriate measures. Based on these metrics, we can compare different models and choose the best one for a specific dataset. At the same time, the metrics can provide us with insights which

we can employ to conduct improvements on the models.

2.3.1 Mean Absolute Percentage Error (MAPE)

The Mean Absolute Percentage Error reflects the difference between real value and predicting results. The formula is shown in (10):

$$M = \frac{1}{n} \sum_{t=1}^{n} \left| \frac{At - Ft}{At} \right| \quad (10)$$

MAPE is defined as the sum of the individual absolute forecast errors, divided by the actual values for each period. However, there is problematic situation: if one of the actual values is zero, the MAPE metric is not computable, since the denominator is zero. In this case, it is better to choose some other metric instead of changing the datasets. Nevertheless, it is in general suitable metric to measure the accuracy of the predictions.

2.3.2 Mean Squared Error (MSE)

Mean Squared Error measures the average of the squares of the difference between the actual value and predicted value. The formula is shown in (11):

$$MSE = \frac{1}{n} \sum_{t=1}^{n} (At - Ft)^2 \quad (11)$$

MSE is a metric of the quality of an estimator [40]. It is always a positive value, which will be smaller if the forecasting is more accurate.

2.3.3 Mean Absolute Error (MAE)

MAE measures errors between paired observations expressing the same phenomenon [41]. The definition of MAE is shown in (12):

$$\frac{1}{n} \sum_{t=1}^{n} |A_t - F_t| \quad (12)$$

It is an arithmetic average of the absolute error, which uses the same scale as the data being

measured [41].

Chapter 3

Design of the Experimental Study

This chapter introduces the design of the experiments in the thesis. The datasets we used are mentioned in Section 3.1. Then, Section 3.2 presents the related parameters in experiments, including the parameters used for dividing datasets and parameters in each model. Section 3.3 is the metrics that are employed to evaluate the performance of the models. Finally, the design of the experiments is shown in Section 3.4.

3.1 Dataset

We have conducted experiments using 2 datasets with different characteristics. These datasets are dataset 1: Steel_industry_data and dataset 2: DailyDelhiClimateTrain. They are all time-series data, and with the features of seasonality, whether it is stationary, respectively. In this section, a series of elements, like size, meaning, and characteristics etc., of these 2 datasets are introduced.

3.1.1 Steel_industry_data

Steel_industry_data is a dataset about the energy consumption in the steel industry. This dataset is sourced from the UCI Machine Learning Repository. The schema of this dataset is shown in Table 2:

Table 2 Head of Steel_industry_data

date	Usage_kWh	Lagging_Current_Reactive.Power_kVarh	Leading_Current_Reactive_Power_kVarh	CO2(tCO2)	Lagging_Current_Power_Factor	Leading_Current_Power_Factor	NSM	WeekStatus	Day_of_week	Load_Type
01/01/2018 00:15	3.17	2.95	0	0	73.21	100	900	Weekday	Monday	Light_Load
01/01/2018 00:30	4	4.46	0	0	66.77	100	1800	Weekday	Monday	Light_Load
01/01/2018 00:45	3.24	3.28	0	0	70.28	100	2700	Weekday	Monday	Light_Load
01/01/2018 01:00	3.31	3.56	0	0	68.09	100	3600	Weekday	Monday	Light_Load
01/01/2018 01:15	3.82	4.5	0	0	64.72	100	4500	Weekday	Monday	Light_Load

There are 11 features in the dataset. The date is continuous from 01/01/2018 00:15, recorded every 15 minutes. Usage_kWh, Lagging_Current_reactive, Leading_Current_reactive, CO2, Lagging_Current_Power_Factor, Leading_Current_Power_Factor all are different elements of steel energy evaluation. NSM means 'Number of Seconds' from midnight. WeekStatus and Day_of_Week both represent the date status. Finally, Load_Type is a category of light load, medium load, and maximum load. Steel_industry_data dataset is stationary since its ADF test result is shown in Figure 18:

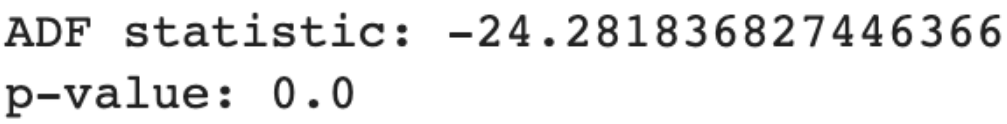


Figure 18 ADF Test of Steel_industry_data

3.1.2 DailyDelhiClimateTrain

The DailyDelhiClimateTrain dataset provides users with the Delhi climate from January 1st, 2013, to April 24th, 2017. This dataset is collected from Weather Underground API. There are 5 features in the dataset, which is shown in Table 3:

Table 3 Head of DailyDelhiClimateTrain

date	meantemp	humidity	wind_speed	meanpressure
2013-01-01	10.0	84.5	0.0	1015.6666666666700
2013-01-02	7.4	92.0	2.98	1017.8
2013-01-03	7.166666666666670	87.0	4.633333333333330	1018.6666666666700
2013-01-04	8.666666666666670	71.33333333333330	1.2333333333333300	1017.1666666666700
2013-01-05	6.0	86.83333333333330	3.7000000000000000	1016.5
2013-01-06	7.0	82.8	1.48	1018.0
2013-01-07	7.0	78.6	6.3	1020.0
2013-01-08	8.857142857142860	63.714285714285700	7.142857142857140	1018.7142857142900
2013-01-09	14.0	51.25	12.5	1017.0
2013-01-10	11.0	62.0	7.4000000000000000	1015.6666666666700
2013-01-11	15.714285714285700	51.285714285714300	10.571428571428600	1016.1428571428600
2013-01-12	14.0	74.0	13.228571428571400	1015.5714285714300
2013-01-13	15.833333333333300	75.16666666666670	4.633333333333330	1013.3333333333300
2013-01-14	12.833333333333300	88.16666666666670	0.6166666666666670	1015.1666666666700
2013-01-15	14.714285714285700	71.85714285714290	0.5285714285714290	1015.8571428571400

From Table 3, we can see that this dataset recorded 4 elements of climate in Delhi: mean temperature, humidity, wind speed, and mean pressure. These 4 parameters will help people grasp the climate characteristics and predict future data based on these. In this experiment, this dataset represents the datasets with seasonality feature. The climate data in a city is a little bit regular each year, and the seasonal composition in STL of the dataset is also shown in Figure 19:

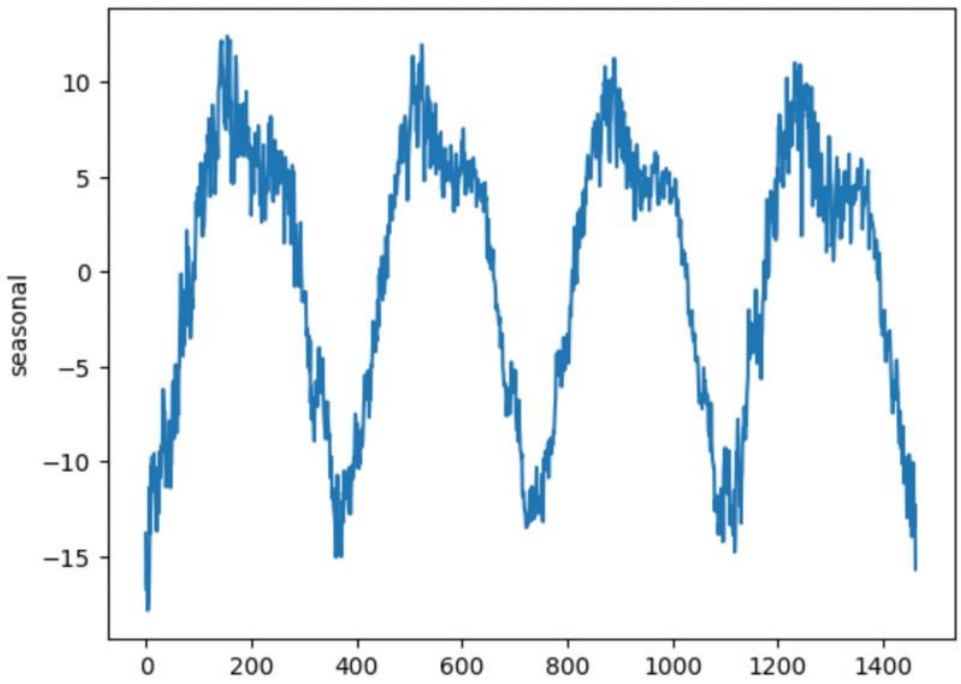


Figure 19 Seasonal Composition of DailyDelhiClimateTrain

3.2 Parameters

3.2.1 Training Size

For the stationary dataset Steel_industry_data, the value of attribute Usage_kWh is between 4 and 160, a huge span. The total size of this dataset is 35040, and they are separated into 3 parts: train, validation, and test sets. The train set is used to train the model and help it learn the pattern and relationship of the data, so there should be enough data in it. The training size is the first 30000 data, which covers almost 85% of data. Then, the validation set covers the following

2000 data. The purpose of the validation set is to avoid overfitting and inspect the model's performance at the end of each epoch since it is not used in the training process. The validation set will also be utilized to monitor the training process, and the training may stop early if the metrics on the validation set are not improved after several epochs. In the end, the last 3040 data is a test set that aims to evaluate the model.

The phenomenon is different in another dataset, DailyDelhaClimate. This dataset is smaller than the previous one, as there are only 1462 data rows. We used these 2 datasets to discover if the dataset size will affect the model and parameter selection. In the DailyDelhiClimate dataset, the first 1032 data is a train set, which is 95% of the whole dataset. Most of the data is used for training the models, and in this way, it can guarantee that models can acquire sufficient information from the data. The following 430 data is divided into 2 parts: validation set (100 data in total) and test set (330 data).

3.2.2 Parameters in Algorithms

Parameters and structure are 2 basic components of models. While implementing a series of models, one of the most essential things is determining the value of each parameter in the model. In this thesis, there are 3 kinds of networks used in experiments: regular Deep Learning networks (LSTM & GRU), bidirectional networks (BiLSTM & BiGRU), and a hybrid network (CNN-LSTM). In this section, the parameters in the models above are introduced.

In the setup of LSTM, there are 3 LSTM layers, 3 Dropout layers, and 1 Dense layer. A multi-layer LSTM network can learn more complex data patterns. Each layer can be built on top of the features extracted in the previous layer, allowing the model to capture more complicated features, patterns, and relationships. This kind of design comes with a problem, however, it

increases the risk of overfitting since models may learn some random errors and outliers in the dataset. So, a Dropout layer is used after each layer of LSTM. It drops some predictions randomly and avoids overfitting to a certain extent. Inside the definition of the LSTM layer, there are 4 parameters mentioned: the number of neurons, activation function, input shape, and return_sequences. The number of neurons determines the output dimension of each LSTM unit, and more neurons can increase the complexity and learning ability of the model but also increase the risk of overfitting and computational burden at the same time. After a large number of experiments of testing and adjusting, we decided on the value 40. The second one is the activation function, which is used to determine whether or not the neurons are activated. In this experiment, we used the Tanh function, the output of which is between -1 and 1. Its production is zero-centred, which helps the data maintain a more stable distribution at each model layer. The definition of the third parameter input shape depends on the shape of the input, and its first parameter specifies the length of the sequence, like how many time steps or data points are in each input, while the second one represents the number of features that will be utilized in the training process. Lastly, the return_sequences parameter means the output of the LSTM layer will return a sequence for each time step or only the last output of the sequence. It is vital in multi-layer networks, and its value is true except in the last layer.

GRU's parameters are similar to those of LSTM since it uses the same architecture. It also consists of 3 layers of GRU networks, each following a layer of Dropout separately. Referring to each layer, each GRU network uses 40 units to calculate as well, and it has the same activation function, input shape, and return_sequences value.

The obvious difference between bidirectional models is that they are implemented with

bidirectional networks that process both forward and backward information simultaneously. In BiGRU and BiLSTM, there are both 2 layers of bidirectional networks that are charged with calculating, extracting features, etc.

The last category is the hybrid network, which is the combination of 2 different types of machine learning models. In the CNN-LSTM model, one CNN layer and 2 LSTM layers are implemented. The definition of LSTM networks is the same as before, while the setup of CNN is distinct. 4 parameters are involved in CNN: the number of filters, kernel size, activation function, and input shape. The function of filters is extracting features while each filter independently extracts features from the input data. According to this, we need to consider the complexity of data when determining the number of it because the more complex the data is, the more filters are needed. In addition, we can use fewer filters in each layer in multi-layer networks since features are progressively refined across multiple layers. In this research, we defined it as 32 after experiments. Then, the kernel size is 1, representing the number of time steps covered by each convolutional kernel is 1. In this way, the convolution operation is performed point by point.

3.3 Metrics

In our experimental study, we used metrics to evaluate the models in terms of 2 aspects: time and accuracy. First, the most important purpose of the experiments is accuracy—we intend to find the most suitable algorithms for datasets with different characteristics. In this work, we use 2 metrics of accuracy: the Mean Absolute Percentage Error (MAPE) and the Mean Squared Error (MSE). MAPE calculates the relative error between predicted and actual values to assess the accuracy of a model, and the expression of it is a percentage which is simple to understand

the meaning of the result. However, the calculation of MAPE may be affected if there is a zero in the input data since the denominator should be a non-zero value. MSE is the difference's square, meaning it is always a non-negative number. One limitation is that it will magnify errors, especially for that large one because it is the square of the difference. In the same way, MSE may be sensitive to the outliers. MAPE and MSE are employed in both the training and the test phases to measure the accuracy of the results and help us understand the suitability of models for the input datasets. Beyond these 2 metrics, we also measured the epochs in each training process of models. The latter is a convenient metric to measure the training time, and it shows how many epochs the models trained, which is a measurement of processing time. It can help us discover the efficiency of the models.

3.4 Design of Experiments

5 kinds of neural network models are implemented in this thesis. These models are used to process datasets in 8 kinds of phenomena to analyze if different horizons and number of features will affect the performance of the models. 8 experiments in each model are shown in Table 4:

Table 4 Experiments in Each Model

Models	Experiments
LSTM	1 feature + horizon (1)
Bi-LSTM	1 feature + horizon (5)
GRU	1 feature + date features + horizon (1)
Bi-GRU	1 feature + date features + horizon (5)
CNN-LSTM	2 features + horizon (1)
	2 features + horizon (5)
	2 features + date features + horizon (1)
	2 features + date features + horizon (5)

In the table, the horizon is the length of time for which forecasts are generated. A timestep will produce one prediction if the horizon is 1. Forecasts are more accurate for shorter than longer time horizons, and the shorter the time horizon of the forecast, the lower the degree of uncertainty [42]. Setting up 2 scenarios of the horizon can give us a direct answer to how it influences the results. Another factor it considered is date features. Energy-related data is time-series data since it is collected from some sensors in factories every minute, every hour, or every day. These data will come with a datetime which shows when the data is received. In time-series data analysis, time is an important component that should be taken into consideration, and some time-related patterns are also essential. For example, using electricity may present some kinds of seasonality, like more in summer and less in winter. According to this, using date features in forecasting will be conducive to the accuracy of results. DateTime feature engineering is the process of creating new features from date and time information in order to

improve predictive accuracy in machine learning models [43]. In this thesis, date features mean creating several new features like a year, month, day, day of the week, and day of the year by time feature in the original dataset. These features can provide us with more information about seasonality, trends, and some special dates like Christmas.

Moreover, applying more than one feature while training the model may positively impact the prediction. For instance, in a factory, energy consumption may result in a change of temperature inside, so the temperature data contributes to predicting the amount of energy. Based on this, not only the main feature, but another feature will also be kept in mind in this thesis while finishing the forecasting.

Therefore, it is clear that these experiments consider several factors, such as horizon, date features, and the number of features. These 8 kinds of experiments will be implemented in each model to find out if these elements will impact the training and prediction. On the other hand, a series of different models is also taken into account. Implementing different models in 2 datasets with different characteristics is an understandable way to explore the suitability and relation between them.

Chapter 4

Description of Model Implementation

Chapter 4 introduces the model implementation of the thesis. Section 4.1 is about data scaling and inversing, which are the preparations for model implementation. Then, the models we chose and their structures are introduced in Section 4.2. Section 4.3 mentions the data window, which is the method we adopted to format the data. Finally, the training and testing process, including dataset splitting, model fitting and compiling, are shown in Section 4.4.

4.1 Scaling and Inversing

Normalization and inverse normalization are 2 kinds of data preprocessing techniques commonly used in machine learning to transform raw data into a specific range or distribution and restore it to the original data when needed. In this thesis, these 2 methods are used to process the input dataset and predictions.

4.1.1 Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data. Data processing is also known as data normalization and is generally performed during the data preprocessing step [44]. Scaling the data can help to balance the impact of all variables on the distance calculation and can help to improve the performance of the algorithm [45]. In some deep learning models like LSTM and GRU, scaling the dataset is a necessary step that can help increase accuracy and efficiency. At first, different features may have various ranges of units, resulting in those with higher values having a more significant impact on the loss function. This can be avoided by scaling since the features will have similar ranges, which can help speed

up the model training process. On the other hand, scaling can prevent the training process from gradient vanishing or explosion by ensuring the input data is in a small range. So, scaling the input dataset is indispensable in neural networks.

Feature scaling in machine learning is one of the most critical steps during the pre-processing of data before creating a machine learning model [46]. 2 common methods to do the scaling are normalization and standardization. The difference between these 2 ways is shown in Figure 20:

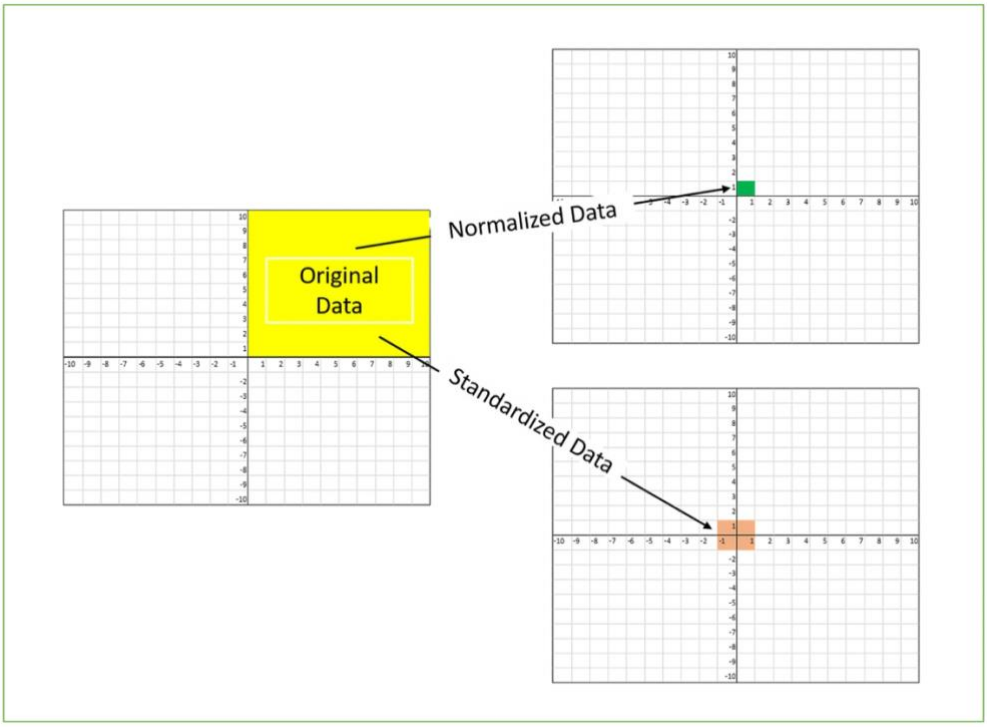


Figure 20 Difference of Normalization and Standardization [46]

From Figure 20, normalization is used to bound the original values between 2 numbers, such as [0,1] or [-1,1], while standardization transforms the data to have a zero mean and a variance of 1. Normalization maintains the sparse structure of the sparse data, and it will not change the distribution of the data. However, it is very sensitive to outliers, and an extreme maximum or minimum value may cause all other values to cluster in a small interval. Standardization can solve this problem since the data will be measured in standard deviation units. However, it will

not retain the original values and zeros in the data. In conclusion, users should choose a suitable scaling method based on their data and models.

This thesis uses the Min-Max scaler in normalization to scale the input dataset. Min-Max normalization is a simple technique that can specifically fit the data in a pre-defined boundary with a pre-defined boundary [47]. The definition of the Min-Max scaler is shown (13) below:

$$A' = \left(\frac{A - \min value\ of\ A}{\max value\ of\ A - \min value\ of\ A}\right) * (D - C) + C \quad (13)$$

In this function, [C, D] is the defined boundary for scaling, and A is the range of original data while B is the mapped data. Min-max scaling maintains the relative relationship of points in the data through a simple linear transformation, so it preserves the original structure of the data.

The implementation of scaling in this thesis is shown in Figure 21:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled = scaler.fit_transform(data.reshape(-1,1))
```

Figure 21 Implementation of MinMaxSacaler

Implementing MinMaxScaler is simple since there is a MinMaxScaler class in the preprocessing class of the sklearn library so that it can be imported directly. What we should only do is fit the data we want to scale on the scaler and then define the boundary of scaling. On the other hand, one technique used in experiments with date features and 2 features is defining several scalers: one for the target feature and others for some other features. In this way, it will be right and understandable while inversing the scaled data. The definition of different scalers is presented in figure 22 and 23 below:

```
scaler_date = MinMaxScaler()
scaled_date = scaler_date.fit_transform(df[date])
scaler_target = MinMaxScaler()
scaled_target = scaler_target.fit_transform(target.reshape(-1,1))
```

Figure 22 Scalers in Date Features Experiments

```
scaler_feature = MinMaxScaler()
scaled_feature = scaler_feature.fit_transform(feature.reshape(-1,1))
scaler_target = MinMaxScaler()
scaled_target = scaler_target.fit_transform(target.reshape(-1,1))
```

Figure 23 Scalers in 2 Features Experiments

Figure 22 shows 2 Min-Max scalers: one for date features and another for target. In this case, we can use scaler_target only to reverse the predictions and do some evaluation. It is the same while processing more than one feature. In those experiments tackling 2 features, 2 scalers are used to complete the scaling step separately. This method can help us save more time at the end of the experiments where the scaled data need to be inversed.

4.1.2 Inversing

Normalization is usually performed on the training data in order to normalize the range of independent variables or features of data. After that, it is needed to convert the scaled data back to the original data range for subsequent analysis and interpretation. This process is called inversing which usually happens after predicting and before evaluation of the model. The important point is that the scaler for scaling before should be used while inversing to guarantee the predictions will be transformed to the original range.

The inversing step in the thesis is shown in Figures 24 and 25:

```
trainpredictions = scaler.inverse_transform(trainpredictions)
train = scaler.inverse_transform(trainY.reshape(-1,1))
testpredictions = scaler.inverse_transform(testpredictions)
test = scaler.inverse_transform(testY.reshape(-1,1))
```

Figure 24 Inversing with the Same Scalers

```
trainpredictions = scaler_target.inverse_transform(trainpredictions)
testpredictions = scaler_target.inverse_transform(testpredictions)
train = scaler_target.inverse_transform(trainY.reshape(-1,1))
test = scaler_target.inverse_transform(testY.reshape(-1,1))
```

Figure 25 Inversing with Different Scalers

In the first experiment, there is only one feature which is scaled, so only one scaler can be used in inversing the predictions. However, the situation is different in the second experiment. There are 2 kinds of features that are scaled at the beginning: one scaler called scaler_date is for date features, and another one called scaler_target is for the target values. So, the scaler_target should be used in inversing the predictions instead of the scaler_date. After that, it is uncomplicated to finish the following evaluation step.

4.2 Models

A total of 5 kinds of models are considered in this thesis: LSTM, Bidirectional-LSTM, GRU, Bidirectional-GRU, and CNN-LSTM. These models are implemented with different architectures to complete the forecasting. They will all be demonstrated in this section.

4.2.1 LSTM

RNN has been widely used in sequential data areas and is efficient and precise in predicting. However, RNNs consisting of sigma cells or tanh cells are unable to learn the relevant information of input data when the input gap is large [48]. Long Short-Term Memory was created to handle this problem of long-term dependencies by introducing gate function into cell structure. Based on this, almost all the exciting results based on RNN have been achieved by

the LSTM [48]. In this thesis, we used 8 kinds of different LSTM to tackle time-series data and complete the experiments. The original code and structure of the network in the first experiment, which processes one feature and horizon value of 1 is shown in Figures 26 and 27:

```
model_LSTM1 = Sequential()
model_LSTM1.add(LSTM(40,activation="tanh", input_shape=(look_back, 1), return_sequences=True))
model_LSTM1.add(Dropout(0.15))
model_LSTM1.add(LSTM(40,activation="tanh",return_sequences=True))
model_LSTM1.add(Dropout(0.15))
model_LSTM1.add(LSTM(40,activation="tanh",return_sequences=False))
model_LSTM1.add(Dropout(0.15))
model_LSTM1.add(Dense(1))
```

Figure 26 Setting of LSTM Model in Experiment 1

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10, 40)	6720
dropout (Dropout)	(None, 10, 40)	0
lstm_1 (LSTM)	(None, 10, 40)	12960
dropout_1 (Dropout)	(None, 10, 40)	0
lstm_2 (LSTM)	(None, 40)	12960
dropout_2 (Dropout)	(None, 40)	0
dense (Dense)	(None, 1)	41

Figure 27 LSTM Architecture in Experiment 1

First, a sequential model of linearly stacked layers in Keras was initialized. Based on this model, data flows from one layer to the next. There are 3 layers of LSTM, and the first 2 are the same in parameter setting. In these 2 layers, each one contains 40 neurons, and the activation function is hyperbolic tangent (tanh). The location of an activation function in a neuron is shown in Figure 28:

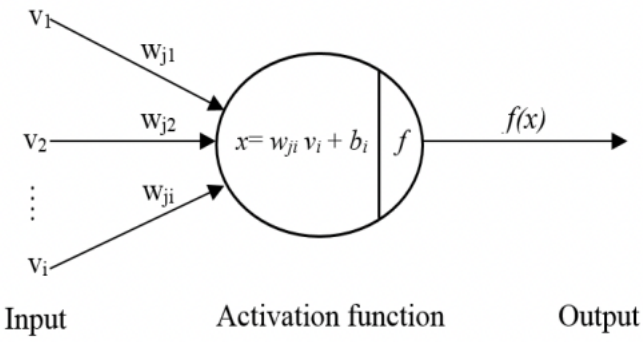


Figure 28 Location of Activation Function [49]

The activation function is used to determine whether a neuron is activated. The hyperbolic tangent function is a variant of the sigmoid function, and it maps values between -1 and 1. Then, the input shape for the first layer is [look-back, 1], which represents the step size of the time-series and the number of features per timestep (it is 1 in experiment 1) separately. Finally, the definition of parameter return_sequences is true, indicating that each step output from this layer is output to the next layer. However, it is false in the last LSTM layer since this layer only needs to output the output of the last timestep and prepare for the final output prediction. Moreover, there are also 3 layers of Dropout after each LSTM layer. Dropout regularization randomly discards the output of neurons at a 15% rate to reduce overfitting. The last one is the Dense layer, whose value is 1, and its purpose is to reduce the output of the LSTM to a single predicted value. The design of this LSTM network helps to capture long-term dependencies in time-series data, and it also avoids overfitting to a certain extent.

The summary of LSTM networks is similar to the first one, and the only differences are in the input_shape and Dense layers. In experiments about date features and more than one feature, the second parameter of input_shape should be changed due to the number of features in total. For instance, the setting of input_shape is [look_back, 6] if the experiment explores the effect of one feature and date features. Moreover, the parameter of the Dense layer should be set up

due to the horizon because it determines how many predictions will be outputted in one timestep in the end.

4.2.2 Bidirectional_LSTM

Bidirectional LSTM is a variant of LSTM, combining 2 separate LSTM networks. In a BiLSTM, one LSTM network is responsible for the forward direction of data (from the past to the future), while another one is in charge of the reverse direction of the data (from the future to the past). The output of each time point is generated by these 2 networks separately, and the output of the whole BiLSTM model is the combination of the results of 2 units. The structure of BiLSTM helps it obtain both the past and future information of a sequence, so it is more powerful while processing data than a traditional LSTM network. In this thesis, the BiLSTM network also is applied to 2 datasets with 8 experiments, and the code and network summary is shown in Figure 29 and Figure 30 below:

```
model_bilstm1 = Sequential()
model_bilstm1.add(Bidirectional(LSTM(40, return_sequences=True), input_shape = (look_back, 1)))
model_bilstm1.add(Bidirectional(LSTM(40)))
model_bilstm1.add(Dense(1))
model_bilstm1.add(keras.layers.Activation('tanh'))
```

Figure 29 Setting of BiLSTM Model in Experiment 1

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirection  (None, 10, 80)            13440
 al)

 bidirectional_1 (Bidirecti  (None, 80)                38720
 onal)

 dense (Dense)               (None, 1)                 81

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 52241 (204.07 KB)
Trainable params: 52241 (204.07 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 30 BiLSTM Architecture in Experiment 1

Different from the setup of the LSTM network, there are a total of 2 layers of BiLSTM. Both of them will create 2 LSTM sublayers: one for the forward direction and another for the reverse. The number of neurons is 40, which is determined by the complexity of both the model and data. It is necessary to apply some experiments and adaptation to determine how many neurons will be used. Choosing a suitable number of neurons can help users avoid overfitting and underfitting. For the first layer, the value of return_sequences is true, demonstrating that this layer will return an output for each timestep. However, in the second layer, there is no definition for this parameter, which means this layer will only return the production of the last timestep. The setup of input_shape is the same as it is in the LSTM network, which represents the step size of the time series and the number of features per timestep. The last layer is the Dense layer, which will prepare the output for the whole network. One apparent difference between the codes of LSTM and BiLSTM is that the definition of activation function in BiLSTM adds it as a separate layer to the end of the network. One of the strengths of this way is that it can provide greater flexibility and clarity since we can replace the activation function more efficiently

instead of changing it inside of the layer structure.

In the other 7 experiments, the changes are similar to the LSTM network. 2 parameters we need to change are the input_shape and the parameter in the Dense layer. In conclusion, this BiLSTM network is made of 2 bidirectional LSTM layers and uses a dense layer to transform the output into individual prediction, in the end, the tanh is applied as an activation function. This kind of network is suitable for processing information before and after the time-series data that needs to be considered simultaneously.

4.2.3 GRU

In 2014, GRU was proposed as a modification for LSTM to do machine learning translation and speed up training for its more straightforward structure and convenience to solve [50]. Traditionally, 2 gates consist of a GRU unit: reset gate and update gate. The structure of the GRU cell is shown in Figure 31:

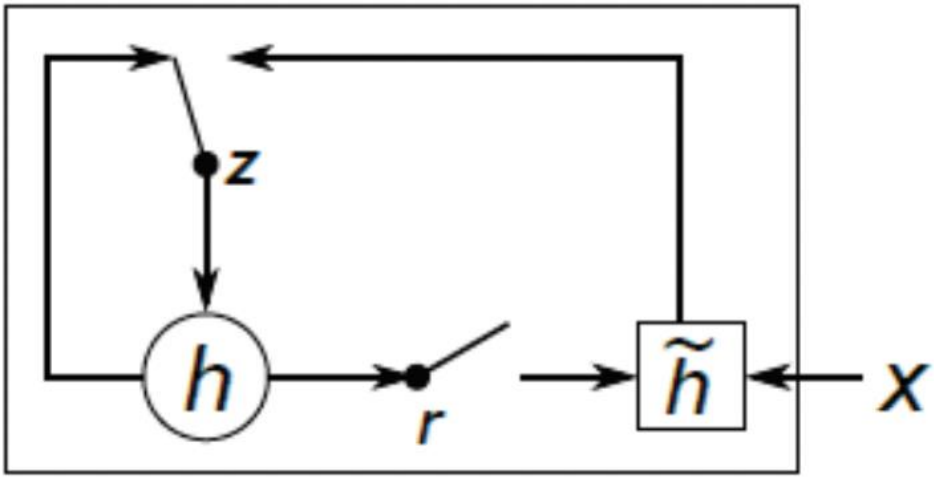


Figure 31 Structure of GRU Cell [50]

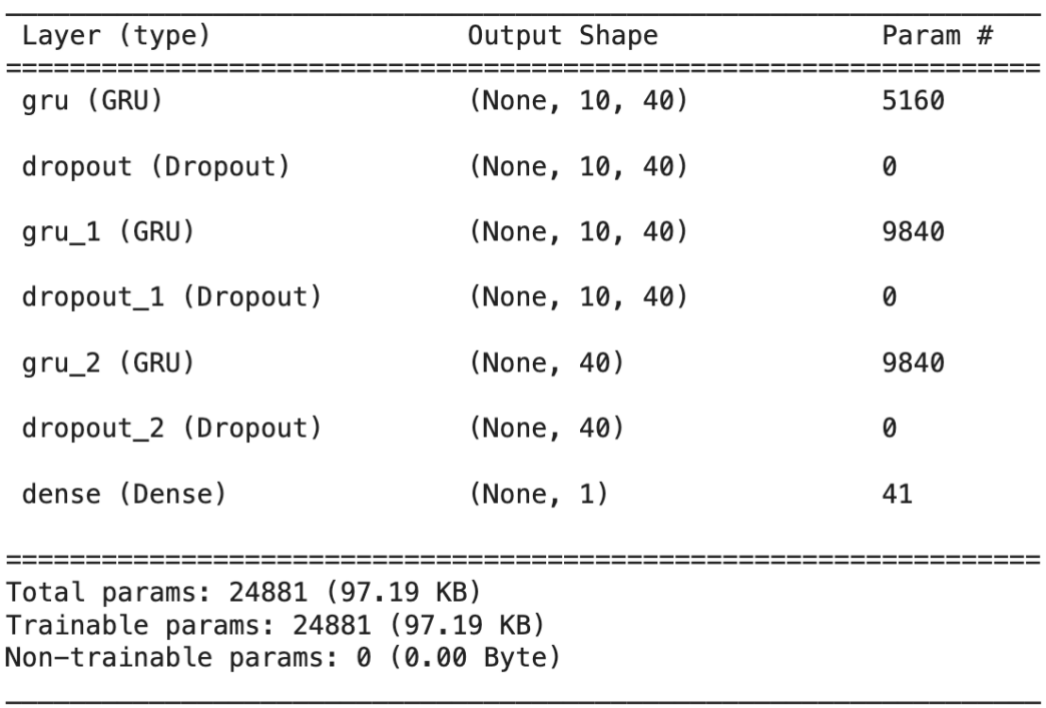
In Figure 31, r represents the reset gate, while z is the update gate. The reset gate determines how much past information needs to be retained and how much new data needs to be added. The value is usually obtained by doing a linear combination of the Sigmoid function with the

current input and previous hidden state. On the other hand, the reset gate makes a decision about the amount of past information that needs to be forgotten. It helps the model discard some irrelevant information to focus on the currently essential inputs and make predictions more precise. Compared with LSTM, GRU has a more accessible structure, fewer parameters, and fewer gating. So, the training process and computing are more straightforward, but it can also provide users with similar performance as LSTM in some cases.

Based on these, GRU is one of the models we chose for the thesis and implemented in experiments. Its performance can give us a way to compare with LSTM. At the same time, using GRU is more efficient and time-saving because of its structure, which helps obtain results quickly. The setup and summary of the GRU model in the first experiment are shown in Figures 32 and 33:

```
model_GRU1 = Sequential()
model_GRU1.add(keras.layers.GRU(40,activation="tanh",return_sequences=True, input_shape=(look_back,1)))
model_GRU1.add(Dropout(0.15))
model_GRU1.add(keras.layers.GRU(40,activation="tanh",return_sequences=True))
model_GRU1.add(Dropout(0.15))
model_GRU1.add(keras.layers.GRU(40,activation="tanh",return_sequences=False))
model_GRU1.add(Dropout(0.15))
model_GRU1.add(Dense(1))
```

Figure 32 Setting of GRU Model in Experiment 1



Layer (type)	Output Shape	Param #
gru (GRU)	(None, 10, 40)	5160
dropout (Dropout)	(None, 10, 40)	0
gru_1 (GRU)	(None, 10, 40)	9840
dropout_1 (Dropout)	(None, 10, 40)	0
gru_2 (GRU)	(None, 40)	9840
dropout_2 (Dropout)	(None, 40)	0
dense (Dense)	(None, 1)	41

Total params: 24881 (97.19 KB)
Trainable params: 24881 (97.19 KB)
Non-trainable params: 0 (0.00 Byte)

Figure 33 Summary of GRU Model in Experiment 1

The model_GRU1 neural network model is built using the Keras library, and it is a recurrent neural network for processing sequential data. The setup of the GRU network is similar to the architecture of LSTM. There are 3 layers of GRU networks in total, and each one consists of 40 neurons. Each neuron is like a processing unit which will perform some computation on the input data. These 40 neurons all perform in this way, and then, in the end, they will determine the output of the layer together. On the other hand, each neuron can learn one type of input data feature, so 40 neurons can help this layer learn up to 40 different input features simultaneously. The activation function of the GRU network is also a hyperbolic tangent function. It is a variant of the sigmoid function, and it maps values between -1 and 1. The input_shape also uses the data window defined before, since the first parameter is still look_back. The meaning of look_back is retrospective period, which indicates the number of previous points in time that are used to predict the next point. The definition of return_sequences is not changed, and it

makes the network return output in each timestep in every layer except the last one. In the last layer of GRU, the value of return_sequnces is false since it will only return the value of the last timestep. Moreover, the dropout layer is also used in this network, and it will help us decrease the risk of overfitting because it will randomly discard 15% of the output of a neuron. Finally, the dense layer is used for preparing the output of the GRU network, and its parameters can determine the horizon of the neural network.

Regarding the other 7 experiments, the architecture of the network is not changed, and only the input_shape and the parameter of the dense layer need to be modified according to different situations. The data window will be different if we want to consider the horizon. At the same time, the second element of input_shape is the number of features that are used in training, so this value is distinct in experiments with date features or some other features. In the introduction of the dense layer, the role of the parameter is mentioned, which determines the number of outputs in a timestep. So, in some experiments referring to the horizon, the value of this parameter should be considered.

GRU is also designed for long-term dependencies and works well with sequential data as does LSTM [51]. At the same time, it is more understandable and easier to implement. In this thesis, we will use it to conduct research on the performance of GRU on time-series data.

4.2.4 Bidirectional GRU

BiGRU is a particular type of recurrent neural network architecture that combines the features of bi-directional networks and GRU. A bidirectional network is one type of deep learning architecture that can process both past and future information, and it is usually used in RNN. In traditional RNN, the network can only access information up to the current time. But in a

bidirectional network, 2 individual networks are combined; one deals with the forward direction of the time series (from the past to the future) while the other tackles the backward direction (from the past to the future). In this way, the model can access and utilize both previous and subsequent information. The general architecture of a bidirectional network is shown in Figure 34:

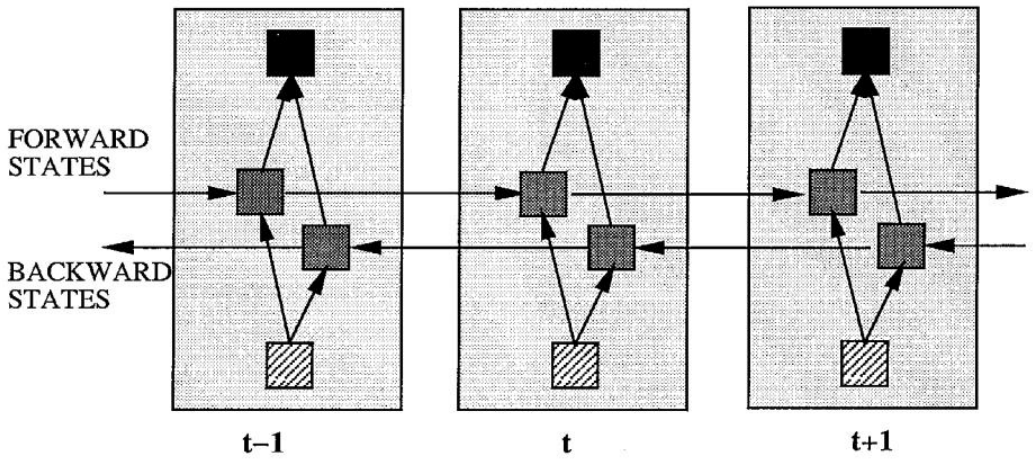


Figure 34 General Architecture of Bidirectional RNN [8]

It can help researchers capture the information in data more comprehensively, and it has been proved that this kind of network is efficient and accurate in some cases when processing sequence data. GRU consists of 2 gates: the reset gate and the update gate. The use and implementation of it is simpler than LSTM but gets similar performance on predicting. Referring to BiGRU, the base unit of its model consists of a forward-propagated GRU unit and a backward-propagated GRU unit [52]. Similar to BiLSTM, it processes data both in these 2 directions, and the result of it is the combination of 2 units. Compared with a standard GRU network, it is more comprehensive since it takes 2 directions into account. But on the other hand, its architecture is more complex, and the processing time may be longer. The definition of Bidirectional GRU and the model are shown in Figures 35 and 36:

```
model_biGRU1 = Sequential()
model_biGRU1.add(Bidirectional(keras.layers.GRU(40, return_sequences=True), input_shape = (look_back,1)))
model_biGRU1.add(Bidirectional(keras.layers.GRU(40)))
model_biGRU1.add(Dense(1))
model_biGRU1.add(keras.layers.Activation('tanh'))
```

Figure 35 Setting of BiGRU Model in Experiment 1

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirection  (None, 10, 80)            10320
 al)

 bidirectional_1 (Bidirecti  (None, 80)                29280
 onal)

 dense (Dense)               (None, 1)                 81

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 39681 (155.00 KB)
Trainable params: 39681 (155.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 36 Summary of BiGRU Model in Experiment 1

The structure of BiGRU refers to the setup of BiLSTM. It imports the bidirectional class from keras and can provide a direct method to build a bidirectional network. There are 2 layers of bidirectional GRU networks in the whole model, and each layer contains 40 neurons which can perform some kind of computation. The definition of return_sequences is the same as before since, in the first layer, it should generate an output in each timestep, while in the other layer, this value is default, which means it will only return the output in the last timestep. The input_shape parameter demonstrates the shape of input data, such as the window size and the number of features. In the first experiment, the element of the dense layer is 1 since the value of the horizon is 1, and it will change due to different experiment setups.

The bidirectional GRU is designed to capture long-term dependencies in serial data more

efficiently, which is suitable for this thesis since it is time-series data. The forward direction GRU layer processes the data in a regular way, and the backward layer will finish this step in the reverse direction. In the end, the outputs of 2 directions are combined to form a composite result that contains both past and future information. In the thesis, we can compare it with standard GRU to see if it is more precise and to check whether the bidirectional GRU requires more resources to train and run due to its structure.

4.2.5 CNN-LSTM

CNN-LSTM is a type of hybrid neural network that combines features of both Convolutional Neural Networks and Long Short-Term Memory Networks. This architecture can help users utilize not only the capabilities of CNN in feature extraction but also the advantages of LSTM in processing time-series data. It is an excellent method for solving problems with spatial features and time series. A hybrid network is a type of model which combines 2 or more different types of neural network architectures. This kind of designation can take advantage of the strengths of the different network architectures, and that is the reason why the hybrid network is typically good at tackling complex problems that are difficult to solve with a single network. A hybrid network can consist of a wide range of networks. Different types of networks have different functions and characteristics, so the strengths of these networks can be combined to solve more complex problems. However, this will also increase the difficulty of training and computing, so choosing the right combination of networks and tuning them to work together is one of the most important points researchers should consider. In this thesis, CNN and LSTM are used to build a hybrid network, which is the most common one. CNN is a kind of feedforward neural network, which has good performance in image processing and natural

language processing [33]. 2 characteristics of CNN are local perception and weight sharing, and these can help reduce the number of parameters and then obtain the target of increased efficiency. On the other hand, LSTM has the characteristic of expanding according to the sequence of time [33], so it can deal with features of data over time. Obviously, this kind of hybrid network is good at tackling video processing or sequence prediction tasks.

The setup of CNN-LSTM and model summary in this thesis are shown in Figures 37 and 38:

```
model_cnn_lstm1 = Sequential()
model_cnn_lstm1.add(Conv1D(filters=32, kernel_size=1, activation='relu', input_shape = (look_back,1)))
model_cnn_lstm1.add(LSTM(40,activation="tanh",return_sequences=True))
model_cnn_lstm1.add(LSTM(40,activation="tanh"))
model_cnn_lstm1.add(Dense(1))
```

Figure 37 Setting of CNN-LSTM Model in Experiment 1

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 10, 32)            64

 lstm (LSTM)                 (None, 10, 40)            11680

 lstm_1 (LSTM)               (None, 40)                12960

 dense (Dense)               (None, 1)                 41

=================================================================
Total params: 24745 (96.66 KB)
Trainable params: 24745 (96.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 38 Model Summary of CNN-LSTM in Experiment 1

At first, there is a one-dimensional convolutional layer that contains 32 kernels, each with a kernel_size of 1, for processing time-series data. Filters are the basic units used for feature extraction in the convolutional layer. Each filter is a set of learned weights that are applied to the input data in a convolutional operation to extract specific types of features. There are 32 filters in total, which means the network can learn and extract 32 different features. The kernel

size is 1 means that only one input value is considered in each convolution operation. It demonstrates that these features are extracted based on a single input value rather than combining multiple neighbouring values. In this layer, the activation function is ReLU which is also widely used in deep learning. The definition of it is shown in (14) below:

$$h(x) = \begin{cases} x\ (x > 0) \\ 0\ (x < 0) \end{cases} \tag{14}$$

There are only 2 values of ReLU: 0 and x. If the value is bigger 0, so the result is itself, otherwise is 0. The function plot of ReLU is shown in Figure 39:

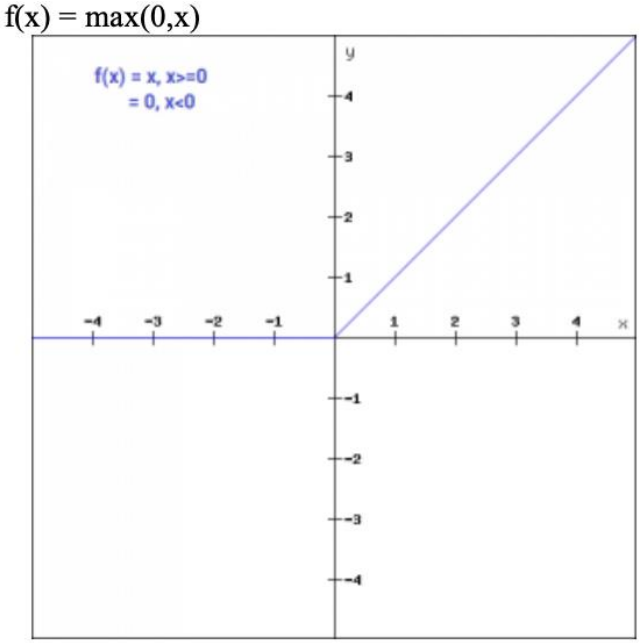


Figure 39 Plot of ReLU [53]

The setting of input_shape and return_sequences in this layer are the same as before. This convolution network layer is used to extract the local data features. It is followed by 2 layers of LSTM networks whose purpose is to tackle these time-series features. These 2 layers contain 40 neurons that can help the model learn 40 kinds of different features of data. The first layer returns a sequence for each timestep, while the second will only output the final state of the sequence.

This kind of CNN-LSTM network architecture is particularly suitable for dealing with complex data where both spatial and time-series features need to be considered, such as in some types of time-series forecasting. Therefore, it is applied in this thesis to do a time-series data prediction and a comparison with some other models.

4.3 Window

Applying Deep Learning for forecasting relies on creating appropriate time windows which allow us to format the data correctly to be fed to neural network models. Data windowing is a process in which users define a sequence of time-series data and separate them into 2 parts, which are inputs and labels. The deep learning model can fit the inputs, generate predictions, compare them to the labels, and repeat this process until it cannot improve the accuracy of its predictions [2]. The simple architecture of a data window is shown in Figure 40:

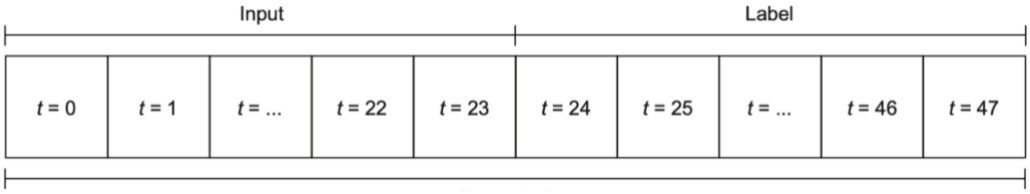


Figure 40 Architecture of Data Window [2]

From Figure 40, it is clear a data window consists of inputs and labels. The whole input dataset will be separated into some windows which will be fed to the models. Implementing different deep learning models can be easier and more correct once applying a window on the dataset. In this section, the data window in this thesis will be introduced.

4.3.1 Original Window Function

The original function of window, which converts an array of values into a dataset matrix, is

shown in Figure 41:

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back) :
  dataX, dataY = [], []
  for i in range( len(dataset) - look_back ) :
    a = dataset[i : i + look_back]
    dataX.append(a)
    dataY.append(dataset[i + look_back])
  return np.array(dataX), np.array(dataY)
```

Figure 41 Original Window Function

The definition of the create_dataset function is presented above, and 2 parameters it will receive: dataset and look_back. The dataset is the input dataset that users want to feed to the model and do the forecasting, and look_back means a retrospective period that indicates the number of previous points in time that are used to predict the next point. If the value of look_back is 1, it will use the value of (t-1) to indicate the value in time t. At the beginning of the function, it initialized 2 lists: dataX and dataY. DataX is used to store the input features, and dataY is used to store the corresponding target values. Then, it will come to a loop that aims to traverse the whole dataset. However, the number of (look_back) time points is needed to complete the prediction so that the loop will end until len(dataset) - look_back. The line of a = dataset[i : I + look_back] is used for extracting the look_back time points starting from index I from the dataset, and these data will be processed for the next value prediction. Then the feature data a will be added to dataX by append. At the same time, the data at the point in time immediately after the look_back time point is a target which should be added to dataY. This step guarantees the corresponding target value is the data at the point in time immediately following feature set

a. In the end, the 2 lists that are initialized at the beginning will be returned as output of this widow function. In total, this function creates some windows from the input dataset by extracting sequence segments, and the output of the function can be fed to the neural network for training.

4.3.2 Window Function with Horizon

In this thesis, the effect of the horizon is also explored by defining different horizon values. The window-creating step may be a little bit different if the value of the horizon is not 1 since the model will generate several predictions in one timestep. So, the window function is not the same as the original one. The window function with the horizon is shown in Figure 42:

```
def create_dataset_horizon(dataset, look_back, horizon):
    X, Y = [], []
    for i in range(len(dataset) - look_back - horizon + 1):
        X.append(dataset[i:(i + look_back)])
        Y.append(dataset[(i + look_back):(i + look_back + horizon)])
    return np.array(X), np.array(Y)
```

Figure 42 Window Function with Horizon

This function will receive 3 parameters, including the value of the horizon. Another difference is the termination condition for traversal, which is (len(dataset) - look_back - horizon + 1). The purpose of this condition is to ensure that the output dataset is created with both enough historical data and a corresponding target data set for each input dataset. The historical data is related to the value of parameter look_back, while the target data is associated with the parameter horizon. Using the length of the dataset, subtracting look_back guarantees the model can process with enough previous values to predict the next value, and subtracting horizon means there is enough future data to form a complete output dataset. Moreover, ranges in

Python are always semi-open, so it is requisite to add 1 in order to include the last point in time and form a complete input and target output. Therefore, this approach avoids boundary problems in the dataset and ensures that each input has a corresponding target output without errors due to missing future data.

Another change in this function is the output list. The line of (Y.append(dataset[(i + look_back):(i + look_back + horizon)])) aims to require the data of the horizon time points starting at (index i + look_back) as target values and add them to the list Y. This means that the corresponding target value is the data at the horizon point in time immediately following each set of features. Therefore, this function can help us get 2 sequences of data which can be processed by the deep learning models directly and in an efficient way.

4.3.3 Window Function with 2 Features

More than one feature to feed the model training may be positive to the accuracy of final results, so the number of features will also be taken into account. However, this affects the data window creation since there is more than one feature in the input dataset, but we only want the target values. So, the definition of the window function is also different in experiments with 2 features, and it is shown in Figure 43:

```
def create_dataset_2fea(dataset, look_back) :
  dataX, dataY = [], []
  for i in range( len(dataset) - look_back ) :
    a = dataset[i : i + look_back]
    dataX.append(a)
    dataY.append(dataset[i + look_back, -1])
  return np.array(dataX), np.array(dataY)
```

Figure 43 Window Function with 2 Features

While tacking with input dataset with more than one feature, the only difference of window function is in output list dataY. As we can see, there is a change when appending the target data into dataY which is (dataY.append(dataset[i + look_back, -1])). In Python, using a negative index represents counting from the end of a list or array, so minus 1 here means selecting the last feature value as the target output. In this way, only the target feature will be appended to the output list instead of the whole features, including some other features which are only used for training the model. This method can also be applied in experiments with more than 2 features.

4.3.4 Window Function with Features and Horizon

After these phenomena, a question comes with the setting of experiments: how about the window function with both features and horizon? As mentioned before, data window functions are different when there is more than one feature or the horizon is not equal to 1. So, the original window function is definitely not suitable for this situation. The definition of the function in this situation is shown in Figure 44:

```
def create_dataset_horizon_date(dataset, look_back, horizon):
    X, Y = [], []
    for i in range(len(dataset) - look_back - horizon + 1):
        X.append(dataset[i:(i + look_back)])
        Y.append(dataset[(i + look_back):(i + look_back + horizon), -1])
    return np.array(X), np.array(Y)
```

Figure 44 Window Function with Features and Horizon

This function seems to combine both the window function with the horizon and the function with 2 features. The termination condition is also (len(dataset) - look_back - horizon + 1), which is used to ensure the output dataset is created with both enough historical data and a corresponding target data set for each input dataset. At the same time, the creation of a target list is the same as the function with 2 features, which uses minus 1 to get the target feature. It is understandable that using this function can deal with experiments that not only refer to the horizon but also the number of features for training models.

4.4 Training and Testing

After all preparations, the most significant step comes to be training and testing. Training means using the input data to train the model used to learn the pattern, structure, or relation of the data. During training, the parameters of the model are continuously adjusted by some optimization algorithms to reduce the difference between the predictions and actual values. Testing is the process of evaluating a trained model using a separate dataset that has not been used in the training phase. The purpose of testing is to evaluate the performance of the model in a different dataset, so it is a good way to find if the model is so accurate that it can be used in reality or the research. So, these 2 stages are crucial in machine learning, which can determine whether the model is suitable and useful finally. In this section, the training and testing process will be introduced, including the dataset splitting and compiling process.

4.4.1 Dataset Splitting

2 datasets are used in the research, and each one is split into 3 parts: training set, validation set, and test set. The method of splitting directly indicates the intervals of each set, and this way can help us determine the size of each set flexibly by only changing the indexes of the sets. For instance, the line of code trainX = X[:-430] valX = X[-430:-330] testX = X[-330:] demonstrated how the dataset X is split. The training set contains all but the last 430 elements of X, and this part of the data is used in the training phase of the model to learn patterns and relationships in the data. The validation set consists of the data in X starting from the 430th last element to the 330th last element (100 elements in total). The validation set is a bit like a test set that the model can peek at to tune its hyperparameters and improve its performance during the model's training [2]. It is used for model tuning and hyperparameter selection, and it helps to understand how the model behaves on unseen data and can be used to avoid overfitting. Finally, the testing set contains the remaining data in X, which has 330 values in total. The test set is used at the end of the entire training process to evaluate the final performance of the model, in particular its ability to generalize on unseen data since the data in the test set is not used in the training step.

Therefore, these 3 sets can provide us with the whole progress of training, validating, and testing the models used. In general, the size of the training set is always more than 70% of the whole original dataset since the models need enough data to finish the training step and learn the relationship and pattern of the data.

4.4.2 Model Compiling and Fitting

In machine learning, it is necessary to compile the model after definition and before the training step. Compiling the model is a key step in the process of preparing and optimizing the model for training. It involves the selection and configuration of the algorithms that the model will be used to learn, the criteria for evaluating its performance, etc. The compilation step finalizes the model and gets it completely ready for use in fitting and predicting [54]. The correct and suitable compilation settings are crucial for training results and well-performing models in machine learning. The compilation step is usually achieved using the compile method in Keras. It configures the learning process, specifying the optimizer, loss function and maybe some metrics to monitor the training process.

In this research, the compile step in one of the experiments is shown in Figure 45:

```
model_LSTM1.compile(optimizer='adam', loss='MSE')
```

Figure 45 Compilation Step in Experiment 1 of LSTM

The optimizer for monitoring it is Adam, and the loss function is Mean Squared Error. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions which is based on adaptive estimates of lower-order moments [55]. Adam combines 2 extended gradient descent algorithms: Momentum and RMSprop. It can adjust the learning rate by calculating the root mean square of the gradient for each parameter, and it achieves the goal of bias correction for momentum and root mean square variables. Adams is one of the most common optimizers in deep learning nowadays. Its efficiency and adaptivity help it become particularly suitable for large-scale datasets and models with a large number of parameters, such as CNN. Adams combines the strengths of Momentum and adaptive learning rate, which

make it effective in tackling some complex problems. Besides, the loss function in the compilation step is a mean squared error, which is usually used in regression problems. It evaluates the average of the squared values of the differences between the predicted and true values, and it can provide us with a simple way to know about the precision of the models. The definition of MSE is shown in (15) below:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (T_i - P_i)^2 \tag{15}$$

In this function, n represents the number of data, Ti is the true value of the ith data, and Pi is the prediction of this value. Obviously, this definition is so simple that can be understood easily. But at the same time, the value of MSE can reflect the error or the inaccuracy of models directly, so it is widely applied in loss function and evaluation metrics.

Then, the compiled will be used to fit the data. Model fitting measures how well a machine learning model generalizes to similar data to that on which it was trained [56]. A well-fitted model can help users obtain more precise and accurate predictions. There are 2 kinds of poor performance of model fitting: overfitting and underfitting. Overfitting means that the neural network at a certain time during the training period does not improve its ability to solve problems anymore, and it starts to learn some random regularity contained in the set of training patterns [57]. In other words, the model is only fitted on the training data, and it even learns some random values and outliers in these data. However, it will perform poorly in data that are not used in training. In contrast, underfitting is that the model is incapable of capturing the variability and characteristics of the data. So, the model is not able to generate predictions since it does not have the ability to learn the pattern and relationship of the data. These 2 problems both have a bad impact on machine learning, and it is necessary to take action to solve them.

The methods we used in the research to avoid overfitting are dividing the validation set and using an early stopping technique while fitting the model. A validation dataset is a series of data that is not used in training the model. While fitting the model, it is applied to evaluate model performance at the end of each epoch. The feedback of this set can help users obtain the performance of the model in unseen data. In the first dataset, the validation set consisted of the data in X starting from the 430th last element to the 330th last element (100 elements in total). In addition, the mechanism of early stopping is used in the research. This technique will terminate the training process if the performance on the validation set does not improve for a certain number of epochs. In this way, it not only prevents the model from learning some random values and errors in the data but also saves training time. The definition of early stopping and the fitting step is shown in Figure 46:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
history = model_LSTM1.fit(trainX, trainY, epochs=200, batch_size=32, validation_data=(valX, valY), callbacks=[early_stopping])
```

Figure 46 Early Stopping and Fitting

To implement the early-stopping technique, it is simple to apply the EarlyStopping function in Keras, which is a callback function used to stop training under certain conditions. In this circumstance, 'Monitor = 'val_loss'' specifies that the validation set loss is monitored to trigger the early stopping mechanism. If the value of 'val_loss' is not improved anymore, the training process will be stopped in advance. Likewise, the patience parameter indicates the number of epochs if there is no improvement before it stops. In this way, patience is 20, which means if 'val_loss' is not decreased for 20 epochs, the training will stop to avoid overfitting and save time. In the end, the setting of restore_best_weights is True, which represents that the model

will be restored to the weights of the one that performed best on the validation set once the early stopping mechanism is triggered. This method guarantees that we will get the most optimal model even if the training process is stopped early. In conclusion, adding an early stopping technique is not only an efficient way for training but also provides a better way to acquire a more precise model.

Then, it comes to the step of fitting the model. It uses trainX and trainY as training datasets where trainX is feature data and trainY is target data. The model will be trained by these and learn how to predict targets using the feature data. In the same way, batch_size specifies that the model will process 32 data points simultaneously in each iteration, and 'epochs = 200' defines that the model should traverse the entire training dataset 200 times, except it triggers an early stopping mechanism. Moreover, the validation set is specified to evaluate the performance of the model at the end of each epoch. It is an ideal method to monitor the model's performance and avoid overfitting since the validation set is unused in training. Finally, the callbacks parameter is implemented to call the EarlyStopping function during the training process. After fitting the model, we will train the model to forecast in the follow-up steps.

Chapter 5

Experimental Results

In this Chapter, the results of experiments are revealed. In Section 5.1.1, we analyze the results of 2 datasets separately and compare the performance of each model. Then, we make a comparison of 2 datasets to explore if there is any relevance between the characteristics of the datasets and the performance of the models.

5.1 Comparison of Algorithms on the Same Dataset

In this section we present 2 sets of results. We conducted 8 experiments for each of the 5 models on the same dataset. The experiments exhibit the impact of the structural differences of models, as well as the impact of the number of features, horizon, and date features .

5.1.1 Results of DailyDelhaClimate Dataset

The results of the first dataset, DailyDelhaClimate is shown in Table 5:

Table 5 Result of DailyDelhaClimate Dataset

					mape train	mape test	mse train	mse test	epoch
lstm	lstm	1 horizon	1feature		8.9449	7.2484	7.0381	6.4735	26
	lstm	5 horizon	1 feature		8.8145	7.2314	6.992	6.6119	86
	lstm	1 horizon	1 feature + date feature		26.5399	24.2597	63.7296	59.7696	94
	lstm	5 horizon	1 feature + date feature		7.2965	6.8348	4.6406	5.6856	89
	lstm	1 horizon	2 features		5.4287	4.8265	2.6843	3.0616	200
	lstm	5 horizon	2 features		10.3991	8.7479	9.8011	9.6594	32
	lstm	1 horizon	2 features + date featur		5.7647	5.4074	2.8026	3.4629	172
	lstm	5 horizon	2 features + date featur		8.7582	7.522	6.6089	6.7688	24
bilstm	bilstm	1 horizon	1feature		5.369	4.6767	2.6955	2.6349	183
	bilstm	5 horizon	1 feature		7.8849	6.7636	5.8473	5.9023	71
	bilstm	1 horizon	1 feature + date feature		26.7139	23.9924	63.8001	58.3185	125
	bilstm	5 horizon	1 feature + date feature		6.7985	6.1171	3.9313	4.8109	80
	bilstm	1 horizon	2 features		5.4852	4.9528	2.7318	3.4181	200
	bilstm	5 horizon	2 features		7.4277	6.403	5.0776	5.3788	88
	bilstm	1 horizon	2 features + date featur		5.7189	4.8737	2.9849	2.9438	92
	bilstm	5 horizon	2 features + date featur		7.0094	6.5012	4.2678	5.359	56
gru	gru	1 horizon	1feature		6.1848	5.7744	3.692	3.9605	146
	gru	5 horizon	1 feature		7.9769	7.035	6.1214	6.3479	147
	gru	1 horizon	1 feature + date feature		26.4733	23.7123	61.8282	56.4854	193
	gru	5 horizon	1 feature + date feature		8.1922	7.1389	5.952	6.4264	34
	gru	1 horizon	2 features		5.6498	5.3404	2.9717	4.1723	112
	gru	5 horizon	2 features		7.9886	6.8854	5.9154	6.1683	98
	gru	1 horizon	2 features + date featur		5.0723	4.4958	2.4375	2.5211	200
	gru	5 horizon	2 features + date featur		6.4429	6.2427	3.5763	4.8027	195
bigru	bigru	1 horizon	1feature		5.4044	4.7382	2.7711	2.7403	105
	bigru	5 horizon	1 feature		7.9107	6.9338	5.9297	6.1118	84
	bigru	1 horizon	1 feature + date feature		26.7376	24.1123	63.7959	58.2654	123
	bigru	5 horizon	1 feature + date feature		6.4676	5.9974	3.6575	4.4793	108
	bigru	1 horizon	2 features		5.7293	5.3846	3.1205	3.8369	89
	bigru	5 horizon	2 features		7.6378	6.6013	5.5048	5.5465	49
	bigru	1 horizon	2 features + date featur		5.3094	4.8672	2.6089	2.8856	60
	bigru	5 horizon	2 features + date featur		6.4647	6.2444	3.6675	5.0584	95
cnn-lstm	cnn-lstm	1 horizon	1feature		5.5426	4.9732	2.9059	2.9232	187
	cnn-lstm	5 horizon	1 feature		7.5178	6.4062	5.2766	5.2403	121
	cnn-lstm	1 horizon	1 feature + date feature		26.5704	24.0902	62.7873	58.6691	121
	cnn-lstm	5 horizon	1 feature + date feature		6.8153	6.2067	3.8557	4.8524	74
	cnn-lstm	1 horizon	2 features		5.453	5.0779	2.8137	4.0914	96
	cnn-lstm	5 horizon	2 features		7.5427	6.5673	5.3267	5.8495	119
	cnn-lstm	1 horizon	2 features + date featur		5.1325	5.1195	2.3529	3.2634	119
	cnn-lstm	5 horizon	2 features + date featur		6.8861	7.0075	3.9795	6.7526	67

To analyze the results of each model more intuitively, we used a bar chart to show the MAPE Train of models and the chart is shown in Figure 47:

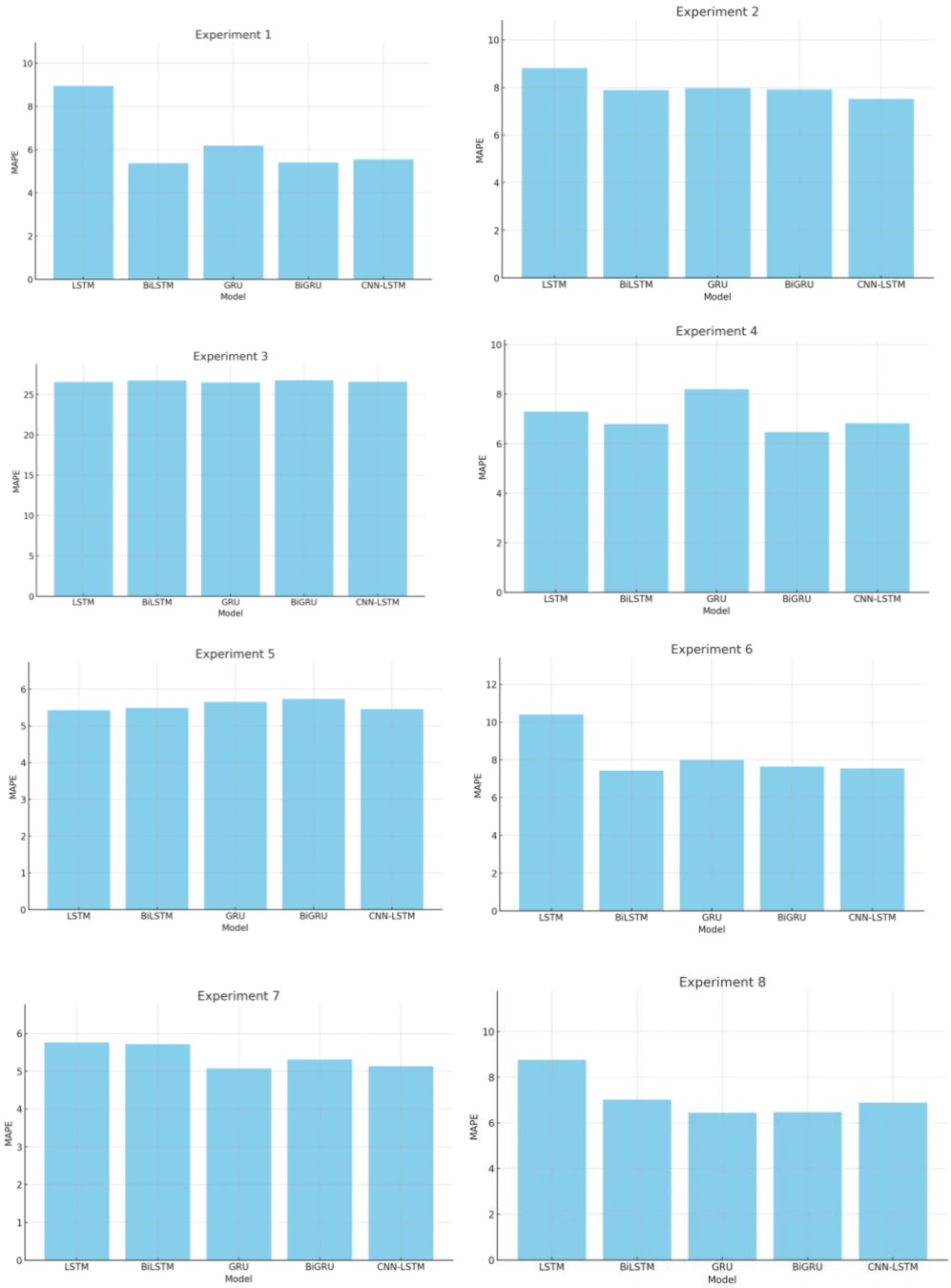


Figure 47 MAPE Train of Models

Table 4 and Figure 47 show the results for the 5 models that we study: LSTM, Bi_LSTM, GRU, Bi_GRU, and CNN_LSTM. In the table, the results are marked with different colours, with red denoting the smallest and green the largest values. The more accurate model is BiGRU, which achieves the lowest error rate in experiments 1,4 and 8, and there are lower error rates in BiGRU, indicated by red areas. In contrast, LSTM is not as precise as other models since there are more

yellow cells in the results for LSTM in the result table. Conversely, LSTM has the fewest epochs of training. From the table, we can see that the number of epochs is the smallest for LSTM in experiments 1, 6, and 8, and this means it is efficient, as the training time is short.

There are some general observations that we can make based on the results of all 8 experiments. All highest errors occurred in experiment 3, for which the horizon value is 1 and which uses date as an additional feature. The reason for this is that the models are more likely to rely on the recent observations when the horizon is 1, and date features may not provide direct information about upcoming changes but may introduce some unwanted noise into the model, especially if these features have no obvious correlation with the target variable. So, we can see that in most of experiments where the horizon is 1, the results are better if there are no date features. However, if the horizon is 5, which means it is long-term forecasting, the results are better with date features. This is because long-term predicting can make use of date features and it may have a positive impact on the results. Additionally, the results are more accurate when the horizon is 1 than when the horizon is 5 in most experiments. Usually, it is easier and more accurate to predict points that are close in the future than those that are farther away since near-term data are more reflective of the current patterns and trends and the farther a data point is in the future, the less it may be affected by such patterns and trends and the more it may be affected by other factors. The more long-term a prediction is, the higher the risk of error because each prediction of the model relies on the predictions of the previous time step. Concerning the number of features, we observe that the higher the number of features, the more accurate a prediction is since the model can learn the data in a more holistic manner. At the same time, the results of experiments with date features are better than those without these features, except for

experiment 3. To a certain extent, date features help models understand and capture the seasonality and trends of this dataset, and models can know about the data clearly by leveraging these date features.

5.1.2 Result of Steel_Industry Dataset

We present results on the stationary dataset Steel_Industry in Table 6:

Table 6 Result of Steel_Industry Dataset

				mape train	mape test	mse train	mse test	epoch
lstm	lstm	1 horizon	1feature		35.0579	149.0395	72.902	55
	lstm	5 horizon	1 feature		85.1129	295.6463	175.6398	74
	lstm	1 horizon	1 feature + date feature		1220.3172	1935.889	3903.529	72
	lstm	5 horizon	1 feature + date feature		59.8932	280.3688	156.6627	62
	lstm	1 horizon	2 features		25.0707	140.407	74.7666	47
	lstm	5 horizon	2 features		57.7902	243.3485	173.089	60
	lstm	1 horizon	2 features + date feature		22.4222	116.5954	67.4426	78
	lstm	5 horizon	2 features + date feature		51.0063	200.0309	168.5372	73
bilstm	bilstm	1 horizon	1feature		19.2459	138.0257	66.2551	59
	bilstm	5 horizon	1 feature		77.003	293.0913	177.4049	62
	bilstm	1 horizon	1 feature + date feature		1220.7313	1948.041	3896.804	68
	bilstm	5 horizon	1 feature + date feature		79.9576	273.6721	175.2882	48
	bilstm	1 horizon	2 features		21.9697	111.9191	68.7674	60
	bilstm	5 horizon	2 features		66.6642	269.6334	181.7328	39
	bilstm	1 horizon	2 features + date feature		29.4872	115.6374	70.5448	54
	bilstm	5 horizon	2 features + date feature		62.2672	225.6964	178.4473	48
gru	gru	1 horizon	1feature		21.5819	146.8385	70.4985	53
	gru	5 horizon	1 feature		86.6396	298.1556	172.5249	64
	gru	1 horizon	1 feature + date feature		1215.7263	1930.712	3883.247	93
	gru	5 horizon	1 feature + date feature		58.6165	242.966	160.9194	82
	gru	1 horizon	2 features		31.725	134.6803	72.0437	46
	gru	5 horizon	2 features		73.2404	235.768	181.1586	58
	gru	1 horizon	2 features + date feature		34.6738	134.8177	72.3026	49
	gru	5 horizon	2 features + date feature		60.8676	183.4564	176.8064	85
bigru	bigru	1 horizon	1feature		26.2516	127.9617	70.2947	65
	bigru	5 horizon	1 feature		81.2424	310.1227	168.1927	44
	bigru	1 horizon	1 feature + date feature		1215.6799	1927.372	3875.165	55
	bigru	5 horizon	1 feature + date feature		79.3443	298.2405	175.7822	45
	bigru	1 horizon	2 features		20.7734	122.8981	69.6891	42
	bigru	5 horizon	2 features		113.179	247.7859	284.9449	42
	bigru	1 horizon	2 features + date feature		28.3685	114.5812	73.069	49
	bigru	5 horizon	2 features + date feature		73.1001	230.8058	183.8588	44
cnn-lstm	cnn-lstm	1 horizon	1feature		33.1102	128.6037	66.8117	74
	cnn-lstm	5 horizon	1 feature		89.6066	302.2672	172.8279	49
	cnn-lstm	1 horizon	1 feature + date feature		1214.8711	1929.287	3868.803	52
	cnn-lstm	5 horizon	1 feature + date feature		74.7876	259.7233	164.883	59
	cnn-lstm	1 horizon	2 features		36.0017	121.0435	68.2845	53
	cnn-lstm	5 horizon	2 features		100.66558	229.3234	206.038	46
	cnn-lstm	1 horizon	2 features + date feature		25.3327	113.0863	71.1488	53
	cnn-lstm	5 horizon	2 features + date feature		56.1351	226.5823	167.227	44

The results of the 8 experiments on the second dataset is shown above, and the comparison of MAPE in each model is shown in Figure 48:

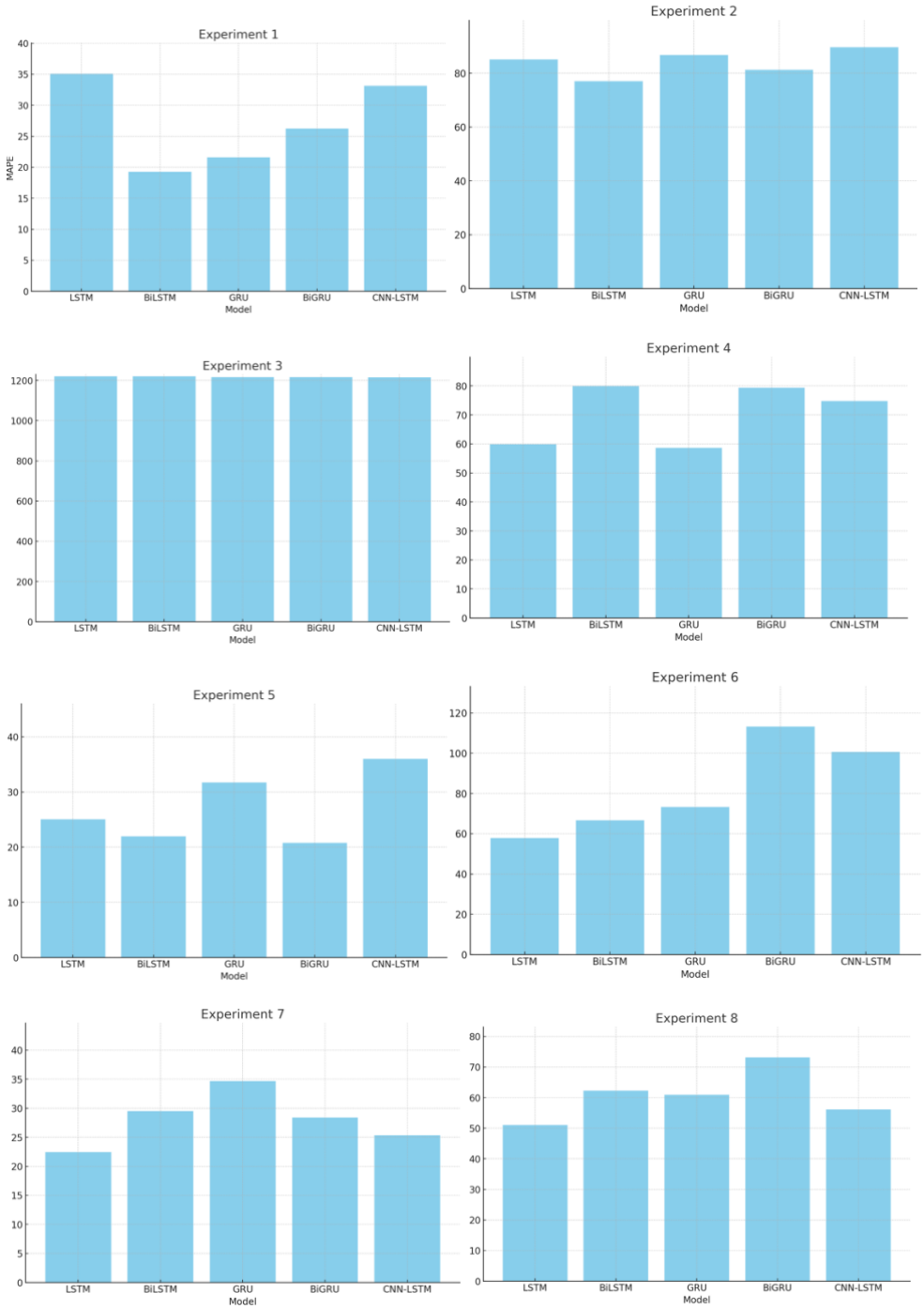


Figure 48 Comparison of MAPE in Each Model

Figure 48 demonstrates the MAPE of the training set in each model by different kinds of experiments. We observe that LSTM is the best model for this dataset in almost all experiments except experiments 1 and 2. In other words, LSTM performs best if there are more features used in the training process. Similarly, CNN-LSTM is also one of the models that performs

well in forecasting this dataset since its result is above the average level among the 5 models. On the other hand, the performance of BiGRU is not as outstanding as it is for the first dataset, especially in experiments 2, 4, 6, and 8. So, BiGRU is not suitable for this dataset when the horizon is 5. Yet, BiGRU training usually takes a shorter time. In the last column, there are more red cells in BiGRU, which indicates that fewer epochs are needed to obtain the most optimal model.

From the results on the second dataset, it is clear that the performance of the models with a horizon of 1 is better than 5 since there are some red-yellow intervals in the table. This means that the models are better at short-term forecasting, and the horizon may affect the precision of it. Models targeting short-term forecasts are often simpler because they only need to capture patterns in recent data. Meanwhile, each prediction of the model depends on the previous output in long-term forecasting, so errors may propagate over time and result in a large error rate in the end. Moreover, experiment 3, for which the horizon is 1 and training includes the date features, still produces the worst result in this dataset. The reason for this may be the same as before because the model is more likely to rely on recent observations when the horizon is 1, and date features may not provide direct information about upcoming changes. Moreover, the results of each experiment in these models present almost the same pattern: results of experiment 7 often achieve the lowest error rate. In experiment 7, models will train with 2 features and date features, and this way can give them more information about the pattern and relationship of the dataset so that the model can learn it more comprehensively.

5.2 Comparison of Algorithms on the Different Datasets

Comparing the results of different datasets can help us determine if there is any relationship

between the characteristics and models. The first is seasonal, while the second one is stationary. Moreover, the Steel_Industry dataset is thirty times as large as the other dataset. We combine the result table we got from datasets 1 and 2 into a new table to make a comparison of how the characteristics of datasets affect the performance of models. The combination results table is shown in Table 7 (the MAPE of the train set in the second dataset is inf, so the MAPE Train in the table is dropped):

Table 7 Combination of Datasets 1 and 2

				mse test 1	mse test2	epoch 1	epoch 2
lstm	lstm	1 horizon	1feature	7.2484	35.0579	26	55
	lstm	5 horizon	1 feature	7.2314	85.1129	86	74
	lstm	1 horizon	1 feature + date feature	24.2597	1220.317	94	72
	lstm	5 horizon	1 feature + date feature	6.8348	59.8932	89	62
	lstm	1 horizon	2 features	4.8265	25.0707	200	47
	lstm	5 horizon	2 features	8.7479	57.7902	32	60
	lstm	1 horizon	2 features + date feature	5.4074	22.4222	172	78
	lstm	5 horizon	2 features + date feature	7.522	51.0063	24	73
bilstm	bilstm	1 horizon	1feature	4.6767	19.2459	183	59
	bilstm	5 horizon	1 feature	6.7636	77.003	71	62
	bilstm	1 horizon	1 feature + date feature	23.9924	1220.731	125	68
	bilstm	5 horizon	1 feature + date feature	6.1171	79.9576	80	48
	bilstm	1 horizon	2 features	4.9528	21.9697	200	60
	bilstm	5 horizon	2 features	6.403	66.6642	88	39
	bilstm	1 horizon	2 features + date feature	4.8737	29.4872	92	54
	bilstm	5 horizon	2 features + date feature	6.5012	62.2672	56	48
gru	gru	1 horizon	1feature	5.7744	21.5819	146	53
	gru	5 horizon	1 feature	7.035	86.6396	147	64
	gru	1 horizon	1 feature + date feature	23.7123	1215.726	193	93
	gru	5 horizon	1 feature + date feature	7.1389	58.6165	34	82
	gru	1 horizon	2 features	5.3404	31.725	112	46
	gru	5 horizon	2 features	6.8854	73.2404	98	58
	gru	1 horizon	2 features + date feature	4.4958	34.6738	200	49
	gru	5 horizon	2 features + date feature	6.2427	60.8676	195	85
bigru	bigru	1 horizon	1feature	4.7382	26.2516	105	65
	bigru	5 horizon	1 feature	6.9338	81.2424	84	44
	bigru	1 horizon	1 feature + date feature	24.1123	1215.68	123	55
	bigru	5 horizon	1 feature + date feature	5.9974	79.3443	108	45
	bigru	1 horizon	2 features	5.3846	20.7734	89	42
	bigru	5 horizon	2 features	6.6013	113.179	49	42
	bigru	1 horizon	2 features + date feature	4.8672	28.3685	60	49
	bigru	5 horizon	2 features + date feature	6.2444	73.1001	95	44
cnn-lstm	cnn-lstm	1 horizon	1feature	4.9732	33.1102	187	74
	cnn-lstm	5 horizon	1 feature	6.4062	89.6066	121	49
	cnn-lstm	1 horizon	1 feature + date feature	24.0902	1214.871	121	52
	cnn-lstm	5 horizon	1 feature + date feature	6.2067	74.7876	74	59
	cnn-lstm	1 horizon	2 features	5.0779	36.0017	96	53
	cnn-lstm	5 horizon	2 features	6.5673	100.6656	119	46
	cnn-lstm	1 horizon	2 features + date feature	5.1195	25.3327	119	53
	cnn-lstm	5 horizon	2 features + date feature	7.0075	56.1351	67	44

From the table, we can observe that the distribution of most of the yellow cells is similar in the results for the 2 datasets, which means that for some experiment setups, the behaviour of the

models is the same for both datasets. Moreover, the green cells in the error rate columns appear for both datasets in the rows of the third experiment for each model. However, there are a lot of cells colored with varying degrees of red. It is apparent that GRU and BiGRU work well on dataset 1, while LSTM is the most suitable model for dataset 2. To a certain extent, these 2 kinds of models are similar since they are both a kind of modification of RNN but with different architectures. LSTM consists of 3 gates, while there are only 2 gates in GRU. The structure determines that GRU is more efficient and time-saving, but LSTM can learn more complex and bigger amounts of data.

Beyond the type of the models, it is apparent from the results that another factor affects the error rate differences for the 2 datasets: the error rate is much higher when the horizon is 5 than when it is 1 in the second dataset, and this kind of discrepancy is not so obvious in the first one. The reason for this is that the first dataset is seasonal, so the larger horizon is instructive for discovering the seasonality and trend of the data, even if it may be detrimental to the precision of results. However, this does not affect the results of the second dataset because it is not seasonal, so long-term forecasting is not as accurate as short-term forecasting.

Chapter 6

Automated Model Selection Technique

The Automated Model Selection technique we proposed is introduced in this chapter. Section 6.1 gives information about the outcomes we obtained from the experiments. Then, the method is introduced in Section 6.2, and its limitations are shown in Section 6.3.

6.1 Outcomes

Based on the thorough analysis of the experimental results presented in Chapter 5, we can extract several outcomes to summarize how to select a model according to the characteristics of input datasets. First, GRU and BiGRU are more efficient and as accurate as LSTM if the size of the dataset is not too big (smaller than 2000 rows). To process larger datasets, LSTM is more suitable since it can capture the complex patterns and relationships of the data. Then, extracting date features from the time feature of a dataset and using them in the training process positively impacts the models' performances if the dataset is seasonal. Moreover, if datasets come with several kinds of features except the target feature, it is better to use more features for training. The more information provided, the more accurate predictions are. At the same time, applying some time-dependent external factors can also help with more accurate forecasting because they provide more information about time. Finally, short-term forecasting (horizon = 1) gives better results than long-term forecasting (horizon = 5) in most experiments. Therefore, the size of the input dataset helps us determine the type of models used for forecasting, the number of features in datasets and if datasets are seasonal, aim to set up the parameters of models. The outcomes of the analysis of results are listed below:

- **Outcome 1:** Choose GRU and BiGRU for processing small time-series datasets.
- **Outcome 2:** Choose LSTM for processing large time-series datasets.
- **Outcome 3:** Extract date features and use them for training if the dataset has seasonality.
- **Outcome 4:** Use more time-dependent features in training if the datasets have more features except the target feature.
- **Outcome 5:** Choose short-term forecasting while doing the time-series forecasting.

6.2 Automated Model Selection Technique

The Automated Model Selection technique we propose is a Decision Tree created based on the experiments and outcomes above, and it is outlined in Figure 49:

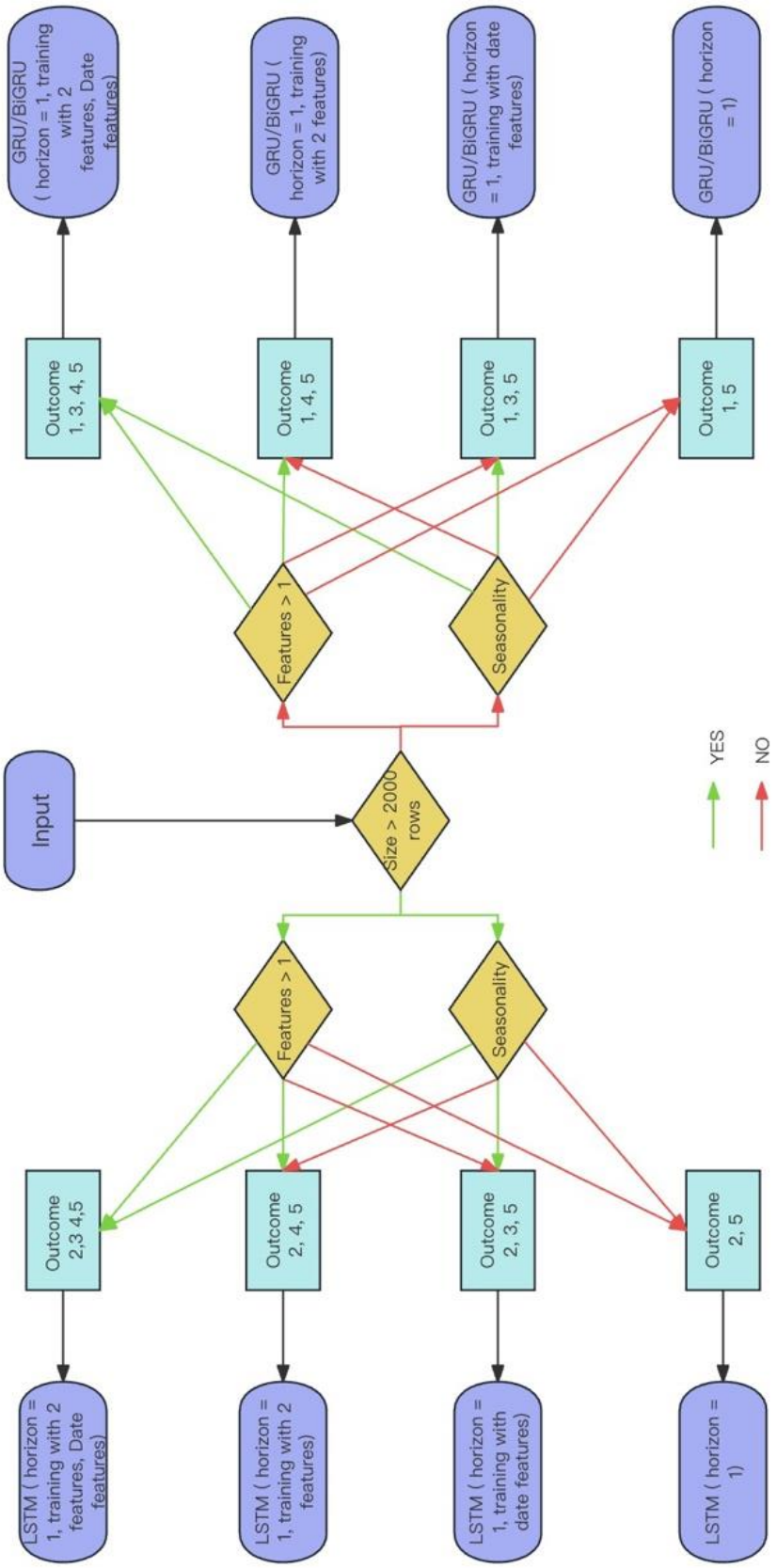


Figure 49 Flow Chart of Automated Model Selection

In Figure 49, we can see that the first characteristic we check is the size of the datasets. According to Outcomes 1 and 2, the length of datasets determines the type of models used for forecasting later. Then, it comes to the decisions of whether the number of features is more than 1 and whether there is seasonality in the datasets. If the datasets come with more than one feature, 2 features are employed to train the models. Especially if it contains another time-dependent feature, priority is given to this kind of feature since it may be an external factor of the target feature. This technique chooses different models based on a series of Outcomes if the input meets with different kinds of conditions. For example, in the left part of the figure where the size of the input is large, the LSTM model with several pieces of setup (horizon =1, training with Date features and 2 features) is selected if the input comes with seasonality and more than 1 feature, which is the first output of the left part of the figure. Therefore, this Automated Model Selection Technique automatically selects a suitable model for the input based on the analysis of the characteristics of datasets and the Outcomes we got from previous experiments.

6.3 Limitations of Technique

There are still several limitations of the proposed Automated Model Selection technique. First, it only considers training models with up to 2 features. In the experiments of this research, we used up to 2 time-dependent features to feed the networks, and this is the reason why this technique only considers training with 2 features if the datasets come with more than one feature. However, the results of models can be more accurate if more external factors are used since they provide more information about time. Moreover, the data type we used in the experiments is scientific data related to climate and energy, so the models we considered target this kind of data. There are several other types of data, and a few models aim at them as well.

For example, the Prophet targets processing business-level datasets, so for this specific kind of data Prophet is more suitable.

Chapter 7

Conclusion and Future Work

The conclusion and future work are introduced in this chapter. Section 7.1 mentions the conclusion of the thesis while the future work are given in Section 7.2.

7.1 Conclusion

Time-series data is related to various domains, and its analysis can promote the development of our lives. However, several specific characteristics of time-series data make it challenging to analyze it. For example, it may be implemented with certain features, such as seasonality, affecting the data's tendency in different seasons. The predictions are inaccurate if this information is not considered while forecasting. Meanwhile, it is time-independent, resulting in its sensitivity to changes in time. A large number of models have been proposed to analyze and predict time-series data. However, it also brings a problem that it is difficult to choose a suitable model among a lot of models. Different models may target different types of datasets, and choosing the suitable one is positive for forecasting results. To tackle this issue, we propose a kind of Automatic Model Selection in this thesis that helps users find a suitable model based on their datasets.

To design this technique, we first collected lots of models and time-series datasets, then chose 5 kinds of models and 2 datasets with different characteristics to conduct experiments. In the research, we did experiments in 2 datasets separately to explore whether the datasets' features affect the models' performance. In this step, we can get the results of how the characteristics of datasets, such as size, seasonality and stationery, influence the models' performances.

Moreover, we applied 8 kinds of settings for each model, which aim to find the best setup of parameters of models. In this part, we considered the horizon, the number of external features, and date features. Finally, we analyzed the experimental results and acquired a series of Outcomes that are the foundation of the Automated Model Selection Technique.

This technique analyzes the characteristics of the time-series datasets at the beginning, then chooses a suitable model based on the Outcomes we got. It achieves selecting the model without human interaction, which is labour-saving and efficient. Meanwhile, its results are accurate since it chooses models according to the characteristics of the input. In conclusion, this Automated Model Selection Technique is helpful if people want to select a suitable model for their time-series datasets.

7.2 Future Work

After this research, some future work still needs to be completed. First, we only employed DL models in this thesis, and it is better to consider more algorithms, especially those targeting time-series data. This kind of model aims at time-series data. Specifically, trends and seasonality are taken into account as well. They use trends and seasonality itself while training and forecasting, while in DL models, we explore these features by extracting date features. At the same time, the result and conclusion will be more comprehensive if a large number of models are used. Second, only two datasets are used in experiments, and both are scientific data, which may limit the outcomes we get. We can acquire more accurate Outcomes based on training on a series of different kinds of datasets and make the technique more general in the future. Third, the automated model selection technique we designed is based on a static decision tree, and it may not guarantee that all kinds of datasets can be addressed. It is possible to create

a dynamic decision tree based on this. We can train the models with a series of metadatasets and acquire more pieces of Outcomes, which shows the relation between the setup or features of datasets and the models. Then, if a new dataset is received, we can compare it with metadatasets and choose the suitable model based on new Outcomes. Finally, this research shows that it is possible to select models automatically. So, the next step is finding a method to achieve this technique and applying it to real life.

References

[1] Esling, P., & Agon, C. (2012). Time-series data mining. *ACM Computing Surveys (CSUR)*, *45*(1), 1-34.

[2] Peixeiro, M. (2022). *Time series forecasting in python*. Simon and Schuster.

[3] Joseph, M. (2022). *Modern Time Series Forecasting with Python: Explore industry-ready time series forecasting using modern machine learning and deep learning*. Packt Publishing Ltd.

[4] Mahalakshmi, G., Sridevi, S., & Rajaram, S. (2016, January). A survey on forecasting of time series data. In *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)* (pp. 1-8). IEEE.

[5] Jha, B. K., & Pande, S. (2021, April). Time series forecasting model for supermarket sales using FB-prophet. In *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 547-554). IEEE.

[6] Mondal, P., Shit, L., & Goswami, S. (2014). Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*, *4*(2), 13.

[7] Box, G. E. P., & Tiao, G. C. (1975). Intervention Analysis with Applications to Economic and Environmental Problems. Journal of the American Statistical Association, 70(349), 70–79.

[8] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE tra nsactions on Signal Processing*, *45*(11), 2673-2681.

[9] Cho, Kyunghyun; van Merrienboer, Bart; Bahdanau, DZmitry; Bougares, Fethi; Schwenk, Holger; Bengio, Yoshua (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation"

[10] Gardner Jr, E. S. (1985). Exponential smoothing: The state of the art. *Journal of forecasting*, *4*(1), 1-28

[11] Rob mulla. (n.d.). *Hour Energy Consumption*. Kaggle. https://www.kaggle.com/datasets/robikscube/hourly-energy-consumption

[12] Csafrit. (n.d.). *Steel Industry Energy Consumption*. Kaggle. https://www.kaggle.com/datasets/csafrit2/steel-industry-energy-consumption

[13] Monik raj behera. (n.d.). *Medium-Ds-Unsupervised-Anomaly-Detection-Deepant-Lstmae*. Github. https://github.com/bmonikraj/medium-ds-unsupervised-anomaly-detection-deepant-lstmae/blob/master/Canadian_climate_history.csv

[14] Sumanthvrao. (n.d.). *Daily Climate Time Series Data*. Kaggle. https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data

[15] Iwasaki, Y., & Levy, A. Y. (1994, August). Automated model selection for simulation. In *AAAI* (pp. 1183-1190).

[16] Calcagno, V., & de Mazancourt, C. (2010). glmulti: an R package for easy automated model selection with (generalized) linear models. *Journal of statistical software*, *34*, 1-29.

[17] Malkomes, G., Schaff, C., & Garnett, R. (2016). Bayesian optimization for automated model selection. *Advances in neural information processing systems*, *29*.

[18] Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, *18*(25), 1-5.

[19] Bentaleb, A., Begen, A. C., Harous, S., & Zimmermann, R. (2020). Data-driven bandwidth prediction models and automated model selection for low latency. *IEEE Transactions on Multimedia*, *23*, 2588-2601.

[20] Ying, Y., Duan, J., Wang, C., Wang, Y., Huang, C., & Xu, B. (2020). Automated model selection for time-series anomaly detection. *arXiv preprint arXiv:2009.04395*.

[21] Wang, C., Chen, X., Wu, C., & Wang, H. (2022). AutoTS: Automatic time series forecasting model design based on two-stage pruning. arXiv preprint arXiv:2203.14169.

[22] Saleem, S., & Kumarapathirage, S. (2023, June). AutoNLP: A Framework for Automated Model Selection in Natural Language Processing. In *2023 18th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-4). IEEE.

[23] M. I. JordanT. M. Mitchel,Machine learning: Trends, perspectives, and prospects.*Science***349**,255-260(2015).DOI:10.1126/science.aaa8415

[24] Zhang, J., Li, Y., Tian, J., & Li, T. (2018, October). LSTM-CNN hybrid model for text classification. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)* (pp. 1675-1680). IEEE.

[25] Simeon kostadinov. (2017, December 16). *Understanding GRU Networks*. Towardsdatascience. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[26] Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM

networks. In Proceedings of the IEEE International Joint Conference on Neural Networks(IJCNN), Montreal, QC, Canada, 31 July–4 August 2005; Volume 4, pp. 2047–2052.

[27] Alawneh, L., Mohsen, B., Al-Zinati, M., Shatnawi, A., & Al-Ayyoub, M. (2020, March). A comparison of unidirectional and bidirectional lstm networks for human activity recognition. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)* (pp. 1-6). IEEE.

[28] Lu, R., & Duan, Z. (2017). Bidirectional GRU for sound event detection. *Detection and Classification of Acoustic Scenes and Events*, 1-3.

[29] Lynn, H. M., Pan, S. B., & Kim, P. (2019). A deep bidirectional GRU network model for biometric electrocardiogram classification based on recurrent neural networks. *IEEE Access*, *7*, 145395-145405.

[30] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.

[31] Shraddha goled. (2022, March 15). *The Beginning of the End for Convolutional Neural Networks?* Analyticsindiamag. https://analyticsindiamag.com/the-beginning-of-the-end-for-convolutional-neural-networks/#:~:text=Yann%20LeCun's%20earliest%20breakthroughs%20came,at%20the%20University%20of%20Toronto

[32] Han, D., Liu, Q., & Fan, W. (2018). A new image classification method using CNN transfer learning and web data augmentation. *Expert Systems with Applications*, *95*, 43-56.

[33] Wang, J., Yu, L. C., Lai, K. R., & Zhang, X. (2016, August). Dimensional sentiment analysis using a regional CNN-LSTM model. In *Proceedings of the 54th annual meeting of the*

association for computational linguistics (volume 2: Short papers) (pp. 225-230).

[34] Kim, T. Y., & Cho, S. B. (2019). Predicting residential energy consumption using CNN-LSTM neural networks. *Energy*, *182*, 72-81.

[35] Ullah, A., Ahmad, J., Muhammad, K., Sajjad, M., & Baik, S. W. (2017). Action recognition in video sequences using deep bi-directional LSTM with CNN features. *IEEE access*, *6*, 1155-1166.

[36] Lu, W., Li, J., Li, Y., Sun, A., & Wang, J. (2020). A CNN-LSTM-based model to forecast stock prices. *Complexity*, *2020*, 1-10.

[37] Liang, S., Zhu, B., Zhang, Y., Cheng, S., & Jin, J. (2020, December). A double channel CNN-LSTM model for text classification. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)* (pp. 1316-1321). IEEE.

[38] S. Siami-Namini, N. Tavakoli and A. Siami Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), Orlando, FL, USA, 2018, pp. 1394-1401, doi: 10.1109/ICMLA.2018.00227.

[39] Meyler, A., Kenny, G., & Quinn, T. (1998). Forecasting Irish inflation using ARIMA models.

[40] Wikipedia contributors. (2023, December 6). Mean squared error. In *Wikipedia, The Free Encyclopedia*. Retrieved 06:25, January 15, 2024,

from https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=1188531059

[41] Wikipedia contributors. (2023, November 25). Mean absolute error. In *Wikipedia, The Free Encyclopedia*. Retrieved 21:29, February 1, 2024, from https://en.wikipedia.org/w/index.php?title=Mean_absolute_error&oldid=1186756191

[42] Oreilly. (n.d.). *PRINCIPLES OF FORECASTING*. OREILLY. https://www.oreilly.com/library/view/operations-management-an/9781118122679/ch8-sec004.html#:~:text=Forecasts%20are%20more%20accurate%20for%20shorter%20than%20longer%20time%20horizons,much%20in%20the%20short%20run.

[43] Joshua Gordon. (2023, June 20). *Practical Guide for Feature Engineering of Time Series Data*. Dotdata. https://dotdata.com/blog/practical-guide-for-feature-engineering-of-time-series-data/#:~:text=DateTime%20feature%20engineering%20is%20the,accuracy%20in%20machine%20learning%20models

[44] Raghav vashisht. (2021, January 26). *When to Perform a Feature Scaling?* Atoti. https://www.atoti.io/articles/when-to-perform-a-feature-scaling/#:~:text=Feature%20scaling%20is%20a%20method,during%20the%20data%20preprocessing%20step

[45] Wei hao khoong. (2023, January 21). *Why Scaling Your Data Is Important*. Medium. https://medium.com/codex/why-scaling-your-data-is-important-1aff95ca97a2#:~:text=Scaling%20the%20data%20can%20help,for%20it%20to%20work%20well

[46] Baijayanta Roy. (2020, April 6). *All about Feature Scaling Scale Data for Better Performance of Machine Learning Mode*. Towardsdatascience.

https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

[47] Patro, S. G. O. P. A. L., & Sahu, K. K. (2015). Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

[48] Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, *31*(7), 1235-1270.

[49] Lau, M. M., & Lim, K. H. (2018, December). Review of adaptive activation function in deep neural network. In *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)* (pp. 686-690). IEEE.

[50] Fu, R., Zhang, Z., & Li, L. (2016, November). Using LSTM and GRU neural network methods for traffic flow prediction. In *2016 31st Youth academic annual conference of Chinese association of automation (YAC)* (pp. 324-328). IEEE.

[51] Shewalkar, A., Nyavanandi, D., & Ludwig, S. A. (2019). Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU. *Journal of Artificial Intelligence and Soft Computing Research*, *9*(4), 235-245.

[52] She, D., & Jia, M. (2021). A BiGRU method for remaining useful life prediction of machinery. *Measurement*, *167*, 108277.

[53] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.

[54] Zach deane-mayer. (n.d.). *Build and Compile a Model*. Datacamp. https://campus.datacamp.com/courses/advanced-deep-learning-with-keras/the-keras-functional-api?ex=5#:~:text=Finally%2C%20you%20must%20compile%20the,compilation%2C%20you

%20select%20an%20optimizer

[55] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[56] Satyam. (2022, March 24). *Everything You Need to Know about Model Fitting in Machine Learning*. Medium. https://medium.com/@satyam3196/everything-you-need-to-know-about-model-fitting-in-machine-learning-4f93dccc6bf1

[57] Jabbar, H., & Khan, R. Z. (2015). Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, *70*(10.3850), 978-981.