Department of Mathematics

BSc (Hons) Mathematics for Data Science

Academic Year 2020 - 2024

Study on stock price prediction based on LSTM neural network and random forest

Yiding Zhao

2053770

Supervised by Dr Jia Wei Lim

A report submitted in partial fulfilment of
the requirements for the degree of Bachelor of Science

Brunel University London
Department of Mathematics
Uxbridge
Middlesex
UB8 3PH
United Kingdom
T: +44 1895 203397
F: +44 (0) 1895 251686

Abstract

Predicting stock market prices accurately is a highly challenging task due to its volatile and complex nature, influenced by numerous uncertain factors. To date, numerous studies have attempted to address this complex problem, with machine learning or deep learning methods proving to yield satisfactory results in stock price prediction. This experiment, based on nearly 17 years of data from the Chinese stock market, established various models for predicting stock closing prices using the random forest algorithm and LSTM neural networks. Information obtained from these models was utilized to calculate certain indicators to evaluate the predictive capabilities of different models. Ultimately, two distinct conclusions were drawn: the random forest algorithm exhibited the strongest stock price prediction capability when incorporating data from the preceding thirty trading days, while LSTM also demonstrated good performance in predicting stock price trends.

Keywords: Stock Prediction, LSTM, Random Forest, EDA.

Acknowledgements

Here, I would like to express my sincere gratitude to my mentor, Dr.Lim, for her unwavering support throughout this journey. In our weekly meetings, her insights have sparked many new ideas for me, enabling me to refine my experiments continuously. When I faced challenges, her patient guidance illuminated the path forward for me. Moreover, she has encouraged and assisted me during moments of self-doubt. Therefore, she is not only a dedicated mentor but also a friend whom I deeply respect.

Furthermore, I would like to thank my parent for their unwavering support in my busy and demanding life, providing me with essential support and being my strong pillars.

My last tribute is to those who know I am not perfect but still love me.

Completing the entire project has made me realize that even in the face of numerous challenges, one must believe:

”Per Aspera Ad Astra”

Contents

List of Figures

Chapter 1

Introduction

Due to various factors such as pandemics, wars, natural disasters, and others, despite continuous improvements and enhancements in the global financial condition, stock markets still progress slowly and occasionally face severe shocks. [1].Therefore, investing in the stock market entails considerable risks. Predicting stock prices can effectively forecast future stock trends, thereby mitigating investment risks: When stock prices experience a significant increase accompanied by a corresponding rise in trading volume, the upward trend may likely persist; however, if stock prices continue to rise while trading volume fails to match, it may indicate a decline in market activity, insufficient purchasing power, and the potential for false increases in stock prices, necessitating cautious investment decisions.

Returning to stock price prediction, it can be understood that the essence of the stock price prediction issue lies in time series analysis. Time series data can be defined as a sequence of observations on a selected variable over time. In the context of stock price prediction, the variable of interest is the stock price, which can be either univariate or multivariate. Univariate data may encompass specific features of the stock, such as trading volume or turnover rate, whereas multivariate data include a broader array of factors that may influence stock trends, such as investor psychology and behavior, public sentiment, sensitive financial information, historical prices, political events, and so forth [2]. It is precisely these variables that make accurate prediction of stock prices generally an unattainable task. The myriad complex factors mentioned above can collectively determine the price of a particular stock. This exacerbates the risk in the stock market, which not only potentially entails economic losses for investors but also may engender certain adverse effects on the economic development of enterprises and countries [3]. Many researchers have worked and proposed their ideas to forecast the market price to make a profit while trading using various methods such as algorithms and statistical analysis [4]. Among these methodologies, two are predominantly mainstream: the first being machine learning methods and the second being deep learning methods.

Among these, the mainstream machine learning methods include Artificial Neural Network (ANN) [5], Support Vector Machine (SVM) [6], and Random Forests (RF) [7]. ANN serve as flexible modeling frameworks and versatile approximators applicable to a wide range of time series forecasting tasks with high accuracy. However, despite their numerous advantages, the performance of ANN may not be satisfactory for certain time series [8]. SVM represent a useful technique for data classification [9], regression [10], and prediction [11]. When combined with

financial statement analysis, SVM can effectively enhance stock prediction accuracy by utilizing Gaussian radial (RBF) [12] as the function for constructing prediction models [13]. Random Forests predominantly tackle classification problems. Hence, when applied to predict stock prices, the stock price prediction task needs to be transformed into a classification problem predicting stock price movements.

Regarding deep learning methods, two approaches are most suitable for stock prediction problems: the first involves constructing Recurrent Neural Network (RNN) [14], and the second involves Long Short-Term Memory Recurrent Neural Network (LSTM) [15]. For RNN, it possesses short-term memory characteristics, but it is also due to this feature that RNN can be influenced by short-term memory. When a time series is sufficiently long, RNN may struggle to transmit information from earlier time steps to later ones. Therefore, when attempting to predict on a lengthy time series data, RNN may overlook crucial information from the outset, resulting in the issue of vanishing gradients during backpropagation. Additionally, when computations become overly complex during the information propagation process, gradient explosion may occur in RNN. LSTM represents an enhanced variant of RNN, addressing to a certain extent the issues of vanishing and exploding gradients inherent in traditional RNN, thereby facilitating the learning of long-term dependencies.

This study utilizes a dataset from the Chinese stock market on Kaggle [16] and selects the stock with the code sh600000, namely the stock data of Shanghai Pudong Development Bank (SPDB), as the research data. This study employs Exploratory Data Analysis (EDA) to conduct preliminary analysis and preprocessing of the dataset. After that, using employs the random forest algorithm and an enhanced version of RNN, namely LSTM neural networks, as methods for stock price prediction. Models are constructed and predictions are made using both methods. The effectiveness of the two models is evaluated and compared through the calculation of Mean Squared Error (MSE) and Mean Absolute Error (MAE). Finally, the optimal prediction model is chosen, and a detailed analysis is conducted to compare between the models and explore the specific factors influencing the predictive capability of the models.

1.1 Aim and Objectives

The aim of this study is to compare the accuracy of stock closing price prediction between the random forest method in machine learning and the LSTM method in deep learning, and to identify the optimal approach for predicting the closing prices of sh600000. To achieve this research aim, the following objectives are necessary:

1. Conduct an EDA to analyze and clean the raw dataset.
2. Utilize the random forest algorithm to train the model, make predictions, and calculate the MSE and MAE.
3. Establish the LSTM neural network, train the model, make predictions, and compute the MSE and MAE.

4. Generate visual representations, compare the value of MSE and MAE, and delineate the advantages and disadvantages of each method in predicting stock closing prices.

5. Integrate the analysis results and select the optimal model.

6. Conduct an in-depth analysis of the factors influencing the predictive accuracy of the selected method.

7. Discuss potential areas for further research and improvement in forecasting stock prices.

1.2 Dissertation Outline

This paper is structured into four main chapters. The first chapter serves as an introductory background, providing a comprehensive overview of the current global stock market conditions. It also briefly outlines the prevailing machine learning and deep learning methods in stock price prediction and delineates the specific objectives and implementation methodologies of this study. The second chapter, methodology, primarily elucidates the specifics of EDA, LSTM, and random forest techniques. The third chapter delves into the execution of EDA and presents the findings. The fourth chapter focuses on constructing a random forest model for predicting stock trends and showcases the resultant predictions. The fifth chapter entails the development of an LSTM model for stock trend prediction and presents the forecast outcomes. Finally, the sixth chapter encapsulates a comprehensive analysis of results and draws conclusions ,presenting some personal insights and recommendations.

Chapter 2

Methodology

In this chapter, I will expound upon the specific methodologies employed for data collection, as well as the theoretical models of LSTM and random forest. Furthermore, I will elucidate the theoretical processes involved in EDA.

2.1 Data Collection

This dataset is from Kaggle. The dataset under consideration is an augmented Chinese stock market dataset, comprising not only OHLC (Open, High, Low, Close) prices and trading volumes but also incorporating additional daily financial ratios such as Price-to-Earnings (P/E) ratio, Price-to-Book (P/B) ratio, Price-to-Sales (P/S) ratio, dividend yield, and so on. The data span from January 4, 2005, to May 11, 2022. Additionally, this dataset encompasses a total of 4714 stocks. Given that the objective of this study is to forecast individual stocks, during the data preprocessing phase, I selected a representative stock for analysis, namely the stock of SPDB (with the code sh600000).

2.2 Exploratory Data Analysis

Due to a lack of historical precedents, the specific process of EDA is actually defined by researchers themselves based on practical considerations [17]. Hence, in this article, I primarily delineate the process of EDA into two steps. The first step involves interpreting the data, which includes determining the size of the dataset, the number of variables present, and the types of data within each variable. The second step encompasses data cleansing and visualization, wherein specific data pertinent to the research are filtered from the dataset. Additionally, graphical representations are generated based on certain variables, and specific methodologies are employed to eliminate outliers within the dataset.

2.3 Random Forests

In order to gain insight into random forests, it is imperative to comprehend the concept of decision trees. The random forest algorithm represents a typical instance of ensemble learning,

with decision trees constituting its fundamental units. The figure below depicts the structure of a decision tree.

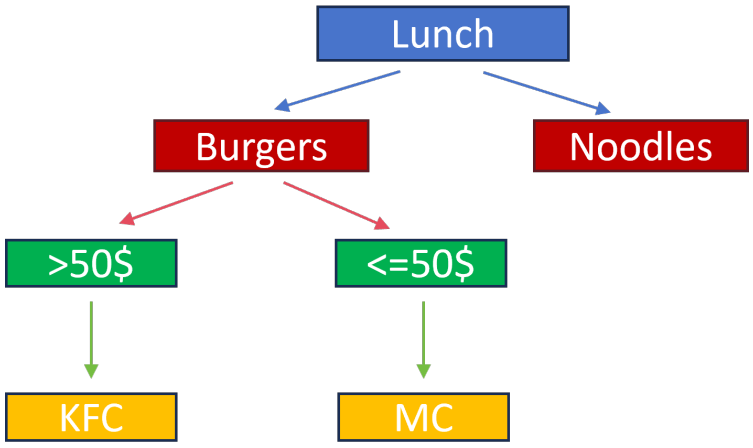

Figure 2.1: Decision tree

As illustrated in Figure 2.1, a decision is made regarding what to have for lunch: if opting for a hamburger and the budget exceeds fifty dollars, Kentucky Fried Chicken (KFC) is a viable option; if the budget is below fifty dollars, McDonald's is a suitable choice. This decision-making process is represented by a binary tree, referred to as a decision tree.

Random forest is a classical ensemble model using the concept of bagging, it composed of multiple decision trees serving as classifiers, with the output category determined by the collective outcomes of individual outputs. To offer a vivid analogy: imagine a forest convening a meeting to deliberate whether to penalize loggers; the ultimate decision relies on the voting outcome, where the category receiving the most votes determines the forest's classification result. Each tree in the forest holds voting power, rendering every tree independent. Predictions from 99.9% of unrelated trees encompass all scenarios, neutralizing each other. Predictions from a minority of outstanding trees transcend the cacophony of "noise," leading to a robust prediction. By aggregating the classification outcomes of several weak classifiers through voting, a strong classifier is formed, reflecting the essence of bagging in random forest.

Figure 2.2: Forest

Figure 2.2 provides a figurative representation of this scenario: red trees cast dissenting votes, while green trees cast affirmative votes, each operating independently and without interference from others.

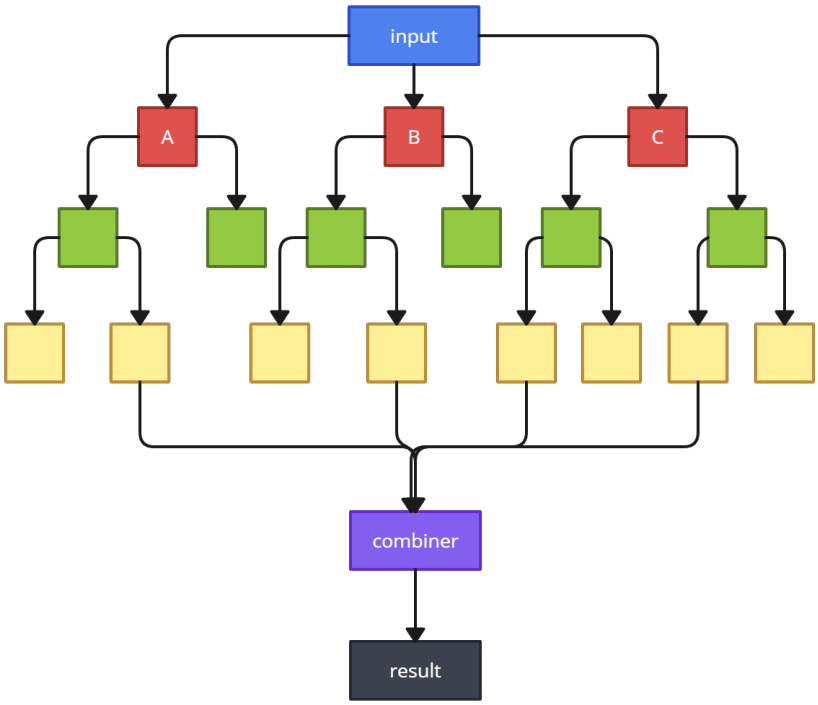

Figure 2.3: Random forest structure

Figure 2.3 depicts the concrete process of the random forest algorithm. It comprises three decision trees denoted as A, B, and C. Within this schematic, the combiner, when dealing with classification tasks, opts for the majority classification outcome as the final result. In regression tasks, it aggregates multiple regression results by averaging them to derive the ultimate outcome.

The generation principle of trees in random forests is as follows:

- If the size of the training set is N, then for each tree, the bootstrap method is utilized, which involves randomly selecting N training samples from the training set with replacement to form the training set for that tree. Two main questions arise here: the first question pertains to why random sampling of the training set is necessary, and the second question addresses why sampling with replacement is employed. Regarding the first question, the primary rationale is that without random sampling, if each tree's training set remains identical, the resulting classification outcomes of the trees would also be identical, rendering the utilization of bagging unnecessary. Concerning the second question, the necessity for sampling with replacement arises from the fact that without it, each tree's training samples would be unique and disjointed, leading to biased and one-sided training. Consequently, each tree would exhibit substantial discrepancies. However, since the ultimate classification in a random forest which relies on the collective decisions of multiple trees (weak classifiers), these decisions should strive for consensus. Therefore, employing entirely distinct training sets for each tree does not contribute to the final classification outcome and merely resembles a blind search.

- If each sample has M feature dimensions, and a constant $m \ll M$ is specified, a random selection of m features is made from the M features. At each split of the tree, the optimal feature is selected from this subset of m features.
- Each tree is allowed to grow to its maximum extent, and there is no pruning process applied.

The classification performance (error rate) of random forests is influenced by two factors: Firstly, the correlation between any two trees in the forest. The higher the correlation, the higher the error rate. Secondly, the classification capability of each tree in the forest. The stronger the classification capability of each tree, the lower the overall error rate of the forest. Therefore, reducing the number of features selected, denoted as m, will correspondingly decrease both the correlation and classification capability of the trees in the forest. Conversely, increasing m will result in an increase in both. Thus, the key lies in selecting the optimal value or range for m, which is the sole parameter of random forests.

2.4 LSTM and RNN

Since the publication of the seminal paper on LSTM [18], this emerging deep learning method has been widely adopted across various domains. Particularly, its impact has been notably significant in language modeling, speech-to-text transcription, machine translation, time series prediction, and other applications [19, 20]. As previously mentioned, LSTM represents a specialized form of RNN, hence RNN can be referred to as the parent network of LSTM [21].

2.4.1 Recurrent Neural Network (RNN)

Therefore, before delving into the explanation of LSTM, it is imperative to comprehend what RNN entails. However, understanding the concept of RNN necessitates a preliminary grasp of the most fundamental single-layer network structure. The following diagram depicts the structure of a single-layer neural network.

Figure 2.4: Single-layer network structure

In Figure 2.4, x represents the input layer of the neural network, while y denotes the output layer. The corresponding mathematical formula is:

$$y = f(W \cdot x + b). \tag{2.1}$$

In Equation 2.1, f denotes the activation function, W represents the weights, and b denotes the bias of the neuron. However, this primitive neural network architecture is ill-suited for processing sequential data. This is primarily because in a single-layer neural network, each output node typically connects only to input nodes without establishing long-term dependencies between nodes. Consequently, single-layer neural networks fail to effectively capture the temporal properties inherent in sequential data. Most significantly, single-layer neural networks lack a memory mechanism; thus, they are unable to establish connections between input samples, thus failing to capture long-term dependencies in sequential data. Therefore, to handle sequential data such as time series data, as in the case of daily stock prices in this study, RNN was introduced. RNN introduce the concept of hidden layers, which can extract features from sequential data and subsequently convert them into outputs. The structure of an RNN is illustrated in the following diagram.

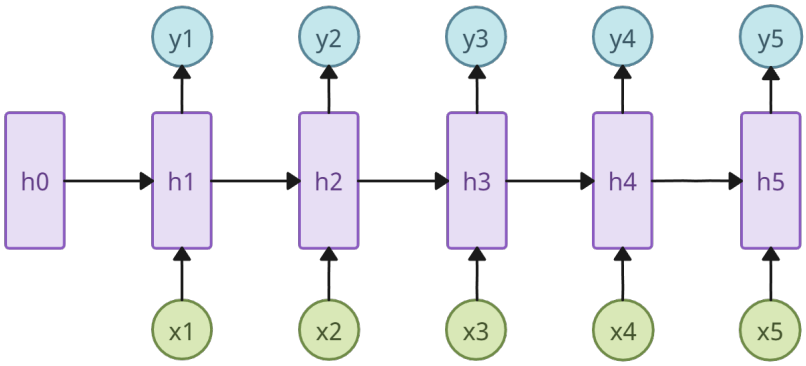

Figure 2.5: RNN structure

In Figure 2.5, for the sake of convenience, only the case of a sequence length of 5 is depicted. In reality, this computation process can continue infinitely. In this figure, both shapes represent vectors, while arrows indicate a transformation applied to the vector. The formula for the hidden layer h_1 is represented as:

$$h_1 = f\left(U \cdot x_1 + W \cdot h_0 + b\right). \tag{2.2}$$

In Equation 2.2, x_1 represents the input at time 1, U denotes the weight matrix from the input layer to the hidden layer, h_1 signifies the value of the hidden layer at time 1, and W refers to the weight matrix for passing the value of the previous hidden layer from one time step to the next. In the specific computation process, the parameters U, W, and b used at each step remain constant. In other words, the U, W, and b at different time steps are identical, which is the primary characteristic of RNN: parameter sharing. How can we understand parameter sharing? Despite x_t representing inputs at different time steps $\{x_{t-1}, x_t, x_{t+1}\}$ and t equal to $0, 1, 2, \ldots, n$. When they are input into an RNN, they are not input as individual vectors. Instead, they are input in the form of a matrix such as $[x_0, x_1, x_2, \ldots, x_n]$, and then this matrix is transformed by the weight matrix U. Therefore, the information actually represents inputs at the same time step, albeit with a different computation order. However, in LSTM, the weights are not shared because they are present in two different vectors, whereas in RNN, the weights are contained

within a single vector, albeit at different time steps. This is one of the differences between RNN and LSTM. Consequently, the formula for h_t can be summarized as:

$$h_t = f\left(U \cdot x_t + W \cdot h_{t-1} + b\right) \quad t = 1,2,3.....n\,. \tag{2.3}$$

In Equation 2.3, the values of parameters U, W, and b are identical.

After elucidating the input section, we now proceed to explain the output section. The method to obtain output values involves direct computation using h_t. Taking y_1 as an example, the formula for y_1 is:

$$y_1 = g\left(V \cdot h_1 + c\right). \tag{2.4}$$

In Equation 2.4, V represents the weight matrix from the hidden layer to the output layer, and c denotes the bias. Similarly, RNN shares parameters for the output layer. It is noteworthy that the activation function here is denoted as $g(x)$, which differs from the activation function $f(x)$ used from the input layer to the hidden layer. In practical applications, $f(x)$ is generally chosen as the hyperbolic tangent ($tanh$) function. It will be mentioned later that one of the limiting factors of RNN is related to this tanh activation function. On the other hand, the choice of $g(x)$ is typically determined based on the task requirements. For example, for binary classification tasks, the $sigmoid(x)$ function can be used, while for multi-class classification tasks, the $softmax(x)$ function is commonly employed. Generalizing this formula, we obtain Equation 2.5:

$$y_t = g\left(V \cdot h_t + c\right) \quad t = 1,2,3.....n\,. \tag{2.5}$$

2.4.2 The Limitations of RNN

The limitations of RNN manifests in the form of gradient explosion and vanishing gradients. In the subsequent discussion, I will delve into the specific reasons behind the emergence of these issues. The name Recurrent Neural Network is self-explanatory; from the name itself, we understand that it is a neural network containing recurrent connections. How can we interpret these recurrent connections? We can make slight modifications to Figure 2.6 to obtain the following diagram.

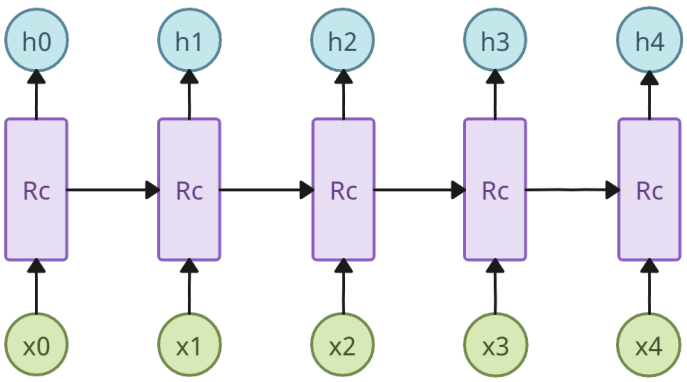

Figure 2.6: Another representation of the RNN structure

Figure 2.6 presents an alternative representation where the output is depicted as the hidden layer h_t rather than y_t. In this structure, each module of the neural network, termed as the RNN cell (abbreviated as *Rc* in the figure), reads a specific input x_t and outputs a value h_t, which is then continuously cycled. If we do not unfold the recurrence, it can be represented using the following diagram.

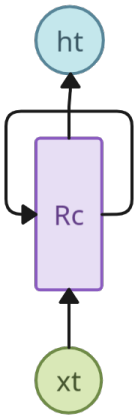

Figure 2.7: Simple RNN structure

Combining Equation 2.3, we can unfold the RNN cell, resulting in the following diagram.

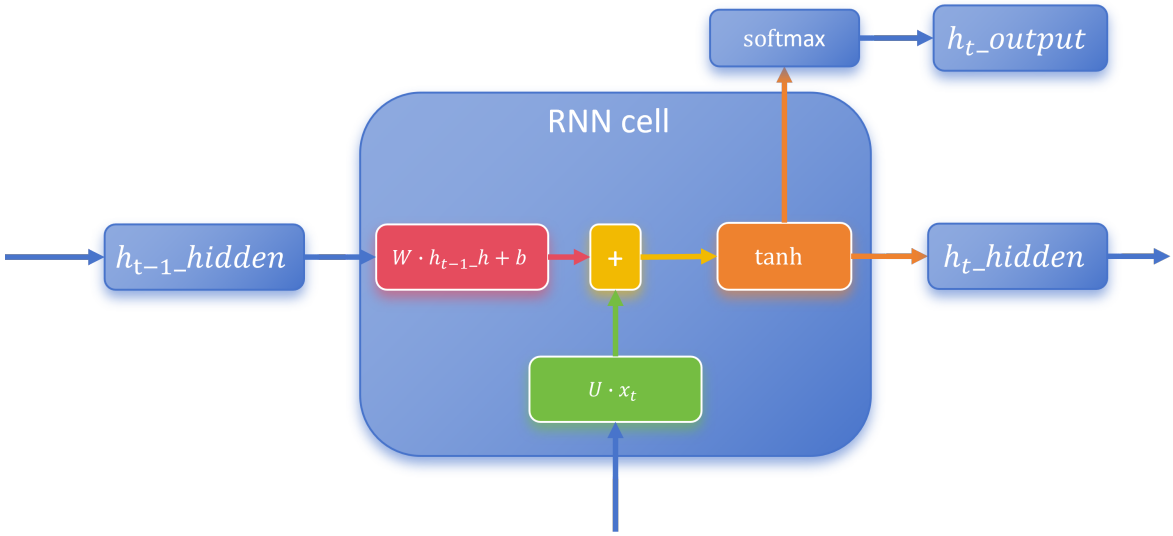

Figure 2.8: RNN cell structure

In Figure 2.8, the dimension of U is $hidden\_size \times input\_size$, while the dimension of W is $hidden\_size \times hidden\_size$. Therefore, each RNN cell has the same dimension. The precise matrix operation can be represented as:

$$U \cdot x_t + W \cdot h_{t-1} = [U \;\; W] \cdot \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}. \tag{2.6}$$

Moreover, as mentioned earlier, the activation function *tanh* is also depicted in the figure. However, the primary reason for the limitations of RNN is closely related to this *tanh* activation function. It is well understood that neural networks are trained by backpropagating errors

through the network, adjusting the weights and biases matrices to minimize errors in future iterations [22]. The backpropagation in RNN, also known as Back Propagation Through Time (BPTT), serves the purpose of computing parameter gradients and updating these parameters continuously to minimize the loss function. Here, the parameter gradients are the partial derivatives of the loss function with respect to the parameters, and the parameter update formula is given by:

$$W = W - \alpha \frac{\partial L}{\partial W}. \tag{2.7}$$

In the Equation 2.7, $\frac{\partial L}{\partial W}$ represents the partial derivative of the loss function with respect to the parameters, and α represents the learning rate. Therefore, the key challenge in the backpropagation process is to compute this partial derivative. To compute this partial derivative, we first need to define the loss function L. Let's assume that the loss function at time t is L_t. Since there are multiple time steps, the total loss function L is the sum of the loss functions at all time steps. Ultimately, we obtain Equation 2.8.:

$$L = \sum_{t=0}^{T} L_t. \tag{2.8}$$

Next, we need to compute the partial derivative of the loss function with respect to the parameters. Since W appears at every time step, the gradient of W at time t is equal to the sum of the gradients of W with respect to all time steps. We obtain the following formula:

$$\frac{\partial L_t}{\partial W} = \sum_{s=0}^{T} \frac{\partial L_t}{\partial W_s}. \tag{2.9}$$

By substituting Equation 2.8 into Equation 2.9, we obtain Equation 2.10. That is, the total gradient $\frac{\partial L}{\partial W}$ of W is equal to the sum of the gradients of W at all time steps:

$$\frac{\partial L}{\partial W} = \sum_{t=0}^{T} \sum_{s=0}^{T} \frac{\partial L_t}{\partial W_s}. \tag{2.10}$$

Finally, we will utilize Equation 2.7 to update the parameters, which completes one full backpropagation process. However, it is during the backpropagation process that the issues of vanishing gradients and exploding gradients may arise. Suppose we have an RNN neural network, but it only has a time series of three time steps. Its forward propagation formula is as follows:

$$\begin{aligned} h_1 &= f(U \cdot x_1 + W \cdot h_0 + b) \qquad & y_1 &= g(V \cdot h_1 + c), \\ h_2 &= f(U \cdot x_2 + W \cdot h_1 + b) \qquad & y_2 &= g(V \cdot h_2 + c), \\ h_3 &= f(U \cdot x_3 + W \cdot h_2 + b) \qquad & y_3 &= g(V \cdot h_3 + c). \end{aligned} \tag{2.11}$$

For clarity, here we still employ Equations 2.3 and 2.5 as the forward propagation formulas for the RNN. After completing the forward propagation, we commence the backpropagation process. At $t = 3$, we compute the partial derivatives of L with respect to W, U, and V, yielding

the following results:

$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial W} + \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W},$$

$$\frac{\partial L_3}{\partial U} = \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial U} + \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial U} + \frac{\partial L_3}{\partial y_3}\frac{\partial y_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial U}, \tag{2.12}$$

$$\frac{\partial L_3}{\partial V} = \frac{\partial L_3}{\partial h_3}\frac{\partial h_3}{\partial V}.$$

From Equation 2.12, it is evident that since U, V, and W are shared across different time steps, when computing the partial derivatives with respect to U, V, and W at a particular time step, we need to consider all preceding information up to that time step. Based on the above expressions, we can derive a general formula for the partial derivatives of L with respect to U, V, and W at time step t. Taking W and U as examples (Because W and U propagate through the hidden layers), we obtain the final general formulas:

$$\frac{\partial L_t}{\partial W} = \sum_{m=0}^{T}\frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\prod_{j=m+1}^{T}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_j}{\partial W},$$

$$\frac{\partial L_t}{\partial U} = \sum_{m=0}^{T}\frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\prod_{j=m+1}^{T}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_j}{\partial U}. \tag{2.13}$$

Substituting Equation 2.3 into the multiplication part, where the activation function is *tanh*, the multiplication part eventually takes the following form:

$$\prod_{j=m+1}^{T}\frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=m+1}^{T}\tanh' \cdot W,$$

$$\prod_{j=m+1}^{T}\frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=m+1}^{T}\tanh' \cdot U. \tag{2.14}$$

Next, we will plot the graphs of the *tanh* activation function and its derivative as shown in Figures 2.9 and 2.10. It can be observed that the derivative of the *tanh* function is always less than 1. This phenomenon implies that when the *tanh* activation function is applied in neural networks, and multiple values less than 1 are multiplied together, the product tends towards 0, leading to the issue of gradient vanishing.

Conversely, when the values of W or U are large, this product tends towards infinity, resulting in the problem of gradient explosion. Gradient explosion causes the gradient values to become extremely large, leading to overly significant parameter updates, destabilizing the network, and potentially preventing convergence.

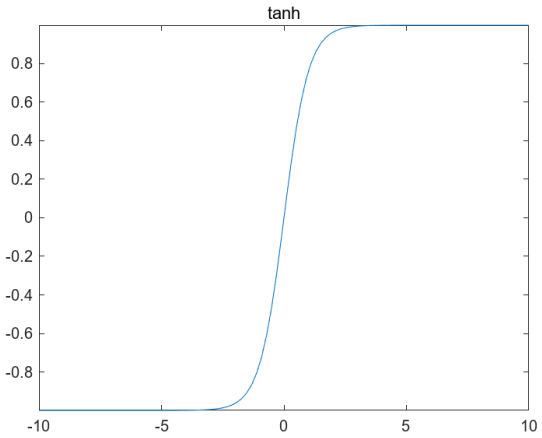

Figure 2.9: tanh function

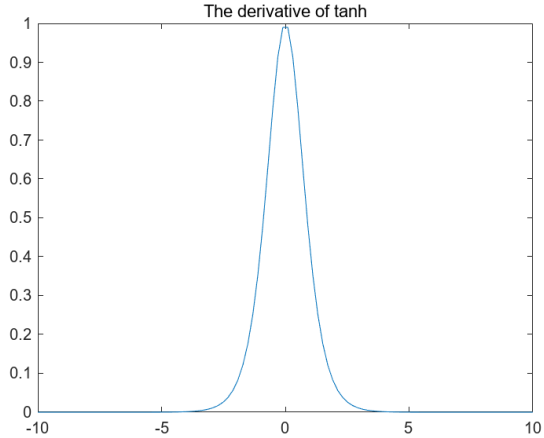

Figure 2.10: tanh_derivative

The cause of the exploding gradient problem is typically attributed to improper choices of W and U, which are not commonly encountered when using RNN to investigate problems. However, the vanishing gradient problem is quite common. When vanishing gradients occur, the memory capacity of the RNN weakens, causing it to forget the content it has previously seen in longer sequences.Therefore, RNN exhibit only short-term memory.

2.4.3 Long Short-Term Memory Neural Network (LSTM)

LSTM is a special form of RNN. Unlike traditional RNN, LSTM effectively addresses the issues of vanishing and exploding gradients, making it widely applicable to various time series problems. For instance, it has been extensively used in traffic prediction [23], wind speed forecasting [24], precipitation time series problems [25], and so on. Meanwhile, LSTM can be represented by a structure similar to Figure 2.7, where the RNN cell is replaced by an LSTM cell. However, the internal structure of the LSTM cell differs significantly from the RNN cell. In the LSTM cell, there are three structures known as gates: the forget gate, the input gate, and the output gate. LSTM controls which information to forget, retain, or pass through unchanged using these three gates, akin to valves in a pipeline regulating the flow. The specific structure of the LSTM cell is depicted in the diagram below.

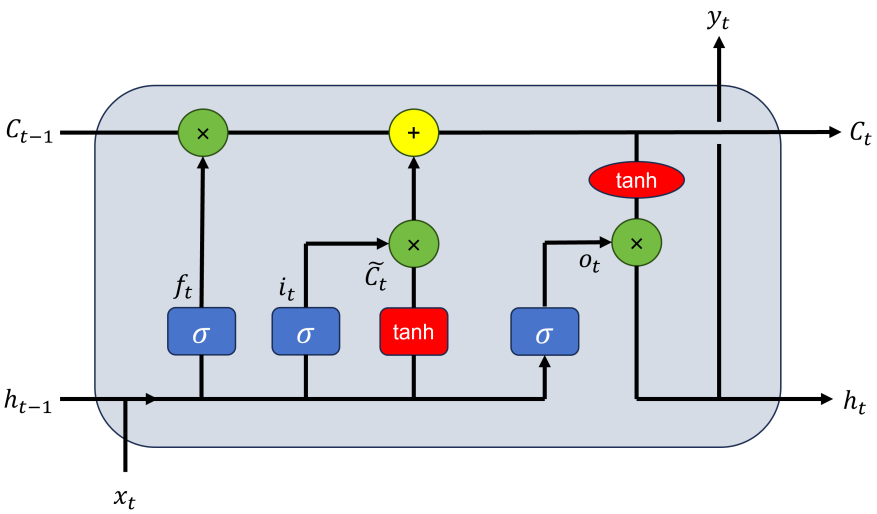

Figure 2.11: LSTM cell

Before explaining the content of each unit individually, we need to clarify 'σ' first, which represents the $sigmoid(x)$ activation function. The plot of the $sigmoid(x)$ function is illustrated in Figure 2.12.

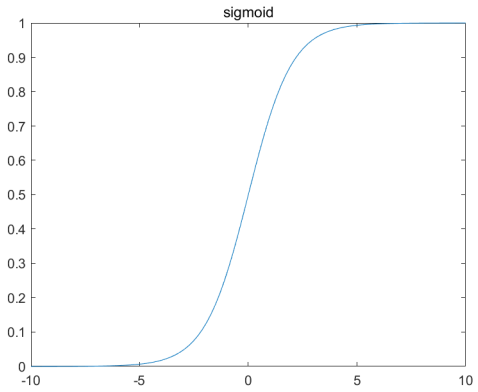

Figure 2.12: sigmoid function

From Figures 2.9 and 2.12, it can be observed that the difference between *tanh* and *sigmoid* lies in their ranges: *sigmoid* compresses values between 0 and 1, while *tanh* compresses values between -1 and 1. This setup aids in updating or forgetting information. Since any number multiplied by 0 yields 0, it gets filtered out; similarly, any number multiplied by 1 results in 1, hence it gets retained. Therefore, 1 corresponds to remembering, while 0 corresponds to forgetting. Having understood the role of the *sigmoid* function, I will now proceed to explain the function of each remaining unit one by one.

Forget Gate

Firstly, we have the forget gate, denoted by f_t in the diagram. The forget gate is very simple, consisting of just a *sigmoid* function. When the hidden state information h_{t-1} from time $t-1$ and the input information x_t at time t enter the forget gate, it combines them and undergoes a non-linear mapping using the *sigmoid* function, ultimately producing a vector f_t. It is noteworthy that each dimension of f_t falls between 0 and 1, with 0 indicating complete forgetting and 1 indicating complete retention. Finally, it is multiplied element-wise with C_{t-1} (where C_t denotes the cell state at time t, which will be discussed in subsequent explanations). The formula for the forget gate f_t can be expressed as:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right). \tag{2.15}$$

It should be noted that W_f in equation 2.15 can be further subdivided into W_{fh} and W_{fx}. Therefore, another way to represent this equation is:

$$f_t = \sigma\left(W_{fh} \cdot h_{t-1} + W_{fx} \cdot x_t + b_f\right). \tag{2.16}$$

Input Gate

The role of the input gate is to determine what kind of information can be stored in the neural network. It is relatively more complex compared to the forget gate and consists of two parts:

the *sigmoid* function layer and the *tanh* function layer. The *sigmoid* function layer determines which values need to be updated, denoted by the symbol i_t, while the *tanh* function layer constructs a candidate cell state, preserving information from x_t and h_{t-1}. Then, it is multiplied by the values of i_t to decide which information is useful. Similar to the forget gate, where 0 means complete forgetting and 1 means complete retention, the information retained here is added as new memory to the new cell state. Therefore, $\tilde{C}_t$ is referred to as the candidate cell state, which is just a waiting selection, and only useful information is added as new memory. The formulas for i_t and $\tilde{C}_t$ are:

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right), \tag{2.17}$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right). \tag{2.18}$$

Similarly, let's expand them,

$$\tilde{C}_t = \tanh\left(W_{Ch} \cdot h_{t-1} + W_{Cx} \cdot x_t + b_C\right), \tag{2.19}$$

$$i_t = \sigma\left(W_{ih} \cdot h_{t-1} + W_{ix} \cdot x_t + b_i\right). \tag{2.20}$$

Cell State

A key aspect of LSTM is the concept of the cell state. Referring back to Figure 2.11, the top horizontal line traversing the entire cell structure represents the cell state, denoted by the symbol C_t. The cell state functions akin to a conveyor belt, operating directly along the entire chain. The information is filtered through the three gate structures and then transmitted to the conveyor belt for computation and transfer. With minimal linear interactions, it is relatively easy for information to remain unchanged as it flows along the conveyor belt. The expression for C_t is as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \tag{2.21}$$

Output Gate

The function of the output gate is to determine the final output, which is represented by h_t. This output is based on the new cell state C_t and is performed in two parts. The first part still utilizes a *sigmoid* function layer to obtain o_t, for example, o_t=[1,1,0,1], which determines which parts of the cell state should be outputted and which parts should be retained. The formula for o_t is:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right). \tag{2.22}$$

Similar to before, it can also be expanded as follows:

$$o_t = \sigma\left(W_{oh} \cdot h_{t-1} + W_{ox} \cdot x_t + b_o\right). \tag{2.23}$$

The second part processes the new cell state through a *tanh* function layer, and finally multiplies it by o_t to obtain the final h_t value. The formula for h_t is:

$$h_t = o_t * \tanh(C_t). \tag{2.24}$$

In the end, excepting h_t, there is also a true output y_t, which has a similar expression to Equation 2.5. Another activation function (*sigmoid* or $softmax$, depending on the task requirements) needs to be applied during output. The expression for y_t is:

$$y_t = \sigma(V \cdot h_t + c). \tag{2.25}$$

2.4.4 The Backpropagation Process in LSTM

Similar to RNN, LSTM also utilize the backpropagation process for weight updates. However, the backpropagation process in LSTM is quite complex, mainly due to the presence of multiple gating mechanisms controlling the output. Therefore, computing the gradients for each gate requires significant effort. The following explanation outlines the backward propagation path, but the equations for the LSTM backpropagation process will not be derived in this essay.

Typically, after the LSTM layer, a fully connected layer (FNN) is connected, followed by a loss function. The total error propagated back from the fully connected layer to the LSTM layer is denoted as E. E is split into two branches upon entering the LSTM layer, one for C_t and the other for h_t. From a macro perspective, each LSTM cell's error propagation starts from C_t and h_t and ends at C_{t-1} and h_{t-1}. The error traverses through four modules in reverse order from the starting point to the end point. From this perspective, we can broadly classify the error propagation process in LSTM into two scenarios from the starting point: the first involves error propagation containing only h_t, such as in the output gate o_t, while the second scenario involves error originating from both C_t and h_t, including the forget gate f_t, input gate i_t, candidate memory cell $\tilde{C}_t$, and C_{t-1}. From the endpoint perspective, the error of C_{t-1} originates from both C_t and h_t, with the propagation path as follows:

$$\begin{cases} E \to C_t \to C_{t-1} \\ E \to h_t \to C_t \to C_{t-1} \end{cases}. \tag{2.26}$$

The error of h_{t-1} originates from four propagation chains, corresponding to the four modules in the LSTM:

- The error of the output gate o_t is propagated from h_t, with the propagation path as follows:

$$E \to h_t \to o_t \to h_{t-1}. \tag{2.27}$$

- The error of the candidate memory cell $\tilde{C}_t$ is propagated from h_t and C_t, with the propagation path as follows:

$$\begin{cases} E \to h_t \to C_t \to \tilde{C}_t \to h_{t-1} \\ E \to C_t \to \tilde{C}_t \to h_{t-1} \end{cases}. \tag{2.28}$$

- The error of the input gate i_t is propagated from h_t and C_t, with the propagation path as

follows:

$$\begin{cases} E \to h_t \to C_t \to i_t \to h_{t-1} \\ E \to C_t \to i_t \to h_{t-1} \end{cases}. \tag{2.29}$$

- The error of the forget gate f_t is propagated from h_t and C_t, with the propagation path as follows:

$$\begin{cases} E \to h_t \to C_t \to f_t \to h_{t-1} \\ E \to C_t \to f_t \to h_{t-1} \end{cases}. \tag{2.30}$$

Further subdividing the four propagation paths for h_{t-1}, we isolate propagation path 2.27 because C_t does not exist in this path. Eventually, we can obtain the complete expressions for computing the derivatives of these three types of propagation paths:

Type 1: Propagation Path 2.26

In 2.26, the complete partial derivative expressions for the two propagation paths are as follows:

$$\frac{\partial E}{\partial C_{t-1}} = \frac{\partial E}{\partial C_t} \cdot \frac{\partial C_t}{\partial C_{t-1}} + \frac{\partial E}{\partial h_t} \cdot \frac{\partial h_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial C_{t-1}}, \tag{2.31}$$

final get:

$$\frac{\partial E}{\partial C_{t-1}} = \left\{ \frac{\partial E}{\partial C_t} + \frac{\partial E}{\partial h_t} \cdot o_t \left[1 - \tan^2 (C_t) \right] \right\} \cdot f_t. \tag{2.32}$$

Type 2: Propagation Path 2.27

In 2.27, the complete partial derivative expressions for this propagation path is as follows:

$$\frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial h_t} \cdot \frac{\partial h_t}{\partial o_t} \cdot \frac{\partial o_t}{\partial h_{t-1}}, \tag{2.33}$$

final get:

$$\frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial h_t} \cdot \tan (C_t) \cdot o_t (1 - o_t). \tag{2.34}$$

Type 3: Propagation Path 2.28, 2.29 and 2.30

- In 2.28, the complete partial derivative expressions for the two propagation paths are as follows:

$$\frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial C_t} \cdot \frac{\partial C_t}{\partial \tilde{C}_t} \cdot \frac{\partial \tilde{C}_t}{\partial h_{t-1}} + \frac{\partial E}{\partial h_t} \cdot \frac{\partial h_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial \tilde{C}_t} \cdot \frac{\partial \tilde{C}_t}{\partial h_{t-1}}, \tag{2.35}$$

  final get:

$$\frac{\partial E}{\partial h_{t-1}} = \left\{ \frac{\partial E}{\partial C_t} + \frac{\partial E}{\partial h_t} \cdot \left[o_t \odot \left(1 - \tan^2 (C_t) \right) \right] \right\} \cdot i_t \cdot \tilde{C}_t \left(1 - \tilde{C}_t \right). \tag{2.36}$$

  Here, the symbol $\odot$ represents the Hadamard product.

- In 2.29, the complete partial derivative expressions for the two propagation paths are as follows:

$$\frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial C_t} \cdot \frac{\partial C_t}{\partial f_t} \cdot \frac{\partial f_t}{\partial h_{t-1}} + \frac{\partial E}{\partial h_t} \cdot \frac{\partial h_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial f_t} \cdot \frac{\partial f_t}{\partial h_{t-1}}, \tag{2.37}$$

final get:

$$\frac{\partial E}{\partial h_{t-1}} = \left\{ \frac{\partial E}{\partial C_t} + \frac{\partial E}{\partial h_t} \cdot \left[o_t \odot \left(1 - \tan^2 \left(C_t \right) \right) \right] \right\} \cdot C_{t-1} \cdot f_t \left(1 - f_t \right). \tag{2.38}$$

- In 2.30, the complete partial derivative expressions for the two propagation paths are as follows:

$$\frac{\partial E}{\partial h_{t-1}} = \frac{\partial E}{\partial C_t} \cdot \frac{\partial C_t}{\partial i_t} \cdot \frac{\partial i_t}{\partial h_{t-1}} + \frac{\partial E}{\partial h_t} \cdot \frac{\partial h_t}{\partial C_t} \cdot \frac{\partial C_t}{\partial i_t} \cdot \frac{\partial i_t}{\partial h_{t-1}}, \tag{2.39}$$

  final get:

$$\frac{\partial E}{\partial h_{t-1}} = \left\{ \frac{\partial E}{\partial C_t} + \frac{\partial E}{\partial h_t} \cdot \left[o_t \odot \left(1 - \tan^2 \left(C_t \right) \right) \right] \right\} \cdot \tilde{C}_t \cdot i_t \left(1 - i_t \right). \tag{2.40}$$

Finally, combining the results from all four modules, we obtain the ultimate expression as follows:

$$\frac{\partial E}{\partial \text{Sum}} = \frac{\partial E}{\partial f} + \frac{\partial E}{\partial i} + \frac{\partial E}{\partial \tilde{C}} + \frac{\partial E}{\partial o}. \tag{2.41}$$

After all the derivative calculations are completed, the final step involves propagating the error back to C_{t-1}, h_{t-1}, and X_t. For h_{t-1} and X_t, two weight matricesΔW_h and ΔW_x need to be computed, while for C_{t-1}, no matrix needs to be derived; the information can be directly passed backward. This is because C_t represents a relatively special propagation chain. During forward propagation, when computing the first time point, the initial value of C_t is a zero matrix. Subsequently, C_t accumulates continuously during the propagation process. At a certain time t when propagating information to the next time point $t+1$, neurons only pass the output state h_t backward as input to the next LSTM layer, while the current layer's C_t value will not be transmitted to the next LSTM layer. This effectively reduces the risk of gradient vanishing. Similarly, during backpropagation, ΔC_t will not be transmitted back; instead, Δh_t from the next layer will propagate backward to the previous layer as the error input. The specific computation process during forward propagation is as follows:

$$\begin{cases} h_{t-1} * W_h = \text{Sum}_h \\ X_t * W_x = \text{Sum}_x \\ \text{Sum}_h + \text{Sum}_x = \text{Sum} \end{cases}. \tag{2.42}$$

In equation 2.42, Sum_h represents the sum of gradients with respect to the output h_t, and Sum_x represents the sum of gradients with respect to the input x_t. Furthermore, the expression for backpropagation can be derived as follows:

$$\begin{cases} \Delta W_h = \left(h_{t-1} \right)^T * \text{Sum} \\ \Delta h_{t-1} = \text{Sum} * \left(W_h \right)^T \\ \Delta W_x = \left(x_t \right)^T * \text{Sum} \\ \Delta x_t = \text{Sum} * \left(W_x \right)^T \end{cases}. \tag{2.43}$$

Finally, the obtained $\frac{\partial E}{\partial \text{Sum}}$ is dimensionally reduced to obtain the bias Δb, concluding the backpropagation process. The clear paths of backpropagation and forward propagation can be ob-

served in Figures A.2 and A.3 [26] in the appendix.

2.4.5 Why does LSTM not Encounter Gradient-Related Issues?

According to Equation 2.14, it can be deduced that the gradient vanishing is related to the *tanh* function. However, from another perspective, the culprit of gradient vanishing is actually $\frac{\partial h_t}{\partial h_{t-1}}$. When $\frac{\partial h_t}{\partial h_{t-1}}$ approaches 0, the gradient vanishes, while when it approaches 1, gradient explosion occurs. The original solution LSTM proposed to address this issue is to ensure that the recursive derivative maintains a constant value, preventing both gradient explosion and vanishing. Therefore, the cell state C_t was introduced into LSTM, with the early formulation of C_t as follows:

$$C_t = C_{t-1} + i_t * \tilde{C}_t. \tag{2.44}$$

However, this formulation also presents a problem, namely uncontrolled growth of the cell state, as there are no clear constraints imposed on the cell state C_{t-1}. Therefore, modern formulations of C_t typically incorporate a forget gate f_t, as shown in equation 2.21, to scale the cell state, thereby addressing the issue of unbounded growth of the cell state. So why does the introduction of C_t not alter the gradient? Because as mentioned earlier, it is the recursive derivative that is the main culprit behind gradient vanishing. Therefore, expanding the full derivative of $\frac{\partial C_t}{\partial C_{t-1}}$, we obtain the following equation:

$$\frac{\partial C_t}{\partial C_{t-1}} = \frac{\partial C_t}{\partial f_t}\frac{\partial f_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial i_t}\frac{\partial i_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial \tilde{C}_t}\frac{\partial \tilde{C}_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial C_{t-1}} + \frac{\partial C_t}{\partial C_{t-1}}. \tag{2.45}$$

Continuing with the calculations, we obtain the following equation:

$$\begin{aligned}\frac{\partial C_t}{\partial C_{t-1}} &= C_{t-1}\sigma' \odot W_f * o_{t-1}\tanh'(C_{t-1}) \\ &+ \tilde{C}_t\sigma' \odot W_i * o_{t-1}\tanh'(C_{t-1}) \\ &+ i_t\sigma' \odot W_i * o_{t-1}\tanh'(C_{t-1}) \\ &+ f_t.\end{aligned} \tag{2.46}$$

This equation is highly complex, governed by six parameters. However, it can be determined that at any time t, its value randomly appears in the range of 0 to 1 or greater than 1. In contrast, the partial derivatives of $\frac{\partial h_t}{\partial h_{t-1}}$ in RNN are either all within the range of 0 to 1 or all greater than 1 for all time steps. Therefore, LSTM avoids the gradient vanishing and exploding issues caused by successive multiplications.

Chapter 3

The Specific Srocess of Exploratory Data Analysis

This chapter will present the specific process of exploratory data analysis conducted in this experiment and the conclusions drawn from the exploratory data analysis process, laying the groundwork for subsequent model development and stock prediction.

3.1 Data Visualization

First, visualize the data by reading it in and converting the date component into an index format. Visualize the first five and last five rows of the data, while obtaining information on its dimensions. The resulting output is as follows.

Table 3.1: **Top five rows**

date	ticker	open	high	low	close	volume	turnover
2005-01-04	sh600000	0.77	0.77	0.75	0.76	3808939.0	0.004232
2005-01-04	sh600297	0.98	0.98	0.90	0.95	81700.0	0.002043
2005-01-04	sz000800	2.20	2.22	2.09	2.13	4837784.0	0.008863
2005-01-04	sh600298	1.10	1.11	1.09	1.10	35200.0	0.001006
2005-01-04	sz000799	2.67	2.72	2.62	2.71	804348.0	0.007500

Table 3.2: **Botton five rows**

date	ticker	open	high	low	close	volume	turnover
2022-05-11	sz301109	27.50	27.80	27.01	27.03	8147809.0	0.144326
2022-05-11	sz002870	31.15	32.70	30.61	31.34	894400.0	0.010063
2022-05-11	sh601198	7.97	8.18	7.95	8.00	4837784.0	0.010100
2022-05-11	sz002890	13.90	14.21	13.87	14.03	35200.0	0.029564
2022-05-11	sz301288	35.00	36.17	32.80	33.94	804348.0	0.571786

These two tables omit some content. In the original dataset, there are a total of seventeen variables and 10,129,344 data points. These seventeen variables are ticker, open, high, low, close, volume, outstanding_share, turnover, pe, pe_ttm, pb, ps, ps_ttm, dv_ratio, dv_ttm, total_mv, and qfq_factor. The displayed tables only show data for seven variables, and among these seven variables, six will be used for modeling in subsequent analysis (excluding ticker, which serves as a label for stock codes). The specific reasons for selecting these six variables will be explained later.

Combining the analysis from Tables 3.1 and 3.2, based on the data, it was found that the time span of this dataset ranges from January 4, 2005, to May 11, 2022, and based on the ticker variable, it was discovered that the dataset contains information for multiple stocks. Exploring the data types further, it was observed that all variables in the dataset, except for ticker, are of type float64, indicating that ticker serves merely as a label for stock codes, consistent with the description provided earlier. Next, calculating the null values in the dataset yielded the following results:

Table 3.3: Null_value

	ticker-turnover	pe	pe_ttm	pb	ps	ps_ttm	dv_ratio	dv_ttm	total_mv	qfq_factor
null_value	0	933946	1336380	94050	5289	10726	1463360	3660488	26	0

From Table 3.3, it can be observed that the majority of missing values are concentrated in variables such as pe, pe_ttm, pb, ps, ps_ttm, dv_ratio, dv_ttm, and total_mv. These variables have limited significance in the stock prediction process, hence removing them is the optimal choice. Additionally, although qfq_factor has no missing values, it is not a primary variable for predicting stock trends and will also be removed. After processing these variables, a stock that has remained relatively stable throughout is selected for analysis—specifically, the stock of Shanghai Pudong Development Bank (stock code: sh600000). This results in a dataset containing only the stock sh600000 with variables open, high, low, close, volume, and turnover. This dataset comprises 4112 entries and spans from January 4, 2005, to May 11, 2022. Plotting the time series of volume and closing price for this stock yields the following results.

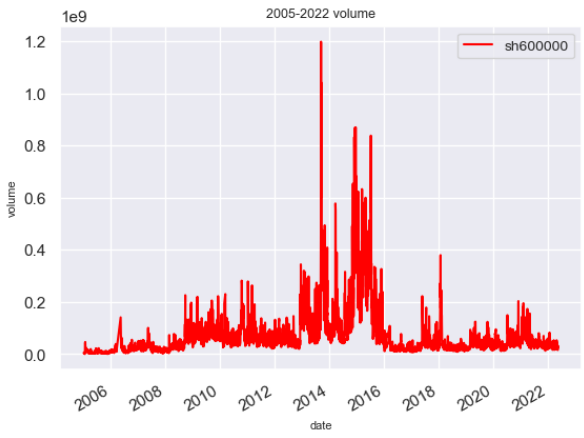

Figure 3.1: Time series of volume

Figure 3.2: Time series of close

From Figure 3.1 and 3.2, it is evident that there are instances of extremely high trading volume and significant declines in closing prices during certain periods. To mitigate the impact

of these extreme data points on subsequent predictions, logarithmic returns can be employed to eliminate outliers from the dataset.

3.2 Eliminate Outliers

The formula for the logarithmic returns of stocks is as follows:

$$\text{The logarithmic returns of stocks} = \ln\left(\frac{R_t}{R_{t-1}}\right). \tag{3.1}$$

In Equation 3.1, R_t represents the return of an asset at time t, and R_{t-1} represents the return of the same asset at time $t-1$. According to this formula, calculate the daily returns and introduce a new variable in the dataset named "earn_rate". Remove any empty values from "earn_rate" and visualize the logarithmic returns to obtain the following graphs:

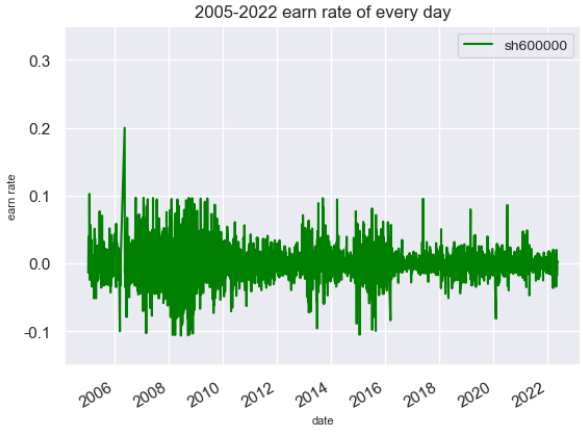

Figure 3.3: Earn rate of sh600000

Figure 3.4: Outliers

From Figure 3.3, it can be observed that there are outliers at certain points, such as those highlighted by red circles in Figure 3.4. Calculating the mean of the daily logarithmic returns reveals that the mean daily logarithmic return for sh600000 is 0.000568. Next, a boxplot is generated, depicted in Figure 3.5.

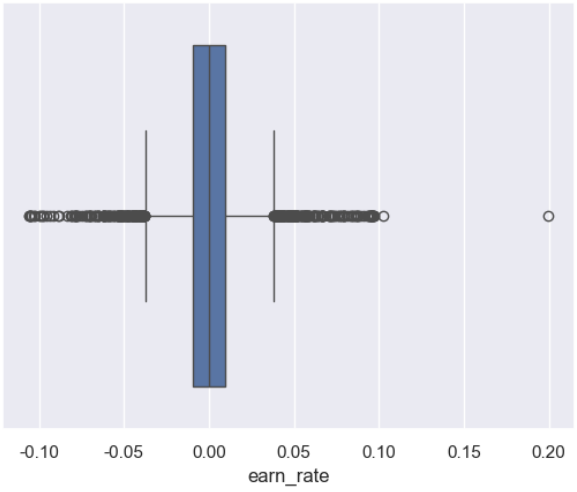

Figure 3.5: Using boxplot to show outliers

Based on the boxplot results, a significant number of outliers are evident, necessitating the use of quartiles to eliminate outliers. The formula for calculating quartiles is:

$$\begin{aligned} Q_1 &= \frac{(N+1)}{4}, \\ Q_3 &= \frac{3\,(N+1)}{4}, \\ \text{IQR} &= Q_3 - Q_1. \end{aligned} \tag{3.2}$$

Utilizing formula 3.2, calculate the quartiles, where IQR represents the interquartile range, Q_1 denotes the lower quartile, and Q_3 denotes the upper quartile. Employing quartiles to remove outliers, redraw the box plot, and observe that the majority of outliers have been eliminated, as depicted in Figure 3.6.

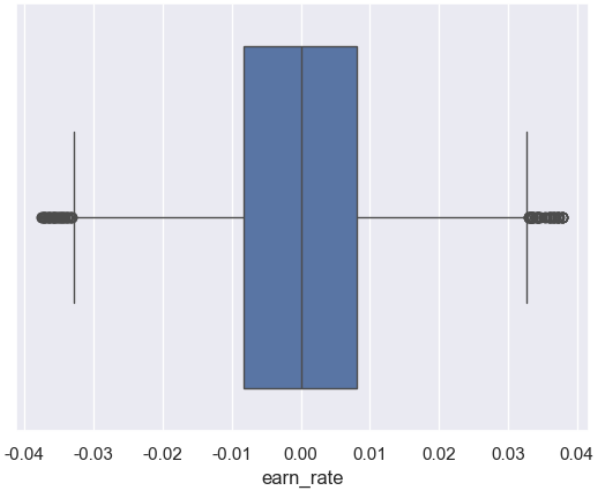

Figure 3.6: Using quartilers to remove outliers

Upon reviewing the data again for missing values, it was found that the 'earn_rate' variable now contains only 3765 non-null values, indicating a total of 347 missing values. Standardizing the dimensions of each variable and eliminating these missing values, the final dataset consists of 3765 entries. All variables, except for 'ticker' which remains an object data type, are represented in float64 format. This concludes the EDA.

Chapter 4

The Implementation of Random Forest

In this chapter, the main focus will be on presenting the results of stock closing price prediction using the random forest algorithm for modeling. In this experiment, three modeling methods were employed, resulting in three different outcomes.

4.1 Modeling and Prediction Using the First Approach

The first modeling approach entails training the model using the testing dataset to predict the stock closing price after thirty trading days. Initially, the dataset needs to be partitioned into training, testing, and validation sets. Since the dataset is a time series, we cannot use the 'train_test_split' function to split the dataset. Therefore, for this experiment, we selected all trading days up to July 12, 2021, as the training set, 99 trading days from July 13, 2021, to December 7, 2022, as the validation set, and 100 trading days from December 8, 2022, to May 11, 2023, as the test set. Additionally, during the dataset preprocessing stage, it is necessary to shift the 'close' column upward by thirty rows and place the shifted data into the 'target' variable, representing the stock closing price thirty trading days later. Using this method, the model is established and predictions are made. Fitting graphs for both the training and testing sets are plotted, resulting in the following outcomes.

Figure 4.1: The train result for RF_1

Figure 4.2: The test result for RF_1

From Figure 4.1, it can be observed that the model fitting is effective; however, as seen from the prediction results in Figure 4.2, there remains a considerable deviation between the actual values and the predicted values. To assess the accuracy of the model, the final step of the experimental process involved calculating the MSE and MAE. The formulas for MSE and MAE are as follows:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2, \tag{4.1}$$

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|. \tag{4.2}$$

In the above formula, $\hat{y}_i$ represents the predicted value. Utilizing equations 4.1 and 4.2 for computation, the results indicate that the MSE for the training set is 0.0481 and the MAE is 0.1490. Similarly, for the testing set, the MSE result is 0.7150 and the MAE result is 0.7880. Thus concludes the modeling and prediction process using the first approach.

4.2 Modeling and Prediction Using the Second Approach

The second modeling approach involves using the trained model to predict the stock closing price one day ahead. Regarding dataset partitioning, the entire process of dividing the dataset is identical to that described in section 4.1, and the dimensions of the test set, training set, and validation set are also the same. However, due to predicting the stock closing price one day ahead, the process of creating the target variable differs from that in section 4.1. It involves shifting the 'close' column upward by one row and placing the shifted data into the target variable, representing the stock closing price one day later. Using this method, the model is established and predictions are made. Fitting graphs are then plotted separately for the testing set and training set, yielding the following results.

Figure 4.3: The train result for RF_2

Figure 4.4: The test result for RF_2

From Figure 4.3, it can be observed that the model's fitting performance is improved compared to that in 4.1. Upon examining Figure 4.4, it is noted that the deviation between actual and predicted values in the testing set has significantly decreased compared to the model in 4.1. Using equations 4.1 and 4.2, the MSE and MAE were calculated. The results for the training set

were MSE: 0.0034 and MAE: 0.0364, while for the test set, MSE: 0.0070 and MAE: 0.0690. Thus, the modeling and prediction process using the second method concludes here.

4.3 Modeling and Prediction Using the Third Approach

This method shares the same objective as described in 4.2, which is to predict the stock closing price one day later. However, the modeling approach is more complex and differs significantly from the conventional modeling methods in 4.1 and 4.2. In the previous two methods, the dataset contained only six features: 'open', 'high', 'low', 'close', 'volume', and 'turnover'. The entire random forest process utilized these six features for modeling and predicting the data, which may not yield optimal model performance. The third modeling method involves adding the past 29 days' closing prices as additional features to the dataset, thereby increasing the total number of features in the dataset from six to 35.

This modeling approach offers several advantages: Firstly, incorporating the closing prices of the past 29 days as features provides richer historical data to the model. This allows the model to consider more time series information and trends during prediction, aiding in capturing both long-term and short-term patterns of stock price fluctuations. Secondly, introducing more features can make the model more complex, potentially improving its prediction accuracy. During training, the model can leverage additional relevant features to learn patterns of stock price changes, thereby enhancing prediction accuracy and generalization capability. Lastly, this approach can reduce data loss. By adding the closing prices of the past 29 days as features and then removing rows containing missing values, we can retain the maximum amount of valid data. This method avoids directly discarding rows with missing values while providing more input features to the model, thereby improving data utilization. Using these 35 features to build the model and predict the stock closing price one day later, we obtained the following results:

Figure 4.5: The train result for RF_3

Figure 4.6: The test result for RF_3

From Figure 4.5, the fitting performance on the training set resembles that of Figure 4.3 in Experiment 4.2. Furthermore, upon comparing Figure 4.6 with Figure 4.4, it is observed that in the new method, the gap between the actual values and the predicted results in the testing set is generally similar to the results obtained in Experiment 4.2. Using equations 4.1 and 4.2,

calculating the MSE and MAE as follows: The training set MSE result is 0.0033 with an MAE value of 0.0356, while the test set MSE result is 0.0065 with an MAE result of 0.0663. Thus concludes the modeling and prediction process using the third approach.

4.4 Analysis of Features Importance

Now, let's analyze the feature importance of sections 4.2 and 4.3. Typically, the method used to calculate feature importance is based on the reduction of impurity within the random forest model, also known as impurity-based feature importance. This method evaluates feature importance by measuring the extent to which each feature reduces impurity (such as Gini impurity for classification trees or MSE for regression trees) at decision tree nodes. The final feature importance values are calculated based on the average reduction in impurity of each feature across all decision trees. Since this experiment involves regression trees, the feature importance is computed based on the reduction in MSE.

The specific process for computing feature importance involves three steps. The first step is to build a regression tree model, which has been completed in sections 4.2 and 4.3.

The second step is to compute the node impurity. For each node t, calculate the MSE of all samples at that node as the node impurity. Let n_t be the number of samples at node t, and $\{y_i\}_{i=1}^{n_t}$ be the true target values of the samples at node t. The formula to compute the node impurity MSE_t for node t is as follows:

$$\text{MSE}_t = \frac{1}{n_t}\sum_{i=1}^{n_t}(y_i - \text{mean}(\{y_i\}_{i=1}^{n_t}))^2. \tag{4.3}$$

In Equation 4.3, $\text{mean}(\{y_i\}_{i=1}^{n_t})$ represents the mean target value of the samples in node t.

The third step involves calculating the reduction in error after node splitting. For each feature X_j, at each non-leaf node t, we consider splitting the node t into left sub-node t_L and right sub-node t_R based on feature X_j, and calculate the total mean squared error after the split. Suppose after splitting node t into t_L and t_R, there are n_{t_L} samples in the left sub-node t_L and n_{t_R} samples in the right sub-node t_R. The formula to calculate the total mean squared error after the split is as follows:

$$\text{Total MSE} = \frac{n_{t_L}}{n_t}\text{MSE}_{t_L} + \frac{n_{t_R}}{n_t}\text{MSE}_{t_R}. \tag{4.4}$$

In Equation 4.4, n_t equal to $n_{t_L} + n_{t_R}$, and MSE_{t_L} and MSE_{t_R} are the mean squared errors of the left and right sub-nodes, respectively. After that, The MSE reduction after node splitting can be calculated as:

$$\text{MSE\_reduction}(t, X_j) = \text{MSE}_t - \text{Total MSE}. \tag{4.5}$$

In the final step, for each feature X_j, the feature importance score is calculated as the weighted average of $\text{MSE\_decrease}(t, X_j)$ across all non-leaf nodes t where the feature is utilized, serving as a metric to gauge the contribution of features to model predictions.

Using the aforementioned method to calculate the importance of each feature as described in Sections 4.2 and 4.3, and visualizing the results, the following outcomes were obtained:

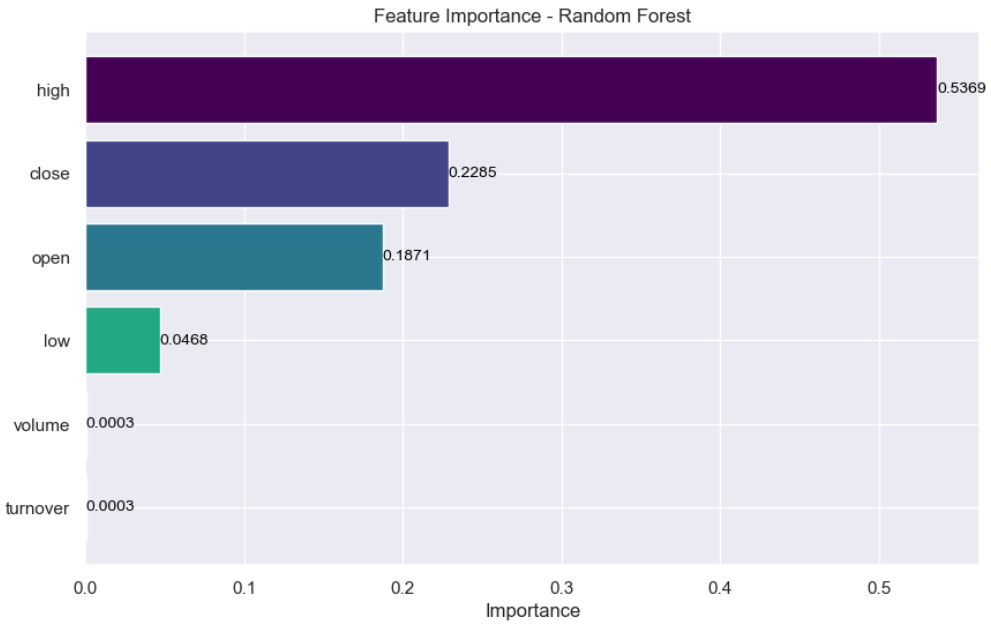

Figure 4.7: The result in Section 4.2

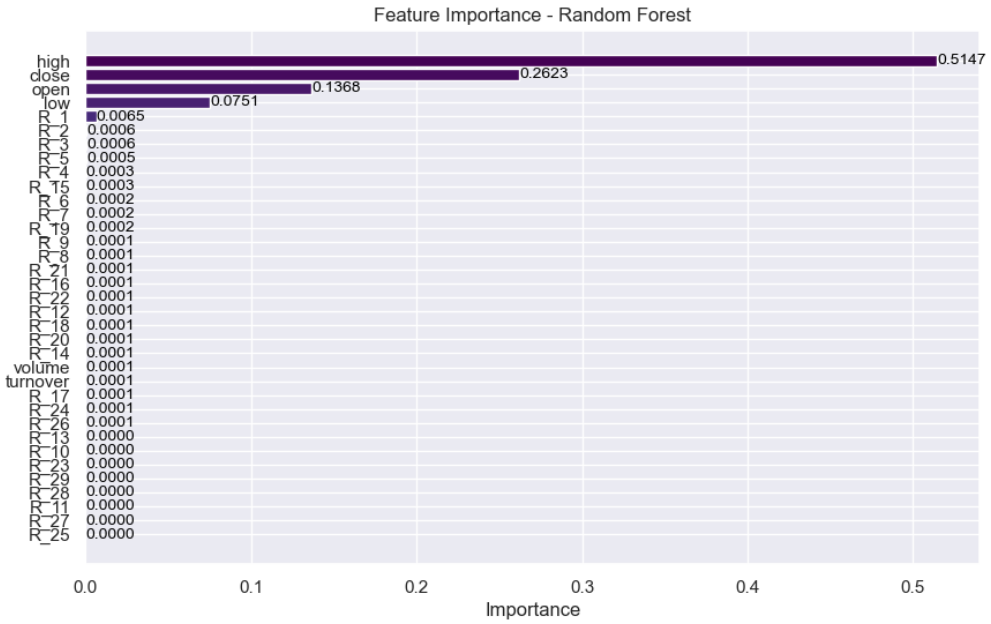

Figure 4.8: The result in Section 4.3

Comparing Figure 4.7 with Figure 4.8, it is observed that among the additional features introduced in the enhanced approach described in Section 4.3, only the feature representing the previous trading day exhibits relatively higher importance, while the information from the remaining trading days' features is not significant. This can explain why the predictive performance of the enhanced approach in Section 4.3 did not exhibit substantial improvement compared to the method utilized in Section 4.2.

Chapter 5

The Implementation of LSTM

This chapter will primarily showcase the results of predicting stock closing prices using the LSTM algorithm.

5.1 Modeling and Prediction in LSTM

Initially, the dataset requires preprocessing due to the wide range of volume compared to the relatively smaller scale of stock prices and turnover rates. Therefore, normalization of the data is necessary to scale everything to a consistent magnitude. In addition, similar to sections 4.2 and 4.3 in Chapter 4, this experiment will use data from the previous thirty trading days to predict the closing price of the following day. Subsequently, a neural network model will be constructed. Specifically, this experiment involves constructing an LSTM neural network comprising two LSTM layers with fifty neurons each. Furthermore, dropout and dense layers were added at the end to prevent overfitting. Explanations for dropout and dense layers will be provided in Section 5.2. Finally, during the model training process, it is necessary to configure the batch size and number of iterations.

In terms of batch size, smaller batches typically provide more stable gradient estimates because each batch's samples may be more representative. Concurrently, smaller batches contribute to improving the model's generalization ability as the model more frequently "observes" different sample data. Additionally, smaller batches can also save memory since fewer samples are loaded at once, thereby occupying less memory space. Conversely, larger batch sizes accelerate the training process because handling more samples per iteration increases the frequency of model parameter updates, thereby speeding up the training process. In this experiment, the batch size was set to 16, which is considered an optimal choice under typical circumstances.

In the context of iteration count, an epoch refers to a complete cycle where the model performs forward propagation (calculating predictions) and backward propagation (computing gradients and updating parameters) using the entire training dataset. During one epoch, the model iterates through all samples in the training dataset. In practical training, multiple epochs are typically conducted to optimize model parameters. The selection of an appropriate number of iterations is a critical aspect of the model training process. Generally, the number of iterations should not be too low, as the model may fail to adequately learn patterns and features within the data. Conversely, excessive iterations should also be avoided to prevent overfitting the model to

the training data, where the model memorizes noise from the training data and fails to generalize well to new data. In the specific experimental process, the appropriate number of iterations can be determined by monitoring the loss function during training or by evaluating performance on a validation set. This approach helps in identifying the optimal number of iterations for model training. In this experiment, the number of iterations is 40, which is determined as the optimal number after multiple selections. Finally, the model was established using the aforementioned parameters, and a summary of the model structure is presented below:

Table 5.1: LSTM Structure

Layer(type)	**Output Shape**	**Param #**
lstm(LSTM)	(None,30,50)	11400
lstm_1(LSTM)	(None,50)	20200
dropout(Dropout)	(None,50)	0
dense(Dense)	(None,1)	51
Total params: 31,651 Trainable params:31,651 Non-trainable params:0		

Subsequently, the training set and testing set were partitioned. The testing set used in this experiment is identical to the training set used in Chapter 4, comprising data from 100 trading days spanning from December 8, 2022, to May 11, 2023. The training set consists of all working days prior to May 11, 2022. Using MSE as the loss function, and after completing all training and prediction processes, the data was inverse-normalized (reverse-scaled), and fitting graphs were plotted for both the training set and testing set, yielding the following results.

Figure 5.1: The train result for LSTM

Figure 5.2: The test result for LSTM

From Figure 5.1, it is evident that the model's fitting performance is not ideal. Similarly, upon examining the prediction results shown in Figure 5.2, significant deviations between actual values and predicted values are observed. Subsequently, by applying formulas 4.1 and 4.2, the MSE and MAE were calculated. The MSE for the training set is 0.3669 and the MAE is 0.4602, while the MSE for the testing set is 0.1706 and the MAE is 0.3073. Thus concludes the prediction process based on LSTM.

5.2 Trying to Improve Model Predictive Performance

Due to the suboptimal predictive performance observed in 5.1, this section will focus on refining the neural network model. The specific improvements include increasing the number of LSTM neurons in the first layer to 128, and adding L2 regularization and Dropout layers. In the second LSTM layer, the number of neurons is reduced to 64, and again, L2 regularization and Dropout layers are added. Finally, a fully connected layer (Dense) with ReLU activation function and 32 neurons is added, along with an increase in the L2 regularization term. The following content will explain all of the aforementioned modifications.

L2 regularization, also known as weight decay, is a commonly used regularization method. It involves adding a penalty term to the model's loss function that is proportional to the sum of the squares of the model weights. This technique helps constrain the size of the model parameters, thereby preventing overfitting to the training data. Let's assume the loss function is represented by $J(w)$, where w denotes the weight parameters of the model. L2 regularization adds a term to the loss function to penalize the sum of squares of the model's weight parameters. Typically, the regularized loss function can be expressed as:

$$J_{\text{reg}}(w) = J(w) + \frac{\lambda}{2}\sum_{i=1}^{n} w_i^2. \tag{5.1}$$

In Equation 5.1, λ is the regularization parameter used to control the strength of the regularization term. A larger λ leads to a more pronounced regularization effect, causing the weight parameters w to tend towards smaller values. Here, $\sum_{i=1}^{n} w_i^2$ represents the sum of squares of all weight parameters, where n is the total number of weight parameters. Therefore, the effect of L2 regularization is to reduce model complexity during optimization by minimizing an additional penalty term that includes the sum of weight squares. This encourages the model's weight parameters to remain relatively small, thereby enhancing the model's generalization and overfitting resistance. In this experiment, λ was set to 0.01.

Dropout refers to the technique of randomly dropping out a portion of neurons in a neural network during the training process, including both input and hidden layer neurons, by setting their outputs to zero. This approach is akin to removing a portion of neurons from the neural network, compelling the network to learn more robust and generalizable features. Specifically, during training, Dropout randomly sets some neurons to zero with a certain probability (typically specified as a dropout rate), for each training sample. This results in a different, partially "pruned" network structure with each training iteration, reducing interdependencies among neurons and enhancing the model's generalization ability. In 5.1, dropout was configured to randomly drop 20% of neurons, whereas in this experiment, dropout was increased to randomly drop 30% of neurons to prevent the model from overfitting to the training data.

Finally, let's explain the fully connected layer. In a neural network, a fully connected layer (Dense layer) refers to a layer structure where each neuron in the layer is connected to every neuron in the adjacent layer. Each connection between neurons has an associated weight, and these weights are learned and adjusted during the training process to transform and represent features effectively. Specifically, when adding a fully connected layer (Dense layer) with a

ReLU activation function, it means introducing a new layer in the neural network where each neuron is connected to all neurons in the preceding layer. Each neuron in this layer applies the Rectified Linear Unit (ReLU) as its activation function. The definition of the ReLU activation function is as follows:

$$f(x) = \max(0, x). \tag{5.2}$$

In Equation 5.2, x represents the input value, and $f(x)$ is the output value after being processed by the ReLU activation function. The ReLU function is characterized by outputting x for $x > 0$ and 0 for $x \leq 0$. Mathematically, the ReLU function can be represented as a piecewise linear function, defined as follows:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}. \tag{5.3}$$

In addition to the aforementioned improvements, this experiment increased the number of iterations to 50 and raised the batch size to 32. Summarizing the model, the following structure was obtained:

Table 5.2: LSTM_new Structure

Layer(type)	**Output Shape**	**Param #**
lstm (LSTM)	(None,30,128)	69120
dropout (Dropout)	(None,30,128)	0
lstm_1 (LSTM)	(None,64)	49408
dropout_1 (Dropout)	(None,64)	0
dense (Dense)	(None,32)	2080
dense_1 (Dense)	(None,1)	33
Total params: 120,641 Trainable params:120,641 Non-trainable params:0		

Utilizing this modified LSTM neural network for prediction yielded the following outcomes:

Figure 5.3: The train result for LSTM_new

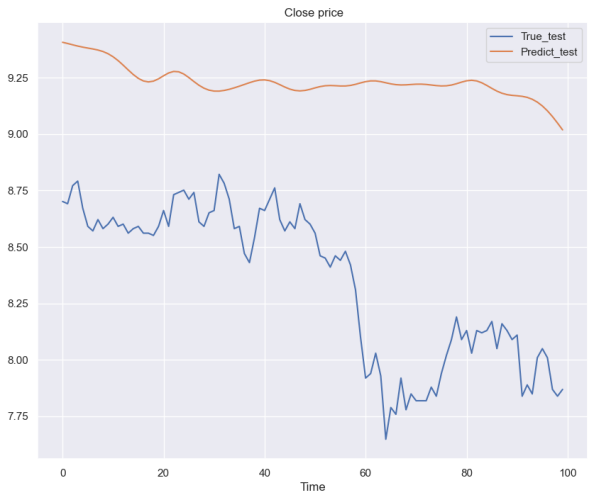

Figure 5.4: The test result for LSTM_new

Interestingly, upon comparing the two figures with Figures 5.1 and 5.2, it was observed

that the prediction performance of the reconstructed neural network did not improve; instead, it declined. Calculating the MSE and MAE for the test set yielded an MSE result of 0.5764 and an MAE result of 0.5523 for the training set, and an MSE result of 0.8792 and MAE result of 0.8850 for the test set, which were far worse than those obtained in 5.1.

Chapter 6

Conclusion and Reflection

In this chapter, we will summarize and compare the results obtained in Chapter 4 and Chapter 5. Following the experiments in Chapter 4 and Chapter 5, five groups MSE and MAE values were obtained. These values have been compiled and organized into the following table of results.

Table 6.1: Results in LSTM and RF

	RF_1		**RF_2**		**RF_3**		**LSTM**		**LSTM_new**	
	train	**test**	**train**	**test**	**train**	**test**	**train**	**test**	**train**	**test**
MSE	0.0481	0.7150	0.0034	0.0070	0.0033	0.0065	0.3669	0.1706	0.5764	0.8792
MAE	0.1490	0.7880	0.0364	0.0690	0.0663	0.0663	0.4602	0.3037	0.5523	0.8850

Before analyzing the results, it is important to review the experimental objectives: to predict stock closing prices using LSTM and random forest algorithms, and to select the optimal predictive model.First, comparing the outcomes of the same method, for the Random Forest algorithm, it was found that the third modeling approach was the most effective. This approach involved augmenting the dataset with data from the preceding thirty trading days as feature values for modeling, aiming to predict the closing price of the second trading day. For the LSTM, the refined neural network's predictive performance was significantly inferior to that of the original one. Next, we will compare the results between two different methods. Since the LSTM in this experiment predicts the closing price for the next trading day, the comparison between the two methods should involve selecting the optimal results from the second and third random forest modeling methods and comparing them with the results of the unrefined LSTM.

Comparing the MSE results reveals that the random forest algorithm exhibits superior model fitting compared to LSTM. Additionally, in terms of predictive performance, the random forest algorithm also outperforms the LSTM. This conclusion similarly applies to the results shown by MAE. However, since outliers were removed during the EDA phase and had minimal impact on the experimental process, the conclusions primarily rely on the results indicated by MSE. Therefore, whether considering the numerical comparison results of MSE or the visual comparison results depicted in Figures 4.5, 4.6, 5.3, and 5.4, it is evident that the predictive performance and model fitting of the random forest algorithm outperform those of the LSTM. This is a rather intriguing conclusion, as LSTM has traditionally been considered an effective method for stock price prediction in mainstream research. However, considering an alternative perspective, previous studies have primarily focused on accurate price forecasting. If we shift

this perspective to predicting trends, LSTM's performance can also be relatively strong. According to Figure 5.2, LSTM accurately predicted a downward trend in stock closing prices. Therefore, using LSTM to forecast stock price trends is justified.

Based on the comprehensive analysis, the following conclusions can be drawn: incorporating features from the previous thirty trading days, the random forest algorithm is the most accurate for predicting the stock closing price on the second day. However, while LSTM may not precisely predict the closing price for the second trading day, it is effective in forecasting stock trends.

6.1 Future Work

During the specific implementation of this project, I found that while there is a risk of vanishing gradients when using RNN to handle longer time series data, this issue may not have affected the experiment in question. Although the original dataset was extensive, this experiment only involved predicting the closing price of a single stock. After handling missing values, the dataset was reduced to approximately three thousand data points. Hence, in future experiments, it would be feasible to incorporate an RNN-based prediction method. Following predictions using RNN, a comparison with the two existing methods could be conducted to determine the optimal prediction approach.

Furthermore, due to hardware constraints, I may be limited in adding additional neurons and layers when constructing the LSTM neural network. Additionally, during subsequent neural network training processes, I may be unable to increase the number of iterations and batch size. Blindly altering these parameters without considering hardware limitations could potentially prolong neural network training time and may not necessarily enhance the network's performance. Therefore, in future experimental processes, it would be beneficial to modify the aforementioned parameters and compare different combinations to assess their impact on the LSTM. This exploration aims to identify an optimal parameter configuration.

Thirdly, in this experiment, LSTM and random forest were used separately for prediction. However, combining LSTM with random forest could potentially improve prediction performance [27]. This also leads to a new research direction where multiple methods can be combined to forecast stock prices. For example, in previous studies, combining LSTM with Convolutional Neural Networks (CNN) [28, 29] has been shown to effectively enhance stock price prediction [30]. This schematic diagram of the neural network can be found in Appendix Figure A.1 [31].

Lastly, as mentioned before, this experiment focused solely on predicting the stock price of a single stock. The stock market is vast, with numerous stocks worthy of study, as evidenced by the abundance of stock labels in the original dataset. At times, different stocks can influence each other. Therefore, in future experimental processes, it would be beneficial to consider the impact of multiple stocks when modeling. For example, incorporating combined information from multiple stocks into the model could potentially enhance the accuracy of predictions.

6.2 Reflection on Project

In the LSTM part, I attempted to enhance the predictive capability of the LSTM neural network by modifying its structure. Section 5.2 presented only one method of improvement, but in the actual experimental process, I tried several changes. For example, I increased the number of LSTM layers to four for prediction, and I increased the number of neurons in each layer to 128 before making predictions, among other modifications. In theory, these improvements could enhance prediction performance, but in practice, the results were contrary. Not only did the predictive capability not improve, but it actually decreased. Why theoretical methods that should improve prediction performance do not actually enhance it? This is a question that I could not resolve during my research process. Furthermore, according to the experimental results, the performance of the LSTM neural network is not superior to that of the random forest. I speculate that this discrepancy may be attributed to the inadequacy of the sample size. Through perusal of blogs and literature, I have become acquainted with a statistical rule of thumb: the minimum number of samples should be 30 times the number of model parameters [32]. For instance, in Section 5.1, where I have fifty neurons, each with fifty weights, the required number of samples would be 75,000. However, my dataset comprises only a little over three thousand time points, which is significantly insufficient.

6.3 Reflection on Personal

My FYP (Final Year Project) can be broadly divided into three phases. The first phase involved planning and preparation, the second phase focused on coding, and the third phase was dedicated to writing the paper. In the first phase, I felt that my planning was not comprehensive enough because the timeline for the entire FYP spanned over the New Year period. I did not consider my ability to execute the plan during the New Year, which resulted in no progress on my FYP during that time. This led to increased pressure to complete subsequent tasks. However, it is worth acknowledging that I adhered to the plan throughout the entire process, allowing me to complete all tasks on time in the end.

In the second module, I anticipated that writing the code would be relatively straightforward, so I did not give it much attention. However, the reality was that while the code for EDA and the random forest section was relatively simple, the LSTM portion involved complex operations. For instance, I needed to write a new function specifically to partition the required data. As a result, I underestimated the difficulty of writing the code, leading to excessive time spent in this area.

In the third phase, the process of writing the entire paper went smoothly. However, I spent considerable time searching for references because I did not initially filter the content of the references. This resulted in reading a large number of references, but in reality, only a few were actually useful. If I had overcome or addressed these issues earlier, I believe I could have been more efficient in completing my FYP.

Appendix A

Appendices

A.1 Supplementary Figures

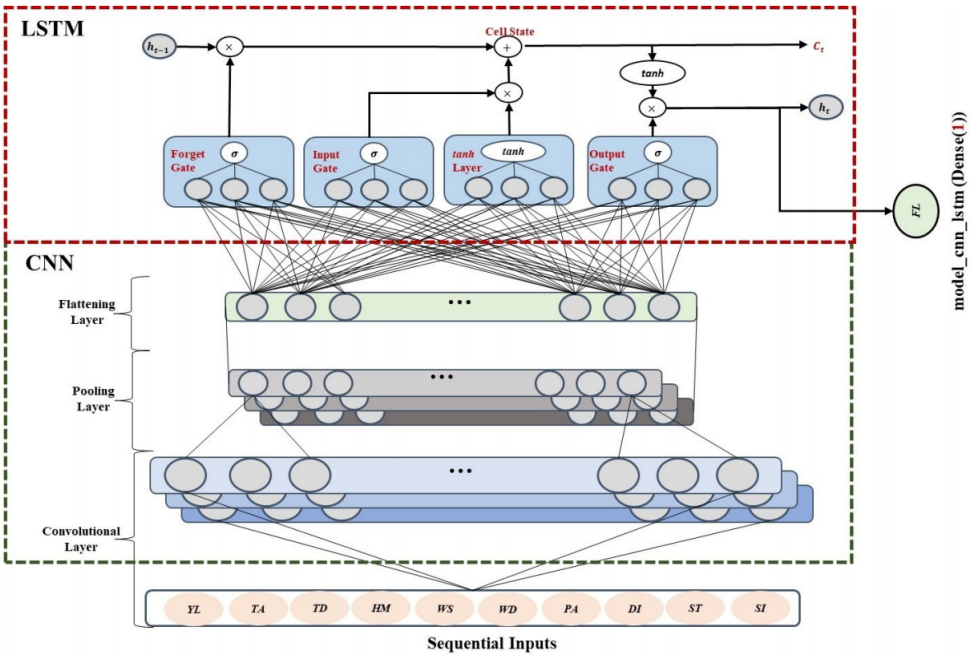

Figure A.1: CNN—LSTM

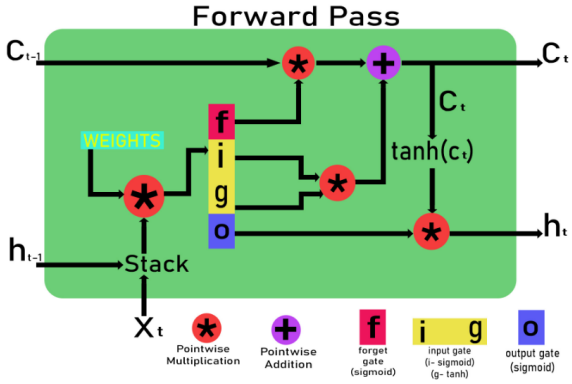

Figure A.2: Forward Pass

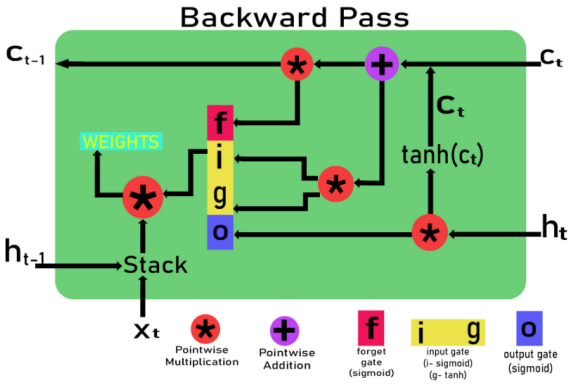

Figure A.3: Backword Pass

Bibliography

[1] Rui Li, DianZheng Fu, and Zeyu Zheng. An analysis of the correlation between internet public opinion and stock market. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 150–153, 2017.

[2] Yilin Ma, Ruizhu Han, and Xiaoling Fu. Stock prediction based on random forest and lstm neural network. In *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, pages 126–130. IEEE, 2019.

[3] Zhigang Jin, Yang Yang, and Yuhong Liu. Stock closing price prediction based on sentiment analysis and lstm. *Neural Computing and Applications*, 32:9713–9729, 2020.

[4] Vijay Singh Rathore, Marcel Worring, Durgesh Kumar Mishra, Amit Joshi, and Shikha Maheshwari. Emerging trends in expert applications and security proceedings of iceteas 2018. *Proceedings of ICETEAS*, page 1, 2018.

[5] Ritika Singh and Shashi Srivastava. Stock prediction using deep learning. *Multimedia Tools and Applications*, 76:18569–18584, 2017.

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[7] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[8] Mehdi Khashei and Mehdi Bijari. An artificial neural network (p, d, q) model for time-series forecasting. *Expert Systems with applications*, 37(1):479–489, 2010.

[9] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[10] AJ Smola and B Schölkopf. A tutorial on support vector regression: Neurocolt, technical report nc-tr-98-030. *Royal Holloway College, London*, 1998.

[11] K-R Müller, Alexander J Smola, Gunnar Rätsch, Bernhard Schölkopf, Jens Kohlmorgen, and Vladimir Vapnik. Predicting time series with support vector machines. In *International conference on artificial neural networks*, pages 999–1004. Springer, 1997.

[12] Michael JD Powell. Radial basis functions for multivariable interpolation: a review. *Algorithms for approximation*, pages 143–167, 1987.

[13] Shuo Han and Rung-Ching Chen. Using svm with financial statement analysis for prediction of stocks. *Communications of the IIMA*, 7(4):8, 2007.

[14] Yongqiong Zhu. Stock price prediction using the rnn model. In *Journal of Physics: Conference Series*, volume 1650, page 032103. IOP Publishing, 2020.

[15] David MQ Nelson, Adriano CM Pereira, and Renato A De Oliveira. Stock market's price movement prediction with lstm neural networks. In *2017 International joint conference on neural networks (IJCNN)*, pages 1419–1426. Ieee, 2017.

[16] FRANCISCO FENG. Augmented chinese stock data w/ frs and fundamentals, 2022. https://www.kaggle.com/datasets/franciscofeng/augmented-china-stock-data-with-fundamentals/data.

[17] John T Behrens. Principles and procedures of exploratory data analysis. *Psychological methods*, 2(2):131, 1997.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Henry W Lin and Max Tegmark. Criticality in formal languages and statistical physics. *arXiv preprint arXiv:1606.06737*, 2016.

[20] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International conference on big data (Big Data)*, pages 3285–3292. IEEE, 2019.

[21] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[22] Christian Bakke Vennerød, Adrian Kjærran, and Erling Stray Bugge. Long short-term memory rnn. *arXiv preprint arXiv:2105.06756*, 2021.

[23] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.

[24] Hui Liu, Xiwei Mi, and Yanfei Li. Smart multi-step deep learning model for wind speed forecasting based on variational mode decomposition, singular spectrum analysis, lstm network and elm. *Energy Conversion and Management*, 159:54–64, 2018.

[25] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

[26] Mango. Derivation of backpropagation through time for lstm, 2021. https://www.imangodoc.com/93676.html.

[27] Kofi O Nti, Adebayo Adekoya, and Benjamin Weyori. Random forest based feature selection of macroeconomic variables for stock market prediction. *American Journal of Applied Sciences*, 16(7):200–212, 2019.

[28] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[29] Jake Bouvrie. Notes on convolutional neural networks. 2006.

[30] Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, and Jingyang Wang. A cnn-lstm-based model to forecast stock prices. *Complexity*, 2020:1–10, 2020.

[31] Pyae Phyo and Yungcheol Byun. Hybrid ensemble deep learning-based approach for time series energy prediction. *Symmetry*, 13:1942, 10 2021.

[32] Ignatius Sapto Condro. What is the minimum data size for applying an lstm on a time series, 2022. https://www.quora.com/What-is-the-minimum-data-size-for-applying-an-LSTM-on-a-time-series.