RNNs for Stock Price Prediction

A. Introduction

In recent years, the application of deep learning techniques has gained significant attention in various domains, including financial markets. Among these techniques, Recurrent Neural Networks (RNNs) have proven to be highly effective in predicting sequential events, making them an ideal choice for stock price prediction. This project aims to design experiments that utilizes different RNN architectures to predict stock prices and subsequently compare their performance based on real historical data.

A.1. Problem Statement

Predicting stock prices is a challenging task due to the complex and dynamic nature of financial markets. Traditional time-series analysis approaches often fall short in capturing the non-linear patterns and dependencies present in historical stock price data. This limitation has spurred the exploration of machine learning models, particularly RNNs, as a promising solution for stock price prediction.

RNNs belong to a category of artificial neural networks specifically designed for handling sequential data. Distinguishing themselves from traditional feedforward neural networks, RNNs possess a hidden state that enables them to retain memory from previous time steps. This characteristic makes them well-suited for tasks involving sequential data, particularly in the realm of financial time series prediction.

In this research, we aim to construct various RNN architectures to forecast the stock price of AAL (American Airlines Group Inc.), a representative stock listed on NASDAQ. Additionally, we will extend evaluation to include different Airline Stocks operating in the same field as AAL, serving as a comprehensive testing dataset to assess the performance of the developed models.

A.2. Comparison with Leading Competitors

Several machine learning techniques have been employed in stock price prediction, including support vector machines, decision trees, and traditional time-series models like ARIMA (AutoRegressive Integrated Moving Average) [10]. However, RNNs have shown promise in capturing complex patterns that are often missed by these methods [1]. In this project, we aim to compare the performance of different RNN architectures to evaluate their effectiveness in stock price prediction.

To date, prominent competitors in the realm of stock price prediction have harnessed a diverse array of deep learning models, notably Long Short-Term Memory (LSTM) networks and Gated Recurrent Unit (GRU) networks. These models have exhibited considerable potential and have gained widespread adoption. Our experiment seeks to conduct a comprehensive comparative analysis of these well-established architectures while incorporating certain enhancements, juxtaposed with traditional RNNs.

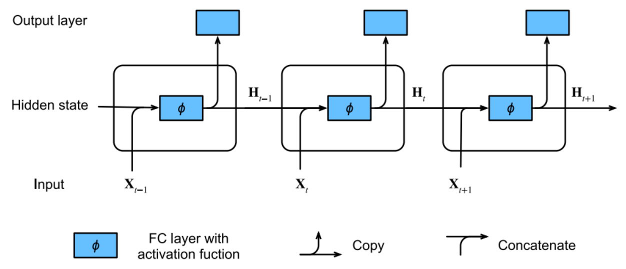

Figure 1. A RNN with a hidden state

B. Method Description

B.1. The Basic Architecture of RNNs

A RNN is a type of neural network architecture designed for processing sequential or time-series data. RNNs feature recurrent connections that allow information to flow from one time step to the next. This inherent cyclic structure makes RNNs particularly well-suited for handling variable-length sequences. RNNs maintain an internal state, known as the hidden state, which evolves at each time step based on the current input and the previous state, enabling the network to capture contextual information within the sequence, as showing Fig. 1

In Fig. 1, the computational process of an RNN is depicted across three consecutive time steps. At each time step t, the computation of the hidden state can be described as follows, firstly, it involves the concatenation of the current time step's input $\mathbf{X}_t$ with the hidden state $\mathbf{H}_{t-1}$ from the previous time step $t-1$. Subsequently, this concatenation result is passed through a fully-connected layer with an activation function φ. The outcome of this fully-connected layer becomes the hidden state $\mathbf{H}_t$ for the current time step t. In this context, the model parameters comprise the concatenation of $\mathbf{W}_{xh}$ and $\mathbf{W}_{hh}$, along with a bias term $\mathbf{b}_h$. Notably, the hidden state of the current time step t, denoted as $\mathbf{H}_t$, plays a pivotal role in the computation of the subsequent time step's hidden state $\mathbf{H}_{t+1}$ for time step $t+1$. Furthermore, $\mathbf{H}_t$ is also utilized in the fully-connected output layer to calculate the output $\mathbf{O}_t$ at the current time step t. The process can be described as (1).

$$\begin{aligned}\mathbf{H}_t &= \phi\left(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h\right)\\ \mathbf{O}_t &= \mathbf{H}_t\mathbf{W}_{hq} + \mathbf{b}_q\end{aligned} \tag{1}$$

However, standard RNNs have limitations, including issues like vanishing and exploding gradients, which can hamper their performance on longer sequences. Conse-

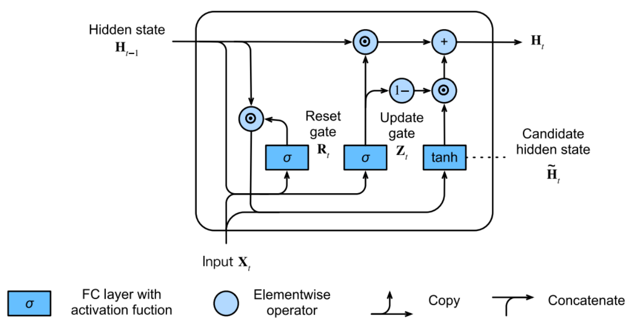

Figure 2. The Cell of GRU

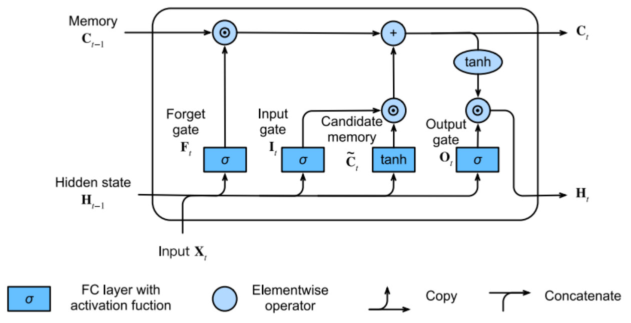

Figure 3. The Cell of LSTM

quently, more advanced RNN variants like LSTM and GRU have been developed to address these challenges and better capture long-term dependencies, often yielding improved results in various sequence modeling tasks.

B.2. GRU

A GRU [3] was developed as an enhancement over the standard RNNs, especially those related to capturing long-term dependencies in sequential data. The key innovation in a GRU is the introduction of gating mechanisms, which help control the flow of information through the network and mitigate the vanishing gradient problem.

A GRU has two main gates, the reset gate and the update gate. The reset gate decides which information from the previous time step to forget, allowing the model to adapt to changing patterns in the data. The update gate, on the other hand, determines how much of the previous hidden state should be retained and how much of the new candidate state should be considered, enabling the model to capture relevant information while discarding irrelevant information. The standard cell of GRU is shown as Fig. 2

Mathematically, for a given time step t the reset gate $\mathbf{R}_t \in \mathbb{R}^{n\times h}$ and $\mathbf{Z}_t \in \mathbb{R}^{n\times h}$, as (2).

$$\begin{aligned}\mathbf{R}_t &= \sigma\left(\mathbf{X}_t\mathbf{W}_{xr} + \mathbf{H}_{t-1}\mathbf{W}_{hr} + \mathbf{b}_r\right)\\ \mathbf{Z}_t &= \sigma\left(\mathbf{X}_t\mathbf{W}_{xz} + \mathbf{H}_{t-1}\mathbf{W}_{hz} + \mathbf{b}_z\right)\end{aligned} \tag{2}$$

The following candidate hidden state $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n\times h}$ at time step t can be described as (3)

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t\mathbf{W}_{xh} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right)\mathbf{W}_{hh} + \mathbf{b}_h\right) \tag{3}$$

The update gate $\mathbf{Z}_t$ determines the extent to which the new hidden state $\mathbf{H}_t \in \mathbb{R}^{n\times h}$ is just the old state $\mathbf{H}_{t-1}$ and by how much the new candidate state $\tilde{\mathbf{H}}_t$ is used. This leads to the final update equation for the GRU, as following (4)

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t \tag{4}$$

In short, GRUs have two distinguishing features. Reset gates capture short-term dependencies in sequences and update gates capture long-term dependencies in sequences.

B.3. LSTM

A LSTM [6] is an evolution of the standard RNNs, particularly designed to address issues associated with capturing long-term dependencies in sequential data. The LSTM architecture introduces a higher level of complexity, featuring memory cells and a suite of gates, which collectively make it a potent tool for modeling sequential data.

At the heart of the LSTM's innovation lies its intricate gating mechanisms, encompassing three pivotal gates: the input gate, the forget gate, and the output gate. Each gate is purpose-built to govern the information flow within the network.

The input gate orchestrates the incorporation of new information into the memory cell, enabling the model to update the cell state with pertinent data. On the other hand, the forget gate plays a crucial role in deciding which information from the previous time step should be discarded. This adaptive mechanism allows the model to accommodate changing patterns in the data, simultaneously addressing the issue of vanishing gradients.

The output gate serves to determine the information to be propagated as the output of the current time step, affording the model the ability to capture the most relevant features of the data.

LSTM's prowess stems from its capacity to maintain and update an internal memory state, bolstered by the finesse of its gating mechanisms. This makes it exceptionally adept at modeling the intricate and long-term dependencies inherent in sequential data. The standard LSTM cell is visually represented in Fig. 3.

Just like in GRUs, the data feeding into the LSTM gates are the input at the current time step and the hidden state of the previous time step. They are processed by three fully connected layers with a sigmoid activation function to compute the values of the input, forget, and output gates.

Mathematically, the gates at time step t, the input gate is $\mathbf{I}_t \in \mathbb{R}^{n\times h}$, the forget gate is $\mathbf{F}_t \in \mathbb{R}^{n\times h}$ and the output gate is $\mathbf{O}_t \in \mathbb{R}^{n\times h}$. They are calculated as following (5).

$$\begin{aligned} \mathbf{I}_t &= \sigma\left(\mathbf{X}_t\mathbf{W}_{xi} + \mathbf{H}_{t-1}\mathbf{W}_{hi} + \mathbf{b}_i\right) \\ \mathbf{F}_t &= \sigma\left(\mathbf{X}_t\mathbf{W}_{xf} + \mathbf{H}_{t-1}\mathbf{W}_{hf} + \mathbf{b}_f\right) \\ \mathbf{O}_t &= \sigma\left(\mathbf{X}_t\mathbf{W}_{xo} + \mathbf{H}_{t-1}\mathbf{W}_{ho} + \mathbf{b}_o\right) \end{aligned} \tag{5}$$

The input gate, denoted as $\mathbf{I}_t$, plays a crucial role in determining the extent to which new data is incorporated into the system through the intermediary $\tilde{\mathbf{C}}_t$. Conversely, the forget gate, represented by $\mathbf{F}_t$, is responsible for regulating the retention of information from the previous memory cell content, denoted as $\mathbf{C}_{t-1} \in \mathbb{R}^{n\times h}$. This intricate balance between integrating fresh data and preserving historical context is a fundamental aspect of the LSTM's ability to manage and model sequential data effectively. The update equations as following (6).

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t \tag{6}$$

When the forget gate consistently approximates 1 and the input gate remains consistently close to 0, the past memory cells denoted as $\mathbf{C}_{t-1}$ persistently endure, safeguarded and transmitted to the current time step. This architectural choice serves the dual purpose of mitigating the vanishing gradient problem and improving the LSTM's capacity to capture extensive dependencies within sequential data.

In the LSTM framework, this mechanism essentially manifests as a controlled transformation of the hyperbolic tangent (tanh) of the memory cell content. This approach ensures that the values of $\mathbf{H}_t \in \mathbb{R}^{n\times h}$ consistently reside within the interval (-1, 1), as indicated by equation (7). When the output gate approximates 1, it effectively permits the unobstructed passage of all memory information to the predictor. Conversely, when the output gate approaches 0, it restricts the information exclusively within the memory cell, refraining from further processing. This adaptive behavior underpins the LSTM's effectiveness in managing information flow and enables it to learn and model intricate patterns within sequences.

$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh\left(\mathbf{C}_t\right) \tag{7}$$

B.4. Deep RNNs

Deep RNNs incorporate multiple layers of recurrent units, as Fig. 4, allowing for hierarchical feature extraction and representation learning.

In a Deep RNN, multiple recurrent layers are stacked on top of each other. Each layer in the stack processes the input sequence sequentially, and the output of one layer becomes the input for the next layer. This stacking of layers enables the network to capture features at different levels of abstraction.

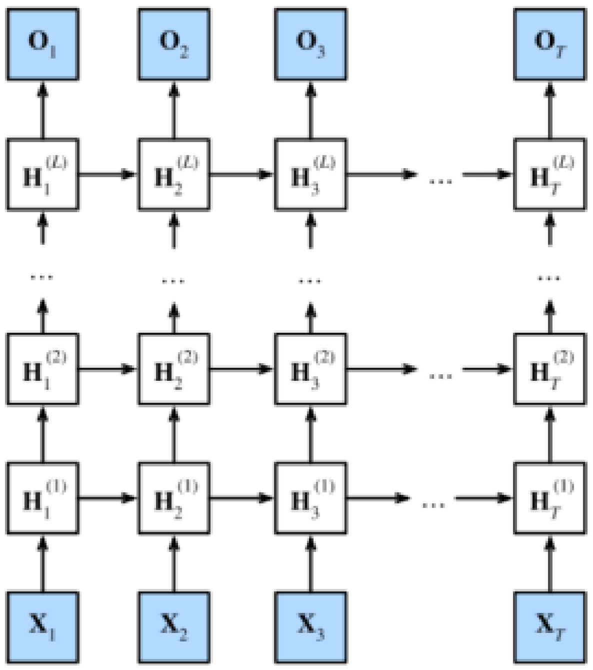

Figure 4. The Architecture of Deep RNNs

Deep RNNs create a feature hierarchy where lower layers capture local, short-term dependencies in the data, and higher layers learn more global and long-term dependencies. This hierarchical representation allows the model to extract relevant information from different time scales. By introducing multiple layers with nonlinear activation functions, Deep RNNs make it easier to train networks on sequences with long-term dependencies.

Mathematically, the hidden state of the l^{th} hidden layer that uses the activation function ϕ_l is expressed as (8)

$$\mathbf{H}_t^{(l)} = \phi_l\left(\mathbf{H}_t^{(l-1)}\mathbf{W}_{xh}^{(l)} + \mathbf{H}_{t-1}^{(l)}\mathbf{W}_{hh}^{(l)} + \mathbf{b}_h^{(l)}\right) \tag{8}$$

The calculation of the output layer is only based on the hidden state of the final L^{th} hidden layer, as (9).

$$\mathbf{O}_t = \mathbf{H}_t^{(L)}\mathbf{W}_{hq} + \mathbf{b}_q \tag{9}$$

The number of hidden layers L and the number of hidden units h are hyperparameters that this research will tune them in the later chapter.

B.5. Bi-RNNs

Bidirectional Recurrent Neural Networks (Bi-RNNs) processes sequential data in both forward and backward directions, as Fig. 5. They are designed to capture information from past and future time steps simultaneously. This bidirectional processing allows the network to capture both

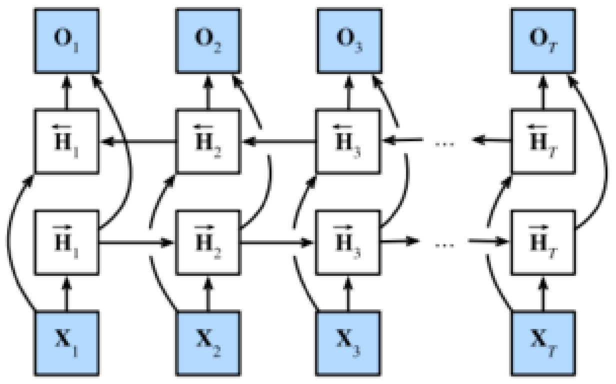

Figure 5. The Architecture of Bi-RNNs

past and future context for each time step, which means a Bi-RNN maintains two sets of hidden states for each time step, one for the forward pass and another for the backward pass.

Mathematically, the forward and backward hidden state updates as (10).

$$\begin{aligned}\overrightarrow{\mathbf{H}}_t &= \phi\left(\mathbf{X}_t\mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1}\mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}\right)\\ \overleftarrow{\mathbf{H}}_t &= \phi\left(\mathbf{X}_t\mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1}\mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}\right)\end{aligned} \tag{10}$$

The final output of a Bi-RNN is typically obtained by concatenating the forward and backward hidden states, $\overrightarrow{\mathbf{H}}_t$ and $\overleftarrow{\mathbf{H}}_t$,to obtain the hidden state $\mathbf{H}_t \in \mathbb{R}^{n\times 2h}$ to be fed into the output layer, as (11).

$$\mathbf{O}_t = \mathbf{H}_t\mathbf{W}_{hq} + \mathbf{b}_q \tag{11}$$

The merged representation encodes information from both directions and serves as the foundation for making predictions or facilitating subsequent processing.

C. Methods Implementation

https://github.com/hahawang1986/deep-Learning.git

assignment3 Ning Ni a1869549.ipynb was the main program of this assignment, and data_load.py is the customized package for running assignment3 data_load.ipynb.

.pth files were the result of various models which can be reloaded by the main program to show figures.

D. Experiments and Analysis

D.1. The Design and Uesage of Dataset

The raw training dataset comprises information on the stock of AAL spanning from 2013 to 2018, as illustrated in 6. For this experiment, a specific segment of the dataset was utilized, focusing on the period from October 2013 to February 2016. This subset, which encompasses 608 trading days, was employed as the training dataset.

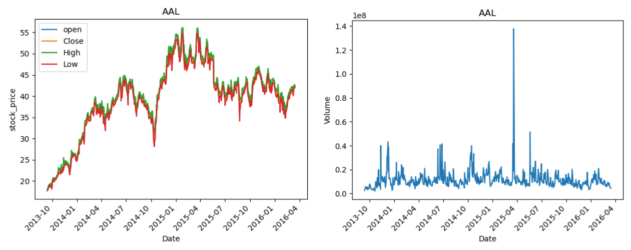

Figure 6. The Raw Information of AAL

The remaining subset from March 2016 to April 2016 was used as the validation dataset to fine-tune various hyperparameters and assess the performance of specific models.

The test datasets contain information on four different airline stocks, namely Delta Air Lines Inc. (DAL), United Airlines Holdings Inc. (UAL), Southwest Airlines Co. (LUV), and the aforementioned AAL. The trading days considered for the test datasets ranged from March 2016 to June 2016, constituting new and unseen data for the developed models. These datasets will be used to evaluate the general performance and robustness of the models.

D.2. The Design of Target Variable

Utilizing Returns (Percentage Change in Return) as the target variable in stock price prediction, expressed as (12), is for several reasons.

Firstly, it addresses the non-stationarity inherent in stock prices by measuring price changes over time rather than absolute levels. Additionally, it mitigates the impact of absolute values, ensuring more consistent predictions across stocks with varying market capitalizations. The focus on relative changes in returns enhances the model's generalization performance, enabling better adaptation to diverse time periods and different stocks. Moreover, employing returns helps avoid information leakage, as predicting closing prices on the same day using returns provides a more realistic simulation of market dynamics by forecasting relative changes from one day's close to the next.

$$\mathbf{Returns} = \frac{\mathbf{Close} - \mathbf{Open}}{\mathbf{Open}} \tag{12}$$

When models yield prediction results, the inverse process of recovering stock prices from the predicted returns can be accomplished using (12). It facilitates the translation of model-generated return predictions back into their corresponding stock price estimates.

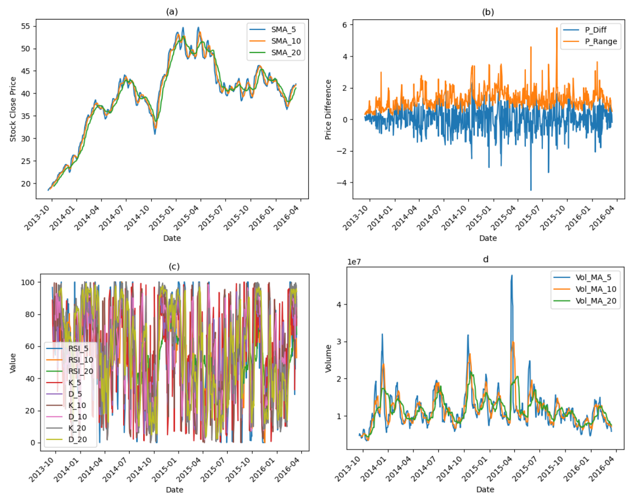

Figure 7. Feature Engineering. (a) Simple Moving Average (SMA) is a calculation that helps smooth out price data by creating a constantly updated average price over 5,10,20 trading day. (b) Price_diff refers to the absolute difference in price between Open and Close. Price_range refers to the absolute difference in price between High and Low. (c) Relative Strength Index (RSI) [9] is used in technical analysis to help identify overbought or oversold conditions in stock market. The "K" and "D" are called the Stochastic Oscillator. The "K" line is a measure of the momentum of a stock's price, while the "D" line is a smoothed version of the "K" line. These three indicators are created by constantly updated average value over 5,10,20 trading day. (d) The average volume of stock over 5,10,20 trading day.

D.3. Feature Engineering

The raw dataset comprises five fundamental features: Open, Close, High, Low, and Volume. The experimental phase extends these basic features to a set of twenty-four derived features, including SMA, RSI, K-D, price_diff, price_range, and volume_change, etc, as following Fig. 7. These additional features provide a multifaceted perspective, capturing nuances and changes from various angles in the stock data.

Subsequently, Principal Component Analysis (PCA) is employed to condense the dimensionality of the extended feature set from 24 to 10 features, effectively retaining 99% of the variance. The output of PCA is as the input of models. The specific procedures for Feature Engineering and PCA implementation can be referenced in the accompanying code file.

D.4. Model Design

This research introduces a custom model class named 'LSTM_GRU_RNN', as detailed in the provided code. This tailored model class streamlines the experimentation process, facilitating a straightforward comparison of various RNN architectures and hyperparameters. The inclusion of this custom class enhances the convenience and efficiency of the experimental setup, enabling a systematic exploration of different configurations to assess their impact on model performance.

D.5. Performance Metrics

This experiment is a regression task, so the performance metrics contains MSE, RMSE, rRMSE, R-square and Win-Loss Ratio.

Especially, rRMSE [7] is a normalized version of RMSE that represents the percentage error relative to the range of the target variable. In the stock market, where prices can vary significantly, rRMSE provides a more interpretable measure of prediction accuracy by scaling the error relative to the price range. It helps to assess the model's performance in a way that is proportional to the magnitude of stock price movements.

The Win-Loss Ratio [11] is a metric used to evaluate the profitability of a trading strategy based on the model's predictions. It is calculated as the ratio of the total number of winning trades to the total number of losing trades. In the context of stock prices, a higher Win-Loss Ratio indicates a more profitable trading strategy based on the model's predictions. It provides insights into the model's ability to generate accurate directional predictions, helping to assess its practical utility in making profitable investment decisions. In the experiments, the Win-Loss Ratio serves as a valuable metric for evaluating the model's ability to predict the directional trends of rising and falling in daily stock market prices.

D.6. Adjustment of Hyperparameters

In the following experiments, it utilizes the LSTM model to assess the impact of hyperparameters on performance. The assumption is that all other hyperparameters remain constant while the current hyperparameter is being adjusted. Each experiment is run for 1000 epochs, repeated ten times. The reported result is the average value obtained from these ten repetitions.

All the experiments can have a good learning result from train set, like Fig. 8, so this research doesn't discuss the training performance in later part, just pay attention on the validation and test.

D.6.1 Bi-Direction

Under identical parameter settings, the price predictions of Bi-LSTM and Uni-LSTM demonstrate similarity, as depicted in Table 1. However, it is noteworthy that despite the comparable predictions, Bi-LSTM (Win-Loss Ratio = 6.0) outperforms Uni-LSTM (Win-Loss Ratio = 3.0) in effectively capturing the daily trends of stock price fluctuations, as Fig. 9. This suggests that the bidirectional nature

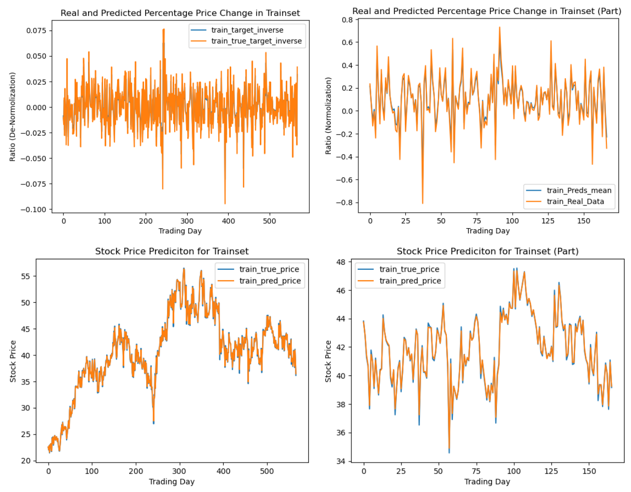

Figure 8. Prediction of Returns and Stock Price in Trainset

Table 1. Comparisons of Bi-LSTM and Uni-LSTM

Model	RMSE	RRMSE	R-square	W-L Ratio
Bi-LSTM	0.85	0.099	0.86	6.0
Uni-LSTM	0.75	0.086	0.89	3.0

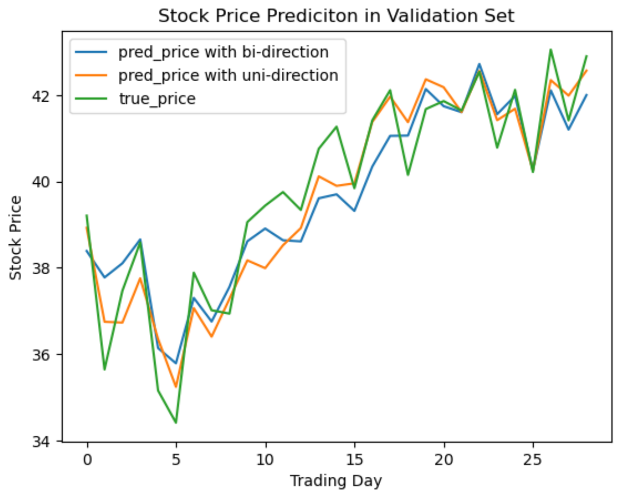

Figure 9. The Prediction with Different Direction

Table 2. Comparisons of Different Stacked Layers

Layers	RMSE	RRMSE	R-square	W-L Ratio
1	0.91	0.106	0.84	13.0
3	0.86	0.100	0.86	13.0
5	0.78	0.091	0.88	4.6
10	0.76	0.088	0.89	4.6

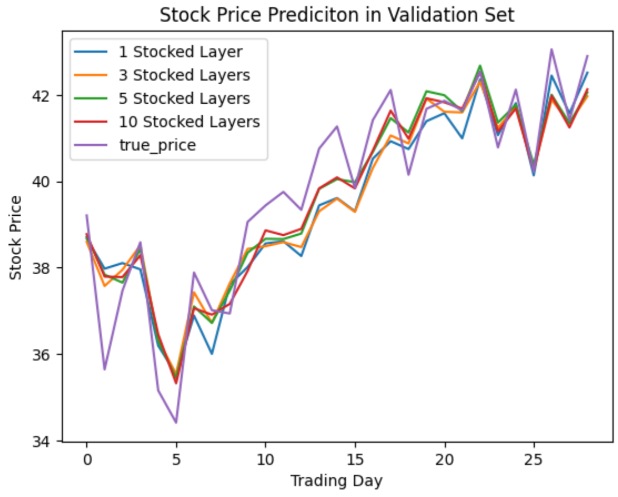

Figure 10. The Prediction with Different Stocked Layers

of Bi-LSTM contributes to a more accurate reflection of the nuanced patterns associated with the rise and fall of daily stock prices.

Unidirectional LSTM, relying solely on past information, may fail to capture sudden market shifts and events influencing future prices trend. In contrast, Bi-LSTM's bidirectional processing comprehensively grasps trends influenced by both past and future data, providing a holistic understanding of dynamic stock market dynamics. Bi-LSTM's superiority is in feature learning, where considering both past and future information allows it to adapt to evolving market conditions and uncover patterns that unidirectional models might overlook. The expanded memory capacity of Bi-LSTM, achieved by processing future information, enhances its adaptability to changes and trends, contributing to more accurate predictions in dynamic financial markets.

D.6.2 Stacked Layers

As the number of stacked layers increases, there is a continuous improvement in RMSE and R-square, indicating enhanced precision in predicting stock prices. However, a notable trade-off emerges as the Win-Loss ratio decreases with the growing number of stacked layers, as Table 2 and Fig. 10. This suggests a potential overfitting issue, which may cause the prediction of the trend of the rise and fall of daily stock becomes unstable.

The observed trade-off between the increasing number of stacked layers and the associated improvements in RMSE and R-square versus the diminishing Win-Loss ratio can be attributed to the delicate balance required in model complexity. As the model becomes more intricate with additional layers, it gains the capacity to capture nuanced patterns and complexities within the training data, thereby enhancing its precision in predicting stock prices as reflected

Table 3. Comparisons of Different Sequence Length

Length	RMSE	RRMSE	R-square	W-L Ratio
3	0.86	0.099	0.86	4.6
5	0.85	0.098	0.87	4.6
10	0.87	0.101	0.86	13.0
20	0.81	0.094	0.88	8.3
30	0.90	0.104	0.85	13.0

in lower RMSE and higher R-square values. However, this heightened sensitivity to the training data comes at the cost of reduced generalization, leading to a decline in the Win-Loss ratio. Overfitting occurs when the model essentially memorizes the training data instead of learning its underlying patterns, causing it to struggle when faced with new, unseen data.

In summary, the increased complexity of the model does contribute to enhanced accuracy metrics, but the declining Win-Loss ratio indicates a potential risk of overfitting. In the context of this experiment, it appears that using three stacked layers serves as an optimal parameter.

D.6.3 Sequence Length

In Table 3 and as illustrated in Fig. 11, it is evident that LSTM models with varying sequence lengths exhibit comparable performance in terms of RMSE and R-square. This suggests that, within the confines of the current training set, both shorter and slightly longer past sequences can effectively capture and reflect the nuances of the current stock price. However, a notable distinction emerges in the Win-Loss ratio, revealing a discernible trend. Specifically, the Win-Loss ratio is relatively low for shorter sequence lengths (3 and 5) and gradually increases for longer sequence lengths (10, 20, and 30).

The discerned pattern in the Win-Loss ratio suggests that, although both shorter and moderately longer sequence lengths contribute to accurate stock price predictions, extending the sequence length enhances the model's performance in terms of Win-Loss ratio. This improvement can be ascribed to the longer sequence to capture intricate patterns and trends associated with the rise and fall of stock prices on each trading day, facilitated by a more extensive historical context. The longer sequence length enables the model to better grasp the nuanced dynamics of market fluctuations, potentially leading to more informed and profitable predictions. However, it is essential to strike a balance, as excessively long sequences may risk overfitting to the training data and may not generalize well to unseen data, so the parameter is set by 10 in this project.

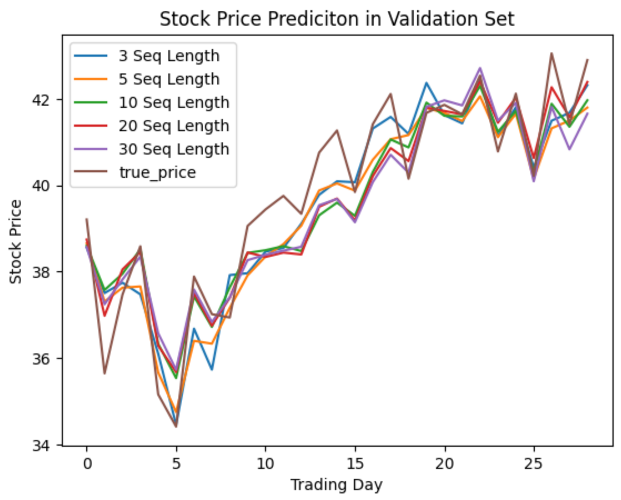

Figure 11. The Prediction with Different Sequence Length

Table 4. Comparisons of Different Hidden State

Length	RMSE	RRMSE	R-square	W-L Ratio
32	0.88	0.102	0.85	6.0
64	0.82	0.095	0.87	8.3
128	0.69	0.080	0.91	13.0
256	0.65	0.076	0.92	6.3

D.6.4 Hidden State

As the hidden state size increases from 32 to 256, there is a consistent improvement in precision metrics, such as decreasing RMSE and RRMSE. This suggests that larger hidden states contribute to enhanced accuracy in predicting stock prices. While larger hidden states lead to improved precision, the Win-Loss Ratio exhibits fluctuations. In particular, a peak is observed at a hidden state size of 128, suggesting an optimal balance between precision and generalization, as Table 4 and as illustrated in Fig. 12

The decrease in the Win-Loss Ratio for a hidden state size of 256 indicates a potential overfitting issue. This decline in the model's ability to profitably predict stock trends may signify a sensitivity to the training data, leading to instability in predicting the rise and fall of daily stock prices.

The experiment demonstrates that increasing the hidden state size in the RNN model generally improves precision in predicting stock prices. However, larger hidden states may lead to overfitting, causing a decline in the Win-Loss Ratio and impacting the stability of predictions.

D.7. The Performances of Different Models in Validation Set

Table 6 and Fig. 13 illustrates a comparison of stock prediction results using different models which include RNN,

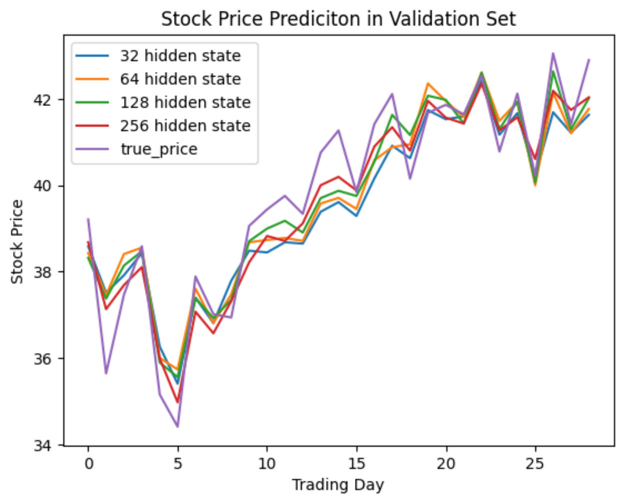

Figure 12. The Prediction with Different Hidden Layers

Table 5. Optimized Hyperparameters

Parameters	Value
input_dim	10
output_dim	1
stocked_layers_dim	3
hidden_state_dim	128
seq_length	10
num_epochs	1000
repeated_trials	10
validation_size	30

LSTM, and GRU. The parameters are identical and set by the optimized values in the last chapter, as Table 5. Observing the results, both RNN and LSTM models exhibit similar performance across most metrics, showcasing lower RMSE and RRMSE values, indicative of more accurate predictions and smaller relative errors.

Additionally, their R-square values are comparable, suggesting good fitting of predicted values to actual values. Notably, the Win-Loss Ratio, a measure of trading strategy success, is highest for the LSTM model, signifying its superior accuracy in predicting market directions and potentially offering more profitable trading strategies.

On the other hand, the GRU model shows relatively lower metrics suggesting a comparatively weaker performance in terms of accuracy and fitting. The Win-Loss Ratio for GRU is also the lowest among the models, indicating potential challenges in generating profitable trading strategies.

In scenarios with shorter sequence lengths (10 in this experiment), RNNs may excel in capturing short-term dependencies due to their innate ability to handle short-term memory. Conversely, more sophisticated models like GRU are known for their prowess in addressing long-term dependencies, so RNNs may prove more effective in this experiment.

Table 6. Comparisons of Different Models

Model	RMSE	RRMSE	R-square	W-L Ratio
RNN	0.69	0.080	0.91	4.6
LSTM	0.69	0.080	0.91	13.0
GRU	0.83	0.097	0.87	1.8

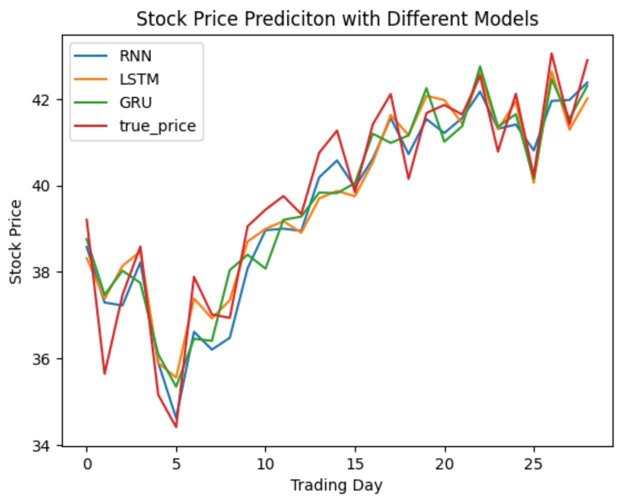

Figure 13. The Prediction with Different Models

While both RNN and LSTM demonstrate robust performance in stock prediction, the LSTM model stands out, particularly in terms of trading strategy success, making it a preferable choice for this specific stock prediction task.

D.8. The Performances of Optimized Model in Test Set

Table 7 and Fig. 14 summarizes the performance metrics of the developed LSTM model trained on AAL data and tested on various airline stocks, including DAL, UAL, and LUV.

AAL exhibits the lowest RRMSE (0.078), indicating accurate predictions with minimal error. The high R-square (0.90) underscores the model's ability to explain a significant portion of AAL's stock price variance. The W-L Ratio (5.6) indicates a favorable balance of correct trend predictions to incorrect ones.

For DAL and UAL, RRMSE values (0.089 and 0.086, respectively) suggest reasonably accurate predictions. High R-square values (0.86 for DAL, 0.91 for UAL) indicate good explanatory power. However, varying W-L Ratios (8.3 for DAL, 2.1 for UAL) suggest differences in accurately capturing daily trends.

LUV shows a higher RRMSE (0.118) with a good R-square (0.83) but a moderate W-L Ratio (3.0), indicating

Table 7. Comparisons in Different Test Dataset

Stocks	RRMSE	R-square	W-L Ratio
AAL	0.078	0.90	5.6
DAL	0.089	0.86	8.3
UAL	0.086	0.91	2.1
LUV	0.118	0.83	3.0

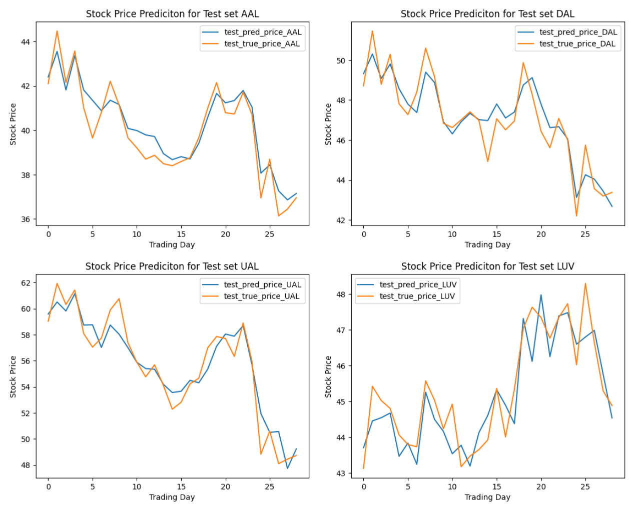

Figure 14. Prediction of Test Datasets

decent performance in predicting trends.

The LSTM model demonstrates good performance on AAL, primarily due to its training on the specific AAL dataset. While its effectiveness varies across other airline stocks, the model performs well overall by accurately predicting the prices and trends in different stocks. The observed variations can be attributed to distinct market behaviors and stock-specific characteristics, making such diversity in performance reasonable. The model showcases satisfactory generalization capabilities, although nuanced considerations are necessary for handling the intricacies associated with diverse financial instruments.

From Fig. 15, the complete learning process and overall performance at different stages are evident. The optimized LSTM model in this experiment exhibits strong performance in the training set, fitting the original values perfectly and capturing both rising and falling trends in stock prices. During the validation step, the model maintains exceptional performance. While there is a decreasing trend in predictive accuracy and trend on the test dataset, the model continues to accurately reflect stock prices and trends, showcasing good generalization.

The decision to use a sequence length of 10 emphasizes short-term stock price changes, allowing the model to predict short-term stock prices based on recent information. However, as the future predicted range of trading days increases, the predictive performance of the model may gradually decline.

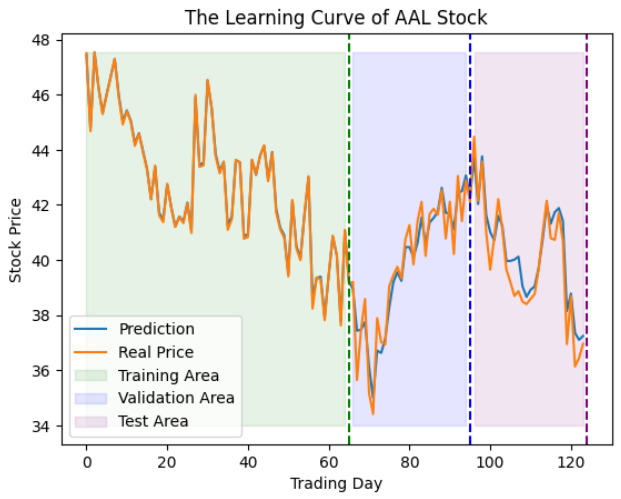

Figure 15. The Learning Curve of AAL Stock
Train Area is the tailed part of training set.

Simultaneously, as D.6.3 demonstrated, longer sequences also do not effectively improve predictive performance. Future long-term trends and complex market dynamics may not be easily predicted with both past short-term and long-term information.

Analyzing the reasons, short-term trends are more apparent. (1) Shorter sequence lengths may be more suitable for capturing short-term market trends, as recent stock price changes may more directly impact upcoming trends. If the market exhibits clear short-term trends, shorter sequence lengths may more sensitively capture these changes. (2) Local data is important. In certain cases, stock price fluctuations may be more influenced by recent events, news, or market sentiment, and longer historical sequences may contain more outdated information. Shorter sequence lengths focus more on recent data, better reflecting the current local market conditions. (3) Model complexity and overfitting. Using shorter sequence lengths can reduce model complexity and mitigate the risk of overfitting. Longer sequence lengths may lead the model to learn too much noise or patterns specific to certain periods during training, resulting in poor generalization to new data.

The disadvantages of using short-term sequence is also obvious. (1) Shorter sequence lengths contain limited historical information and may not fully reflect the long-term trends of stock prices. Long-term predictions typically require more historical data to capture potential long-term patterns and trends, whereas shorter sequence lengths may lead to information loss. (2) Noise and volatility. Long-term predictions are susceptible to noise and transient fluctuations in the market. Shorter sequence lengths may be more prone to interference from these transient changes that are not crucial for long-term trends.

In conclusion, predicting short-term stock prices with shorter sequence lengths appears more reliable. However, both short and long sequence lengths face challenges in accurately predicting long-term stock prices.

E. Conclusion

This research embarked on the task of predicting stock prices using various RNNs, focusing on the stock of AAL as the primary dataset. The exploration encompassed different RNN models, including the basic RNN, LSTM, and GRU, as well as variations such as Bi-LSTM, Deep RNNs with stacked layers, and adjustments to hyperparameters.

Key project design choices include the utilization of Returns as the target variable, extensive feature engineering involving 24 derived features and PCA for dimensionality reduction, and the development of a custom LSTM model class 'LSTM_GRU_RNN' to streamline experimentation.

Notable findings of experiments include the superiority of Bi-LSTM in capturing nuanced patterns associated with daily stock price fluctuations, the delicate trade-off between the number of stacked layers and model performance metrics, the impact of sequence length on prediction accuracy, and the sensitivity of the model to the hidden state size.

The comparison of different models in the validation set revealed that both RNN and LSTM models exhibited robust performance, with LSTM standing out, particularly in terms of the Win-Loss Ratio, indicating its superior accuracy in predicting market directions.

Furthermore, the optimized LSTM model demonstrated strong generalization capabilities in the test set, accurately predicting stock prices and trends for various airline stocks. However, nuanced variations were observed across different stocks, emphasizing the need for careful consideration of specific market behaviors and stock characteristics.

Finally, the research findings suggest that leveraging past short-term information for predicting future short-term stock prices is a reliable approach. However, when aiming to predict future long-term stock prices, it becomes imperative to explore and incorporate additional methods to enhance predictive accuracy and account for the complexities inherent in long-term market dynamics.

Based on the results and limitation of this experiment, the future work may contain (1) Explore the potential benefits of ensemble models, combining predictions from multiple deep learning architectures or incorporating other machine learning models [4]. (2) Implement more sophisticated hyperparameter tuning techniques, such as Bayesian optimization [2], to further refine the model's parameters. This can potentially lead to better-performing models with improved generalization. (3) Develop interpretable models to better understand the features and patterns driving the predictions. This could involve techniques such as attention mechanisms [8]. (4) Investigate dynamic sequence length strategies that adapt to changing market conditions. For example, implementing models that automatically adjust the sequence length based on market volatility or other relevant factors [5].

References

[1] AV Alzheev and RA Kochkarov. Comparative analysis of arima and lstm predictive models: Evidence from russian stocks. *Finance: Theory & Practice*, 24(1):14–23, 2020. 1

[2] Hyunghun Cho, Yongjin Kim, Eunjung Lee, Daeyoung Choi, Yongjae Lee, and Wonjong Rhee. Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE access*, 8:52588–52608, 2020. 10

[3] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014. 2

[4] Mudasir A Ganaie, Minghui Hu, AK Malik, M Tanveer, and PN Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022. 10

[5] Mark Harmon and Diego Klabjan. Dynamic prediction length for time series with sequence to sequence networks. *arXiv preprint arXiv:1807.00425*, 2018. 10

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[7] Duc Huu Dat Nguyen, Loc Phuoc Tran, and Vu Nguyen. Predicting stock prices using dynamic lstm models. In *Applied Informatics: Second International Conference, ICAI 2019, Madrid, Spain, November 7–9, 2019, Proceedings 2*, pages 199–212. Springer, 2019. 5

[8] Jiayu Qiu, Bin Wang, and Changjun Zhou. Forecasting stock prices with long-short term memory neural network based on attention mechanism. *PloS one*, 15(1):e0227222, 2020. 10

[9] Ugur Sahin and A Murat Ozbayoglu. Tn-rsi: Trend-normalized rsi indicator for stock trading systems with evolutionary computation. *Procedia Computer Science*, 36:240–245, 2014. 5

[10] Daiyou Xiao, Jinxia Su, et al. Research on stock price time series prediction based on deep learning and autoregressive integrated moving average. *Scientific Programming*, 2022, 2022. 1

[11] Xianghui Yuan, Jin Yuan, Tianzhao Jiang, and Qurat Ul Ain. Integrated long-term stock selection models based on feature selection and machine learning algorithms for china stock market. *IEEE Access*, 8:22672–22685, 2020. 5