Neural Network-Based Strategies for Robust Stock Market Predictions

Thi Kim Oanh Nguyen
a1879781

1. Introduction and Background

1.1. Problem Statement

The stock market serves as a platform for the transfer, trade, and distribution of stocks, acting as a vital avenue for large companies to raise funds from stockholders. The issuance of stocks channels a substantial capital influx into the market, fostering capital concentration and organic corporate configuration [7]. Examining the stock market involves gathering, organizing, and merging diverse pertinent information. This practice aids in comprehending and anticipating stock price trends, facilitating informed investment decisions aimed at mitigating risks and maximizing returns [5]. Accurate and timely predictions of stock movements can pave the way for suitable regulations and prudent guidance within the stock market, laying a robust foundation for sustained economic growth.

However, predicting stock prices is widely perceived as a complex process due to inherent characteristics like noise and volatility. The intricate development mechanism of stock price prediction involves the interplay of various factors, encompassing political, economic, and market dynamics, technology, and investor behavior. These factors collectively contribute to the fluctuations in stock prices [12]. This continual change in stock prices fosters an environment for speculative activities while elevating market risks. Such risks not only pose potential economic losses for investors but also carry consequential impacts on enterprise and national economic constructs.

Forecasting stock market movements is widely recognized as among the most intricate issues in time-series prediction. It has emerged as a prominent area of study within economic science, fueled by investor concerns and the allure of potentially lucrative returns. De Gooijer et al. (2005) [3] conducted a review of papers published in journals overseen by the International Institute of Forecasters, covering the Journal of Forecasting from 1982 to 1985 and the International Journal of Forecasting from 1985 to 2005. Their findings revealed that more than a third of the papers published in these journals were centered on time-series forecasting. These methodologies primarily center on the time series data without considering additional contextual factors. The stock market, functioning as a complex system influenced by various factors and uncertainties, often demonstrates robust nonlinear traits. This complexity diminishes the effectiveness of conventional analytical methods. Moreover, the substantial volume of information involved in modeling and forecasting the stock market poses significant challenges for algorithm design. In the financial sector, machine learning models have gained extensive usage in analyzing time-series data. Furthermore, deep learning has emerged as a new trend within machine learning owing to its remarkable capacity to delineate nonlinear connections. Deep learning boasts robust capabilities in handling data, effectively addressing challenges stemming from the intricacies of financial time series [7].

This study aims to forecast closing stock market prices utilizing various stock datasets sourced from diverse companies across different sectors. The primary contributions of this research can be outlined as follows: Initially, we predict residual prices by considering retrospective periods of 10, 20, or 30 days. Subsequently, our model is applied to Amazon, Google and Commonwealth Bank datasets. Additionally, we employ LSTM models to evaluate and select optimal hyperparameters, such as hidden size, number of layers, learning rate and optimizer. Lastly, we explore additional neural network models such as RNN, GRU and CNN-LSTM to discern potential variations or enhancements in performance. The rest of the paper is organized as follows. Section 2 describes the method, Section 4 presents the experimental results, and Section 5 discusses future directions of our research.

1.2. Related Work

This section offers an overview of relevant previous literature concerning the prediction of stock market trends. Zhao et al. (2017) [15] conducted experiments using various time-weighted functions within the LSTM model and figured out the varying relationship between data importance and their time-series using the CSI 300 index. Zhang et al. (2018) [14] combined CNN with RNN for getting the correlation between various temporal data, addressing the significant limitations of general RNN models in achieving this objective. This approach led to a 30% reduction in the

mean squared error for predictions compared to standard RNN models. Kim and Won (2018) [8] utilized KOSPI 200 index data to compare single models, including GARCH, exponential GARCH, exponentially weighted moving average, deep feedforward neural network (DFN), and LSTM, as well as hybrid DFN models incorporating a DFN with one GARCH-type model. Jin et al. (2019) [7] incorporated investor sentiment trends into their model analysis, combining empirical modal decomposition (EMD) with LSTM to achieve improved accuracy in stock forecasts. While the attention mechanism-based LSTM model is prevalent in speech and image recognition, its application in finance remains relatively uncommon. Nguyen and Yoon (2019) [10] employed a pre-trained model utilizing LSTM cells with suitable input features and optimal initial training parameters, referred to as the deep transfer with related stock information (DTRSI). Derakhshan and Beigy (2019) [4] forecasted stock prices by employing model-based opinion mining, utilizing a part-of-speech graphical model to extract user opinions. Their approach was tested across two distinct datasets, one in English and another in Persian. Banyal et al. (2020) [1] introduced a Stacked LSTM and Multilayer Perceptron (MLP) model to augment stock prediction accuracy. Qiu et al. (2020) [13] conducted a comparison between the LSTM model alongside wavelet denoising and the GRU model using datasets from S&P500, DJIA, and HSI. Both models showcased a mean square error lower than 0.05.

In this study, we made data stationary by applying a logarithmic transformation to stabilize the variance, while differencing the data to eliminate the trend component and subsequently utilizing a moving average to reduce noise or fluctuations, helping to highlight trends or patterns by smoothing out short-term variations. Subsequently, we conducted experiments involving various hyperparameters for LSTM, comparing its performance with RNN, GRU, and CNN-LSTM models to enhance predictive capabilities. From our experiments, we observed promising outcomes across all models. We attained RMSE and MAE scores lower than 0.05 for RNN, GRU, LSTM, and CNN-LSTM models. This includes LSTM models experimented on different datasets from Amazon, Google, and Commonwealth Bank.

2. Method Description

2.1. RNN Model

The Recurrent Neural Network (RNN) is a type of neural network designed to handle sequential data by retaining memory of past inputs through loops within the network (Figure 1). This enables the RNN to exhibit temporal dynamic behavior and process sequences of varying lengths.

Mathematically, an RNN operates recursively at each

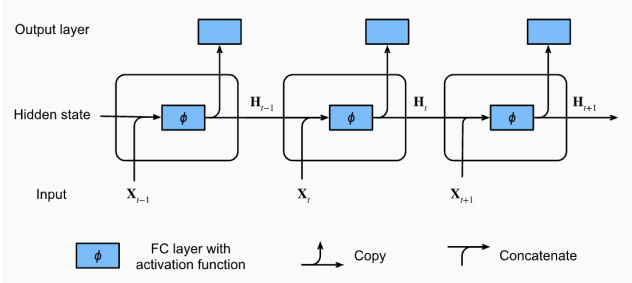

Figure 1. Structure of RNN block. Source: https://d2l.ai

time step t and consists of three main components: an input at time t denoted by x_t, a hidden state at time t denoted by h_t, and an output at time t denoted by y_t. The hidden state h_t is computed based on the current input x_t and the previous hidden state h_{t-1} using a set of weight matrices W and U, and an activation function ϕ:

$$h_t = \phi(W \cdot x_t + U \odot h_{t-1}) \tag{1}$$

Here, W represents the weights for the current input, U represents the weights for the previous hidden state, and ϕ is usually a non-linear activation function such as the hyperbolic tangent (tanh) or Rectified Linear Unit (ReLU).

The output y_t at each time step can be computed based on the hidden state h_t:

$$y_t = V \cdot h_t \tag{2}$$

Where V represents the weights for the output layer.

This recursive nature allows RNNs to capture temporal dependencies in sequential data. However, traditional RNNs suffer from vanishing or exploding gradient problems when dealing with long sequences, which can hinder their ability to retain information over long-term dependencies.

The vanishing and exploding gradient problems are inherent challenges in training Recurrent Neural Networks (RNNs), especially when dealing with long sequences. These issues arise due to the backpropagation of errors through multiple time steps, leading to either vanishingly small gradients or explosively large gradients.

The vanishing gradient problem occurs when the gradients during backpropagation diminish rapidly as they propagate backward through time. Mathematically, the gradients in RNNs are calculated using the chain rule, where the gradient at each time step is the product of the gradients from subsequent time steps. For instance, consider the gradient computation for a specific weight W at time step t:

$$\frac{\partial Loss}{\partial W_t} = \frac{\partial Loss}{\partial h_t} \cdot \frac{\partial h_t}{\partial W_t} = \left(\prod_{k=t}^{1} \frac{\partial h_k}{\partial h_{k-1}} \right) \cdot \frac{\partial h_t}{\partial W_t} \tag{3}$$

In situations where the recurrent weight matrices $\frac{\partial h_k}{\partial h_{k-1}}$ have eigenvalues less than 1, the product $\prod_{k=t}^{1} \frac{\partial h_k}{\partial h_{k-1}}$ approaches zero as t increases, causing the gradient to vanish. This phenomenon leads to the inability of the network to learn long-term dependencies [11].

Conversely, the exploding gradient problem occurs when the gradients grow exponentially as they propagate backward through time. This can result from weight matrices having eigenvalues greater than 1, causing the product of these values to explode to extremely large values. Consequently, the network experiences unstable training, making it difficult to update the model parameters effectively.

Both the vanishing and exploding gradient problems affect the RNN's ability to capture and retain information over long sequences, hindering its performance in learning and recalling long-term dependencies.

2.2. LSTM Model

The Long Short-Term Memory (LSTM) architecture is a specialized form of recurrent neural network (RNN) designed specifically to overcome the limitations of standard RNNs in capturing and retaining long-term dependencies within sequential data [10]. One of the primary challenges of conventional RNNs is the vanishing gradient problem, where gradients diminish significantly during backpropagation through time, affecting the network's ability to capture long-range dependencies.

LSTMs are equipped with gating mechanisms that enable them to control the flow of information, allowing selective retention and utilization of information at different time steps. This mechanism enables LSTMs to effectively learn and retain temporal patterns over extended sequences of data.

The LSTM cell, depicted in Figure 2, operates at each time step within a sequence. It comprises three essential components: the input gate (i_t), the forget gate (f_t), and the output gate (o_t). These gates regulate the flow of information by deciding what information to keep, discard, or output to the next cell state or hidden state. The forget gate determines the information to discard from the previous cell state, the input gate controls the updates to the cell state based on the current input and previous hidden state, and the output gate determines the output based on the updated cell state and hidden state.

Mathematically, the operations within an LSTM cell involve various transformations and activations. The forget gate (f_t), input gate (i_t), and output gate (o_t) are calculated using sigmoid activation functions:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{4}$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{5}$$

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \tag{6}$$

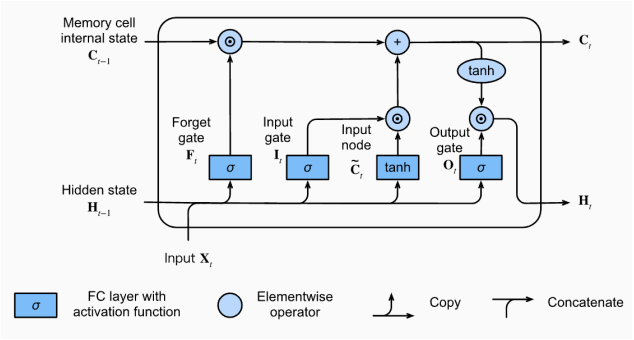

Figure 2. Structure of LSTM block. Source: https://d2l.ai

Here, h_{t-1} represents the previous hidden state, x_t is the current input, and W_f, W_i, and W_o are weight matrices corresponding to the forget, input, and output gates, respectively. b_f, b_i, and b_o denote the respective bias terms.

The cell state c_t is updated by applying the forget gate to the previous cell state c_{t-1} and adding the new information obtained from the input gate and tanh activation:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \tag{7}$$

The hidden state h_t is derived by passing the updated cell state through the output gate and applying the tanh activation:

$$h_t = o_t \odot \tanh(c_t) \tag{8}$$

The tanh activation function is defined as $\tanh(z) = \frac{2}{1+e^{-2z}} - 1$. It's utilized to update the hidden state within the LSTM architecture. This function helps manage the values of the hidden state, ensuring they fall within the range $[-1, 1]$.

The sigmoid activation function is defined as $\sigma(z) = \frac{1}{1+e^{-z}}$. This function constrains the output values within the range of $[0, 1]$. A value close to 0 implies that most information is discarded or lost, while a value closer to 1 signifies that more information is retained or allowed [6].

Overall, these mechanisms allow LSTMs to effectively capture long-term dependencies and handle sequential data by selectively retaining and utilizing relevant information.

2.3. GRN Model

GRU, a variant of the Recurrent Neural Network (RNN), is a simplified form of the Long Short-Term Memory (LSTM) model. Unlike LSTM's three gates, GRU comprises two gates: the reset gate (r_t) and the update gate (z_t).

Initially, the reset gate (r_t) determines how to combine the new input with the previous memory, allowing the network to decide which information to discard or update. Simultaneously, the update gate (z_t) dictates the extent to which the previous memory should be retained or overwritten by the new input [2].

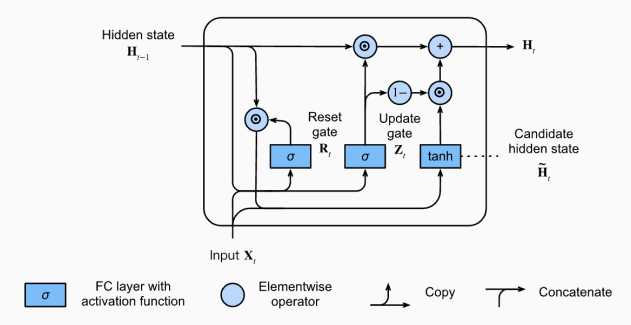

Figure 3. Structure of GRU block. Source: https://d2l.ai

The fundamental equations for the GRU model are as follows:

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1}) \tag{9}$$

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1}) \tag{10}$$

$$\tilde{h}_t = \tanh(r_t \odot U \cdot h_{t-1} + W \cdot x_t) \tag{11}$$

$$h_t = (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} \tag{12}$$

Here, r_t and z_t represent the reset and update gates, respectively. x_t denotes the current input, h_{t-1} is the previous hidden state, $\tilde{h}_t$ is the candidate activation, and h_t is the current hidden state. W_r, U_r, W_z, and U_z are weight matrices for the reset and update gates, while W and U are weight matrices for the candidate activation. The σ function represents the sigmoid activation, and tanh represents the hyperbolic tangent activation. The $\odot$ symbol denotes element-wise multiplication.

These equations depict the flow of information (Figure 3) within the GRU architecture, capturing how the gates interact with input and previous hidden states to update the current hidden state.

2.4. CNN-LSTM Model

The CNN-LSTM model is a hybrid architecture that combines the Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) models. This combined model leverages the strengths of CNNs in spatial feature extraction and LSTMs in sequential modeling, making it effective for tasks involving both spatial and temporal dependencies, such as video analysis, time series forecasting, and natural language processing.

The architecture typically involves two main components: a CNN module followed by an LSTM module.

The CNN module performs feature extraction from the input sequence. Let X represent the input sequence data at time t, which is passed through convolutional layers followed by pooling layers to capture spatial patterns. The output of the CNN module, denoted as C, is the learned spatial features:

$$C = \text{CNN}(X) \tag{13}$$

The LSTM module processes the learned spatial features C to capture temporal dependencies and generate predictions. Let H_t represent the hidden state of the LSTM at time t. The LSTM computations involve gates (input, forget, output gates) and memory cells to manage information flow. The LSTM's hidden state H_t is updated based on the input C and the previous hidden state H_{t-1}:

$$H_t = \text{LSTM}(C, H_{t-1}) \tag{14}$$

The LSTM output Y_t at each time step t is derived from the hidden state H_t using an output layer:

$$Y_t = \text{OutputLayer}(H_t) \tag{15}$$

In summary, the CNN-LSTM model first extracts spatial features from the input sequence using CNN layers, then uses the LSTM to capture temporal dependencies and generate predictions based on the learned features [9].

This architecture is particularly useful in tasks where both spatial and temporal information play crucial roles, enabling the model to effectively capture complex patterns and dependencies in data sequences.

2.5. Performance evaluation metric

Common evaluation metrics used for regression models include Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) [7].

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{16}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{17}$$

Concerning, n is the number of samples. y_i represents the actual value of the target variable for the i-th sample. $\hat{y}_i$ represents the predicted value of the target variable for the i-th sample.

3. Method Implementation

Our codes are here: https://github.com/OanhOlivia/Deep-Learning/blob/main/a1879781_DL_A3.ipynb

4. Experiment and Analysis

4.1. Data Preprocessing and Feature Selection

Preprocessing steps are applied to all feature columns (Open, High, Low, Close, Adj Close, and Volume) to transform and prepare the raw stock market data for machine learning models. We utilize log transformation to stabilize

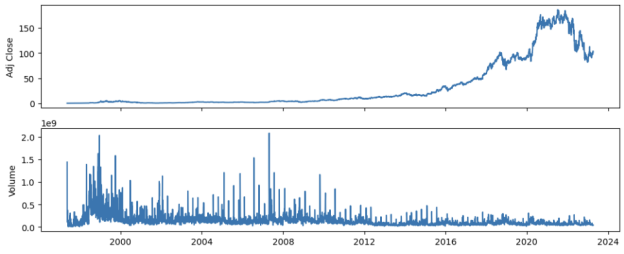

Figure 4. Original Data (Adj Close and Volume Features)

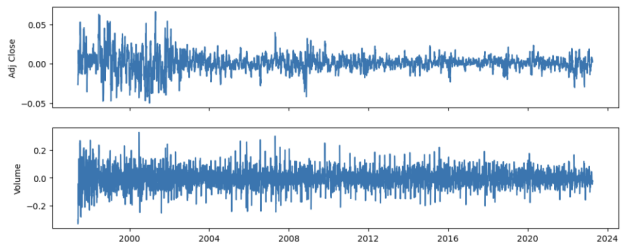

Figure 5. Transformed Data (Adj Close and Volume Features)

and scale the data and differencing to eliminate trends, especially in cases where the data exhibits varying scales and exponential growth, as depicted in Figure 4. The moving average technique is employed to diminish noise and capture underlying trends by averaging values within a specified window size. Smoothing the data aids in identifying patterns or trends by eliminating short-term fluctuations. As shown in the results from Figure 5, the data displays increased stationarity and exhibits consistent variance. The Close column is shifted by one day to align it with the data for the following day. This step creates the target variable for a supervised learning setup where the model aims to predict tomorrow's closing price based on today's information.

The creation of lag features aims to empower the model to learn from historical observations and capture temporal dependencies within the dataset. This methodology enables the model to establish connections between the current Close price and its past values at various time steps (Close(t-1), Close(t-2), etc.), offering a sequential perspective that proves valuable in time series forecasting tasks. We experiment with different time steps – 10, 20, and 30 days – during Lookback Periods to assess how these features influence the performance of the models. The lag features, combined with the initial six features, form the set of features used to predict the stock price.

4.2. Stock Datasets

We experiment with stock prediction models across datasets from Amazon, Google, and Commonwealth Bank, representing diverse sectors including commerce, technology and finance. The Amazon dataset spans from

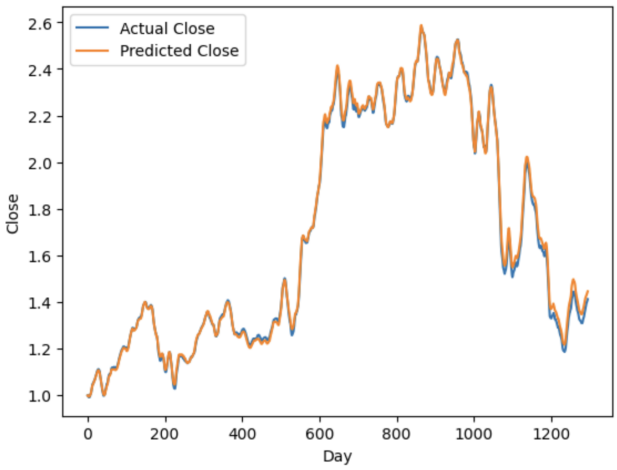

Figure 6. Amazon Dataset

15/5/1997 to 5/4/2023, the Google dataset from 3/1/2005 to 15/11/2023, and the Commonwealth Bank dataset from 1/1/1992 to 22/11/2023, sourced from Yahoo Finance. Testing models across different datasets helps evaluate their generalization ability. A model that performs well across multiple datasets indicates robustness and applicability in diverse scenarios. Stocks from different companies or industries exhibit varying behaviors due to their unique market influences. Experimenting across these helps in understanding how models respond to diverse market behaviors. Relying on models that perform consistently across different datasets reduces the risk associated with overfitting to a specific dataset. It provides confidence that the model is not capturing idiosyncrasies of a particular stock or industry. Different datasets challenge models differently. Testing on varied datasets helps in understanding a model's sensitivity to different market conditions, data qualities, or stock characteristics. Figures 6, 7, and 8 demonstrate LSTM models predict stock prices well across these three datasets.

4.3. Lookback Periods

Considering previous days' data in stock price prediction is crucial for models to learn from long-term trends and behaviors that affect stock prices over extended periods. Firstly, stock prices exhibit temporal dependencies, meaning today's price is often related to past prices. By looking back at historical data, the model can capture trends, seasonality, and cyclic patterns that influence stock prices. Secondly, stock prices follow a sequence. Previous days' prices can provide crucial information about the direction and magnitude of future prices. Utilizing this sequential information helps in understanding how the price evolves over time. Finally, lag features derived from past stock prices act as additional features for the model. These lag features en-

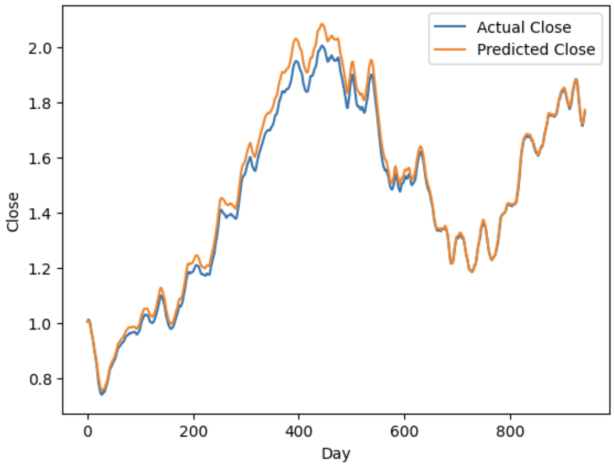

Figure 7. Google Dataset

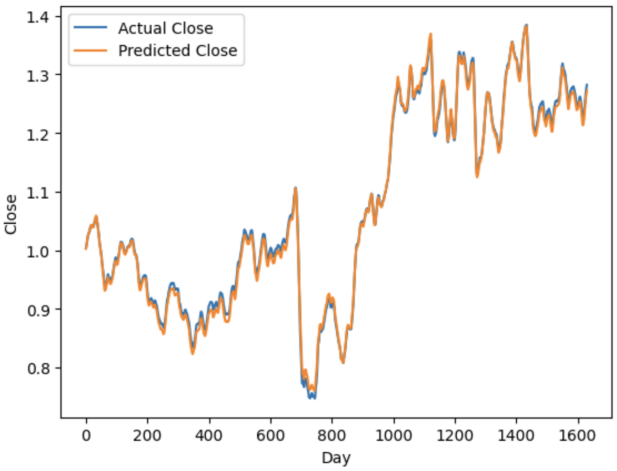

Figure 8. Commonwealth Bank Dataset

able the model to relate the current price to its historical values at different time steps. This sequential perspective aids in making more informed predictions. When experimenting with various lookback periods (10, 20, 30 days) on Amazon and Google datasets in Tables 1, 2, 30-day lookback period shows the lowest error scores. With a longer window, such as 30 days, the model filters out short-term noise or price fluctuations, emphasizing underlying trends and influential patterns in price movements. However, the effectiveness of the lookback period relies on the dataset's specific characteristics and the nature of the analyzed stock market. Therefore, we opt for a 10-day lookback to maintain simplicity in our models.

Lookback	RMSE	MAE
10	0.0258	0.0187
20	0.0262	0.0206
30	0.0183	0.0132

Table 1. Amazon Dataset

Lookback	RMSE	MAE
10	0.0474	0.0430
20	0.0516	0.0457
30	0.0384	0.0296

Table 2. Google Dataset

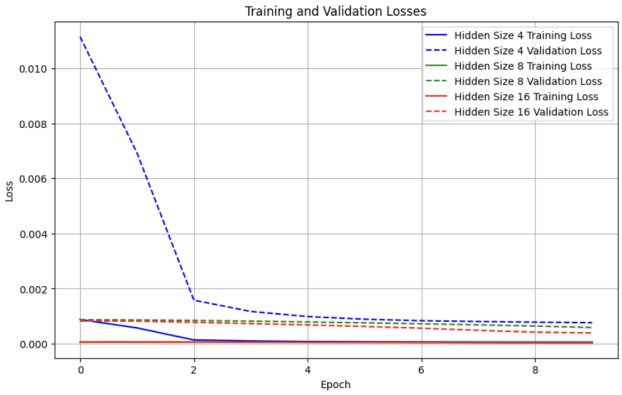

Figure 9. Training and Validation Loss of different Hidden Sizes

4.4. Hyperparameters Tuning

4.4.1 Hidden Sizes

Hidden size significantly impacts the model's ability to understand and capture data patterns. We conduct experiments with different hidden sizes — 4, 8, and 16 — using the LSTM model with 1 stacked layer and learning rate 0.001 and Adam optimizer. Larger hidden sizes tend to perform better on training data but might struggle with generalizing to new, unseen data. Conversely, smaller hidden sizes are less prone to overfitting but might lack the capacity to capture complex patterns. What is important to consider is finding a size that strikes a balance between performance and generalization. From Figures 9, 10, a hidden layer size of 4 appears to be more suitable.

4.4.2 Model Layers

Experimenting with different numbers of stacked LSTM layers (with hidden sizes set to 4, learning rate at 0.001, and using the Adam optimizer) involves evaluating the depth and complexity of the model. As the number of stacked

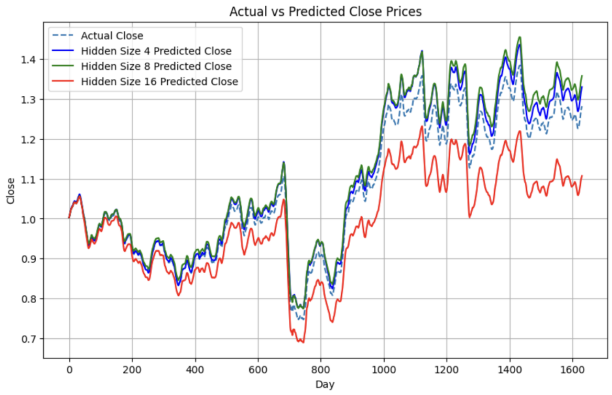

Figure 10. Actual and Predicted Price of different Hidden Sizes

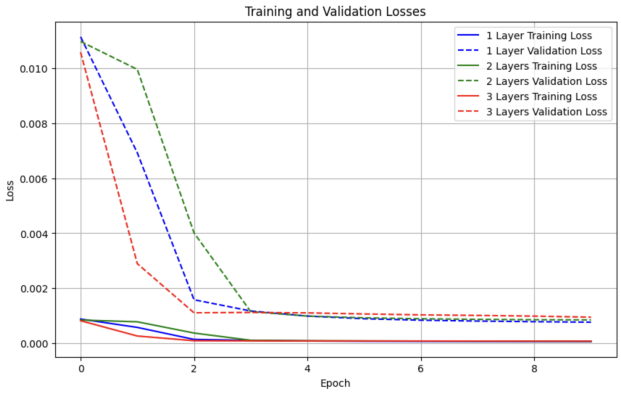

Figure 11. Training and Validation Loss of different Numbers of Layers

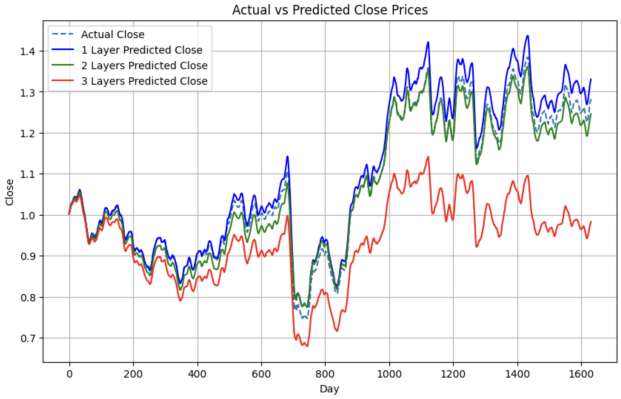

Figure 12. Actual and Predicted Price of different Numbers of Layers

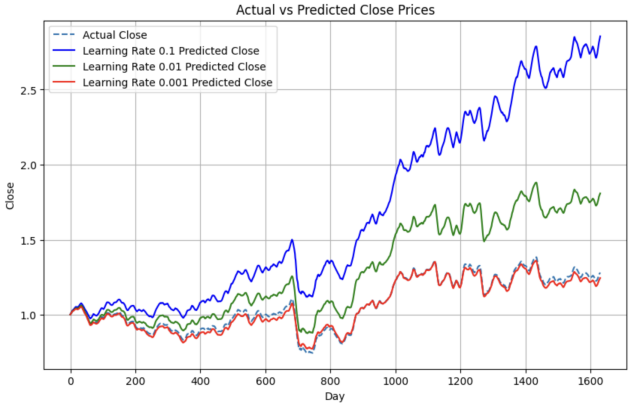

Figure 13. Actual and Predicted Price of different Learning Rates

layers increases (from 1 to 3), the model gains the ability to capture more intricate patterns and dependencies within the data. However, with added layers, the model's complexity rises, potentially leading to issues during training, such as vanishing or exploding gradients. These challenges can impact the model's convergence and training time. Similar to selecting the hidden size, we aim to choose a two-layer configuration that strikes a balance between model complexity, training dynamics, and generalization performance, as illustrated in Figures 11 and 12.

4.4.3 Learning Rates

The learning rate determines the size of the step taken during the optimization process. Higher learning rates allow for faster convergence but might overshoot the optimal values and cause instability or prevent convergence. Conversely, very low learning rates might converge slowly or get stuck in local minima. When experimenting with different learning rates (0.001, 0.01, 0.1) for an LSTM model with a batch size of 16, two layers, a hidden size of 4, and Adam optimizer, the learning rate of 0.001 produces the best result (Figure 13). However, learning rates can be influenced by various factors such as optimizers (Adam, SGD), batch size, and model architecture. The depth, complexity, and type of the model, including LSTM layers and parameter numbers, impact the learning rate's effect on convergence. This suggests the potential for better performance through exploring different combinations of learning rates and other parameters. Further investigation might unveil more optimal settings for improved model performance and convergence.

4.4.4 Optimizers

Following above experiments, we identify the optimal LSTM model using the Adam optimizer, featuring two layers, a hidden size of 4, and a learning rate set at 0.001. However, similar to the selection of the learning rate, the choice of optimizer also depends on its interaction with other parameters. Figures 14 and 15 along with Table 3 showcases that the LSTM model, employing the SGD optimizer with a learning rate of 0.01 and a momentum of 0.9 — maintain-

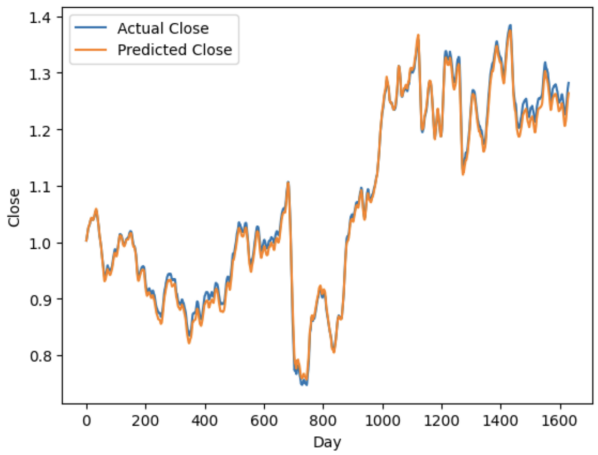

Figure 14. Actual and Predicted Price of LSTM - SGD Model

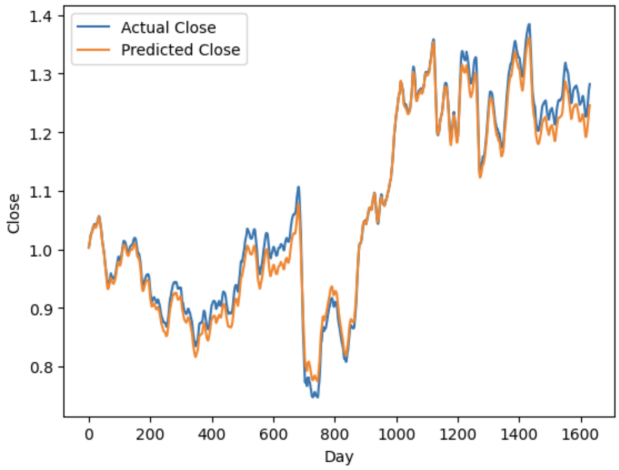

Figure 15. Actual and Predicted Price of LSTM - Adam Model

Optimizer	RMSE	MAE
SGD	**0.0099**	**0.0085**
Adam	0.0194	0.0163

Table 3. Error Metric of LSTM - SGD and Adam Model

ing a single layer and a hidden size of 4 — outperforms the equivalent Adam-based model. As a result, this LSTM configuration using the SGD optimizer is considered the best-performing model for comparison against other neural network models in the next section.

4.5. Models Experiment

LSTM model demonstrated the best performance when configured with a single layer, a hidden size of 4, SGD optimizer with a learning rate of 0.01, and a momentum of 0.9. RNN model exhibited competitive results with 2 stacked

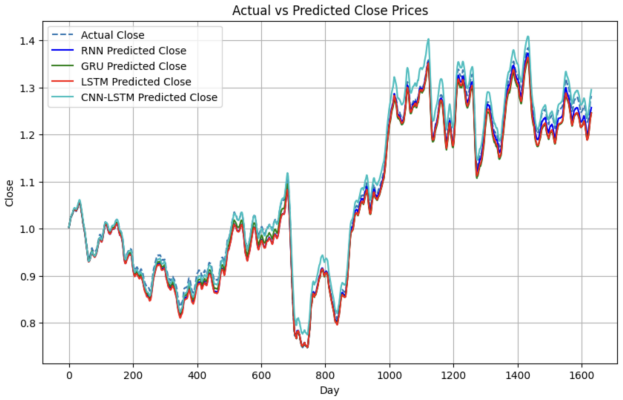

Figure 16. Actual and Predicted Price of all Models

Model	RMSE	MAE
RNN	0.0154	0.0129
GRU	0.0183	0.0160
LSTM	**0.0099**	**0.0085**
CNN-LSTM	0.0178	0.0143

Table 4. Error Metric of all Models

layers, a hidden size of 16, a learning rate of 0.001, and the Adam optimizer. GRU model performed well with 1 stacked layer, a hidden size of 16, a learning rate of 0.001, and the Adam optimizer. CNN-LSTM model, configured with 1 stacked layer, a hidden size of 4, 16 filters, a filter size of 2, a learning rate of 0.001, and the Adam optimizer, also demonstrated strong performance.

Looking at Figure 16, it is challenging to discern the model that performs the best since all predicted lines closely align with the actual close line. However, Table 4 illustrates that the LSTM achieves the best results, displaying an RMSE of 0.0099 and an MAE of 0.0085. While the LSTM model showcases optimal performance in this research, it is important to note that its superiority within this dataset does not ensure similar outcomes across all datasets. Further experimentation and careful selection of hyperparameters might reveal enhancements in other models.

5. Reflection on Project

5.1. Conclusion

In this study, various neural network architectures, including LSTM, RNN, GRU, and CNN-LSTM, were examined and evaluated for their efficacy in predicting stock prices. The experimentation involved fine-tuning hyperparameters like hidden size, stacked layers, optimizers, and learning rates across LSTM models, alongside exploring different lookback periods. Remarkably, a lookback period

of 30 days appeared to exert a more significant influence on the models' performance.

The findings revealed that while LSTM model exhibited strong performance — specifically, the LSTM model with a single layer, hidden size of 4, utilizing the SGD optimizer with a learning rate of 0.01 and a momentum of 0.9 showcased superior predictive capabilities. However, RNN, GRU and CNN-LSTM models also demonstrated competitive performances, indicating the significance of thoroughly exploring various architectures and hyperparameters.

Notably, the optimal model's selection varied based on the dataset characteristics, emphasizing the necessity of tailoring model architectures to the dataset specifics. Moreover, the performance of models across various datasets might differ, implying the importance of considering the dataset's peculiarities when choosing the appropriate architecture.

This research underscores the significance of balancing model complexity and performance to achieve accurate predictions in stock price forecasting. Further exploration and experimentation with different architectures and hyperparameters are recommended to ascertain the ideal model for specific datasets and promote robust predictive models in stock price prediction.

5.2. Future Work

Future work in this domain could explore several avenues to enhance the predictive capabilities of models in stock price prediction. Firstly, the incorporation of external factors such as news sentiment analysis, economic indicators, or social media trends could enrich the models with additional contextual information, potentially improving their forecasting accuracy. Additionally, ensemble techniques or hybrid models combining multiple architectures (LSTM, CNN-LSTM, attention mechanisms) might offer improved predictive power by leveraging the strengths of different models. Exploring more advanced optimization techniques, such as learning rate schedules, adaptive learning rates, or different weight initializations, could further enhance the training dynamics and convergence speed of the models. Furthermore, the application of transfer learning or pre-trained models might aid in capturing intricate patterns in stock market data, particularly in scenarios with limited training data. Addressing interpretability and explainability concerns by employing techniques like attention mechanisms or model visualization methods could enhance the transparency of the models' predictions, crucial for stakeholders in the financial domain. Lastly, considering various time horizons for predictions (short-term vs. long-term) and conducting extensive experiments across diverse market conditions and sectors could provide a comprehensive understanding of model performance under varying circumstances.

References

[1] Banyal, S, Goel, P Grover, D 2020, 'Indian Stock-Market Prediction using Stacked LSTM AND Multi-Layered Perceptron', *International Journal of Innovative Technology and Exploring Engineering*, vol. 9, no. 3, pp. 1051–1055. 2

[2] Chung, J, Gulcehre, C, Cho, K Bengio, Y 2014, 'Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling', *ArXiv.Org*. 3

[3] De Gooijer, JG, Hyndman, RJ, 2005, '25 years of IIF time series forecasting: a selective review', *Soc Sci Electron Publ*, vol. 22(3), pp. 443–473. 1

[4] Derakhshan, A Beigy, H 2019, 'Sentiment analysis on stock social media for stock price movement prediction', *Engineering Applications of Artificial Intelligence*, vol. 85, pp. 569–578. 2

[5] Fama, EF, 1998, 'Market efficiency, long-term returns, and behavioral finance', *J Financ Econ*, vol. 49(3), pp. 283–306. 1

[6] Farzad, A, Mashayekhi, H Hassanpour, H 2019, 'A comparative performance analysis of different activation functions in LSTM networks for classification', *Neural Computing Applications*, vol. 31, no. 7, pp. 2507–2521. 3

[7] Jin, Z, Yang, Y Liu, Y 2020, 'Stock closing price prediction based on sentiment analysis and LSTM', *Neural Computing Applications*, vol. 32, no. 13, pp. 9713–9729. 1, 2, 4

[8] Kim, HY Won, CH 2018, 'Forecasting the volatility of stock price index: A hybrid model integrating LSTM with multiple GARCH-type models', *Expert Systems with Applications*, vol. 103, pp. 25–37. 2

[9] Liu, S, Zhang, C Ma, J n.d., 'CNN-LSTM Neural Network Model for Quantitative Strategy Analysis in Stock Markets', in *Neural Information Processing*, Springer International Publishing, Cham, pp. 198–206. 4

[10] Nguyen, T-T Yoon, S 2019, 'A Novel Approach to Short-Term Stock Price Movement Prediction using Transfer Learning', *Applied Sciences*, vol. 9, no. 22, pp. 4745-. 2, 3

[11] Pascanu, R, Mikolov, T Bengio, Y 2013, 'On the difficulty of training Recurrent Neural Networks', *ArXiv.Org*. 3

[12] Patel, J, Patel, M Darji, M 2018, 'Stock Price prediction using clustering and regression: A Review', *IJSRCSEIT*, vol. (3) 1, pp. 1967-1971. 1

[13] Qiu, J, Wang, B Zhou, C 2020, 'Forecasting stock prices with long-short term memory neural network based on attention mechanism', *PloS One*, vol. 15, no. 1, pp. e0227222–e0227222. 2

[14] Zhang, R, Yuan, Z Shao, X 2018, 'A New Combined CNN-RNN Model for Sector Stock Price Analysis', in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, IEEE, pp. 546–551. 1

[15] Zhao, Z, Rao, R, Tu, S Shi, J 2017, 'Time-Weighted LSTM Model with Redefined Labeling for Stock Trend Prediction', in *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, pp. 1210–1217. 1