Stock Prediction using RNN

Yuchen Liu
The University of Adelaide
yuchen.liu01@adelaide.edu.au

Abstract

This study examines the performance of vanilla Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models in predicting Google stock prices. The study focuses on evaluating the impact of differect time series lengths and data preprocessing techniques, including addressing anomalies in the "Close" column. The results show that both RNN and LSTM models perform reasonably well, with LSTM generally achieving better accuracy. Additionally, straightforward preprocessing methods proved to be the most effective in improving model performance for this study. This work highlights how choices in input data and preprocessing affect prediction outcomes, offering practical insights for using deep learning in financial time-series forecasting. The code for this project is available on the GitHub page (https://github.com/MorainingErin/COMP_SCI_7318_DLF/tree/main/A3).

1. Introduction

Financial data is essential for decision-making in areas like investment, risk management, and economic forecasting. Among various types of financial data, stock prices stand out for their dynamic and often unpredictable nature, providing insights into the behavior of financial markets. The accurate stock price prediction is a long-standing challenge in quantitative finance. For traders and analysts, these predictions are not just number; they are translations of what the traind in markets financial decision-making could possibly be.

In recent years, deep learning has opened exciting possibilities regarding tackling the complexities of stock price prediction. Recurrent Neural Networks (RNNs)[3], along with their variants such as Long Short-Term Memory (LSTM)[6] and Gated Recurrent Unit (GRU)[2] networks, have gained recognition for their effectiveness in processing sequential data due to their ability of capture patterns and dependencies over time. These models have been successfully applied to a wide range of time-series predicting tasks, including financial market predictions where understanding temporal relationships is crucial.

In this assignment for the course "Deep Learning Fundamentals" [9], the performance of RNNs for a stock price prediction task is evaluated. Specifically, the study explores the usage of vanilla RNN and LSTM architectures to predict Google stock prices. Through a series of experiments, this study examines how varying time series lengths, different data preprocessing strategies, and different normalization methods affect model performance. By analyzing these factors, this report provides insights into the predictive behavior of vanilla RNN and LSTM models in financial time-series forecasting and how these models respond to changes in input data and preprocessing approaches.

The organization of this report is as follows: Sec.1 introduces the background and motivation for the assignment. Sec.2 summaries the related research on stock price prediction using RNNs and LSTMs. Sec.3 defines the problem and outlines the experimental methodology. Section 4 presents the results of our experiments, including an analysis of time series length and preprocessing techniques, and also discusses the findings and their implications. The code for this assignment is available on the GitHub page (Sec.5). Finally, Section 6 concludes the report and proposes directions for future research.

2. Related Works

Stock price prediction has been a challenging and extensively studied problem in finance and machine learning. Traditional approaches relied on statistical methods such as AutoRegressive Integrated Moving Average (ARIMA)[1] and other econometric models. While these methods have demonstrated strong performance in modeling linear relationships, their ability to capture the nonlinear and dynamic patterns inherent in financial time series is limited. Moreover, these models often struggle to handle the noise and complexity of real-world stock market data, necessitating more robust solutions.

In recent years, the deep learning methods has provided a transformative approach to stock price prediction, leveraging its ability to model complex, nonlinear dependencies.

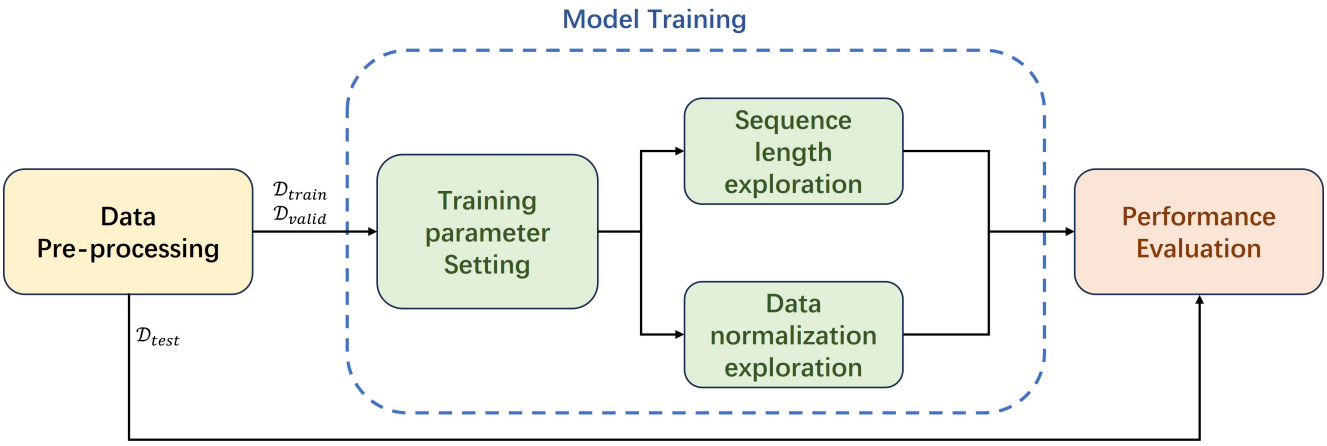

Figure 1: **Overview of the model training pipeline.** The pipeline consists of three main stages: data preprocessing, model training, and performance evaluation. During model training, two experimental explorations—sequence length and normalization strategies—are conducted. The results from the experiments are fed into the performance evaluation stage for analysis.

Among these deep learning architectures, Recurrent Neural Networks (RNNs)[3] have been the most successful for time-series predicting tasks due to their inherent design for sequential data. RNNs capture temporal dependencies by having a hidden state that changes over time, enabling the network to learn patterns from historical data. The vanilla RNN model often suffers from several drawbacks. For example, the vanishing gradients makes RNN difficult to capture long-term dependencies in sequential data. This limitation reduces their effectiveness for tasks like stock price prediction, where understanding trends over extended periods is crucial.

To overcome these challenges, advanced RNN variants were developed, such as Long Short-Term Memory (LSTM) networks [6] and Gated Recurrent Units (GRUs) [2]. These models use gating mechanisms to control the flow of information, allowing them to recognize important patterns while reducing vanishing gradient problems. Therefore, LSTMs and GRUs are well-suited for tasks with complex temporal relationships, such as stock price forecasting, and have become popular choices in time-series prediction.

In this assignment, we focuses on vanilla RNNs and LSTMs for stock price prediction. The vanilla RNN is included as a baseline due to its simplicity, while the LSTM is selected for its ability to capture long-term dependencies in financial data. By examining how these two models perform under different sequence lengths and preprocessing strategies, this study provides a better understand to the role of architecture and data handling in prediction accuracy.

3. Method

The model training pipeline is depicted in Figure 1, involving data preprocessing, parameter setting and evaluation.

3.1. Problem Definition

For the stock price prediction problem, the objective is to predict the stock price in the future. The prediction process can be defined as follows:

$$f : \{\mathbf{x}_{t-T+1}, \ldots, \mathbf{x}_{t-2}, \mathbf{x}_{t-1}\} \rightarrow \mathbf{x}_t, \tag{1}$$

where $\mathbf{x}_i$ represents the stock price at the time i. t refers to the predicted time, and T denotes the window size, indicating the amount of history data used for prediction. In this formulation, the input data is vector-based, as multiple price-related features may contribute to accurate prediction. In this work, the features include the open and close price, total daily volume, and the minimum and maximum price for each day, which are considered essential for stock price prediction.

A deep neural network is employed in this study to model the function f. The network takes the historical features as input and predicts the stock price for the next day as its output.

3.2. Data Preprocessing

In this assignment, the Google Stock Price dataset[11], denoted as $\mathcal{D}$, was utilized. The dataset, originally provided on Kaggle, was split into a training dataset and a test dataset ($\mathcal{D}_{test}$). To further optimize model training and evaluation, the training dataset was divided into a training subset ($\mathcal{D}_{train}$) and a validation subset ($\mathcal{D}_{valid}$). The test dataset ($\mathcal{D}_{test}$) is reserved exclusively for final evaluation, while $\mathcal{D}_{train}$ is used for training the network and $\mathcal{D}_{valid}$ for hyperparameter tuning and model selection during the training process.

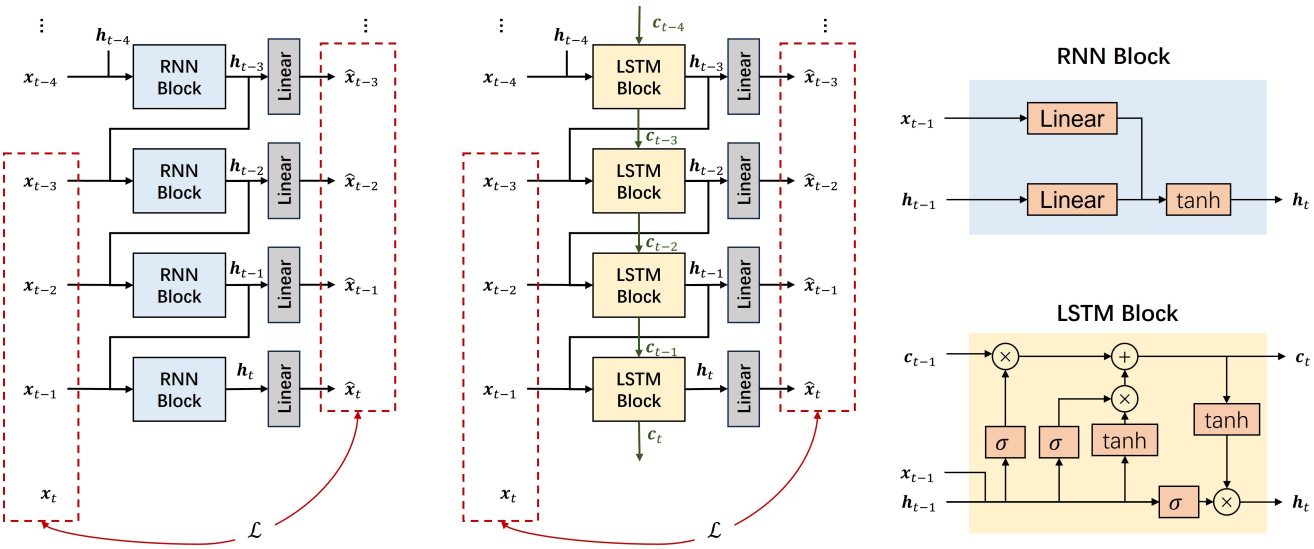

Figure 2: **Model architectures of vanilla RNN and LSTM.** The left diagram illustrates the vanilla RNN architecture, where each RNN block processes sequential data and outputs hidden states (h_t), followed by a linear layer to generate predictions ($\hat{x}_t$). The middle diagram depicts the LSTM architecture, where each LSTM block generates both hidden states (h_t) and cell states (c_t), enabling better handling of long-term dependencies. The right panels provide detailed structures of the RNN and LSTM blocks, showcasing the operations within each block, including linear transformations, activation functions (e.g., $tanh$), and gating mechanisms in the LSTM. The red dashed lines indicate the recurrent connections and loss computation ($\mathcal{L}$).

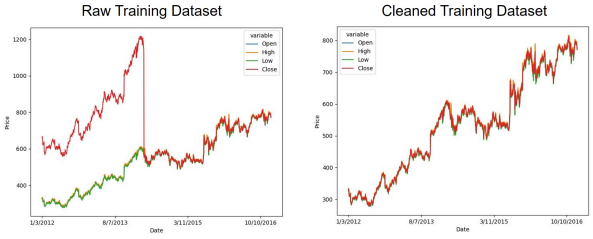

Figure 3: **Visualization of anomalies in the "Close" column and the cleaned dataset.** The left plot shows the raw training dataset, where an anomaly in the "Close" column is visible as a sharp spike. The right plot shows the cleaned dataset with anomalies in the "Close" column corrected.

During preprocessing, anomalies were detected in the "close" column of the original training dataset ($\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$). Specifically, there were instances where the "close" price was roughly double the "high" price of the same day (as shown in Figure. 3), which contradicted the expected relationship between these values. To address this issue and ensure the data's reliability, three preprocessing strategies were designed:

1. **Original Data:** The dataset was used as-is, including the anomalies in the "close" column, to train and validate the model.

2. **Column Removal:** The "close" column, identified as problematic, was entirely removed from the dataset. Models trained on this version of the data do not use the "close" price for training or prediction.

3. **Data Cleaning:** The erroneous "close" price values were adjusted by multiplying these values by 0.5 to recover the expected range and relationship between prices. The cleaned dataset was then used for training and validation.

Additionally, multiple normalization techniques were applied to scale the data appropriately, including the global min-max normalization, Windowed min-max normalization, Logarithmic transformation and log returns normalization. A detailed discussion of these normalization methods is provided in Sec.3.3.3.

3.3. Model Training

3.3.1 Model Architecture

This assignment evaluates two RNN-based architectures, vanilla RNN and Long Short-Term Memory (LSTM), for stock price prediction. Both models are designed to handle sequential data and have been widely applied in time-series forecasting tasks. The general framework of these architectures is shown in Fig. 2.

Vanilla RNN: [3] Vanilla RNN is one of the simplest architectures for sequential data modeling. It processes input data step by step, maintaining a hidden state that evolves over time to capture temporal dependencies. However, vanilla RNNs often struggle with the vanishing gradient problem, which limits their ability to learn long-term dependencies effectively.

LSTM: [6] LSTM, introduced by Hochreiter and Schmidhuber [6], was specifically designed to address the limitations of vanilla RNNs. By using gating mechanisms, LSTM can manage the flow of information and retain relevant features over extended sequences.

By comparing the performance of vanilla RNN and LSTM models, this study aims to explore their strengths and limitations under different preprocessing methods and time-series lengths.

3.3.2 Training Parameter Selection

Due to time constraints, a full parameter search for each network architecture was not conducted. Instead, the parameters were selected based on established practices in deep learning and prior experience with similar tasks.

Optimizer: The Adam optimizer was chosen for all networks. This optimizer provides consistent performance across various architectures, making it a suitable choice for training our network models.

Batch size: A batch size of 8 was applied across all networks. This relatively small batch size is appropriate for the limited size of the dataset. The small batch size also helps the model capture finer patterns in the data, enhancing their ability to utilize temporal dependencies.

Sequence length: The sequence length of 8 was selected for this study. The models use the past 7 days of stock prices as input to predict the latest day's price. This choice balances the need to capture sufficient temporal dependencies while avoiding excessive computational overhead and potential overfitting.

Learning rate: For the learning rate, a cyclic learning rate policy was adopted to identify the optimal initial learning rate. This approach progressively adjusts the learning rate during training, helping the models converge efficiently and avoid suboptimal local minima. The details of this learning rate schedule are outlined in Sec. 4.2.

Loss function. As stock price prediction involves quantitative response variables, the average L1 loss function was used as the training loss function. L1 loss, commonly applied in regression tasks, calculates the mean absolute error between the predicted values ($\hat{y}$) and the actual values (y). The loss function is defined as:

$$\mathcal{L} = \frac{1}{T-1} \sum_{i=t-T+2}^{t} |\mathbf{x} - \hat{\mathbf{x}}_i|_1, \tag{2}$$

where T represents the sequence length, $\mathbf{x}_i$ is the actual value, and $\hat{\mathbf{x}}_i$ is the predicted value at the same time.

By combining these parameter choices, this study aims to achieve a balance between training efficiency and predictive accuracy.

3.3.3 Further Explorations

To further enhance the performance of the networks, several modifications and experimental designs were implemented to investigate factors affecting model accuracy and generalization. These adjustments focused on exploring the effect of the length of historical data, different normalization approaches, and variations in the dataset preprocessing strategies, as defined in Sec.3.2.

The length of historical data was varied to determine its impact on capturing temporal dependencies. By including longer sequences of past stock prices, the networks were expected to leverage additional contextual information. However, increasing the sequence length also adds computational complexity and may lead to overfitting, especially in smaller datasets. This study systematically tested different sequence lengths to identify the optimal configuration for both vanilla RNN and LSTM models.

Four normalization techniques were evaluated for their impact on model performance in stock price prediction. Global min-max normalization scales data to a fixed range, typically $[0, 1]$, using the minimum and maximum values of the entire dataset[8]. Windowed min-max normalization calculates scaling parameters within a rolling window (e.g., 8 days) to capture local dynamics[5]. Logarithmic transformation reduces data variance and stabilizes time-series trends[7]. Finally, log returns normalization, defined as $r' = log(p_t) - log(p_{t-1})$, is a standard in finance for analyzing relative changes[4]. These approaches were compared to understand their influence on predictive accuracy and stability in financial forecasting tasks.

Through these experiments, this study evaluates the combined effects of data length, normalization techniques, and preprocessing methods on the predictive accuracy of vanilla RNN and LSTM models. These explorations and their impact on network performance are discussed in detail in Sec. 4.4.

3.4. Performance Evaluation

After completing the model training, performance is evaluated on the test dataset $\mathcal{D}_{test}$. To prevent data leakage, for all training parameters setting and improvements above, only the validation dataset $\mathcal{D}_{valid}$ and $\mathcal{D}_{train}$ are used.

4. Experiments

In this section, the experimental setup is detailed, and the effectiveness of the improvement methods is demonstrated.

Model	Dataset	Normalization	Open Avg.	High Avg.	Low Avg.	Close Avg.	Volume Avg.
RNN	Original	log-abs	7.816	7.983	9.716	10.499	**433,521**
		global-minmax	5.137	6.881	**5.666**	**5.518**	481,845
		win-minmax	9.585	14.433	9.460	13.679	841,374
		log-return	10.288	15.294	13.237	11.926	642,819
RNN	Removed	log-abs	19.717	18.436	19.677	-	474,418
		global-minmax	5.413	5.819	6.737	-	449,110
		win-minmax	15.032	18.819	12.702	-	1,050,123
		log-return	37.804	9.513	16.131	-	763,929
RNN	Clean	log-abs	15.080	14.079	15.972	15.821	484,041
		global-minmax	**4.481**	**5.038**	5.698	5.924	455,916
		win-minmax	11.599	15.237	10.362	13.956	652,226
		log-return	32.597	42.068	31.399	40.626	758,740
LSTM	Original	log-abs	4.893	6.539	5.469	5.882	401,255
		global-minmax	**3.934**	5.267	5.685	**5.277**	460,364
		win-minmax	20.726	27.112	17.492	27.652	1,689,386
		log-return	28.856	23.579	15.943	30.499	1,044,033
LSTM	Removed	log-abs	5.710	6.380	7.569	-	383,605
		global-minmax	4.351	5.381	5.943	-	442,935
		win-minmax	16.910	27.672	18.724	-	1,565,077
		log-return	21.909	10.978	23.474	-	1,006,653
LSTM	Clean	log-abs	4.853	7.825	**4.186**	6.479	398,492
		global-minmax	4.438	**4.921**	5.844	5.657	**357,543**
		win-minmax	21.290	29.632	18.410	23.693	1,743,166
		log-return	19.777	13.946	12.929	12.941	1,593,707

Table 1: **Experimental results of RNN and LSTM models.** The table presents the average absolute difference for each feature (Open, High, Low, Close, and Volume) under different normalization strategies (log-abs, global-minmax, win-minmax, log-return) and dataset types (Original, Removed, Clean). Bold values indicate the best performance for each model, and underlined values highlight the best performance for each dataset.

4.1. Experimental Details

The Google Stock Price dataset was sourced from Kaggle[11] and consists of 1258 days of stock price data for training and additional 20 days for testing. Each record contains six columns: Date, Open, High, Low, Close, and Volume.

In this assignment, the original training dataset was randomly split into $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$ with an 80:20 ratio for training and validation, ensuring sufficient data for both tasks. The test set, $\mathcal{D}_{test}$, was researved for final evaluation. This structured division enables a comprehensive assessment of the model's performance across training, validation, and testing phases.

All experiments were conducted on a GPU platform (RTX2080Ti), and the neural network implementation was carried out using PyTorch[10].

4.2. Learning Rate Setting

To determine an appropriate learning rate, a cyclical learning rate (CLR) [12] approach was employed. Starting with a small value (1×10^{-10}) and gradually increasing, this approach helps identify the optimal learning rate without the need for extensive grid searches. Figure 4 illustrates the learning rate schedules and corresponding training curves.

Using insights from the CLR experiment, an initial learning rate of 10^{-2} was chosen for all networks. This initial rate provided stable convergence while accommodating the varying depths and complexities of each network. For subsequent training, this learning rate will be decreased by half if there is no improvement in validation accuracy for 50 epochs. This reduction allows for fine-tuning as the models approached optimal convergence. By following this dynamic adjustment strategy, each model was effectively trained with an appropriate learning rate throughout the training process.

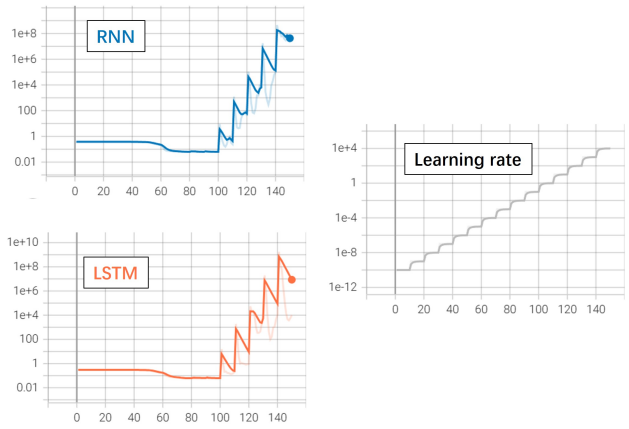

Figure 4: **Learning rate schedules and training curves for determining optimal starting learning rates.** The figure illustrates the cyclic learning rate (CLR) schedules and the corresponding training curves for RNN and LSTM models. The learning rate is dynamically adjusted over iterations, enabling efficient identification of optimal learning rates without extensive grid search.

4.3. Main Experiments

Model performance was evaluated using the average absolute difference on the test dataset for all columns: Open, High, Low, Close (except for the "Removed" dataset), and Volume. For comparison, two models—vanilla RNN and LSTM—were evaluated on three types of datasets: the original dataset (referred to as "Original"), the cleaned dataset ("Clean"), and the dataset with the "Close" column removed ("Removed"). The quantitative results under different normalization strategies are presented in Table 1, with the best performance for each model highlighted in bold and the best performance for each dataset underlined. Figure 5 shows the predicted values for each model on the test dataset.

Based on the results, both RNN and LSTM achieved similar accuracy in this stock prediction task, with LSTM generally outperforming vanilla RNN. Regarding dataset preprocessing, fixing the "Close" column led to slight improvements in most cases, while removing this column tended to decrease estimation accuracy.

4.4. Ablation Study

In this paper, two aspects of network training factors are further explored: normalization strategies and sequence length for network learning. Ablation experiments were conducted in this section to study these factors in detail. tion appears to better preserve the original data features, en-Firstly, experiments were conducted using different normalization strategies, as introduced in Sec.3.3.3. Four normalization methods were applied: global min-max normalization, which scales data to a fixed range ("global-minmax"); windowed min-max normalization, which computes scaling parameters within a sliding window ("win-minmax"); logarithmic transformation to reduce data variance ("log-abs"); and log returns normalization, focusing on analyzing relative changes ("log-return"). The experimental results are shown in Table 1. Interestingly, the simpler normalization methods ("global-minmax" and "log-abs") yielded better results. This may be attributed to the relatively small size of the training and testing datasets. With the dataset's fixed range, global min-max normalization appears to better preserve the original data features, enabling the network to more effectively learn local patterns.

Secondly, an ablation study was conducted on sequence length. For both RNN and LSTM, this experiment varied the sequence length used for training while keeping other parameters fixed. Figure 6 presents the average loss under different sequence lengths. As the sequence length increases, the networks achieve better performance, likely due to the ability to leverage longer historical data.

Interestingly, vanilla RNN and LSTM did not exhibit significant differences in performance across the experiments. One potential reason for this outcome is the relatively short sequence length used in this study, with a maximum of 20 days of historical data. Such a limited sequence length may not fully highlight the drawbacks of vanilla RNNs, such as their inability to capture long-term dependencies effectively, which typically become more pronounced with longer sequences. Additionally, the stock price dataset used in this work is relatively simple and structured, which might reduce the need for the advanced gating mechanisms of LSTMs to handle complex dependencies or noise. Further studies using longer sequence lengths or more complex datasets might better illustrate the differences between these architectures.

5. Code

The code for this project is available on my GitHub page (`https://github.com/MorainingErin/COMP_SCI_7318_DLF/tree/main/A3`). A readme file is also attached with the project for better understanding.

6. Conclusion

In this study, the capabilities of vanilla RNN and LSTM models were evaluated for stock price prediction using Google stock price data. Experiments highlighted the influence of two key factors: Sequence length and data preprocessing strategies. The results revealed that both vanilla RNN and LSTM could achieve reasonable performance, with LSTM slightly outperforming vanilla RNN. Among the data preprocessing approaches, fixing anomalies in the "Close" column yielded slight improvements, while remov-

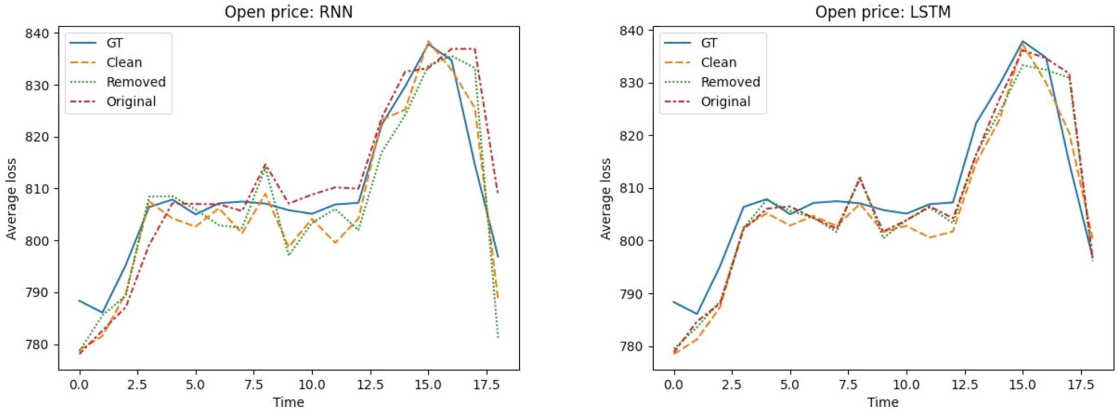

Figure 5: **Predicted values for RNN and LSTM models on the test dataset.** The plots compare the predicted "Open" prices generated by RNN (left) and LSTM (right) models across different datasets: Original, Clean, and Removed. The ground truth (GT) is also included for reference. The results highlight the models' ability to approximate the true values under varying preprocessing conditions.

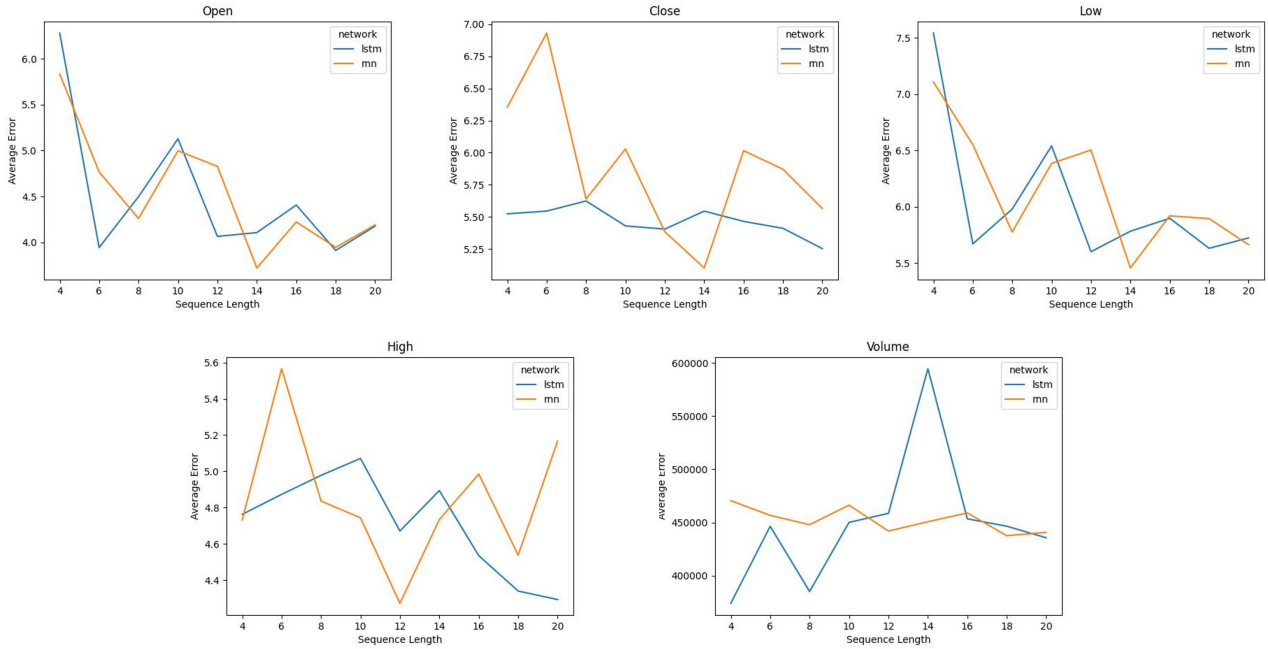

Figure 6: **Impact of sequence length on model performance.** The plots depict the average loss for each feature (Open, Close, Low, High, and Volume) under varying sequence lengths for LSTM and RNN models. The results indicate that increasing sequence length generally improves performance by enabling the models to leverage longer historical data for better predictions.

ing the column reduced prediction accuracy. Furthermore, global min-max normalization emerged as the most effective normalization method, likely due to the small size and fixed range of the dataset.

Limitations and Future Work: This research focused on a specific dataset and two basic model architectures, which may limit its generalizability to other financial datasets and tasks. The relatively small size of the dataset also constrained the exploration of more complex modeling techniques. Additionally, while the study evaluated multiple normalization and preprocessing strategies, it did not explore advanced methods such as feature engineering or

alternative loss functions tailored to financial forecasting.

In the future, this work could be expanded by testing on larger and more diverse datasets. Exploring other model architectures, such as GRUs or Transformer-based models, could also provide new insights. Additionally, experimenting with more advanced preprocessing methods might help improve prediction accuracy. To make the evaluation more practical, alternative metrics like directional accuracy or profit-based measures could be included, offering a broader understanding of the models' performance in real-world financial applications.

References

[1] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[2] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[3] Jeffrey L Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[4] Gregory Gundersen. Log returns: A simple introduction, 2022. Accessed: 2024-12-01.

[5] Vibhuti Gupta and Rattikorn Hewett. Adaptive normalization in streaming data. In *Proceedings of the 3rd International Conference on Big Data Research*, pages 12–17, 2019.

[6] Sepp Hochreiter and J"urgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[7] Machine Learning Mastery. How to apply log transformation to time series data, 2022. Accessed: 2024-12-01.

[8] Machine Learning Mastery. How to normalize and standardize time series data in python, 2022. Accessed: 2024-12-01.

[9] The University of Adelaide. Course outlines: COMP SCI 7318 - Deep learning fundamentals. `https://www.adelaide.edu.au/course-outlines/110026/1/sem-2/2020/`, 2024. [Accessed 01-10-2024].

[10] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

[11] Rahul Sah. Google stock price dataset, 2023. Accessed: 2024-12-01.

[12] Leslie N Smith. Cyclical learning rates for training neural networks. In *IEEE winter conference on applications of computer vision (WACV)*, pages 464–472. IEEE, 2017.