Stock Price Prediction Using Recurrent Neural Networks: Insights from Vanilla RNN, GRU, and LSTM Architectures

Xunyi Zhao
University of Adelaide
a1896243@adelaide.au

Abstract

The prospering integration of AI in financial decision-making has highlighted the need for sophisticated stock price prediction models. This study aims to improve forecasting accuracy by transitioning from traditional single-variable RNN models to advanced multivariate RNN variants, specifically vanilla RNN, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). Employing a dataset comprising 3,020 data points of AMAZON stock prices over 20 months, listed in NASDAQ market, we contrast the effectiveness of these models. Our analysis demonstrates that the multivariate GRU model markedly outperforms its vanilla RNN counterpart, achieving an 29.6% reduction in Root Mean Squared Error (RMSE) and a 66% decrease in model loss. These results underline the superior ability of the deep learning models to capture the complex interdependencies among multiple stock-related variables, suggesting that it offers more robust capabilities for accurate stock price forecasting. This paper underscores the potential of advanced AI techniques to enhance financial decision-making and adapt to the inherent volatility of stock markets, thereby offering substantial benefits to investors.

1. Introduction

A key component of financial research is stock price prediction, with precise forecasting being essential for influencing investment choices and controlling financial risks. However, because financial markets are so intricate and dynamic, forecasting stock market behavior is still a difficult undertaking. A number of factors, such as the state of the economy, business performance, investor attitude, firm location and political events, affect stock prices. The efficacy of conventional forecasting techniques is frequently called into question by the unpredictable interaction of various variables. Enormous benefits can stem from the ability to generate high accuracy financial information forecasts. Legislators, banks and investors can make informed decisions and optimize their portfolios for better returns thanks to precise stock predictions while venture capital fund can minimize their loss using good forecasts. Additionally, improved prediction models support economic stability by enabling better risk assessment. Long-established financial technical analysis and statistical models nowadays are failing to account for the nonlinear linkages and temporal dependencies present in the stock market. As an alternative, deep learning models—in particular, RNNs—are good at catching these patterns, thus better at time-series forecasting jobs. The ability of three RNN architectures—Vanilla RNNs, Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRU)—to forecast stock values is examined in this article. Besides, univariate and multivariate approaches will be utilized in the study in order to investigate the models' ability to learn complex correlations in stock data, thus enhance prediction performance. This study assesses the models' accuracy and error minimization using a stock dataset from AMAZON. The findings reveal valuable insights into the strengths and drawbacks of each RNN architecture, emphasizing the advantages of advanced deep learning techniques for addressing the unpredictable nature of financial markets. This research highlights the transformative role of AI-driven methodologies in financial forecasting, providing a foundation for more reliable and effective stock price prediction systems.

The study is organised as follows: the Literature Review section reviews earlier work in time-series forecasting while the Methodology portion describes the dataset, preprocessing procedures, and model architectures employed. Comparing model performances and presenting experimental results are done in the Implementation and Results section. The difficulties encountered and possible areas for development are covered in the Limitations and Future Work section. The study's main findings and their significance for stock price prediction are finally highlighted in the Conclusion.

2. Literature Review

Time series forecasting plays a critical role in various predictive applications, with stock price prediction being one of the most prominent. Early research in this area relied heavily on statistical methods, starting with the autoregressive (AR) model proposed in 1927. This model used past values of a time series to forecast future points through regression techniques. Building upon this concept, Autoregressive Integrated Moving Average (ARIMA) models were introduced in the 1970s by George Box and Gwilym Jenkins, becoming a standard approach in time series forecasting [1]. ARIMA models combine autoregression, differencing, and moving averages, with further adaptations like VRIMAX and SARIMAX addressing external predictors, multivariate data, and seasonal variations. Despite their solid performance, ARIMA models are resource-intensive, requiring intricate analysis and expert knowledge to select the right parameters.

As time progressed, machine learning methods gained traction, offering more automated and flexible alternatives for time series prediction. However, traditional machine learning models faced difficulties in capturing temporal relationships in data, often leading to issues like overfitting. This limitation made them less effective when compared to statistical approaches. In response, neural networks emerged, particularly Recurrent Neural Networks (RNNs), designed specifically to handle sequential data [6]. Although vanilla RNNs showed promise in capturing sequential patterns, they suffered from challenges like vanishing and exploding gradients, which restricted their ability to model long-range dependencies effectively.

To overcome these issues, Sepp Hochreiter and Jürgen Schmidhuber proposed the Long Short-Term Memory (LSTM) model in 1997 [5]. LSTMs introduced mechanisms to manage gradients more effectively, making it easier for the network to retain information over longer sequences. This innovation allowed LSTM networks to become a dominant approach in time series forecasting, particularly when long-term dependencies were involved. Following this success, Kyunghyun Cho and colleagues introduced the Gated Recurrent Unit (GRU) in 2014 [2]. The GRU is a simplified version of LSTM that reduces the number of parameters by eliminating the output gate. This reduction made GRUs more computationally efficient while maintaining similar predictive capabilities. Despite their improvements, both LSTMs and GRUs still face limitations when working with very long sequences, often requiring a fixed look-back window to maintain relevant information.

The introduction of the Transformer model by Google in 2017 marked a significant shift in how sequential data was processed [8]. Unlike RNNs, the Transformer relies on an attention mechanism, which allows the model to focus on important parts of the input sequence, enabling it to handle long-range dependencies more effectively. Although initially developed for natural language processing, Transformers have been adapted for time series forecasting as well, demonstrating their versatility. However, these models tend to be computationally expensive, particularly during the inference stage, and they still face challenges in capturing extremely long-term dependencies in time series data.

Recently, specialized models such as SOFTS [4], Tiny Time Mixers (TTMs) [3], and TSMamba [7] have emerged, offering significant advances in time series forecasting with greater efficiency. These models are designed to manage longer sequences while reducing computational overhead, making them suitable for large-scale applications. Nevertheless, this study focuses on evaluating the performance of three fundamental deep learning models—Vanilla RNN, LSTM, and GRU—in the context of stock price prediction. By comparing these classic models, this paper seeks to better understand their respective strengths and weaknesses in the domain of time series forecasting, specifically within financial prediction tasks.

3. Methodology

3.1. Data Preprocessing

Our dataset consists of seven columns: **date**, **open**, **high**, **low**, **close**, **volume**, and name. For our analysis, we exclude the name column, as it does not provide relevant information for the prediction task. The date column is set as the index, providing a temporal structure to the dataset. The dataset is complete, with no missing values, ensuring that all data points are available for model training.

The primary variable we aim to predict is the **close** price, as it reflects the final price of the stock at the end of the trading day, which is often considered the most significant data point for investors and analysts. Predicting the close price offers valuable insights into the market's overall direction and performance for that day.

Given the sensitivity of deep learning models to feature scaling, it was necessary to normalize the data to bring all variables within a consistent range. We chose to apply the MinMaxScaler, which transforms the data into a specified range, in our case [0, 1]. This scaling technique is particularly suitable for our dataset, as stock prices do not typically follow a standard distribution, making other scaling methods, such as StandardScaler, less ideal for this context.

For time series forecasting with RNN-based models, the data must be converted into sequences. We chose a look-back period of 20 days by default, meaning that each input sequence for the model will consist of the previous 20 days of stock data, used to predict the close price on the following day. In this study, we compare two approaches to sequence preparation: the univariate model, which uses

only the close price as the input feature, and the multivariate model, which incorporates all five features (open, high, low, close, volume) to predict the close price. Sequences will be generated for both types of models, allowing us to evaluate the performance of each approach.

After the sequences are prepared, we split the data into three sets: training, validation, and test, following a ratio of 80%, 10%, and 10%, respectively. The training set is used to train the models, the validation set is used for fine-tuning hyperparameters, and the test set is reserved for evaluating the final model's performance. The single-variate model will have an input dimension of 1 (using only the close price), while the multivariate model will have an input dimension of 5 (incorporating all available features).

3.2. RNN Architectures

Recurrent Neural Networks (RNNs) are designed to process sequential data by leveraging temporal dependencies, making them highly suitable for tasks such as time-series forecasting. Unlike conventional feedforward networks, RNNs incorporate feedback loops that allow them to retain information from previous time steps using a hidden state. This capability enables RNNs to model patterns and trends over time in both univariate data (a single variable over time) and multivariate data (multiple variables over time). Their versatility makes RNNs widely applicable in areas like stock price prediction, demand forecasting, and weather modeling, where short-term variations and long-term trends must be captured. Among the common RNN architectures, Vanilla RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU) provide varying levels of sophistication in handling sequential dependencies.

3.2.1 Vanilla RNN

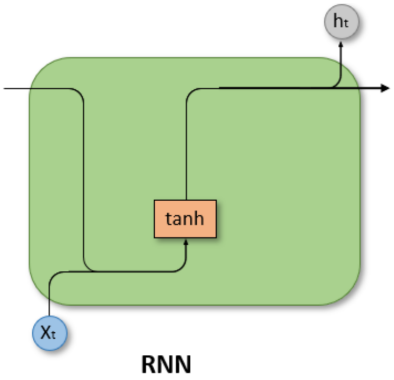

Figure 1. Vanilla Recurrent Neural Network

The **Vanilla RNN** represents the foundational RNN architecture, as we see in Figure 1 consisting of an input layer, a recurrent hidden layer, and an output layer. The hidden state h_t is updated at each time step based on the current input x_t and the previous hidden state h_{t-1}, as follows:

$$h_t = \tanh(W_h h_{t-1} + W_x x_t + b_h) \tag{1}$$

Here, W_h, W_x, and b_h are learnable parameters. Furthermore, the hidden size, which defines the dimensionality of h_t, dictates the layer's ability to capture and represent temporal patterns. The recurrent connections in the hidden layer enable the network to capture sequential dependencies, while the output layer translates the hidden state into predictions. While Vanilla RNNs can effectively learn short-term patterns, they struggle with capturing long-term dependencies due to the vanishing or exploding gradient problem. During backpropagation through time (BPTT), gradients are repeatedly multiplied by weight matrices over many time steps. This can lead to exponentially small gradients (vanishing gradients), making it difficult to learn long-range dependencies, or excessively large gradients (exploding gradients), which destabilize training. This inherent limitation in handling long sequences prompted the development of more advanced architectures like LSTMs and GRUs.

3.2.2 LSTM

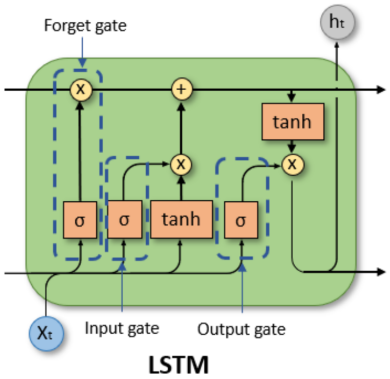

Figure 2. Long Short-Term Memory(LSTM)

The **Long Short-Term Memory (LSTM)** network improves on Vanilla RNNs by introducing memory cells and gating mechanisms to manage information flow. This architecture is designed to overcome the vanishing gradient problem and to capture long-term dependencies effectively. An LSTM uses a memory cell and three gates shown in Figure 2:

1. **Forget Gate**: Decides how much of the previous memory to retain:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2}$$

2. **Input Gate and Candidate Cell State**: Determines which new information to incorporate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4}$$

3. **Cell State Update**: Combines retained memory and new input:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{5}$$

4. **Output Gate and Hidden State**: Determines the output and the updated hidden state:

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

$$h_t = o_t \odot \tanh(C_t) \tag{7}$$

The gates and the memory cell adjust their states using the current input and the previous hidden state, guided by their unique weights and biases. In an LSTM, the hidden size directly influences the dimensions of the memory cell and gate vectors. These mechanisms allow LSTMs to effectively manage both short-term and long-term dependencies, making them suitable for complex time-series data. However, the increased complexity and computational demands of LSTMs have led to the development of more efficient alternatives like GRUs.

3.2.3 GRU

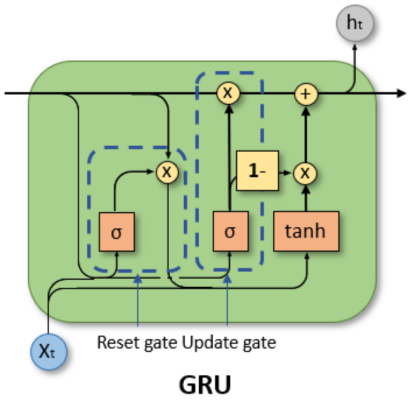

Figure 3. Gated Recurrent Unit (GRU)

The **Gated Recurrent Unit (GRU)** simplifies the LSTM by merging the forget and input gates into a single update gate and eliminating the separate memory cell. Instead, the GRU relies solely on the hidden state to manage information. The hidden size here defines the size of the hidden state and the gate vectors. The two key gates in a GRU shown in Figure 3 are:

1. **Update Gate**: Balances the retention of past information with the incorporation of new information:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{8}$$

2. **Reset Gate**: Controls how much of the previous hidden state is ignored:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{9}$$

The candidate hidden state is computed as:

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \tag{10}$$

Finally, the hidden state is updated using:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \tag{11}$$

By reducing the number of gates and parameters, GRUs are computationally more efficient than LSTMs while retaining the ability to model both short-term and long-term dependencies. This makes them a practical choice for time-series forecasting tasks, especially when computational resources are limited.

3.3. Optimizer, Loss Function, and Evaluation Metric

3.3.1 Optimizer

The Adam optimizer, short for Adaptive Moment Estimation, was used throughout this study with a fixed learning rate of 0.001. Adam combines the benefits of two optimization methods—momentum and RMSProp—making it a robust and efficient choice for training deep learning models. It dynamically adjusts the learning rate for each parameter based on estimates of the first and second moments of the gradients. The parameter updates are calculated as follows:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{12}$$

where: - θ_t represents the parameter at iteration t, - η is the learning rate, - $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moment estimates, - ϵ is a small constant for numerical stability.

Adam was chosen because it performs well across a wide range of tasks with minimal need for hyperparameter tuning, making it suitable for this study's experimental setup.

3.3.2 Loss Function

The Mean Squared Error (MSE) loss function was employed to measure the error between the predicted and actual values. MSE calculates the average squared differences, emphasizing larger errors more than smaller ones, which is particularly useful for regression tasks. The formula for MSE is:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{13}$$

where: - n is the number of samples, - y_i is the actual value, - $\hat{y}_i$ is the predicted value.

MSE was selected because it directly measures prediction accuracy in continuous tasks, such as stock price forecasting, and aligns well with the goal of minimizing prediction errors.

3.3.3 Evaluation Metric

The Root Mean Squared Error (RMSE) was used as the primary evaluation metric. RMSE is the square root of MSE, providing an interpretable measure in the same units as the original target variable. Its formula is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{14}$$

To compute RMSE, the predicted values were first reverse-transformed back to their original scale to ensure that the evaluation reflects real-world performance accurately. RMSE was chosen because it penalizes large errors and provides an intuitive measure of prediction accuracy, making it a reliable metric for assessing model performance in stock price prediction tasks.

4. Experiments and Insights

4.1. Experimental setup

To ensure the reproducibility of our results, we set the random seed with a fixed value of 42. This is important as it allows for consistent initialization of the model's weights, data shuffling, and random number generation, ensuring that experiments can be repeated under the same conditions.

We selected a batch size of 64, which is commonly used in practice for its balance between computational efficiency and stable gradient estimates. A larger batch size may lead to more accurate gradient estimates but can require more memory, while a smaller batch size might introduce more noise in the gradient calculations. Batch size 64 typically provides a good trade-off, offering sufficient stability and efficient training.

For the training data, we shuffle the dataset to ensure that the model does not learn any unintended patterns from the order of the data. Shuffling during training helps improve generalization by exposing the model to varied sequences in different orders. However, we refrain from shuffling the test and validation sets, as they should remain consistent and represent the actual distribution of data for final evaluation.

As discussed in the data preparation section, we prepare the data for both univariate and multivariate models. By default, the hidden size for all networks is set to 64, and the lookback period is set to 20 days, meaning that the model uses the past 20 days of data to predict the next day's closing price.

We define a training function to facilitate model training and evaluation, and a separate function to plot the results for later analysis. The training function incorporates model evaluation by reversing the transformation of scaled predictions back to the original target values. We then calculate the Root Mean Squared Error (RMSE) for performance assessment. The loss function used during training is Mean Squared Error (MSELoss), and the optimizer is Adam with a learning rate of 0.001. For each model, we train for a maximum of 100 epochs by default.

Finally, to evaluate model performance, we plot the predicted values and the actual values as line graphs. This allows for a visual comparison of how well the model's predictions align with the true stock prices.

4.2. Experiment and Results

4.2.1 Model Comparison: Multivariate vs. Univariate RNN, LSTM, and GRU

The performance comparison between the multivariate and univariate models across different architectures (RNN, LSTM, and GRU) is depicted in the Figure 4. In general, the multivariate models consistently outperform their univariate counterparts. Among the different models tested, the GRU model demonstrates the best performance in both the multivariate and univariate cases. On the other hand, in the univariate setting, the Vanilla RNN performs the worst, while in the multivariate case, the LSTM model shows the least effectiveness.

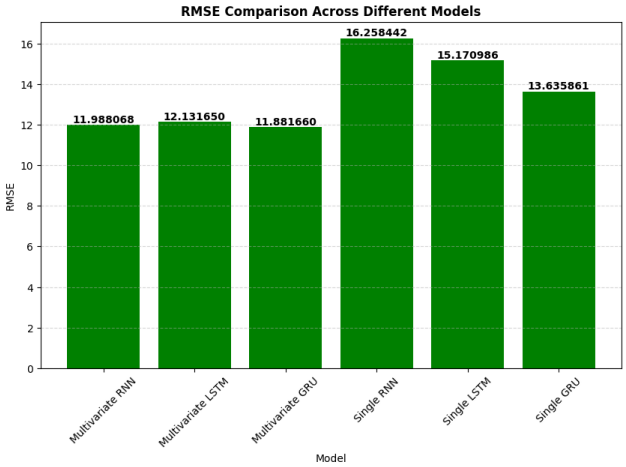

Figure 4. Model RMSE Comparison

Notably, the multivariate GRU model achieves nearly 30% lower RMSE than the single-variable Vanilla RNN. This significant improvement highlights the advantage of using multiple features (including the closing price) for pre-

diction, which allows the model to learn more comprehensive temporal dependencies.

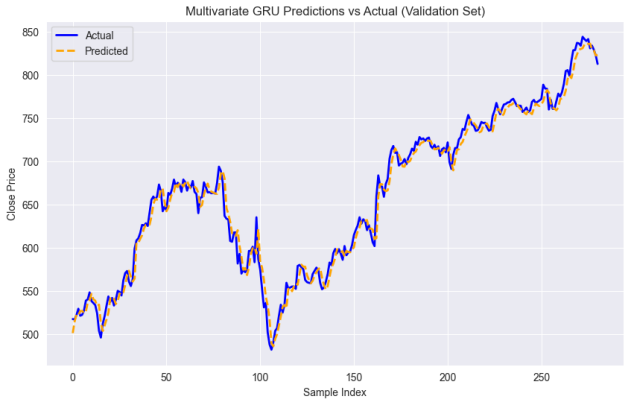

Figure 5. Multivariate GRU Performance on the Validation Data

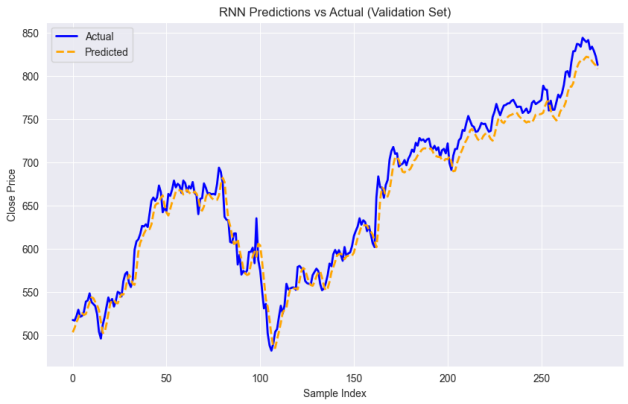

Figure 6. Univariate Vanilla RNN Performance on the Validation Data

Further examination of the prediction curves between the multivariate GRU and single-variable Vanilla RNN models reveals clear differences. In Figure 5 The predictions from the multivariate GRU are tightly aligned with the actual values, with minimal deviation. In contrast, in Figure 6 the Vanilla RNN shows a noticeable gap between predicted and actual values, especially towards the end of the prediction horizon. This visual difference emphasizes the superior predictive accuracy of the multivariate GRU model.

Based on these results, we selected multivariate GRU as the best-performing model for further optimization and hyperparameter tuning.

4.2.2 Hyperparameter Tuning

For hyperparameter tuning, we focused on two key parameters: the lookback period and the hidden dimension size. We experimented with three different lookback periods ([10, 20, 30]) and three hidden dimension sizes ([32, 64, 128]). Among the various combinations, the configuration with a lookback of 30 days and a hidden dimension of 128 yielded the best performance as we can see in Table 1, achieving an RMSE of 11.45, a slight improvement over the previous configuration with an RMSE of 11.88.

Lookback (Days)	Hidden Dimension	RMSE
10	32	11.92
10	64	14.71
10	128	12.52
20	32	13.72
20	64	11.88
20	128	11.49
30	32	11.71
30	64	17.52
30	128	**11.45**

Table 1. RMSE for different combinations of lookback period and hidden dimension.

4.2.3 Final Model and Performance on Test Data

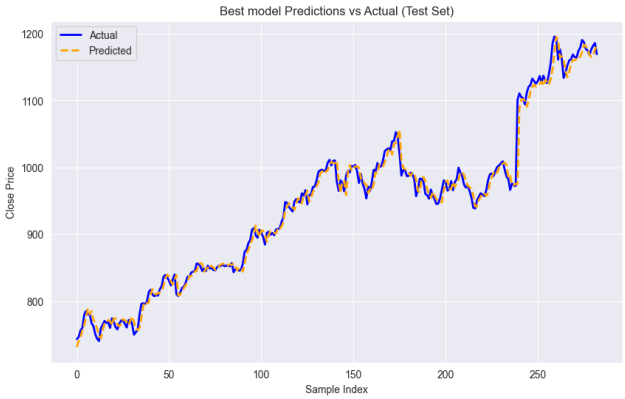

Figure 7. Best model performance on test data

After tuning the hyperparameters, we selected the multivariate GRU model with a lookback period of 30 days and a hidden dimension size of 128 as the final model. We evaluated the performance of this model on the test data, and as shown in the Figure 7, the model continued to perform exceptionally well, demonstrating strong generalizability. This confirms that the multivariate GRU model with the optimized hyperparameters is capable of providing accurate predictions on unseen data, validating its potential for real-world stock price forecasting.

4.3. Insights

4.3.1 Model performance Insights

From the model performance analysis, it is clear that multivariate models consistently outperform their univariate counterparts across all architectures. This is likely attributed to the inclusion of multiple features in the multivariate models, which enables them to capture more intricate relationships and dependencies within the data. Specifically, the Multivariate GRU model yields the lowest RMSE of 11.45, highlighting its effectiveness in capturing both short-term and long-term patterns. In contrast, the univariate models struggle more, with the GRU again performing better than the LSTM and RNN. The simplified structure of the GRU helps it avoid overfitting, making it better at handling temporal dependencies and leading to superior performance compared to the more complex LSTM and RNN models. The higher RMSE for the univariate models can be attributed to the absence of additional features, making it harder for these models to capture the full complexity of the data.

When comparing the architectures, GRUs emerge as the top performer in both the univariate and multivariate settings, followed by RNNs and LSTMs. The slightly higher RMSE observed with the LSTM models could be due to their complexity, which requires more careful tuning and a larger volume of data to reach their full potential. RNNs, while simpler, face challenges in learning long-term dependencies due to the vanishing gradient problem, resulting in relatively higher RMSE values compared to GRUs.

4.3.2 Hyperparameter Tuning Insights

The hyperparameter tuning process, which varied the look-back window and hidden dimension sizes, revealed that the best-performing configuration for the Multivariate GRU involved a look-back window of 30 days and a hidden dimension of 128. A 30-day look-back strikes a balance by capturing sufficient historical data without introducing excessive noise or overcomplicating the model. While a shorter look-back of 10 days might miss crucial long-term patterns, a 30-day window allows the model to better account for longer-term dependencies, leading to improved performance. On the other hand, reducing the look-back window to 20 days, while still optimal, results in a slightly higher RMSE of 11.71.

For the hidden dimensions, the model with 128 units achieves the best performance. Smaller hidden dimensions, such as 32 or 64, may lack the capacity to fully model the complex relationships present in multivariate data, leading to suboptimal predictions. However, increasing the hidden dimension size provides the necessary flexibility to model intricate patterns, although this must be done cautiously to avoid overfitting. Overall, the 30-day look-back with a hidden dimension of 128 represents the most effective balance between model complexity and predictive accuracy.

5. Limitations and Future Work

5.1. Limitations

This study is subject to several limitations that may have influenced the results. A primary limitation is the lack of extensive hyperparameter tuning. Due to time constraints, we selected the Multivariate GRU with its default settings as the best-performing model. While this model performed well, it is possible that other configurations, particularly for Multivariate RNN, with different hyperparameter settings, could lead to better performance. Therefore, the model choice made in this work may not be the optimal one, and further optimization could yield improvements.

Another limitation arises from our approach to feature selection. We did not engage in feature engineering, using all available features in their raw form for the multivariate model. This approach may not have maximized the potential of the data. It's likely that a more thoughtful selection or transformation of features could improve the model's predictive accuracy and provide a more refined understanding of the important factors driving stock price movements.

Additionally, the scope of model architectures explored was narrow. We focused on basic implementations of RNN, LSTM, and GRU, each with a single layer. While these models serve as a good starting point, they are relatively simplistic and may not capture the full complexity of the data. Future work could explore more advanced architectures, such as deeper networks or models with additional layers, to better capture intricate patterns and dependencies.

Lastly, the optimization process in this study was limited to the Adam optimizer with a fixed learning rate of 0.001. While Adam is a robust choice, its performance might be further improved with other optimizers, or by incorporating adaptive learning rate schedules. This limitation in optimization could have affected the model's convergence speed and overall performance.

5.2. Future Work

Looking ahead, a major area for improvement is the hyperparameter optimization process. A comprehensive search for the best hyperparameters across all models will allow for a more rigorous comparison and could potentially reveal configurations that outperform the models tested in this study. By fine-tuning these parameters, we may be able to improve model performance and gain deeper insights into the strengths of each approach.

Another key direction for future research is the exploration of more advanced RNN variants, such as multi-layer LSTM or bidirectional LSTM models. These more com-

plex architectures could offer better performance by capturing dependencies in both directions of the sequence, which could be especially valuable in time series forecasting tasks where context from both past and future data is important.

Moreover, exploring other optimizers with adaptive learning rates or advanced techniques like learning rate schedules could improve convergence and performance across all models. Testing a range of optimizers beyond Adam, such as SGD with momentum or RMSProp, could lead to better tuning and potentially enhanced model performance.

In addition, future work should investigate the impact of different feature subsets for the multivariate model. By carefully selecting the most relevant features, we could improve the model's performance while reducing the computational load. This could lead to a more targeted analysis of the factors that most strongly influence stock price predictions, resulting in a more streamlined and effective model.

Finally, modern time series forecasting methods like TSMamba should be explored. These newer models are specifically designed to handle long sequences more efficiently and could offer significant advantages over traditional RNN models in terms of both performance and computational efficiency. A comparative study could provide useful insights into the trade-offs between these newer approaches and the classical models.

6. Code

The GitHub link to the code can be found at https://github.com/billzhao1030/Deep-Learning-Assignment-3, which includes the dataset and all the experiment code in Python.

7. Conclusion

This study explores the application of Recurrent Neural Networks (RNNs), including Vanilla RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU), for stock price prediction using both univariate and multivariate approaches. The findings reveal that multivariate models consistently outperform univariate models, with the GRU emerging as the best-performing architecture. The multivariate GRU model demonstrates a 29.6% reduction in RMSE compared to the baseline Vanilla RNN, highlighting its superior ability to capture complex temporal dependencies and interrelationships among multiple stock-related features.

Hyperparameter tuning further enhances model performance, with the optimal configuration achieved using a 30-day lookback period and a hidden dimension size of 128, resulting in an RMSE of 11.45. These results underscore the importance of leveraging multivariate data and optimizing hyperparameters for accurate financial forecasting. The study also highlights the computational efficiency and effectiveness of GRUs in handling time-series data, making them a practical choice for stock price prediction tasks.

Despite its contributions, this research is subject to limitations, including limited hyperparameter tuning, lack of advanced feature engineering, and the use of basic RNN architectures. Future work should address these limitations by exploring more sophisticated models, advanced optimization techniques, and modern time-series forecasting methods such as TSMamba. Additionally, a deeper investigation into feature selection and engineering could further enhance prediction accuracy.

In conclusion, this study emphasizes the transformative potential of advanced deep learning techniques, particularly multivariate GRUs, in financial forecasting. By leveraging these models, investors and analysts can make more informed decisions, navigate market volatility, and optimize investment strategies, ultimately contributing to more robust and reliable stock price prediction systems.

References

[1] George Box and Gwilym Jenkins. Time series analysis: forecasting and control reaming. *San Francisco: Holden-Day*, 1970. 2

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

[3] Vijay Ekambaram, Arindam Jati, Pankaj Dayama, Sumanta Mukherjee, Nam H Nguyen, Wesley M Gifford, Chandra Reddy, and Jayant Kalagnanam. Tiny time mixers (ttms): Fast pre-trained models for enhanced zero/few-shot forecasting of multivariate time series, 2024. 2

[4] Lu Han, Xu-Yang Chen, Han-Jia Ye, and De-Chuan Zhan. Softs: Efficient multivariate time series forecasting with series-core fusion, 06 2024. 2

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1–42, 01 1997. 2

[6] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015. 2

[7] Haoyu Ma, Yushu Chen, Wenlai Zhao, Jinzhe Yang, Yingsheng Ji, Xinghua Xu, Xiaozhu Liu, Hao Jing, Shengzhuo Liu, and Guangwen Yang. A mamba foundation model for time series forecasting, 2024. 2

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 06 2017. 2