COMP SCI 7318 Deep Learning Fundamentals - Assignment 3

Lalitphan Sae-teoh (a1932456)

University of Adelaide

Abstract

One of the challenging tasks in financial analysis is stock price forecasting. This assignment focuses on implementing NVDA stock closing price forecasting, where NVDA is the stock name of NVIDIA, a world-leading artificial intelligence computing company. To predict complexity stock data, there are 3 models performed in this experiment, including Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM). The study is designed to train and evaluate these models' performance regarding predictive accuracy, robustness, and efficiency. The findings demonstrate which models deliver the most reliable forecast, providing practical guidance for future financial modelling applications.

1. Introduction

Many businesses and investors use stock price prediction for their financial analysis to make better decisions. However, it is challenging because stock prices are extremely volatile, non-linear, and sensitively influenced by external factors. Sequence-based models have shown significant performance since the rise of deep learning in tasks such as time-series prediction, especially those based on Recurrent Neural Networks (RNNs), Gated Recurrent Units (GRUs), and Long-Short-Term Memory (LSTM).

This assignment focuses on implementing and comparing these three models to forecast the closing prices of NVIDIA (NVDA) stock the next day. The practical application of this research is to evaluate their predictive accuracy and efficiency in a real stock market scenario while addressing the complexities of stock market data. The models are trained and tested on historical stock price data and separated into 2 time step configurations between 30 and 90 days. Their performance is analyzed using key metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and accuracy within a defined threshold.

This study investigates how each model handles the temporal dependencies and patterns in stock prices to identify the model that delivers the most reliable forecasts. The findings provide valuable insights for researchers and practitioners seeking effective methods for financial time-series prediction.

Figure 1. NASDAQ: NVDA (NVIDIA) 5 Years Historical Chart (Source: finance.yahoo.com)

1.1. Why NVIDIA Stock?

NVIDIA Corporation has been one of the most outstanding technology companies in recent years. It excels in graphics processing units (GPUs) and artificial intelligence (AI). Although it has been in the NASDAQ exchange market since 1999 with NVDA named, during these 5 years, its stock price has exponentially grown by almost 2,700%, shown in Fig. 1. Its growth can be attributed to several factors, especially in the era of artificial intelligence and machine learning. NVIDIA dominates the GPU market, particularly in gaming and professional applications. With the rise of AI, NVIDIA GPUs have become indispensable for training and deploying complex models. Demand drives revenue growth, and it is undeniable that NVDA stock is worth keeping an eye on for growth.

NVIDIA's stock is attractive for forecasting because of its dynamic character and applicability in rapidly growing industries. As a market leader in GPUs and AI technologies, NVIDIA's stock has fluctuated widely and gained momentum frequently due to new product releases, global tech trends, and general market conditions. However, several external factors, such as the pace of AI adoption, the worldwide demand for semiconductors, and economic con-

ditions affecting the tech sector, affect the company's success and make predicting more difficult. This volatility makes predicting its stock price both challenging and rewarding. NVIDIA is a perfect case study for using complicated time-series prediction models because of its high volatility, market sensitivity, and significance to cutting-edge technologies. It provides a great chance to investigate the potential and real-world difficulties of deep learning in financial analysis.

1.2. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) were introduced in the 1980s by researchers David Rumelhart, Geoffrey Hinton, and Ronald J. Williams [8]. Today, RNNs continue to impact AI research and applications, having established the groundwork for advanced sequential data processing techniques like time-series analysis and natural language.

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for sequential data, where the order of data points carries significant meaning. Unlike traditional feedforward neural networks, RNNs have a unique architecture that allows them to maintain a "memory" of previous inputs through internal states. It is achieved by looping connections in their hidden layers, enabling the model to retain information from earlier steps in the sequence while processing later ones [4].

RNNs are particularly well-suited for tasks involving time series data, natural language processing, and other applications requiring context awareness. For instance, stock price forecasting relies primarily on past prices to predict future value. RNNs' recurrent structure allows them to identify the trends, patterns, and temporal dependencies required for precise forecasting [5].

Despite their strengths, RNNs face challenges like the vanishing gradient problem, which occurs during backpropagation through time (BPTT). This issue makes it difficult for the network to learn long-term dependencies in sequences as gradients diminish or explode. Because of these limitations, more advanced architectures like Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks have been developed, incorporating gating mechanisms to capture long-range dependencies better.

This study uses RNNs as a baseline model to compare how well more complex models like GRU and LSTM perform. Recognising RNN advantages and disadvantages provides significant background information for evaluating how well these architectures manage the challenges of stock price prediction.

1.2.1 RNNs Architecture

Recurrent Neural Networks (RNNs) are designed to process sequential data, allowing it to model temporal dependencies. The main idea behind RNNs is the capacity to retain information from previous inputs through recurrent connections. This memory mechanism is achieved by feeding the hidden state from one time step back into the network for the next time step, creating a chain-like structure of repeating modules. Fig. 2 [6]

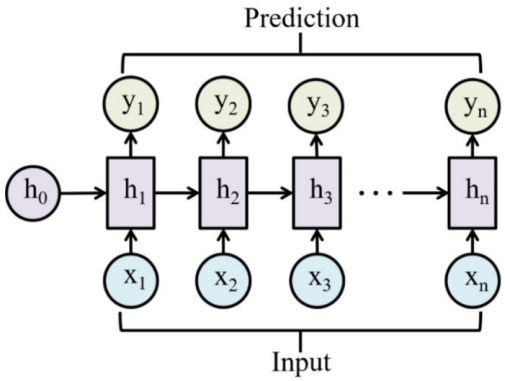

Figure 2. Basic RNNs Architecture [6]

Input Layer (x_t): Accepts sequential data, where each time step corresponds to a feature vector.

Hidden Layer (h_t): Computes the hidden state h_t at each time step using the input x_t and the previous hidden state h_{t-1}:

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

- W, U, b: learnable parameters
- f: an activation function, typically a tanh or ReLU

Output Layer (y_t): Produces the output y_t at each time step, which can be used for predictions. The output is often a single value per time step for regression tasks:

$$y_t = g(V \cdot h_t + c)$$

- V, c: additional learnable parameters
- g: an activation function suited to the task

1.3. Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a type of recurrent neural networks (RNNs) architecture introduced to handle the drawbacks of traditional RNNs, particularly the vanishing gradient problem and difficulty in learning long-term dependencies. GRUs use gating mechanisms to regulate the flow of information, making them more efficient and capable of capturing both short-term and long-term dependencies in sequential data [2].

Unlike Long Short-Term Memory (LSTM) networks, GRUs have a more straightforward structure with fewer parameters. This simplicity often leads to faster training while maintaining comparable performance, making GRUs especially popular for time-series prediction tasks, including stock price forecasting.

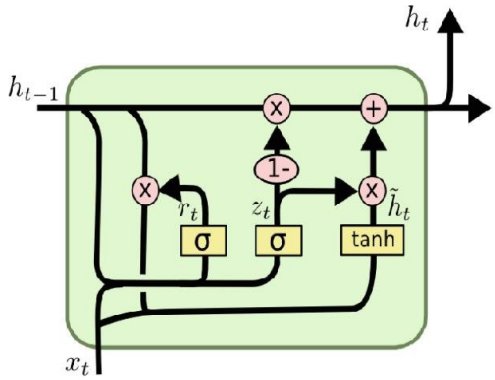

Figure 3. GRU Architecture [7]

1.3.1 GRU Architecture

GRUs improve upon traditional RNNs by introducing two gating mechanisms: Reset Gate and Update Gate. The following are the components of GRU architecture Fig. 3 [7]:

Reset Gate (r_t): Determines how much of the past information should be forgotten at a given time step. It helps the model reset its memory when necessary:

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

Update Gate (z_t): Controls how much of the past information to carry forward and how much new information to incorporate:

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

Candidate Activation ($\tilde{h}_t$): Calculates the candidate memory content using the reset gate:

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

Final Hidden State (h_t): Combines the previous hidden state and the candidate activation based on the update gate:

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

1.4. Long-Short Term Memory (LSTM)

Long short-term memory (LSTM) networks are a specialized form of recurrent neural networks (RNNs). They were introduced to solve the vanishing gradient issue that prevents RNNs from recognizing long-term dependencies. LSTMs can effectively capture both short-term and long-term temporal relationships in sequential data. [3]

To accomplish this, LSTMs integrate a memory cell and several gating mechanisms controlling the network's information flow. Because of their architecture, LSTMs are perfect for time-series forecasting, natural language processing, and other applications involving sequential data. Their architecture allows them to retain relevant data across several steps while eliminating irrelevant elements.

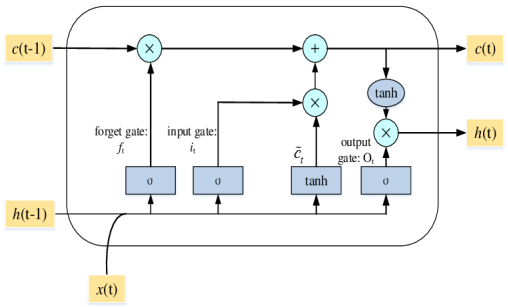

Figure 4. LSTM Architecture [1]

1.4.1 LSTM Architecture

LSTMs extend the traditional RNNs architecture by introducing a cell state and three gates (Forget Gate, Input Gate and Output Gate): Fig. 4 [1]

Forget Gate (f_t): Determines how much information from the previous cell state (C_{t-1}) should be discarded:

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

- σ: the sigmoid activation function
- W_f, b_f: learnable parameters
- h_{t-1}: the previous hidden state
- x_t: the current input

Input Gate (i_t): Controls how much new information is added to the cell state:

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

Cell State Update (C_t): The cell state is updated using a combination of the previous state and the new candidate values:

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

- ($\tilde{C}_t$): candidate value

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

Output Gate (O_t): Determines the output hidden state (h_t) by filtering the updated cell state:

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

$$h_t = o_t \odot tanh(C_t)$$

2. Code

The implementation code for this assignment can be found as follows source-codes.

According to the experiments performed in this assignment, there are 2 notebooks as follows:

1. 30-day Time Step Config Models
2. 90-day Time Step Config Models

Column Name	Definition
Date	Trading date
Open	Open price of trading day
High	Highest price of trading day
Low	Lowest price of trading day
Close	Close price of trading day
Volume	Total number of shares traded during day

Table 1. NVDA Stock Data Definition

Column	Mean	STD	Min	Max
Open	19.85	30.85	0.48	149.35
High	20.21	31.40	0.49	152.89
Low	19.45	30.20	0.47	146.26
Close	19.85	30.83	0.48	148.88
Volume	4.67e+08	2.53e+08	4.56e+07	3.69e+09

Table 2. NVDA Statistical Information

3. Model Implementation

This section explains the process for training and evaluating the RNN model as a baseline model: data preprocessing, model training, and evaluation process.

3.1. Data Preprocess

All 3 models use the same dataset and data preprocessing in the study.

3.1.1 Dataset

This assignment uses NVIDIA company's daily stock price to forecast, named NVDA from the NASDAQ exchange. The data is downloaded from the Nasdaq official website NVDA Historical Data. There are 6 columns in the dataset, which are Date, Open, High, Low, Close, and Volume Tab. 1, with a total of 2,516 records. It is 10 years of historical data, starting from 05-Dec-2014 to 04-Dec-2024. The statistical information of the data can be found in this table Tab. 2.

3.1.2 Feature Scaling

The input features were scaled before splitting using Min-Max normalization to transform them between 0 and 1. It is commonly used in machine learning and deep learning tasks, including time-series data like stock price forecasting, to ensure efficient training and model performance.

The formula for Min-Max Normalization is:

$$x^{'} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Parameters	RNN	GRU	LSTM
Units	50	50	50
Activation Function	tanh	tanh	tanh
Epoch	20	20	20
Batch Size	32	32	32
Learning Rate	0.001	0.001	0.001
Optimizer	Adam	Adam	Adam

Table 3. Baseline Model Parameters Settings

- $x^{'}$: Normalized value in the range [0,1]
- x: Original value
- x_{min}: Minimum value in the dataset
- x_{max}: Maximum value in the dataset

3.1.3 Data Splitting

As the objective of this assignment is to predict the next day's stock closing price, the column "Close" is the target data and feature. There are 2 sets of features prepared to experiment with different time step configurations by dividing sequentially into 80% of training and 20% of testing set:

1. **30-days time step** (including Training: 1,988 records, Testing: 498 records)
2. **90-days time step** (including Training: 1,940 records, Testing: 486 records)

3.2. RNN Model Training

After preprocessing the data, the RNN baseline model was implemented. The goal was to train the model to predict the next day's closing price. The RNN is set with 50 units, the number of hidden states in each network layer, and an activation function of tanh. The training was repeated for 20 epoch iterations with 32 batch sizes, a default learning rate of 0.001 and an Adam optimizer. Tab. 3

3.3. Evaluation

As stock price prediction is a regression task, the metrics for the evaluation process are Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) to quantify the loss between predicted and actual values. Accuracy is used to measure how well the model is predicted.

3.3.1 MSE

MSE calculates the average of the squared differences between predicted values ($\hat{y}$) and actual values (y):

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

Metrics	RNN 30-day	RNN 90-day	%Diff
MSE	0.0031	0.0016	-49%
MAE	0.0355	0.0259	-27%
RMSE	0.0554	0.0400	-28%
Accuracy	52.81%	62.76%	+10%

Table 4. RNN Model Testing Result

3.3.2 MAE

MAE measures the average magnitude of errors, regardless of their direction, by taking the absolute differences between predicted and actual values:

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

3.3.3 RMSE

RMSE is the square root of the MSE, combining the benefits of MSE's sensitivity to large errors with a more interpretable unit:

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

3.3.4 Accuracy

The accuracy is calculated as the ratio of correctly predicted samples to the total number of samples in the test set:

$$Accuracy = \frac{Numberofcorrectpredictions}{Totalnumberofpredictions}$$

3.3.5 RNN Model Result

During training, validation is set to 20%. There are two sets of results, one for the 30-day time step and one for the 90-day time step. Both summaries of the testing results of the RNN baseline model can be found in Tab. 4. The result shows that RNN with the 90-day time step configuration has lower loss and better accuracy with 62.76%, which is 10% higher than the 30-day time step configuration.

4. Experiments and Analysis

There are 2 models performed to compare with RNN baseline model in this assignment:

1. Gated Recurrent Unit (GRU)
2. Long-Short Term Memory (LSTM)

Metrics	GRU 30-day	GRU 90-day	%Diff
MSE	0.0006	0.0006	Slightly
MAE	0.0159	0.0160	Slightly
RMSE	0.0239	0.0241	Slightly
Accuracy	77.11%	79.22%	+2%

Table 5. GRU Model Testing Result

Metrics	LSTM 30-day	LSTM 90-day	%Diff
MSE	0.0024	0.0009	-62%
MAE	0.0333	0.0202	-39%
RMSE	0.0490	0.0300	-39%
Accuracy	44.38%	63.58%	+19%

Table 6. LSTM Model Testing Result

4.1. GRU Model Result

The GRU model is implemented with the same approach as the RNN baseline model in terms of the goal to predict the next day's closing price, validation set splitting, time step configuration (30-day and 90-day) and model parameter settings (Tab. 3).

There are two sets of results, one for the 30-day time step and one for the 90-day time step. Both summaries of the testing results of the GRU model can be found in Tab. 5. The result shows that GRU, between two time step configurations, has almost the same loss, and the accuracy of the 90-day time step is 79.22%, which is higher by 2% compared to the 30-day time step.

4.2. LSTM Model Result

The LSTM model is implemented with the same approach as the RNN baseline model in terms of the goal to predict the next day's closing price, validation set splitting, time step configuration (30-day and 90-day) and model parameter settings (Tab. 3).

There are two sets of results, one for the 30-day time step and one for the 90-day time step. Both summaries of the testing results of the LSTM model can be found in Tab. 6. The result shows that LSTM has a large difference in loss where the 90-day time step is lower, which means it has a better prediction. It reflects the accuracy of the 90-day time step significantly improves from the 30-day time step from 44.37% to 63.58%, which is almost 20% higher.

4.3. Models Comparison Analysis

4.3.1 30-day Time Step Model Performance Evaluation

The chart Fig. 5 is visualized comparing each model prediction and actual with 30-day time step, how well each

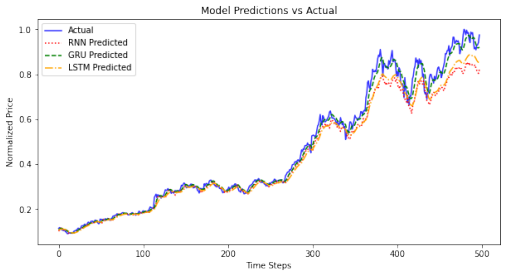

Figure 5. 30-day Time Step Test Prediction and Actual Chart

Metrics	RNN	GRU	LSTM
MSE	0.0031	0.0006	0.0024
MAE	0.0355	0.0159	0.0333
RMSE	0.0553	0.0024	0.0490
Accuracy	52.81%	77.11%	44.38%

Table 7. 30-day Time Step Test Result

model predicted. The 30-day time step result between the three models found that the GRU model significantly outperformed the RNN and LSTM models across all metrics Tab. 7.

The GRU MSE of 0.0006 and RMSE of 0.0024 show precision in predictions, while the MAE of 0.0159 highlights its ability to minimize absolute errors. The GRU accuracy of 77.11% also shows its effectiveness in capturing complex patterns in stock price data, benefiting from its efficient gating mechanisms and reduced parameter count compared to LSTM.

At the same time, the RNN model demonstrated moderate performance, with an RMSE of 0.0553 and an accuracy of 52.81%. While it managed to capture basic patterns in the time-series data, its relatively high MSE and MAE indicate the limitation of traditional RNN due to the vanishing gradient problem in learning long-term dependencies.

In comparison, the LSTM model showed mixed performance. While its RMSE of 0.0490 and MAE of 0.0333 were slightly better than the RNN baseline model, its MSE and accuracy lagged behind GRU's performance. Even though the LSTM model in this experiment was intended to manage long-term dependencies more effectively than RNNs, it might have had inadequate tuning or data volume, which resulted in less than ideal accuracy of 44.38%.

4.3.2 Different Time Steps Model Performance Evaluation

The accuracy performance comparison of RNN, GRU, and LSTM models using two different time steps, the 30-day and 90-day time steps, can be found in Tab. 8.

LSTM demonstrated the most noticeable improvement with the longer time step Tab. 6. MSE decreased dramatically by approximately 62%, and MAE reduced by 39%, indicating an increase in predictive accuracy. The RMSE decreased significantly, demonstrating the model's improved capacity to reduce deviations. Because of its ability to capture long-term dependencies, the LSTM profited greatly from the longer sequential input, as evidenced by the over 20% improvement in accuracy.

Time Step	RNN	GRU	LSTM
30-day	52.81%	77.11%	44.38%
90-day	62.76%	79.22%	63.58%

Table 8. Different Time Step Accuracy Comparison

All metrics of RNN significantly improved with an increased time step Tab. 4. Better predictive performance was demonstrated by the MSE, MAE, and RMSE decreasing by roughly 49%, 27%, and 28%, respectively. Additionally, a significant improvement of over 10% in accuracy indicates that the RNN was able to capture more temporal dependencies with a longer time step.

However, GRU performance stays consistent with the longer time step compared to the other two models Tab. 5. MSE, MAE, and RMSE of GRU show a slight change, almost the same as the 30-day time step. There is a marginal improvement of around 2% in accuracy, indicating that the model was already successfully capturing most temporal patterns with the original configuration.

4.4. Hyperparameter Tuning Experiments

This section shows the performance of the hyperparameter tuning, using the best-performed GRU model, to predict the NVIDIA stock closing price the next day in the previous experiment. In this assignment, hyperparameter tuning is performed using a cross-validation technique in a random search that divides the dataset into k folds and trains the model on k-1 folds while testing the remaining fold to find the best combination of sample hyperparameters. There are 3 hyperparameters in the experiment to search, which are units or hidden states [16, 32, 50, 64, 128], learning rate [0.01, 0.001, 0.0001] and batch size [16, 32, 64].

In 10 trials of random searching for the 30-day time step, the best parameters were 64 units, a 0.001 learning rate, and a 32-batch size. After applying the hyperparameter tuning, the results improved and are shown in Tab. 9. All error metrics decreased, with an MSE of 22.3%, indicating more precise predictions. The accuracy increased by 4.42%, reflecting the model's enhanced ability to capture patterns in the data.

In 10 trials of random searching for the 90-day time step, the best parameters were 64 units, a 0.01 learning rate, and a 16-batch size. After applying the hyperparameter tuning,

Metrics	Baseline	Tuned	%Diff
MSE	0.0006	0.0004	-22.3%
MAE	0.0159	0.0138	-12.8%
RMSE	0.0239	0.0210	-11.9%
Accuracy	77.11%	81.53%	+4.42%

Table 9. 30-day Time Step GRU Tuned Performance

Metrics	Baseline	Tuned	%Diff
MSE	0.0006	0.0003	-49.3%
MAE	0.0160	0.0111	-30.6%
RMSE	0.0241	0.0172	-28.8%
Accuracy	79.22%	91.36%	+12.14%

Table 10. 90-day Time Step GRU Tuned Performance

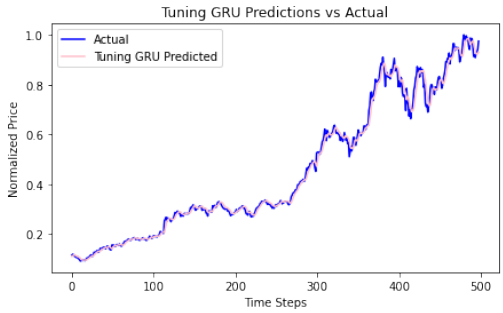

Figure 6. 90-day Time Step GRU Tuned Prediction and Actual

it dramatically improved, and the result is shown in Tab. 10. MSE decreased by 49.3%, and MAE and RMSE also saw significant reductions of 30.6% and 28.8%, respectively. The accuracy jumped by an impressive 12.14%, showing that the tuned GRU model with a 90-day time step outperformed all other configurations Fig. 6.

5. Summary

This assignment focused on predicting NVIDIA's stock closing price using three deep learning architectures: RNN, GRU, and LSTM. The objective was to compare the model performance using key evaluation metrics, including Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Accuracy. Additionally, the study examined the impact of hyperparameter tuning and different time-step configurations (30-day and 90-day) to identify the most efficient model and configuration. The baseline evaluation showed that GRU consistently outperformed RNN and LSTM, achieving lower error rates and higher accuracy. Increasing the time step from 30 to 90 days generally improved performance across all models, with LSTM demonstrating the most noticeable gains.

Hyperparameter tuning significantly enhanced the models' predictive capabilities, with the tuned GRU model using a 90-day time step achieving the best results. In comparison to its baseline, the tuned GRU increased accuracy by 12.14% and decreased MSE by over 50%. With the lowest error rates and the maximum accuracy of 91.36%, this demonstrated how well GRU captures complex temporal dependencies while maintaining computational efficiency.

The findings suggest that GRU is an optimal model for NVIDIA stock price forecasting due to its ability to balance accuracy and computational efficiency. The study underscores the importance of hyperparameter tuning and the choice of time step in enhancing model performance. The longer 90-day time step provided additional context, enabling the models to capture long-term dependencies more effectively.

There are room to improve, the future work could explore the external factors such as market sentiment, macroeconomic indicators, or sector performance to improve prediction accuracy. Furthermore, hybrid designs that combine attention mechanisms with GRU or LSTM may improve long-term dependency modelling. Lastly, testing out with Transformers may offer important information on how well they work for stock price forecasting jobs.

In conclusion, this study highlights the effectiveness of RNN models, particularly GRU, for financial time-series forecasting. By leveraging hyperparameter tuning and longer time-step configurations, predictions can be significantly enhanced, paving the way for practical applications in stock market analysis.

References

[1] Faisal Butt, Lal Hussain, Anzar Mahmood, and Kashif Lone. Artificial intelligence based accurately load forecasting system to forecast short and medium-term load demands. *Mathematical Biosciences and Engineering*, 18:400–425, 2021. 3

[2] Kyunghyun Cho, Bart 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] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 1997. 3

[4] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. The MIT Press, 2016. 2

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

[6] Ibomoiye Domor Mienye, Theo G Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information (Basel)*, 15 (9):517, 2024. 2

[7] Adnan Riaz, Muhammad Nabeel, Mehak Khan, and Huma Jamil. Sbag: A hybrid deep learning model for large scale traffic speed prediction. *International Journal of Advanced Computer Science and Applications*, 11:287–291, 2020. 3

[8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 2