# Recurrent Neural Networks for Stock Price Prediction

Simon B Siagian
The University of Adelaide
simonbaharja.siagian@student.adelaide.edu.au

## Abstract

*This study evaluates the effectiveness of Recurrent Neural Networks (RNNs) in predicting stock prices using historical data from Apple Inc. (AAPL) between 2010 and 2023. Three RNN variants—Vanilla RNN, Long Short-Term Memory (LSTM), and Gated Recurrent Unit (GRU)—were analyzed for their predictive performance and ability to capture temporal dependencies. Results indicate that GRU achieved the best validation performance with a Mean Squared Error (MSE) of 7.84, outperforming LSTM (7.86) and Vanilla RNN (8.1) under optimal configurations.*

## 1 Introduction

Total market capitalization has experienced a significant increase in the last four decades. Starting from around USD 2.5 trillion in the 1980s, it reached over USD 70 trillion at the end of the 2010s [1]. Following this significant increase, today researchers have focused on financial forecasting using artificial intelligence and machine learning. This is done by treating financial or stock price data as time series data to identify patterns, periods, and the cycle trend within them, also known as time series forecasting [2].

According to [2], there are two types of time series algorithms: linear and nonlinear. The difference is that linear models use predefined mathematical equations to fit a model to a univariate time series. In contrast, non-linear models such as deep learning use the historical data to fit the model and try to identify the hidden and underlying patterns through the training process. The disadvantage of the linear model is that it can only consider univariate time series data without identifying the dependencies among various stocks. This problem is then taken into account by deep learning algorithm by learning the dependencies among the data.

In recent years, researchers have used various deep learning algorithms for time series prediction tasks. Algorithms such as CNN, RNN, GRU, and LSTM have been evaluated to predict the trend of time series data, especially stock price data. This helps investors make decisions about a particular stock. In this study, we evaluate the performance of a Recurrent Neural Network (RNN) for stock price prediction using historical data.

## 2 Related Works

Several traditional and deep learning methods have been evaluated to predict the stock price trend. Before the development of deep learning algorithms, methods such as Autoregressive Integrated Moving Average (ARIMA) [3] were used to predict time series data with an assumption that a linear correlation exists among the data. However, this algorithm struggled to capture non-linear relationships [4]. Researchers then attempted to use non-linear models for prediction; the initial model included a neural network and support vector machine [5]. Zhang [6] compared the performance of ARIMA and neural networks for stock price prediction. The result showed that neural networks outperform ARIMA in nonlinear data prediction. However, the accuracy of the model needs to be improved.

Hsieh et al. [7] used a recurrent neural network to predict the price index of several stocks. However, it could not solve the problem of time dependencies, which grows significantly with larger datasets. To overcome this problem, LSTM and GRU are introduced. Nelson et al. [8] assess the effectiveness of the LSTM network in stock price prediction by evaluating a series of multiple LSTM networks. They successfully used LSTM network to predict Ping An Bank's closing price. In further development, the GRU network is used for stock market prediction. In 2018, Dang et al. [9] evaluated a GRU-based model for financial analysis and successfully applied it to S&P500 stock price forecasting. Chen et al. [10] proposed a GRU application for predicting 10 different stocks in the same country, and the result concludes that GRU models could predict with the lowest MAE of 15.

# 3 Method

## 3.1 Dataset

This study used a historical stock price dataset retrieved from the Yahoo Finance API [11]. The dataset covers the stock price data of Apple Inc. (AAPL) from January 2010 to December 2023. The price unit in this dataset is US Dollar. The dataset consist of the following columns [2]:

1. **Date**: The trading date for the corresponding stock price data. It serves as the primary temporal reference for the dataset.
2. **Open**: The opening price of Apple's stock on a given trading day, reflecting the first trade price at the market open.
3. **High**: The highest price Apple's stock reached during the trading day.
4. **Low**: The lowest price Apple's stock fell to during the trading day.
5. **Close**: The closing price of the stock, representing the final trade price at the end of the trading session.
6. **Volume**: The total number of shares traded during the trading day, offering insights into market activity and investor interest.
7. **Adjusted Close**: The closing price adjusted for stock splits, dividends, and other corporate actions. This provides a more accurate representation of the stock's value over time for analysis purposes.

## 3.2 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a type of artificial neural network designed to process sequential data such as speech or time series data [12]. This study will evaluate three RNN models for stock price prediction: Simple or Vanilla RNN, Long Short-term Memory (LSTM), and Gated Recurrent Unit (GRU).

### 3.2.1 Vanilla Recurrent Neural Network

Vanilla RNN is the most basic model of RNN. The architecture of simple RNN is shown in Figure 1. The equation for vanilla RNN can be described as :

$$h_t = f_W(h_{t-1}, x_t) \tag{1}$$

$$y_t = W_{hy} h_t \tag{2}$$

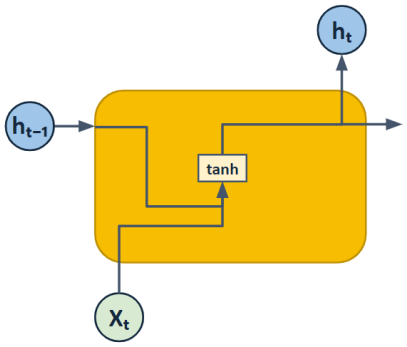


Figure 1: Architecture of Vanilla RNN

where $x_t$ is input at time t, $h_t$ is hidden state at time t, $W_{hy}$ is weight matrix, and $f_W$ is nonlinear activation function. Normally, *tanh* activation function is applied in the hidden state and output layer [Saud].

However, vanilla RNNs are prone to vanishing and exploding gradient problems due to their sequential nature and reliance on backpropagation through time (BPTT) to update weights. During this process, gradients are repeatedly multiplied by the same weight matrices as they propagate backwards through the time steps. If the weights are small, the gradients shrink exponentially, leading to the vanishing gradient problem, which hinders learning long-term dependencies [13]. Conversely, if the weights are large, the gradients grow exponentially, causing the exploding gradient problem, which destabilizes training. This sensitivity to weight scaling makes vanilla RNNs struggle with capturing long-range patterns in data [14].

### 3.2.2 Long Short Term Memory (LSTM)

LSTM is a variant of RNN that can capture long-term dependencies. Compared to vanilla RNN architecture, LSTM has a 'gate' structure. This gate structure is designed to overcome the vanishing/exploding gradient problem [4]. LSTM consists of three gates:

1. Input gate: Receives the previous information and retains the information in cell state.
2. Forget gate: Decide what information can be deleted or forgotten in cell state.
3. Output gate: Decide whether the hidden state is used for the output generated in the cell.

The gate in LSTM can be described in equations as follows:

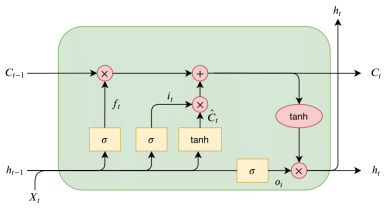


Figure 2: Architecture of LSTM

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

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

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{5}$$

where $i_t$,$f_t$,$o_t$ represent input, forget, and output gate respectively, $x_t$ is the input vector, and $h_{t-1}$ represent previous hidden state. The hidden state can then be calculated as :

$$h_t = o_t * tanh(C_t) \tag{6}$$

where $C_t$ is the current cell state.

#### 3.2.3 Gated Recurrent Unit (GRU)

GRU is another variant of LSTM and has a slightly different architecture from LSTM. This type of RNN was also designed to avoid exploding and vanishing gradient [15]. GRU has only two gates: the reset gate and the update gate. The reset gate controls how much previous information will be ignored, and the update gate determines how much information is reserved [4]. Figure 3 depicts the architecture of GRU and is mathematically described as follows:

$$r_t = \sigma(x_t W_x r + h_{t-1} W_{hr} + b_r) \tag{7}$$

$$z_t = \sigma(x_t W_x z + h_{t-1} W_{hz} + b_z) \tag{8}$$

where $r_t$ and $z_t$ are reset and update gates, respectively.

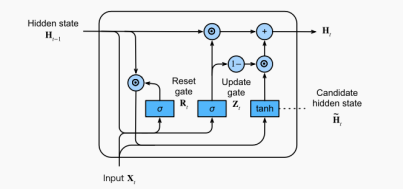


Figure 3: Architecture of GRU

## 3.3 Modelling and Evaluation

### 3.3.1 Experiment Setup

In this study, we evaluate the performance of vanilla RNN, LSTM, and GRU for stock price prediction. This involves training each model on APPL stock price data and tuning hyperparameters such as the number of layers, unit per layer, and batch size to find the best-performing model.

| Date | Low | Open | Volume | High | Close |
|---|---|---|---|---|---|
| 04-01-2010 | 7.585000038146970 | 7.622499942779540 | 493729600 | 7.660714149475100 | 7.643214225769040 |
| 05-01-2010 | 7.6160712242126500 | 7.664286136627200 | 601904800 | 7.699643135070800 | 7.656428813934330 |
| 06-01-2010 | 7.526785850524900 | 7.656428813934330 | 552160000 | 7.68678617477417 | 7.534643173217770 |
| 07-01-2010 | 7.466071128845220 | 7.5625 | 477131200 | 7.5714287757873500 | 7.520713806152340 |
| 08-01-2010 | 7.466429233551030 | 7.510714054107670 | 447610800 | 7.5714287757873500 | 7.570713996887210 |
| 11-01-2010 | 7.444643020629880 | 7.599999904632570 | 462229600 | 7.607142925262450 | 7.503929138183590 |
| 12-01-2010 | 7.372142791748050 | 7.471070766448980 | 594459600 | 7.491786003112790 | 7.4185709953308100 |
| 13-01-2010 | 7.289286136627200 | 7.423929214477540 | 605892000 | 7.533214092254640 | 7.523213863372800 |
| 14-01-2010 | 7.465000152587890 | 7.503929138183590 | 432894000 | 7.5164289474487300 | 7.479642868041990 |
| 15-01-2010 | 7.352499961853030 | 7.533214092254640 | 594067600 | 7.557143211364750 | 7.354642868041990 |
| 19-01-2010 | 7.401429176330570 | 7.440357208251950 | 730007600 | 7.685357093811040 | 7.679999828338620 |

Figure 4: Input Sequence of features and output

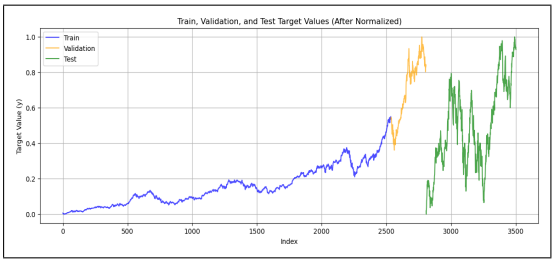


Figure 5: Train, validation, and test target after data normalization

#### 3.3.2 Data Preparation

Before feeding the data to the model, data normalization is needed to eliminate the effect of unit differences among the features. All columns in the dataset are normalized using MinMaxScaler [4]. The formula for this normalization is described as:

$$x'_t = \frac{x_t - min(x_t)}{max(x_t) - min(x_t)} \tag{9}$$

where $x'_t$ is the normalized data, this will make all data scaled between 0 and 1. The fluctuation for the close price of AAPL from 2010 to 2023 after normalisation is shown in Figure 5. To avoid data leakage, we fit different data scales for train validation and test sets.

In this study, we also evaluate the effect of window size or lookback size on model performance. Thus we use $X_t = \{x'_{t-n}, x'_{t-n+1}, ..., x'_t\}$, with $n$ is the length of window size. All columns except Adjusted Close are used for prediction. Figure 4 illustrates the sequence of input with 10 window sizes (orange box), and the prediction is the close price (yellow box) of the following day.

We split the data into train, fix validation and test set. Since we performed time-series data prediction, data is not splitted using random shuffle, instead first 70% data are used for training, next

10% data for validation, and last 20% data for testing.

#### 3.3.3 Model Evaluation

To measure the performance of the model, we use Mean-Square Error (RMSE), as this metric is commonly used and useful for regression tasks and time series analysis since it penalises a large error more heavily [4]. The formula of this metric is defined as follows:

$$MSE = \frac{1}{T}\sum_{t=1}^{T}(Y_t - Y_t^*)^2 \tag{10}$$

where $Y_t^*$ is the inverse scaled predicted values at time t, and T is the number of samples.

## 4 Result and Analysis

The type of RNN task in this study is many-to-one since we use a sequence of input features to predict a single value. A baseline model using a simple moving average algorithm was used first to be compared to the model performance.

**Baseline Model**. Simple Moving Average (SMA) predicts the next value as the average of the past $N$ observations. This study evaluated SMA with various window sizes for the last 30% of the data. Since the daily close price fluctuates, SMA performs better for smaller window sizes. Smaller window sizes allow the model to react quickly to short-term changes, shown in Figure 6. Moreover, bigger window sizes perform poorly because they take the average of more data and produce smoother changes. Thus, it can be concluded that smaller window sizes perform better when there is high variability within the data. MSE for window sizes 10, 30, and 60 are 17.43, 59.48 and 118.30, respectively.

**RNN (VRNN, LSTM, and GRU)**. Firstly, we compare *tanh* and *relu* activation functions in vanilla RNN. Results show that using *relu* activation function improves the model significantly for this task. With the same window size, unit, and batch size configuration, a model with *tanh* activation function yields an MSE of 30.59. In contrast, the model with *relu* performs on validation data with an MSE of 8.7. This finding shows that the *relu* model better captures long-term dependencies. *Tanh* model often leads to vanishing gradient problems during backpropagation for long-term dependencies [source].

Secondly, we compare VRNN, LSTM and GRU models across different lookback window sizes (10, 30, 60) by assesing the validation MSE. All models show increasing validation with longer lookbacks, suggesting overfitting or reduced learning efficiency. GRU performs consistently well, achieving the best validation MSE (7.84) for lookback 10 and maintaining competitive results for larger lookback size, making it the most robust model in term of time dependencies. LSTM performs moderately with stable MSE trends, while VRNN exhibits low Train MSE but struggles with high Validation MSE, particularly for longer lookbacks, indicating poor generalization. Figure 7 shows the error comparison of the model with various lookback sizes. Moreover, this could indicate difficulty learning long-term dependencies, a common symptom of vanishing gradients. This is noticeable for VRNN, with a higher rise of MSE for a bigger lookback size, whereas GRU and LSTM mitigate this issue better.

Subsequently, we evaluate the effect of unit size in RNN model on model performance. To assess this, we use different unit sizes (32, 64, 128) in VRNN. With a smaller unit size of 32, the model achieves a higher Validation MSE (8.747), indicating that the model lacks the capacity to capture the data's complexity, leading to underfitting. Increasing the unit size to 64 reduces the Validation MSE to 8.1, demonstrating that this moderate unit size yields an optimal balance by providing sufficient capacity for learning while maintaining generalization. However, increasing the unit size to 128 leads to a higher Validation MSE (8.53), likely due to overfitting. A larger unit size increases the model's complexity, which can enhance training performance but reduces its ability to generalize to unseen data. The error comparison across various unit sizes is shown in Figure 9.

Results also show that VRNN, LSTM and GRU yield different performances with different batch sizes for training. With the same lookback and unit size, VRNN perform better with a higher batch size; this suggests that VRNN, with its simple architecture, may generalize better due to the stabilization of gradient updates during training. In contrast, for more complex architecture like LSTM and GRU, a larger batch size negatively impacts the performance. Both LSTM and GRU show an increase in validation error with a batch size of 64, despite a slight decrease in training error. This behaviour indicates potential overfitting or reduced generalization when the batch size increases. Table 1 shows the comparison of different batch sizes on model performance. For a fair comparison, we then use the same unit size and batch size for VRNN, LSTM, and GRU.

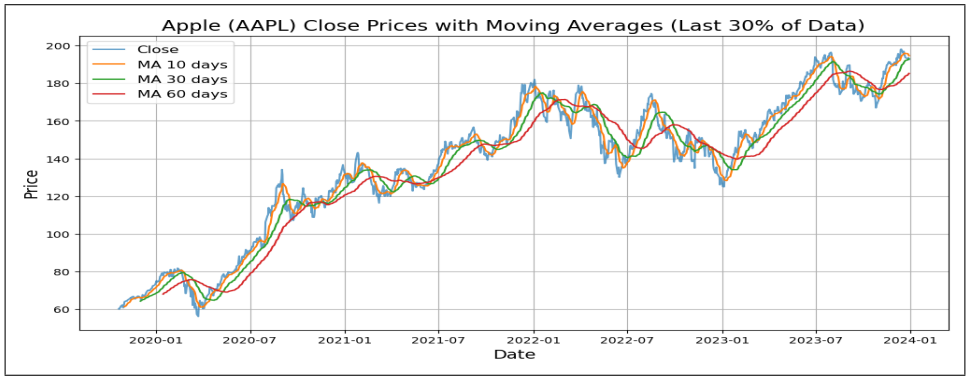


Figure 6: Simple Moving Average Model

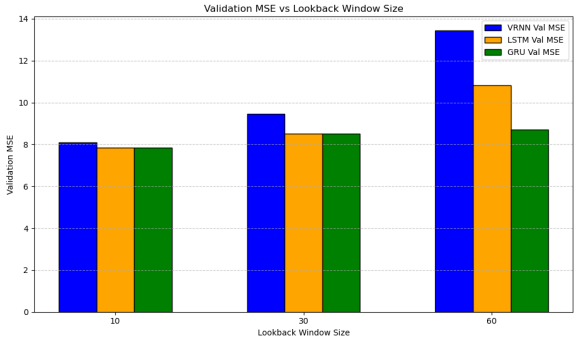


Figure 7: Validation error for various lookback size

Table 1: Model vs Batch Size (Lookback 10, Unit 64)

| **Model** | **Batch Size** | **Train MSE** | **Val MSE** |
|---|---|---|---|
| VRNN | 32 | 0.279 | 8.17 |
| | 64 | 0.27 | 8.1 |
| LSTM | 32 | 0.3357 | 7.856 |
| | 64 | 0.36 | 9.11 |
| GRU | 32 | 0.411 | 7.84 |
| | 64 | 0.342 | 9.525 |

The experiment shows that all RNN models work better by adding one dense layer. Smaller hidden layer size is proven to be better for this task, with only one hidden size and 16 neurons. Adding multiple dense layers yields a higher validation MSE for all models, which is a sign of overfitting. This confirms that RNN models are inherently complex, with gate mechanisms already able to capture complex relationships within the data; adding more layers will introduce more model complexity, hence overfitting. The number of parameters for each model configuration is shown in Table 2. Considering its architecture configuration, VRNN has the lowest number of parameters, and LSTM has the highest number.

Table 2: Number of parameters in each model

| **Model** | VRNN | LSTM | GRU |
|---|---|---|---|
| **#Parameters** | 4545 | 16833 | 13053 |

For further comparison, we also compare LSTM and GRU with their variants: Bidirectional-LSTM and Bidirectional-GRU. Both Bi-LSTM and Bi-GRU process the data in forward and backward directions [source]. However, the latest algorithm did not perform better than LSTM and GRU. With the same configuration, Bi-LSTM and Bi-GRU yielded prediction MSE of 13.6 and 8.5 respectively. This might be because the data is strictly forward-moving, and backward passes may not provide meaningful information or even introduce noise. The increased complexity in the bidirectional model also doubles the number of parameters, making the theme prone to overfitting.

Finally, after tuning the hyperparameters in all models by evaluating the validation error, we compare the best model from VRNN, LSTM, and GRU. Figure 8 shows the comparison of the prediction of all models with the ground truth validation data. Results show that all models perform well on validation data. The prediction graph of VRNN, LSTM, and GRU models aligns closely with the ground truth. There are no noticeable deviations, indicating that these models are effectively capturing the underlying patterns in the data. However, Table 3 show the more comprehensive comparison of train and validation errors for each model. Based on validation performance, GRU performs best among all models with slightly different errors than LSTM. This might be due to its simplicity, but it can still generalize well to time series data. GRU can capture temporal dependencies effectively without the potential overfitting risk.

The GRU model is then tested against the test set to find the final model performance. The pre-

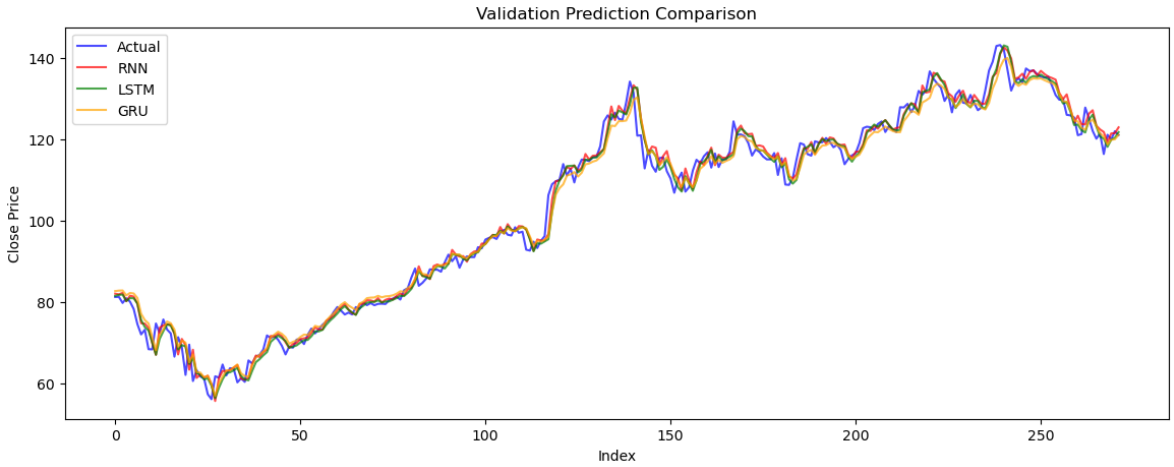


Figure 8: Validation prediction

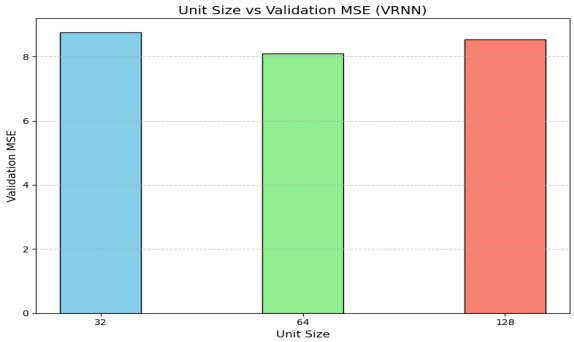


Figure 9: Unit size vs Validation MSE

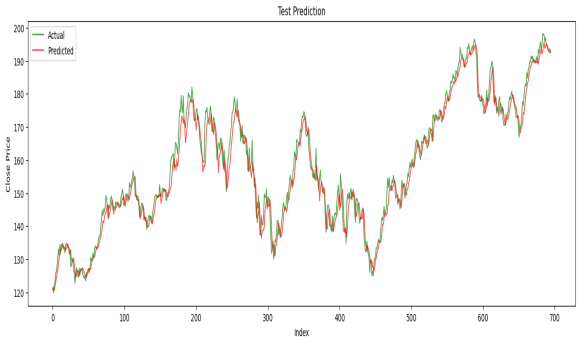


Figure 10: Best model (GRU) prediction on test set

Table 3: Model Performance Comparison

| Model | Train MSE | Val MSE |
|---|---|---|
| VRNN | 0.27 | 8.1 |
| LSTM | 0.336 | 7.856 |
| GRU | 0.411 | 7.84 |

diction result is shown in Figure 10, with test MSE 10.94, which is not significantly different from the validation error, indicating that the model has achieved good generalization.

# 5 Conclusion and Future Work

This study evaluates the performance of RNN in stock price prediction. Compared to moving average as a simple baseline model, RNN works better in predicting stock price with lower MSE at a looking back size of 10. The results also show that RNN with gate architecture such as GRU and LSTM work better in this study, especially with larger looking back sizes. This proves that the LSTM and GRU models work better in handling data with temporal dependencies. The results demonstrated that GRU outperformed both LSTM and Vanilla RNN in terms of validation MSE. LSTM also performed competitively, but its higher complexity made it prone to overfitting in certain scenarios. Due to vanishing gradient issues, Vanilla RNN struggled with generalization, particularly with longer lookback windows. The study also revealed that smaller hidden layer sizes and reduced model complexity yield better generalization for time-series tasks.

Further study could explore integrating external features such as market trends or sentiment data for improved accuracy. Additionally, transfer learning and hyperparameter optimization using advanced techniques could further enhance performance and robustness in stock price prediction tasks.

# 6 Code

The code for this study is available on GitHub repository.

# References


[1] A. G. A. Md. Arif Istiake Sunny, Mirza Mohd Shahriaar Maswood, "Deep learning-based stock price prediction using lstm and bi-directional lstm model," *2nd Novel Intelligent and Leading Emerging Sciences Conference*, pp. 87–92, 2020.

[2] G. E. V. K. M. S. K. Sreelekshmy Selvin, Vinayakumar R, "Stock price prediction using lstm,rnn and cnn-sliding window model," pp. 1643–1647.

[3] D. Faruk, "A hybrid neural network and arima model for water quality time series prediction," *Engineering Applications of Artificial Intelligence*, vol. 23, pp. 586 – 594, 2010.

[4] Z. W. Yiwei Liu, "Application of regularized gru-lstm model in stock price prediction," *2019 IEEE 5th International Conference on Computer and Communications*, pp. 1886 – 1890, 2019.

[5] J. W. L. Q. Wenjie Lu, Jiazheng Li, "A cnn-bilstm-am method for stock price prediction," *Neural Computing and Applications*, vol. 33, pp. 4741–4753, 2020.

[6] X. S. Ruixun Zhang, Zhaozheng Yuan, "A new combined cnn-rnn model for sector stock price analysis," *1028 42nd IEEE International Conference on Computer Software Applications*, pp. 546–551, 2018.

[7] W. Y. T. Hsieh, H. Hsiao, "Forecasting stock market using wavelet transform and recurrent neural networks," *Applied Soft Computing Journal*, vol. 11, pp. 2510–2525, 2011.

[8] R. O. D.M Nelson, A.C Pereira, "Stock merket price movement prediction with lstm neural networks," *IJCNN*, vol. 6, pp. 1419–1426, 2017.

[9] M. Dang. L, "Deep learning approach for short-term stock trends prediction based on gru," *IEEE Accsess*, vol. 6, pp. 55392–55404, 2018.

[10] X. W. Chen, C Xue, "Research on improved gru-based stock price prediction method," *Applied Sciences*, vol. 13, 2023.

[11] Y. Finance, "Yahoo finance api," n.d. Retrieved December 1, 2024, from https://finance.yahoo.com.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. Available online: https://www.deeplearningbook.org.

[13] F. M. D. S. Karim, M.E, "Stock price prediction using bi-lstm and gru-based hybrid deep learning approach," *Proceedings of Third Doctoral Symposium on Computational Intelligence*, vol. 479, 2023.

[14] S. S. Arjun Singh Saud, "Analysis of look back period for stock price prediction with rnn variants: A case study on banking sector of nepse," *International Conference on Computational Intelligence and Data Science (ICCIDS 2019)*, vol. 167, pp. 788–798, 2020.

[15] V. M. K. Cho and C. Gulcehre, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," February 2015.

[16] X. X. Minrong Lu, "Trnn: An efficient time-series recurrent neural network for stock price prediction," *Information Sciences*, vol. 657, 2023.

[17] Z. L. Shui-Ling Yu, "Stock price prediction based on arima-rnn combined model," *4th International Conference on Social Science (ICSS 2017)*, pp. 17–25, 2017.