Recurrent Neural Networks for Stock Price Prediction

Hu Xiaolong
The University of Adelaide
Adelaide SA 5005
A1912235@adelaide.edu.au

Abstract

This paper develops the effectiveness of RNN-based architectures for predicting Google stock closing prices using historical data from 2012 to 2016. Recurrent Neural Networks (RNNs), especially their variants such as Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks, are well suited for analyzing sequential data such as stock prices. The task involves predicting the next day's closing price using the stock data (open, high, low, closing, and volume) for the past 30 days. The dataset was preprocessed and the Vanilla RNN, GRU, and LSTM models were trained using the same hyperparameters. Finally, predictions were made for future closing prices and the results were compared with the actual stock prices. The performance of the models was evaluated based on their ability to capture trends and minimize prediction errors. This paper discusses the design choices, performance analysis, and limitations of the models, providing insights for future improvements and practical applications in stock market prediction.

1. Introduction

Stock price prediction is a cornerstone problem in financial analysis and holds tremendous value for investors seeking to make strategic decisions. The task is inherently challenging due to the volatility and nonlinear nature of stock market data. Traditional statistical methods such as Autoregressive Integrated Moving Average (ARIMA) and exponential smoothing have been widely used, but they have limited ability to model complex temporal dependencies. Recurrent Neural Networks (RNNs) have emerged as a powerful tool for time series forecasting due to their ability to learn sequential patterns.[1] In particular, advanced variants such as Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) networks address key issues such as vanishing gradients, making them suitable for modeling long-term dependencies.[2]

1.1. Purpose

This paper focuses on predicting Google's stock closing price by leveraging RNN-based architectures. Unlike simple statistical methods, RNNs can effectively leverage historical trends to make future predictions. Specifically, we trained and compared three architectures: Vanilla RNN, GRU, and LSTM. The models were evaluated based on their ability to predict the stock price in January 2017 using historical data from 2012 to 2016. The task is designed to mimic realistic financial forecasting, using the past 30 days of stock data to predict the next day's closing price.

Through comparative analysis, it aims to highlight the strengths and weaknesses of each RNN architecture in processing stock market data, determine which architecture can best capture the time dependency and trends of stock market data, thereby minimizing prediction errors and providing actionable insights for decision-making in the financial market.

1.2. Process

We mainly collected historical Google stock data from 2012 to 2016, including opening price, highest price, lowest price, closing price, and trading volume. The data was normalized and divided into training, validation, and test sets. Vanilla RNN, GRU, and LSTM models were implemented using PyTorch. All models have the same architecture in terms of input size, hidden layers, and hyperparameters to ensure fair comparison. Afterwards, the models were trained using the training set and the hyperparameters were adjusted using the validation set. Then, forecasts for January 2017 were generated using the last 30 days of December 2016 as input. These forecasts were compared with actual data to evaluate the accuracy and trend matching ability of the models.

Finally, the results were analyzed to evaluate the strengths and limitations of the models. Insights on the performance of the models and potential improvements are discussed in the context of real-world applications.

2. Method Description

The following will explain the algorithm and the model used in detail and illustrate some of their challenges.

2.1. Vanilla Recurrent Neural Network (RNN)

The Vanilla Recurrent Neural Network (RNN) is the simplest form of RNN, designed to process sequential data by maintaining a hidden state that evolves over time. It is an extension of a feedforward neural network that incorporates loops to allow information to persist across time steps.[4] Vanilla RNNs are particularly well suited for tasks involving temporal dependencies, such as time series forecasting.

We use PyTorch to implement its architecture which consists of the following components:

1. **Input layer:** The input data is represented as a sequence of vectors, where each vector corresponds to the stock features (open, high, low, close, volume) over the past 30 days. Each sequence has a shape of **N x T x F**, where **N** is the batch size **T = 30** is the sequence length, and **F = 5** is the number of features.
2. **Recurrent Layer:** A single-layer RNN cell processes the input sequentially. At each time step **t**, the cell updates its hidden state $\boldsymbol{h_t}$ based on the current input $\boldsymbol{x_t}$ and the previous hidden state $\boldsymbol{h_{t-1}}$. The state transition can be expressed as:
$$\boldsymbol{h_t = tanh(W_h h_{t-1} + W_x x_t + b)}$$
where $\boldsymbol{W_h}$ and $\boldsymbol{W_x}$ are weight matrices, and **b** is the bias vector.
3. **Output Layer:** After processing all time steps, the output from the final hidden state is passed to a fully connected layer, which predicts the closing price for the next day.

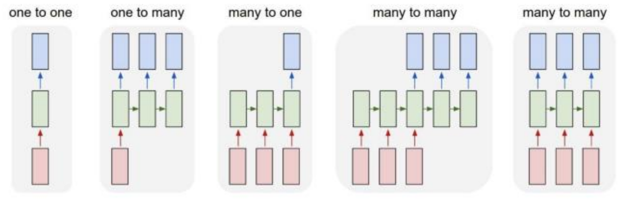


Figure 1. Different (non-exhaustive) types of Recurrent Neural Network architectures. Red boxes are input vectors. Green boxes are hidden layers. Blue boxes are output vectors[4]

Vanilla RNNs are used as a baseline model to provide insight into the performance of direct sequential methods for stock price prediction. While Vanilla RNNs are conceptually simple, they suffer from limitations such as the vanishing gradient problem, which hinders their ability to learn long-term dependencies. These challenges motivated this study to include advanced architectures such as GRU and LSTM for comparison.

2.2. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) specifically designed to address the limitations of Vanilla RNNs, particularly the vanishing gradient problem. By incorporating gating mechanisms, LSTMs can effectively capture long-term dependencies, making them well-suited for time-series prediction tasks such as stock price forecasting.[5]

We use PyTorch to implement its architecture with the following key component:

1. **Input Layer:** The input data comprises sequences of stock features (open, high, low, close, volume) over the past 30 days. Each input sequence has a shape of **N x T x F**, where **N** is the batch size **T = 30** is the sequence length, and **F = 5** is the number of features
2. **LSTM Layer:** LSTM introduces three gates, input, forget, and output gates, that regulate the flow of information within the network:
   - **Forget Gate**: Decides what information to discard from the previous cell state$\boldsymbol{C_{t-1}}$:
   $$\boldsymbol{f_t = \sigma\left(W_f[h_{t-1}, x_t] + b_f\right)}$$
   - **Input Gate**: Determines what new information to store in the cell state:
   $$\boldsymbol{i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)}$$
   $$\boldsymbol{\widetilde{C}_t = tanh(W_C[h_{t-1}, x_t] + b_C)}$$
   - **Output Gate**: Controls what part of the cell state contributes to the final output:
   $$\boldsymbol{o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)}$$
   $$\boldsymbol{h_t = o_t \cdot tanh(C_t)}$$
3. **Fully Connected Layer**: The output from the final hidden state is passed through a fully connected layer to predict the closing price for the next day.

LSTMs excel at capturing both short-term and long-term dependencies due to their gating mechanisms. This makes them more effective for predicting stock prices compared to Vanilla RNNs. However, due to its complex architecture, LSTM is more computationally expensive than Vanilla RNN. Training can be slower, especially when working with large datasets.

2.3. Gated Recurrent Unit (GRU)

Gated Recurrent Units (GRUs) are a more streamlined version of Long Short-Term Memory (LSTM) networks, designed to address the same issues as LSTMs but with a simpler architecture. GRUs have fewer parameters and computational demands compared to LSTMs, making them faster to train while retaining comparable performance in many applications, including time-series prediction tasks.[3]

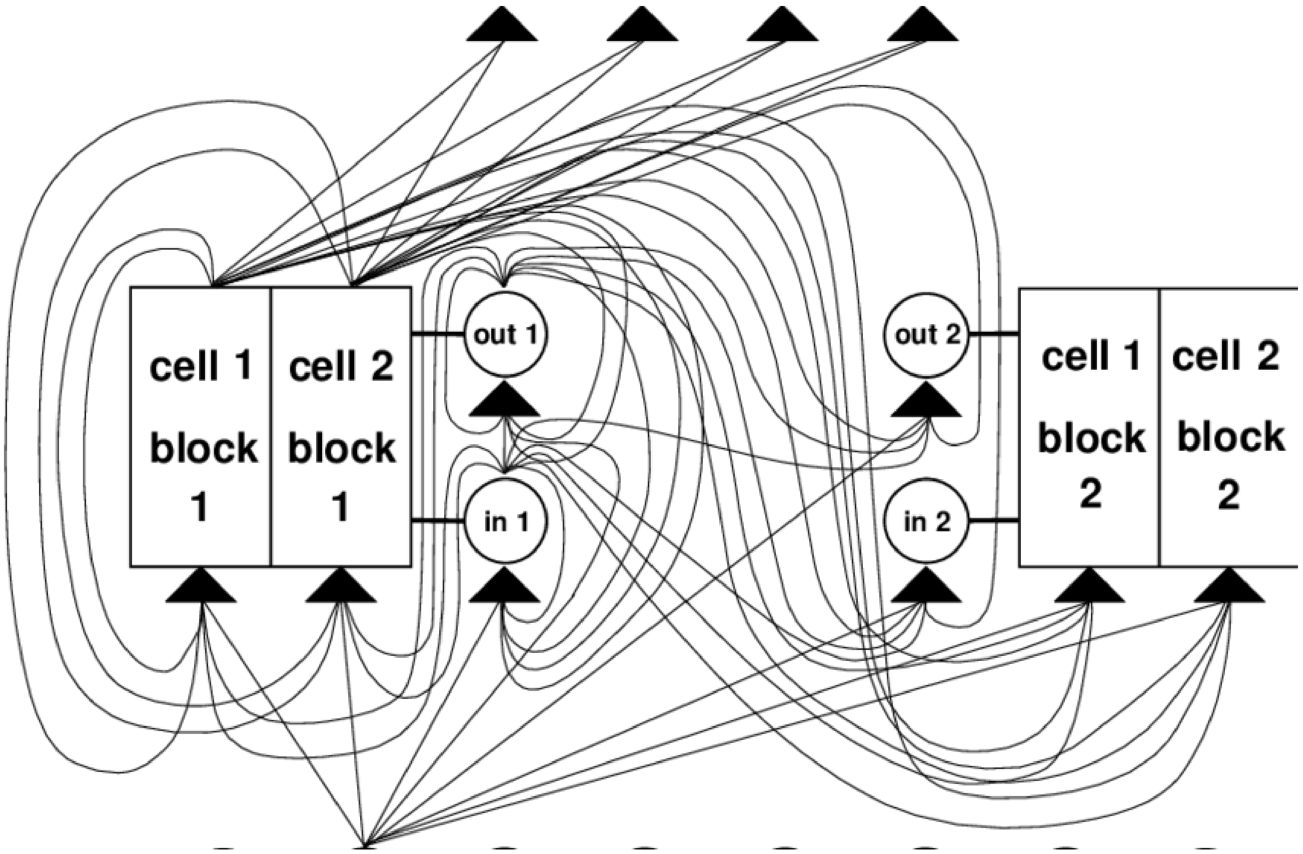


Figure 2. Example of a LSTM net with 8 input units, 4 output units, and 2 memory cell blocks of size 2. As soon as the error stream wants to leave the memory cell or gate cell, it is truncated. Therefore, any connection shown above cannot propagate errors back to the cell of origin of the connection (except for the connection to the output cell), although the connection itself is modifiable. This is why the truncated LSTM algorithm is so effective, despite its ability to bridge very long-time lags.[5]

We us PyTroch to implement its with the following components:

1. **Input Layer**: The input data comprises the past 30 days of stock features (open, high, low, close, volume). The shape of the input tensor is **N x T x F**, where **N** is the batch size **T = 30** is the sequence length, and **F = 5** is the number of features
2. **GRU layer:** GRUs simplify the structure of LSTMs by combining the forget and input gates into a single update gate and removing the cell state $\boldsymbol{C_t}$, relying only on the hidden state $\boldsymbol{h_t}$. The key equations governing GRUs are:
   - **Update Gate:** Determines how much of the previous hidden state to retain:
     $$\boldsymbol{z_t = \sigma(W_z[h_{t-1}, x_t] + b_z)}$$
   - **Reset Gate:** Controls how much of the previous hidden state contributes to the new candidate state:
     $$\boldsymbol{r_t = \sigma(W_r[h_{t-1}, x_t] + b_r)}$$
   - **Candidate State:** Represents the new content to be added to the hidden state:
     $$\boldsymbol{\widetilde{h}_t = \ tanh(W_h[r_t \cdot h_{t-1}, x_t] + b_h)}$$
   - **Final Hidden State:** Combines the previous hidden state and the candidate state based on the update gate:
     $$\boldsymbol{h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \widetilde{h}_t}$$

   These mechanisms allow GRUs to capture temporal dependencies efficiently while maintaining a reduced computational overhead.
3. **Fully Connected Layer**: The output from the final hidden state is passed through a dense layer to predict the closing price for the next day.

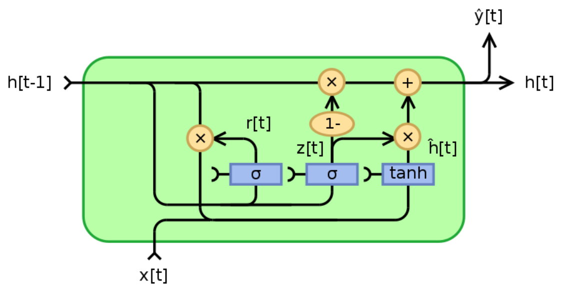


Figure 3. Gated Recurrent Unit Architecture[3]

GRUs are computationally faster and more memory efficient than LSTMs. They simplify the architecture without compromising performance for many time series forecasting tasks. While simpler than LSTMs, GRUs may not be able to effectively capture extremely long-term dependencies in some cases.

The GRU model balances performance and computational efficiency, making it a valuable alternative to Vanilla RNNs and LSTMs. This study incorporates it to provide insights into the trade-off between complexity and accuracy for stock price prediction tasks.

3. Experimental Test

The following shows the data preprocessing steps, the experimental setup for testing the models, and the results obtained from different configurations.

3.1. Method Implementation

Data Preparation: Only the Open, High, Low, Close, and Volume columns are used. The Closing price is chosen as the prediction target. The data is scaled between 0 and 1 using Min-Max scaling to stabilize the training and improve convergence. We design to use a sequence length of 30 days (N = 30) as input to predict the next day's closing price (M = 1). After that, the dataset is split into training and validation subsets with 8/2. The training set contains data from January 2012 to December 2016, and the test set covers January 2017.

Denormalization: Since the input data is normalized to a smaller range [0, 1] before training to speed up the training process and improve numerical stability, the model's predicted output is also in the same normalized range. However, in order to compare the predicted results with the real data, the predicted values must be denormalized to the original data scale and the actual range of the data must be restored through the inverse transformation:

Original Value = Normalized Value * (Max – Min) + Min

Training Setting: All models accept a 3D tensor of shape (batch_size, sequence_length, features) corresponding to a 30-day window and five features. Meanwhile, Mean squared error (MSE) is used as the loss function to minimize the difference between the predicted closing price and the actual closing price. In addition, on the optimizer, the Adam optimizer is selected with a learning rate of 0.001 to balance efficiency and stability. And, the batch size is set to 64 for mini-batch training, the models are trained for 500 epochs to ensure sufficient time for convergence.

Learning rate: 0.001
Optimizer: Adam
Loss function: MSE
Number of epochs: 500

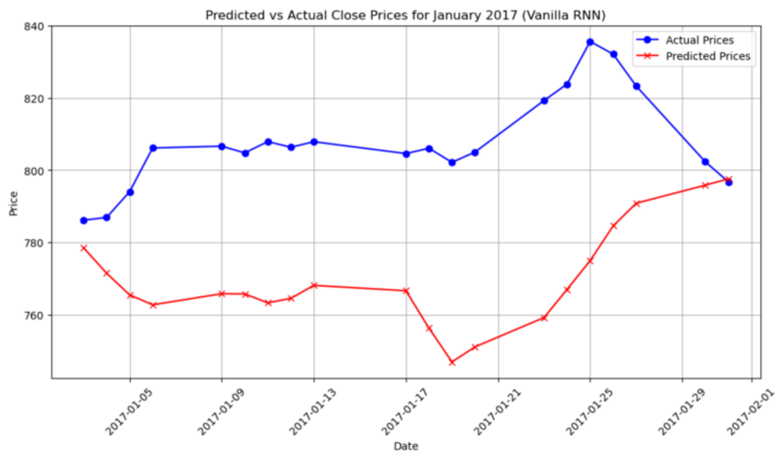
After training, we also use mean squared error (MSE) and root mean square error (RMSE), which are calculated based on the original scale to provide more interpretable performance evaluation.

Trend Match: In the test, we use trend match to calculate the degree of fit between the predicted value and the actual value. The calculation principle is based on the judgment of trend changes in stock forecasts. It determines whether the trend matches by comparing the predicted closing price of each day with the actual closing price. Specifically, it focuses on the turning point of the trend, that is, when the predicted value and the actual value have different changes (i.e., the trend rises or falls), it is considered a match. If the predicted value and the actual value change in the same direction, it is considered a mismatch.

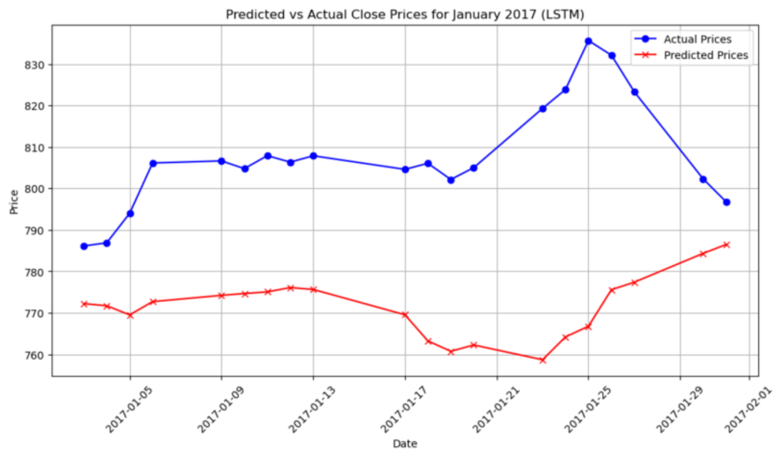
The core rules of Trend Match calculation: If the change direction of the predicted value is different from the change direction of the actual value, that is, the closing price of the predicted value and the actual closing price of the previous moment change in opposite directions, then the point is considered to be matched. If the change direction of the predicted value is the same as the change direction of the actual value, then the point is considered to be mismatched.

All model training is performed on a NVIDIA CUDA-enabled 3070laptop version GPU to speed up computation. During training, models are evaluated on a validation set to monitor performance and prevent overfitting. At the end of training, the model with the lowest validation loss is saved for testing.

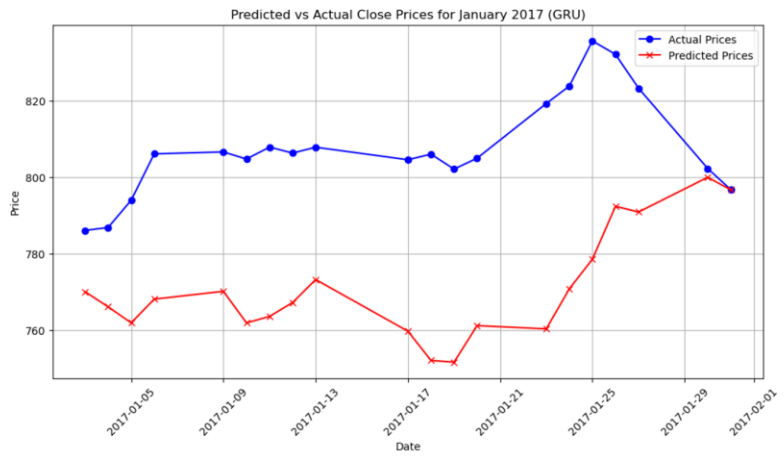
This comprehensive implementation provides a solid foundation for experimental evaluation and allows straightforward reproduction of results. Next, we present the experimental tests and results obtained from these settings.



	Date	Predicted Close	Actual Close	Trend Match
0	2017-01-03	778.496216	786.14	Match
1	2017-01-04	771.449402	786.90	Mismatch
2	2017-01-05	765.453491	794.02	Mismatch
3	2017-01-06	762.698608	806.15	Mismatch
4	2017-01-09	765.759521	806.65	Mismatch
5	2017-01-10	765.707581	804.79	Match
6	2017-01-11	763.247437	807.91	Mismatch
7	2017-01-12	764.467773	806.36	Match
8	2017-01-13	768.077820	807.88	Mismatch
9	2017-01-17	766.591919	804.61	Match
10	2017-01-18	756.226807	806.07	Mismatch
11	2017-01-19	746.872131	802.17	Match
12	2017-01-20	751.038452	805.02	Mismatch
13	2017-01-23	759.166077	819.31	Mismatch
14	2017-01-24	766.987549	823.87	Mismatch
15	2017-01-25	774.929260	835.67	Mismatch
16	2017-01-26	784.564941	832.15	Match
17	2017-01-27	790.836792	823.31	Match
18	2017-01-30	795.839966	802.32	Match
19	2017-01-31	797.594055	796.79	Mismatch



	Date	Predicted Close	Actual Close	Trend Match
0	2017-01-03	772.223267	786.14	Match
1	2017-01-04	771.714417	786.90	Mismatch
2	2017-01-05	769.525696	794.02	Mismatch
3	2017-01-06	772.714539	806.15	Mismatch
4	2017-01-09	774.227295	806.65	Mismatch
5	2017-01-10	774.666748	804.79	Match
6	2017-01-11	775.063782	807.91	Mismatch
7	2017-01-12	776.106445	806.36	Match
8	2017-01-13	775.647705	807.88	Mismatch
9	2017-01-17	769.549377	804.61	Match
10	2017-01-18	763.243774	806.07	Mismatch
11	2017-01-19	760.743408	802.17	Match
12	2017-01-20	762.262512	805.02	Mismatch
13	2017-01-23	758.694641	819.31	Mismatch
14	2017-01-24	764.213928	823.87	Mismatch
15	2017-01-25	766.771973	835.67	Mismatch
16	2017-01-26	775.580872	832.15	Match
17	2017-01-27	777.403748	823.31	Match
18	2017-01-30	784.344116	802.32	Match
19	2017-01-31	786.481873	796.79	Match



	Date	Predicted Close	Actual Close	Trend Match
0	2017-01-03	770.130798	786.14	Match
1	2017-01-04	766.174316	786.90	Mismatch
2	2017-01-05	761.987793	794.02	Mismatch
3	2017-01-06	768.183655	806.15	Mismatch
4	2017-01-09	770.227966	806.65	Mismatch
5	2017-01-10	761.970032	804.79	Match
6	2017-01-11	763.667236	807.91	Mismatch
7	2017-01-12	767.287170	806.36	Match
8	2017-01-13	773.294495	807.88	Mismatch
9	2017-01-17	759.796448	804.61	Match
10	2017-01-18	752.161072	806.07	Mismatch
11	2017-01-19	751.715942	802.17	Match
12	2017-01-20	761.251099	805.02	Mismatch
13	2017-01-23	760.409851	819.31	Mismatch
14	2017-01-24	770.891357	823.87	Mismatch
15	2017-01-25	778.611511	835.67	Mismatch
16	2017-01-26	792.458374	832.15	Match
17	2017-01-27	790.949646	823.31	Match
18	2017-01-30	800.020630	802.32	Match
19	2017-01-31	796.738953	796.79	Match

Figure 4. Comparison chart of the predicted closing price of January 2017 by vanilla RNN, LSTM, and GRU models and the actual closing price, and detailed data

3.2. Vanilla RNN Result

The Vanilla Recurrent Neural Network (RNN) was evaluated on the stock price prediction task using Google’s historical stock price data. The results were analysed for both training performance and test predictions. During training, the Vanilla RNN demonstrated consistent improvement in minimizing the loss function across 500 epochs.

Training Performance: The initial loss value at epoch 10 was 0.0582, which decreased steadily to 0.0013 by epoch 500. Most of the significant reductions in loss occurred during the first 100 epochs, indicating rapid learning during the early training phase. Beyond epoch 200, the loss plateaued, suggesting that the network reached its capacity for learning from the data. The progressive reduction in loss validates the effectiveness of the Vanilla RNN in capturing the temporal patterns in the training dataset
After training process, the vanilla RNN has achieved a low Mean Squared Error (MSE) of 0.0002 and a Root Mean Squared Error (RMSE) of 0.0138 after 500 epochs. These metrics reflect the model's ability to minimize prediction errors during training, though it is essential to assess its generalization on test data.

Prediction Results: We used the last 30 days of December 2016 as input and let the model predict the closing price of stocks in January 2017 day by day.

As can be seen from the table, Vanilla RNN has achieved a trend match accuracy of 40%, correctly predicting 8 out of 20 trends. This low accuracy suggests limited capability in identifying the directional changes of stock prices, particularly during volatile periods.

Predicted closing prices showed moderate alignment with actual values but struggled to capture sharp increases or decreases in stock prices. For instance, predictions on January 25 and 26 underestimated the actual closing prices significantly, missing the trend shift during that period.

Analysis of Performance: The model effectively minimizes the loss during training, as evidenced by the low MSE and RMSE values. Despite its simplicity, the Vanilla RNN can partially identify stock trends during relatively stable periods.

However, the 40% trend accuracy is suboptimal, highlighting that the model does not generalize well to unseen test data. During periods of high volatility, the predictions lag behind the actual values and fail to capture sharp up or down trends (e.g., January 23-26). The vanishing gradient problem may limit the model's ability to exploit long-term dependencies.

3.3. LSTM Result

The LSTM model exhibited strong predictive performance, with an MSE of 0.0007 and an RMSE of 0.0262. These metrics indicate that the model effectively minimized error during prediction, providing a reliable forecast of stock prices

Training Performance: The LSTM model demonstrated rapid convergence, as evidenced by the early reduction in training loss.
Initial losses started relatively high (e.g., 0.0024 at epoch 10) but quickly declined to lower values, reaching 0.0004 at epoch 500.

However, some fluctuation in loss values was observed at intermediate epochs, suggesting sensitivity to local optima or learning rate configuration.

Prediction Result: According to the actual data, the trend matching accuracy of LSTM reached 45%, and 9 out of 20 trends were correctly predicted. indicating moderate success in capturing directional trends in stock price movements.

The predicted price of the LSTM model is close to the true price, but in many cases, the error is concentrated at the critical point of the stock price (e.g., slightly below the critical value of an increase or decrease), resulting in failure of trend matching. On January 6, the actual closing price was 806.15, while the predicted value was 772.71. The absolute value error between the two was large, causing the trend prediction to fail.

Analysis of Performance: The LSTM architecture effectively exploits temporal information, which is reflected in the steady decrease in training loss, while the low RMSE indicates that the predicted prices are very close to the average actual prices.

Although the price predictions are more accurate than the vanilla RNN model, the trend accuracy still needs to be improved. This may be due to the LSTM’s tendency to optimize to minimize absolute error rather than directional consistency.

3.4. GRU Result

Both GRU and LSTM are improved models proposed to deal with the gradient vanishing and gradient exploding problems of traditional RNN in long sequence data. Although the two have many similarities in structure, they also have some key differences that make GRU more efficient or perform better than LSTM in some cases.

Training Performance: The training process of the GRU model shows a rapid decrease in loss in the early stages, but fluctuations in the later stages. The loss rate reaches 0.0024 in the tenth epoch and reaches 0.001 after 250 epochs. And the MSE,RMSE has reached 0.0007,0.0262. These values demonstrate the model's ability to minimize prediction error on the test set.

The model fluctuates in training loss, but the final loss of 0.0004 indicates that the model converges effectively, although the presence of noise in the training loss curve may indicate sensitivity to hyperparameter settings or irregularities in the dataset.

Prediction Result: The figure shows that the GRU model correctly predicted the direction of the stock price trend 45% of the time. In 20 predictions, there were 9 trend matches and 11 trend mismatches, which is similar to LSTM. Although the accuracy is improved compared to the simple model, it highlights the challenge of accurately capturing stock price trends.

From the specific prediction results, on some dates (such as the 3rd, 10th, and 17th), GRU was able to accurately predict the trend of stock prices (such as rising or falling). However, on other dates (such as the 4th and 13th), the trend predicted by GRU was inconsistent with the actual price trend, which once again shows that the GRU model sometimes encounters challenges in trend matching, especially when the market is volatile or the price changes are more complex.

Analysis of Performance: The GRU model effectively captures the general trend of the data with a relatively low RMSE. It successfully matches the actual price trend in nearly half of the cases, which is competitive considering the randomness of stock prices.

However, the model has difficulty coping with large fluctuations in stock prices, resulting in mismatches during volatile periods. The fluctuations in training loss indicate that there is room for improvement in optimization techniques, such as adjusting the learning rate or using advanced gradient clipping methods.

4. Results Analysis

We analyse and compare the performance of three models: Vanilla RNN, LSTM, and GRU on the task of stock price prediction. These models exhibit different strengths and weaknesses under different evaluation metrics. When choosing the most suitable model, we must not only consider the performance indicators of the model (such as loss value, MSE, RMSE, etc.), but also consider its feasibility and practicability in practical applications.

Vanilla RNN: The structure is relatively simple and suitable for processing basic time series data. It can be easily implemented and trained quickly. Compared with LSTM and GRU, Vanilla RNN is computationally lightweight and suitable for use in environments with limited resources.

However, when trained on longer sequences, Vanilla RNN is prone to gradient disappearance or gradient explosion problems, causing its performance to drop significantly when dealing with longer-term dependencies.

LSTM: It can effectively solve the vanishing gradient problem in traditional RNN, allowing it to capture dependencies over a longer time span. In long-term series tasks such as financial data, LSTM can often achieve better prediction results. According to the experimental results, the accuracy of LSTM in trend prediction is significantly higher than that of Vanilla RNN. Although the accuracy is not perfect, it is still better than other models and can stably predict price trends.

However, the structure of LSTM is more complex and requires more computing resources than Vanilla RNN, especially when processing large-scale data sets. Moreover, LSTM is prone to overfitting, especially when the amount of data is insufficient. This can result in a model that performs well on the training set but suffers performance degradation on the validation or test set.

GRU: As a variant of LSTM, the structure is much simplified and the number of parameters is reduced, so it is more computationally efficient. GRU is more efficient than LSTM when processing large-scale data, with better performance. GRU has similar performance to LSTM and performs particularly well in capturing long-term dependencies. Judging from the experimental results, the MSE value of GRU is low, indicating that it has a strong ability to fit the training data.

Although GRU performs better on MSE and RMSE, its accuracy on trend matching is only 45%. This indicates that GRU is less stable than LSTM in capturing the trend of price changes, especially in high-volatility markets.

Based on the above analysis, we compared the advantages and disadvantages of the three models. From the perspective of prediction performance and practicality, there are high requirements for the prediction accuracy of the model, and **LSTM is the best choice**.

LSTM is a very suitable choice in authenticity prediction. It can effectively capture the long-term trends in stock market data, so it performs well in stock market prediction tasks, especially when large-scale historical data is involved. Its high trend prediction accuracy and stability are its main advantages. Although its computational complexity is high, its accuracy in authenticity prediction and its ability to handle long-term dependencies make it highly valuable in actual market applications.

5. Conclusion

In this paper, we comprehensively analyse how three popular recurrent neural network (RNN) models—Vanilla RNN, LSTM, and GRU—predict stock market trends based on historical price data. The main goal of this work is to evaluate and compare the effectiveness of these models in terms of prediction accuracy, computational efficiency, and ability to capture long-term dependencies in time series data.

Our experimental results show that LSTM outperforms Vanilla RNN and GRU in capturing long-term dependencies and providing accurate trend predictions. Despite its higher computational cost, LSTM demonstrates superior accuracy in predicting stock market trends. GRU, while slightly less accurate, is a good choice when computational efficiency is a priority.

Future work could explore further optimizations of these models, such as integrating external market factors, leveraging hybrid models, or applying transfer learning techniques to improve prediction performance.

GitHub links:
https://github.com/SpiderJockey7/Recurrent-Neural-Networks-for-Stock-Price-Prediction

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Institute for Cognitive Science, University of California, San Diego, CA, USA, Tech. Rep. ICS 8504, Sept. 1985.

[2] M. I. Jordan, "Serial order: A parallel distributed processing approach," Institute for Cognitive Science, University of California, San Diego, CA, USA, Tech. Rep. ICS 8604, May 1986.

[3] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," presented at the Université de Montréal, Montréal, QC, Canada, 2014. Available: https://arxiv.org/abs/1406.1078

[4] Calvin Feng, "Recurrent Neural Network," *Machine Learning Notebook*. Available: https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks. [Accessed: Nov. 22, 2024].

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Dec. 1997, doi: 10.1162/neco.1997.9.8.1735