RNNs for stock price prediction

Irfan F Boelhasrin
University of Adelaide
Adelaide, South Australia
irfan.boelhasrin@student.adelaide.edu.au

Abstract

Stock market prediction is a challenging task due to the highly volatile and nonlinear nature of time-series data. Traditional linear models such as AR, ARMA, and ARIMA are often limited in their ability to generalise across different stock datasets. In this study, we explore the use of Recurrent Neural Networks (RNNs) for stock price prediction, which are well-suited for handling sequential data and capturing temporal dependencies. We evaluate the performance of various RNN architectures, including vanilla RNN, LSTM, and GRU, on predicting Google stock prices (GOOG) using daily closing prices as input. Hyperparameter tuning was performed to optimise each model's performance, with metrics such as mean squared error (MSE) and loss curves used for evaluation. Our results show that GRU outperformed both the vanilla RNN and LSTM models, achieving the best performance with an MSE of 0.0066.

1. Introduction

The stock market, comprising the primary and secondary markets, is characterised by highly fluctuating and nonlinear time series data. Traditional linear models such as AR, ARMA, and ARIMA have been utilised for stock market prediction, but their applicability is often limited to specific datasets, making them less effective across different companies or scenarios. [2]

Recurrent Neural Networks (RNNs), a subset of deep learning models, have emerged as a powerful alternative for handling the intricate temporal dependencies inherent in stock market data. Unlike traditional models, RNNs can learn complex patterns and relationships over time, enabling them to generalise well across diverse datasets. These models are particularly suited for predicting stock prices, as they can process sequential data like day-wise closing prices and use previous trends to forecast future values.

This study explores the performance of various RNN models on the Google stock prices (GOOG) dataset, including a vanilla RNN model, LSTM, and GRU. Through a comparative analysis, we aim to assess the strengths and limitations of each model, focusing on their ability to address the diverse challenges posed by a time series dataset.

2. Data and method description

2.1. Dataset

We utilise the daily stock market prices dataset from Kaggle [3], which includes data for Forbes2000, NASDAQ, NYSE, and S&P500 listed companies. The dataset contains information such as the date, stock volume, highest stock price, lowest stock price, and daily closing price for each listed company.

The day-wise closing price is used as the predictive variable in a form of time series data, with previous days' prices serving as input to forecast the next day's closing price. Investors often rely on the day-wise closing price to make decisions about whether to buy or sell a stock [2].

For this study, we focus on predicting Google stock prices (GOOG) using Recurrent Neural Networks (RNNs).

2.2. Recurrent Neural Network

Recurrent Neural Networks (RNNs) are specialised neural network architectures designed to process sequential data by considering temporal dependencies between elements in the sequence. Unlike feedforward networks, RNNs maintain a hidden state that acts as memory, allowing them to incorporate information from both the current input and previous states, making them highly suitable for time-series forecasting, natural language processing, and other sequential tasks [2].

The recurrent nature of RNNs is achieved through a feedback loop, where the output from the hidden state at time step $t-1$ serves as input for the hidden state at time step t. This mechanism enables the network to retain contextual information and dynamically update its memory with each new input. As shown in Fig. 1, RNNs effectively model sequential relationships by iterating through the input data while recursively updating their hidden states.

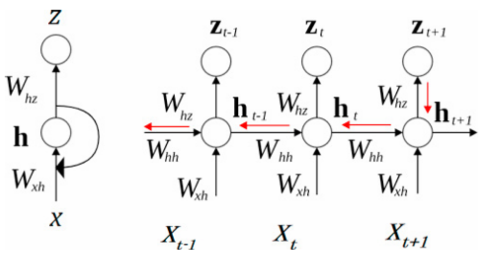


Figure 1. Typical architecture of RNN [2]

The computation within an RNN is defined by two main equations:

$$h_t = g(W_{xh}X_t + W_{hh}h_{t-1} + b_h),$$

where:

- h_t is the hidden state at time t,
- g is the activation function (e.g., $\tanh$),
- W_{xh} is the weight matrix for the input to hidden state,
- W_{hh} is the weight matrix for the hidden state recurrence,
- x_t is the input vector at time t,
- h_{t-1} is the hidden state from the previous time step,
- b_h is the bias term.

The output at each time step is computed as:

$$z_t = g_n(W_{hz}h_t + b_z),$$

where:

- z_t is the output vector at time t,
- W_{hz} is the weight matrix connecting the hidden state to the output layer,
- b_z is the output bias term.

By leveraging these recursive computations, RNNs are capable of capturing complex temporal dependencies, making them an essential tool for sequential data processing tasks. However, basic RNNs face challenges such as vanishing gradients, which limit their ability to learn long-term dependencies, motivating the development of advanced variants like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks.

In this study, we applied three different RNN architecture: vanilla RNN, Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM, considered as a special case of RNN) using Facebook's PyTorch library [1].

3. Implementation

3.1. Dataset preparation

The GOOG stock market price dataset contains Google's stock prices, spanning dates from August 19, 2004, to December 12, 2022, comprising a total of 4,612 data entries. For prediction purposes, we focus on the `Close` column, which represents the stock price at the market's closing on each day. Fig. 2 shows the graph of the GOOG stock's closing prices over this period.

To effectively utilise the data, we preprocess the `Close` column by scaling its values to the range of -1 to 1. During the data splitting process into training, testing, and validation sets, the data is organised in sequential order, as predictions rely on the prices of the previous N days to forecast the next day's price. This approach, known as the sliding window method [4], is implemented with a lookback period N of 20 days. After we split the data, we convert them into tensor, as a requirement for PyTorch library.

3.2. Vanilla Recurrent Neural Network

The vanilla RNN we implemented follows a simple recurrent neural network (RNN) architecture, consisting of a single recurrent layer (`nn.RNN`) followed by a fully connected linear layer (`nn.Linear`). The recurrent layer processes sequential input data by maintaining a hidden state, which is updated at each time step based on the current input and the previous hidden state. The final hidden state is passed through the fully connected layer to produce the output. This architecture employs the tanh activation function by default in the recurrent layer and does not apply any activation function to the output layer. Algorithm 1 shows the algorithm of our vanilla RNN.

Algorithm 1 Basic RNN Model

```
1: Input: Sequential input data $x_{\text{in}}$ with shape $(batch\_size, seq\_len, input\_dim)$
2: Initialise RNN layer rnn with:
3:     Input dimension input_dim
4:     Hidden dimension hidden_dim
5:     Number of layers num_layers
6:     batch_first=True
7: Initialise fully connected layer fc with:
8:     Input dimension hidden_dim
9:     Output dimension output_dim
10: Forward Pass:
11:     Initialise hidden state $h_0$ with zeros:
12:     $h_0 = \text{zeros}(\text{num\_layers}, \text{batch\_size}, \text{hidden\_dim})$
13:     Move $h_0$ to device (e.g., GPU): $h_0 = h_0.to(device)$
14:     Apply RNN layer on $x_{\text{in}}$ and $h_0$:
15:     $\text{out}, h_n = \text{rnn}(x_{\text{in}}, h_0)$
16:     Extract the output of the last time step:
17:     $\text{out} = \text{out}[:, -1, :]$
18:     Apply the fully connected layer:
19:     $y_{\text{out}} = \text{fc}(\text{out})$
20: Output: Predicted output $y_{\text{out}}$ with shape $(batch\_size, output\_dim)$
```

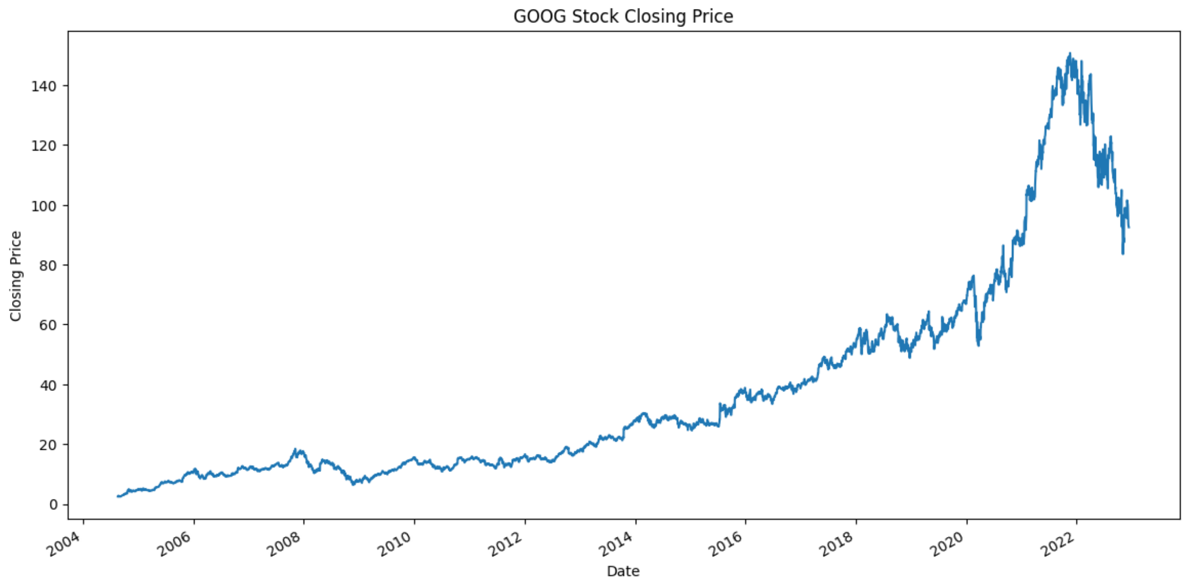


Figure 2. GOOG closing stock price from 19 August 2004 to 12 December 2022

3.3. Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a specialised type of Recurrent Neural Network (RNN) designed to effectively learn long-term dependencies. Introduced by Hochreiter and Schmidhuber in 1997, LSTMs address the vanishing gradient problem often encountered in standard RNNs, enabling them to retain and utilise information over extended time periods. This makes LSTMs particularly suitable for tasks involving sequential data with long-term dependencies, such as speech recognition, time-series forecasting, and natural language processing [2].

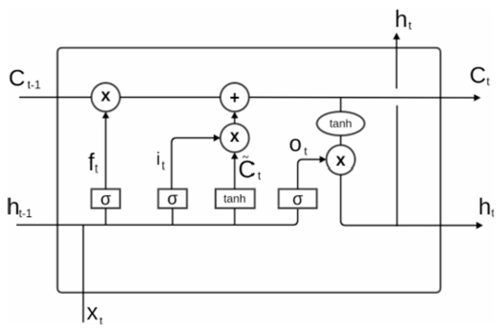


Figure 3. Diagram of LSTM cell [2]

Unlike conventional RNNs, which rely on simple feedback loops, LSTMs consist of memory blocks, referred to as cells, that are equipped with gates to regulate the flow of information. These gates—input gate (i_t), forget gate (f_t), and output gate (o_t)—work together with a cell state (C_t) to manage the storage and retrieval of information. A diagram of an LSTM cell is shown in Fig. 3.

The horizontal line running through the top of the LSTM diagram represents the cell state (C_{t-1}, C_t), which acts as a conveyor belt, carrying information across the network. This cell state allows LSTMs to selectively retain or discard information based on the decisions made by the gates [2].

3.4. Gated Recurrent Unit

Gated Recurrent Unit (GRU) networks are a type of Recurrent Neural Network (RNN) designed to efficiently learn long-term dependencies and address the vanishing and exploding gradient problems encountered in traditional RNNs. Introduced as an improvement over earlier RNN architectures, GRUs achieve a balance between complexity and performance by using fewer parameters than Long Short-Term Memory (LSTM) networks, making them computationally efficient while still maintaining the ability to model sequential data effectively [5].

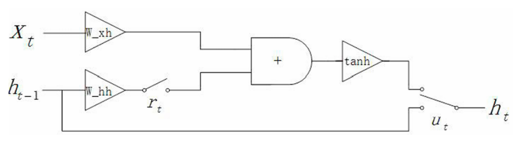


Figure 4. Diagram of GRU cell [2]

The GRU architecture differs from standard RNNs by incorporating gating mechanisms to regulate the flow of information. Specifically, GRUs utilise two gates: the reset gate (r_t) and the update gate (u_t), which together determine the extent to which past information influences the current computation and how much of the current input should be passed to the output. The structure of a GRU cell is depicted in Fig. 4.

The operations within a GRU cell can be summarised as follows:

1. **Reset Gate:** The reset gate controls how much of the previous hidden state (h_{t-1}) should be ignored when computing the current state. It is computed as:

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

   where W_r and b_r are the weights and biases for the reset gate, x_t is the input at time step t, and h_{t-1} is the previous hidden state.

2. **Update Gate:** The update gate decides the extent to which the previous hidden state should contribute to the current hidden state:

$$u_t = \sigma(W_u[h_{t-1}, x_t] + b_u),$$

   where W_u and b_u are the weights and biases for the update gate.

3. **Candidate Hidden State:** A candidate hidden state ($\tilde{h}_t$) is computed by combining the current input and the reset gated previous hidden state:

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

   where W_h and b_h are the weights and biases for the candidate state, and $\odot$ represents element-wise multiplication.

4. **Hidden State Update:** The final hidden state is a combination of the candidate state and the previous hidden state, weighted by the update gate:

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

4. Experiments

In this study, we conducted two experiments. In the first experiment, we used the same hyperparameter values across the three RNN models being tested: Vanilla RNN, LSTM, and GRU. The second experiment involved performing hyperparameter tuning using grid search to find the best model and optimal hyperparameters for predicting the GOOG stock price. For the experiments, we will use mean-squared error (MSE) as our performance metric.

Hyperparameters	Value
Hidden dimension	32
Number of layers	2
Optimiser	Adam
Learning rate	0.01
Number of epochs	100

Table 1. RNN model hyperparameters

4.1. Model training

Our RNN models were trained using the hyperparameters listed in Tab. 1. The Adam optimiser was used with its default learning rate of 0.01. Each model consisted of 32 hidden neurons and 2 stacked RNN layers, making them multi-layered architectures. Training was conducted over 100 epochs to ensure sufficient learning. In all experiments, the mean squared error (MSE) loss function was employed for backpropagation and for evaluating the performance of the models.

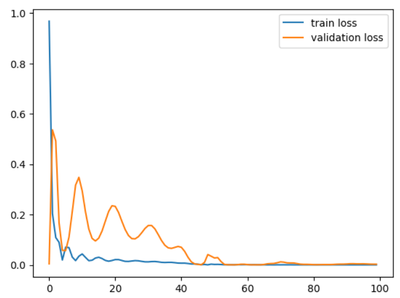
The training results are shown in Fig. 5. Based on the loss curves, we suspect that the LSTM model suffers from underfitting, while the vanilla RNN and GRU models perform relatively well. After fitting the models to the test set, we calculated the mean-squared error (MSE), which resulted in 0.0080 for the vanilla RNN, 0.4021 for LSTM, and 0.0085 for GRU, confirming the underperformance of the LSTM model. The comparison between predicted and actual stock prices, as shown in Fig. 6, further supports this observation of the underperformance of LSTM.

The training processes took between 3 to 11 seconds to complete, indicating they were not computationally intensive. Based on this, we decided to perform hyperparameter tuning on all three models to determine whether the LSTM model's performance could be better or if further improvements could be made to the other two models.

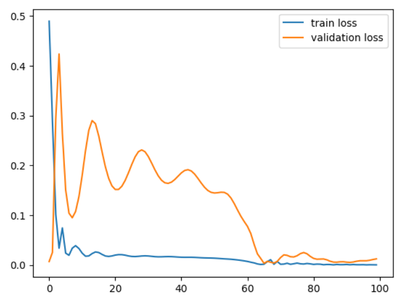
4.2. Hyperparameter tuning

The second experiment we conducted was hyperparameter tuning on all three models. We used grid search for tuning:

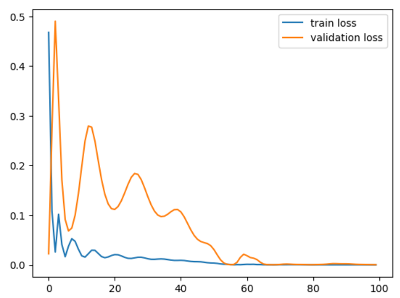
- **Models**: We evaluated all three models: RNN, LSTM, and GRU, to compare their performance.
- **Learning rate**: We kept the default learning rate of 0.01, as it already performed well, and added a smaller rate of 0.001 to test for potential improvements in stability and convergence.
- **Hidden dimension**: To capture more complex patterns in the data, we tested larger hidden dimensions of 64 and 128.



(a) RNN loss over epoch curve

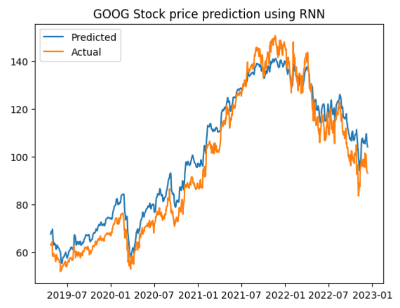


(b) LSTM loss over epoch curve

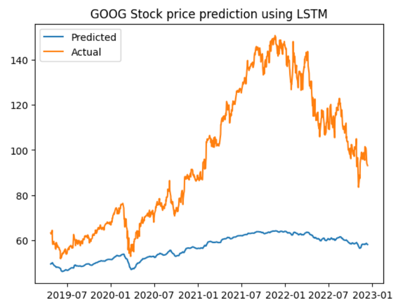


(c) GRU loss over epoch curve

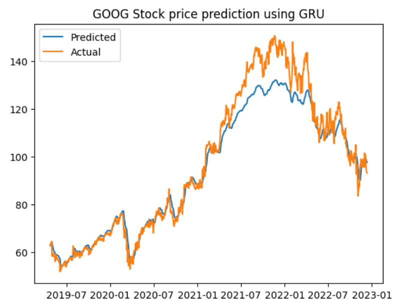
Figure 5. Loss curves of RNN models



(a) RNN model performance on predicting the stock price



(b) LSTM model performance on predicting the stock price



(c) GRU model performance on predicting the stock price

Figure 6. Predictions of GOOG stock prices using RNN models

- **Number of layers**: While the two-layer configuration worked well, we explored using three layers to increase the model's capacity and potentially capture more complex temporal dependencies.
- **Number of epochs**: Observing underfitting in the LSTM model, we increased the number of training epochs to provide the model with more opportunity to learn and improve.

We measured the models' performance by its validation MSE, with results for each hyperparameter combination are shown in Tab. 2. Among the top ten models, none of them is an LSTM model, confirming that it is not ideal for our case. The GRU models seem to be dominating, with the top two showing only a slight difference in MSE values. The only distinction between them and the first model we tested lies in the hidden dimension size and the number of epochs. Fig. 7 shows the loss curves of our best model.

4.3. Performance analysis

To finalise this study, we evaluated our best model using the test set, achieving an MSE of 0.0066, which represents a 17% improvement over the best result from our first experiment. The comparison between the predicted and actual stock prices is shown in Fig. 8.

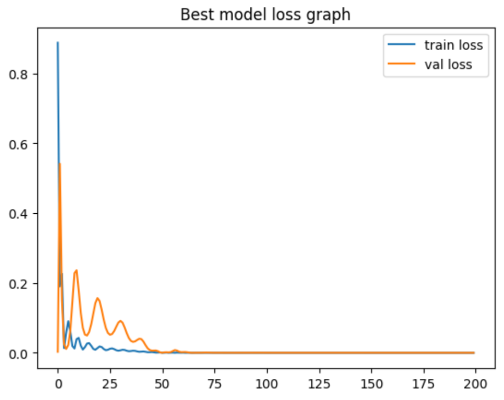


Figure 7. Loss curves of the best model

4.4. Code

The complete code for this study, including different model implementations, experiments, and hyperparameter tuning, is available on GitHub:

GitHub Repository: COMP SCI 7318 Deep Learning Fundamentals Assignment 3: RNNs for stock price prediction (https://github.com/ifboelhasrin/DLF_A3)

The repository is publicly available, so no access permissions are required.

Model	Hidden Dimension	Number of Layers	Number of Epochs	Learning Rate	Mean-squared Error
GRU	64	2	200	0.010	0.020125
GRU	128	2	200	0.010	0.020189
RNN	128	2	200	0.001	0.029123
GRU	32	3	200	0.010	0.029760
GRU	128	3	200	0.010	0.030552
RNN	128	3	200	0.001	0.033798
GRU	128	3	100	0.010	0.033919
GRU	32	2	200	0.010	0.034273
RNN	32	2	200	0.010	0.036491
GRU	128	2	200	0.001	0.037144

Table 2. Hyperparameter tuning results for RNN, LSTM, and GRU models.

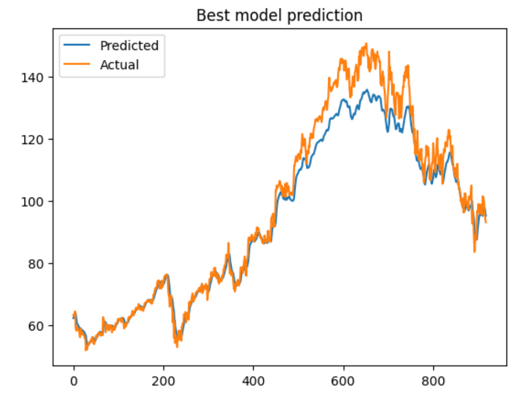


Figure 8. Comparison between

5. Conclusion

In this study, we implemented and evaluated various Recurrent Neural Network (RNN) models, including a vanilla RNN, Long Short-Term Memory (LSTM) networks, and Gated Recurrent Unit (GRU) networks, for time series prediction on GOOG stock market price. After experimenting with different models and fine-tuning key hyperparameters such as hidden dimensions, number of layers, number of epochs, and learning rates, we observed that our GRU model achieved the best performance, with an MSE of 0.0066.

While the vanilla RNN model showed competitive results against the GRU, the LSTM model appeared to underperform in our case, particularly struggling with underfitting. The results of these experiments highlight the importance of selecting the right model and fine-tuning hyperparameters to optimise performance when using time series data for predictions, such as stock market prices.

Future work could explore additional recurrent models, such as attention-based mechanisms or Transformer networks, to better capture long-range dependencies in time series data. It would also be valuable to experiment more with LSTM to find the underlying problem of its underperformance. Additionally, techniques like transfer learning, where pre-trained models are fine-tuned on financial data, or reinforcement learning, for dynamic trading strategies, could be explored to enhance prediction accuracy and model adaptability.

References

[1] A Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2019. 2

[2] M Hiransha, EA Gopalakrishnan, VK Menon, and KP Soman. Nse stock market prediction using deep-learning models. *Procedia Computer Science*, 132:1351–1362, 2018. 1, 2, 3

[3] P Mooney. Stock market data (nasdaq, nyse, sp500), 2023. Accessed: 2024-11-30. 1

[4] S Selvin, R Vinayakumar, EA Gopalakrishnan, VK Menon, and KP Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1643–1647, 2017. 2

[5] G Shen, Q Tan, H Zhang, P Zeng, and J Xu. Deep learning with gated recurrent unit networks for financial sequence predictions. *Procedia Computer Science*, 131:895–903, 2018. Recent Advancement in Information and Communication Technology:. 3