# Deep Learning Fundamentals - Assignment 3

Gauranga Das
University of Adelaide
a1910652@adelaide.edu.au

## Abstract

*This project investigates the use of Recurrent Neural Networks (RNNs) and their derivatives for stock price prediction based on historical data. Vanilla RNNs, Long Short-Term Memory networks (LSTMs), Gated Recurrent Units (GRUs), and GRUs with attention mechanisms were implemented and evaluated. The models were trained on Google's stock price dataset, with hyperparameter tuning conducted for LSTM, GRU, and GRU with Attention using different hidden state sizes. Performance was measured using the Root Mean Squared Error (RMSE) on the validation set, with GRU augmented with attention achieving the lowest RMSE of 1.8624. However, when evaluated on the test set, the best-performing model achieved an RMSE of 13.2492, highlighting challenges such as distribution shifts between the training and test sets and overfitting. The results demonstrate the effectiveness of GRUs and attention mechanisms for sequential data tasks while emphasizing the importance of robust data preprocessing and regularization techniques. Future work could explore advanced architectures like Transformers to further enhance prediction accuracy.*

## 1. Introduction

Predicting stock market prices is challenging due to the volatility of financial markets. Computer systems that can perform accurate stock market prediction can help investors and day-traders in making optimal decisions to maximize their portfolio gains. In this project, the aim is to predict stock market prices using RNNs, which are excellent are processing sequential data, making them suitable for time-series data such as stock prices.

The goal of this project is to perform stock market price prediction using RNNs and their derivatives. The main objective is to explore various types of RNNs, tune their hyperparameters, evaluate their performance, and analyze the results. Multiple types of RNN architectures, including Vanilla RNN [11], LSTM [5], and GRU [4], are trained on stock data and evaluated using the RMSE performance metric to select the best model. Additionally, the effect of the attention mechanism is also explored.

In this project, the past $N = 30$ days of historical data is used to predict the closing price for the next day ($M = 1$). Such a model can assist in making stock purchase decisions based on the past 30 days of data. If the closing price predicted by the model for the next day is significantly higher than today's closing price, it suggests that the stock might be worth buying. Stock market data [9] from Kaggle is used for this project.

## 2. Related work

There are several effective approaches for stock price prediction, each with its strengths and limitations. Traditionally, statistical methods such as Autoregressive Integrated Moving Average (ARIMA) [3, 8] were widely used. These models are particularly effective for capturing linear dependencies but often struggle with the complex, nonlinear dependencies characteristic of stock price data.

Reinforcement Learning (RL) has gained significant popularity for stock market prediction [6, 1]. RL approaches, such as Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO), train agents to make sequential decisions, including when to buy, sell, or hold a stock. Unlike other methods, RL focuses on maximizing cumulative returns rather than directly predicting prices, making it highly effective for dynamic trading environments. However, RL requires large amounts of high-quality data and computational resources, and its performance can be highly sensitive to the design of the reward function and market volatility.

In recent years, Transformer-based architectures [14] have gained prominence, particularly in natural language processing (NLP) tasks. They have also been applied to stock market prediction [15]. Transformers rely on self-attention mechanisms to capture long-term dependencies, making them highly effective for sequential data. However, their computational cost increases significantly with longer sequences, which can make them less practical for tasks involving large datasets or limited computational resources.

In contrast, RNNs and their variants, such as LSTMs and

GRUs, are specifically designed for sequential data and can capture temporal dependencies effectively. Additionally, the incorporation of attention mechanisms enhances RNN performance by allowing the model to focus on relevant parts of the input sequence. Their relatively lower computational cost compared to Transformers and their ability to model long-term dependencies with mechanisms like gating (in LSTMs and GRUs) make them suitable for financial time-series tasks.

## 3. Method

Our goal is to perform stock market prediction the Google Stock Price dataset using different types of RNNs and then compare their performance.

### 3.1. Vanilla RNN

The baseline model will be a vanilla RNN architecture. RNNs are a type of neural network suitable for processing sequential data. RNNs process each element of the sequence one at a time. It maintains a hidden state that captures information about all the elements seen so far. The hidden state is updated using the input at any given time-step and the previous hidden state.

$$\begin{aligned}\text{Hidden State: } h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t)\\ \text{Output: } y_t &= W_{hy}h_t\end{aligned}$$

Vanilla RNNs are updated using Backpropagation Through Time (BPTT), a process that involves repeatedly multiplying matrices during the calculation of gradients. This often causes Vanilla RNNs to struggle with learning long-term dependencies due to the vanishing gradient problem, where gradients shrink exponentially as they propagate backward through time. Despite this limitation, Vanilla RNNs were chosen as a baseline model because they are relatively simple to implement and are computationally inexpensive compared to other more complex neural network architectures.

### 3.2. LSTM

Long Short-Term Memory (LSTM) networks are special type of RNN that is able to overcome many of the limitations of vanilla RNN such as capturing long-term dependencies. LSTM achieves this by using three gating mechanisms: forget gate, input gate and output gate which are used to update the hidden state and memory cell. The memory cell is used to remember long-term information and hidden state is used to remember short-term information and to display output. Forget gate decides which information should be discarded, input gate decides what information should be added to the memory and output gate decides what information should be passed to the next hidden state.

$$\begin{aligned}\text{Input Gate: } i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1})\\ \text{Forget Gate: } f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1})\\ \text{Output Gate: } o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1})\\ \text{Cell Gate: } \tilde{c}_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1})\\ \text{Cell State: } c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t\\ \text{Hidden State: } h_t &= o_t \odot \tanh(c_t)\end{aligned}$$

To improve the performance of LSTM, a two-layer LSTM is used where the hidden states generated from the first LSTM is used as input for the next layer LSTM.

### 3.3. Gated Recurrent Units

Gated Recurrent Units (GRUs) are a simpler variant of LSTM that are less complex while still able to achieve similar performance. GRUs have only two gating mechanisms: update gate and reset gate and maintain only one hidden state. Update gate decides how much of the previous hidden state and current input should be retained. Reset gate is used to decide how much of the previous hidden state to forget.

$$\begin{aligned}\text{Reset Gate: } r_t &= \sigma(W_{xr}x_t + W_{hr}h_{t-1})\\ \text{Update Gate: } z_t &= \sigma(W_{xz}x_t + W_{hz}h_{t-1})\\ \text{Candidate Hidden State: } \tilde{h}_t &= \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}))\\ \text{Hidden State: } h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t\end{aligned}$$

Similar to LSTM, a two-layer GRU is used.

### 3.4. GRU with Attention

To further improve the performance of the GRU, an attention mechanism [2] is applied. Let $h$ represent the hidden state from the final layer of the GRU at the last time step, and let $o_t$ denote the output of the final layer of the GRU at time step $t$. The attention mechanism is described as follows:

$$\begin{aligned}\text{Score: } s_t &= o_t^T h\\ \text{Normalized Scores: } \alpha_t &= \frac{\exp(s_t)}{\sum_{j=1}^{T}\exp(s_j)}\\ \text{Context Vector: } c_t &= \sum_{i=1}^{T}\alpha_t o_t\end{aligned}$$

The dot product of $o_t$ and $h$ is computed to determine the attention score for time step $t$. This score is then normalized using a softmax function to calculate the amount of attention to be assigned to time step $t$. Finally, the weighted sum of all the output vectors is computed as $c_t$ which is the context vector summarizing the relevant information from the entire sequence.

### 3.5. Google Stock Price Dataset

For this project, the Stock Market Data from Kaggle is used. The dataset contains the stock price history of many companies. For the experiments, only Google's stock price data is utilized. The dataset is named `GOOG.csv` and is located in the `sp500` folder. It includes seven features: Date, Low, Open, Volume, High, Close, and Adjusted Close. The entire dataset consists of 4,612 data points.

## 4. Experiments

Different types of RNNs were trained and evaluated on the Google Stock Price dataset. The experiments illustrate their relative strengths and weaknesses.

**Data Preprocessing**. Stock price history of Google is used from Kaggle. The dataset contains seven columns, of which only five are used as input features: Open, High, Low, Close, and Volume. The data is split into training, validation, and test sets in a ratio of 0.8:0.1:0.1. Since the features have different ranges, they are scaled using the Min-Max scaling method [13], which scales the values to the range $[0, 1]$.

**Hyperparameter Optimization**. Four different types of RNNs are trained: Vanilla RNN (baseline), LSTM, GRU, and GRU with Attention. Hyperparameter optimization is performed for each of them except for the baseline model. One important hyperparameter for RNNs is the hidden state size. For each type of RNN, hidden state sizes of 32, 64, and 128 are used to train models, which are then evaluated on the validation set. The results on the validation set are used to select the best model.

**Evaluation Methods**. Different hyperparameters are tested for each type of RNN. For each RNN type and its corresponding hyperparameter, the model is trained on the training set for 100 epochs and then evaluated on the validation set. The input features from the validation set are fed into the model to generate the predicted closing stock prices, while the actual closing stock prices are obtained from the output features of the validation set. The scaler is used to unscale the predicted and actual closing prices. The Root Mean Squared Error (RMSE) [12] is computed between the actual and predicted closing stock prices which is then used as the metric to evaluate the RNN type and it's hyperparameter. The model with the lowest RMSE on the validation set is selected as the best model. Finally, the best model is evaluated on the test set.

**Training**. All models are trained using the Adam [7] optimizer with a default learning rate of 0.001 and a batch size of 32. The mean-squared error (MSE) loss function [10] is used to compute the loss between the predicted prices and the actual prices. The Vanilla RNN is trained with a hidden state size of 16, while the LSTM, GRU, and GRU with attention are trained using hidden state sizes of 32, 64, and 128. Each of these models (except for the Vanilla RNN) is implemented with two layers.

Table 1. RMSE of various models on the validation set

| RNN Type | Hidden State Size | RMSE |
|---|---|---|
| Vanilla RNN | 16 | 2.6009 |
| LSTM | 32 | 2.2974 |
| LSTM | 64 | 3.5564 |
| LSTM | 128 | 2.7797 |
| GRU | 32 | 1.8994 |
| GRU | 64 | 2.8100 |
| GRU | 128 | 2.2614 |
| GRU with Attention | 32 | 1.9777 |
| **GRU with Attention** | **64** | **1.8624** |
| GRU with Attention | 128 | 2.8169 |

Table 2. RMSE of best model on the test test

| RNN Type | Hidden State Size | RMSE |
|---|---|---|
| GRU with Attention | 64 | 13.2492 |

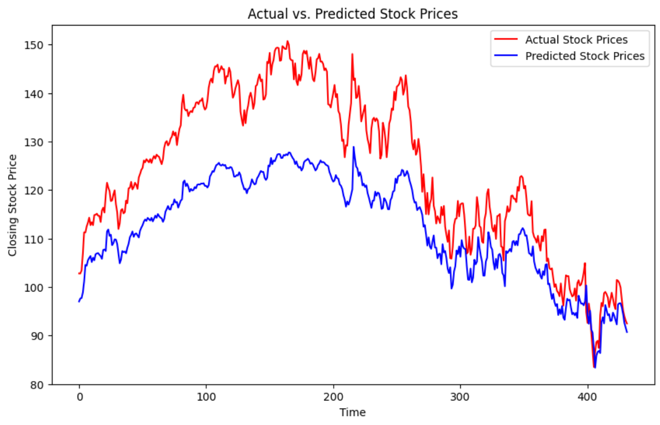

Figure 1. Plot of actual and predicted closing prices of the best model on the test set

**Results**. Table 1 provides the RMSE for different types of RNNs and their corresponding hidden state sizes on the validation set. The Vanilla RNN has a high RMSE of 2.6009, which could be due to its inability to capture long-term dependencies which is caused by vanishing gradients problem. Accurately predicting stock prices requires analyzing many days' worth of data. The LSTM generally performs better than the Vanilla RNN, achieving a lowest RMSE of 2.2974 with a hidden state size of 32. LSTMs are generally better than Vanilla RNNs at capturing long-term dependencies due to their ability to remember long-term information using a memory cell, which explains their better performance. However, increasing the hidden state size to 64 leads to a degradation in performance, with an RMSE of 3.5564, indicating potential overfitting due to the model becoming more complex. Further increasing the hidden state size to 128 lowers the RMSE to 2.7797, but it is still worse than the LSTM with a hidden state size of 32, suggesting that overfitting persists.

The GRU outperforms both Vanilla RNN and LSTM, achieving an RMSE of 1.8994 with a hidden state size of 32. GRUs outperform LSTMs because they are simpler and have fewer parameters, making them more robust to overfitting. However, increasing the hidden state size to 64 and 128 leads to worse RMSE values of 2.8100 and 2.2614, respectively. Finally, adding an attention mechanism to the GRU further improves its performance. The GRU with attention and a hidden state size of 64 achieves an RMSE of 1.8624, making it the best-performing model. This indicates that allowing the model to focus on relevant parts of the sequence improves predictions. The GRU with attention and a hidden state size of 32 achieves a slightly worse RMSE of 1.9777, and increasing the hidden state size to 128 results in a much worse RMSE of 2.8169, likely due to overfitting.

Overall, the GRU with attention performs the best. The Vanilla RNN is too simple and ineffective at modeling long-term dependencies. LSTMs are better, but their complexity makes them prone to overfitting, especially on small datasets. GRUs strike a good balance between simplicity and performance, allowing them to outperform both Vanilla RNNs and LSTMs. Across all RNN types, models with smaller hidden state sizes (32 or 64) perform better than those with larger hidden state sizes. This may be because the dataset is relatively small. Larger models typically require more data to train effectively. When large models are trained on small datasets, they tend to overfit, leading to worse performance on the validation set.

Finally, the best-performing model is evaluated on the test set. Table 2 shows that it achieves an RMSE of 13.2492, which is significantly higher than its validation performance. Several reasons could explain this poor performance on the test set. First, the original data is in chronological order. The training set consists of the first 80% of the data, while the test set comprises the last 10%. Stock prices generally tend to increase over time, meaning the stock prices in the test set are likely much higher than those in the training set and therefore have a different distribution. Since the model was trained on lower stock prices, it struggles to accurately predict the much higher prices in the test set. Figure 1 further illustrates this by plotting the actual and predicted stock prices. Even though the predicted stock prices follow the trend of the actual stock prices, the predicted prices consistently underestimate the actual prices, supporting the earlier explanation of the distribution shift. Another potential reason could be that the model overfitted to the training and validation sets, leading to poor generalization and, consequently, worse performance on the test set.

## 5. Code

GITHUB LINK

## 6. Conclusions

In this paper, various types of RNNs, including Vanilla RNNs, Long Short-Term Memory networks (LSTMs), Gated Recurrent Units (GRUs), and GRUs with attention mechanisms, were used to predict stock prices based on historical data. The main goal was to evaluate multiple models, analyze their results, and select the best-performing one for evaluation on the test set.

Among the models, Vanilla RNNs had the highest RMSE, highlighting their inability to capture long-term dependencies in sequential data. LSTMs slightly improved upon Vanilla RNNs but sometimes performed worse, with a tendency to overfit as the hidden state size increased. GRUs outperformed both Vanilla RNNs and LSTMs, achieving a lower RMSE due to their simplicity and their ability to retain long-term information through gating mechanisms. Finally, GRUs with attention mechanisms performed the best, achieving an RMSE of 1.8624 on the validation set. This result indicates that incorporating attention mechanisms into RNNs can significantly enhance their performance by enabling the model to focus on the most relevant parts of the input sequence.

When evaluated on the test set, the best-performing model achieved an RMSE of 13.2492, which is significantly higher than the RMSE achieved on the validation set. This discrepancy can be attributed to differences in the data distributions of the training and test sets, as stock prices generally increase over time. Another possibility is that the model overfitted the training and validation sets, leading to poor generalization on unseen data.

Future work should address the challenges identified in this study by employing data preprocessing techniques to ensure that the training, validation, and test splits have similar distributions. The significant performance gap between the validation and test sets suggests that distributional shifts negatively impacted the model's generalization. To mitigate this, the parameters of the scaler could be regularly updated in real-world applications to adapt to changing data distributions, ensuring that the data fed to the model matches the distribution of the training data. The experiments also showed that larger hidden state sizes often led to worse performance due to overfitting, particularly for LSTMs and GRUs. To address this, regularization methods such as dropout and weight decay could be applied, which are known to mitigate overfitting in complex models. These techniques would enable larger models to generalize better to unseen data. Lastly, Transformers, which make predictions purely based on attention mechanisms, could be explored as an alternative to RNNs. While GRUs with attention mechanisms demonstrated the best performance in this project, the experimental results highlight the benefits of focusing on relevant parts of the input sequence. Transformers are designed to leverage attention exclusively and have shown exceptional performance in capturing long-term dependencies, suggesting that they may outperform RNN-based architectures for stock price prediction tasks.

In conclusion, this project demonstrates the effectiveness of GRUs and attention mechanisms for stock price prediction, while emphasizing the importance of robust data preprocessing and strategies to mitigate overfitting for real-world applications.

## References

[1] Anil Berk Altuner and Zeynep Hilal Kilimci. A novel deep reinforcement learning based stock direction prediction using knowledge graph and community aware sentiments. *Turkish J. Electr. Eng. Comput. Sci.*, 30:1506–1524, 2021.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[3] George E. P. Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, 1970.

[4] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics, 2014.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Taylan Kabbani and Ekrem Duman. Deep reinforcement learning approach for trading automation in the stock market. *IEEE Access*, 10:93564–93574, 2022.

[7] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[8] Bharat Kumar Meher, Iqbal Thonse Hawaldar, Cristi Spulbar, and Ramona Birau. Forecasting stock market prices using mixed arima model: A case study of indian pharmaceutical companies. *Investment Management and Financial Innovations*, 18(1):42–54, 2021.

[9] Paul Mooney. Stock Market Data (NASDAQ, NYSE, S&P500). `https://www.kaggle.com/datasets/paultimothymooney/stock-market-data`, 2022. Kaggle.

[10] Pytorch. MSELoss. `https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html`. (accessed 15 Nov. 2024).

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

[12] sklearn. mean_squared_error. `https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.mean_squared_error.html`. (accessed 16 Nov. 2024).

[13] sklearn. MinMaxScaler. `https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.MinMaxScaler.html`. (accessed 1 Dec. 2024).

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[15] Jue Xiao, Tingting Deng, and Shuochen Bi. Comparative analysis of lstm, gru, and transformer models for stock price prediction. *arXiv preprint arXiv:2411.05790*, 2024.