Assignment 3 - Deep Learning Fundamentals

Dang Thinh Nguyen
The University of Adelaide
dangthinh.nguyen@student.adelaide.edu.au

Abstract

This study explores the use of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks for predicting stock prices based on historical market data. The dataset, consisting of features such as open, high, low, close prices, and trading volume, was preprocessed to address invalid values, normalize features, and generate temporal sequences. The models were trained and evaluated using various hyperparameter configurations, with performance assessed using Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The LSTM model slightly outperformed the RNN, achieving an MSE of 0.002 and an RMSE of 0.0447 on the validation dataset, compared to the RNN's MSE of 0.00207 and RMSE of 0.04548. On the test dataset, the LSTM achieved an MSE of 0.00711 and an RMSE of 0.08433, demonstrating reasonable generalization to unseen data. While both models effectively captured temporal patterns, the LSTM's architecture provided a marginal edge in accuracy. Challenges such as hyperparameter tuning, sequence length selection, and overfitting were encountered, emphasizing the complexity of financial forecasting tasks. The results suggest that both RNN and LSTM models are viable for stock price prediction, with opportunities for improvement through the integration of external features, hybrid architectures, and automated optimization techniques.

1. Introduction

Forecasting stock prices has long been a challenge for researchers and analysts [1]. Investors are particularly interested in accurate predictions to guide their investment decisions, as understanding the future trend of the stock market is critical for effective and profitable strategies [1]. The complexity of the stock market, influenced by numerous interconnected variables, makes accurate forecasting a demanding task [1].

The advent of machine learning, particularly deep learning methods, has opened new avenues for stock price prediction. One of the most promising techniques is Recurrent Neural Networks (RNNs) designed to handle sequential data and capture temporal dependencies [2]. RNNs utilize feedback loops to retain memory of previous inputs, making them ideal for tasks such as time-series analysis [2]. However, its effectiveness can be hampered by issues like vanishing or exploding gradients [2]. Long Short-Term Memory (LSTM) networks, an advanced variant of RNNs, address these limitations through specialized memory cells and gating mechanisms, enabling them to model long-term dependencies effectively [2].

In this study, we employ RNNs and LSTM networks to predict stock market indices by analyzing historical data. Our objective is to evaluate the effectiveness of these models in forecasting multivariate outputs, thereby providing valuable insights into their capabilities for stock market prediction. The results aim to benefit traders, investors, and analysts by offering a robust framework for understanding market dynamics and making informed decisions.

2. Background

This section describes the general architecture of two methods: Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) Networks, and the potential of their application in this study.

2.1. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks designed for processing sequential data. Unlike traditional feedforward networks, RNNs incorporate loops within their architecture, enabling them to maintain a "memory" of previous inputs [3]. This capability makes them suitable for tasks involving temporal or sequential dependencies, such as time-series forecasting, natural language processing, and speech recognition [3].

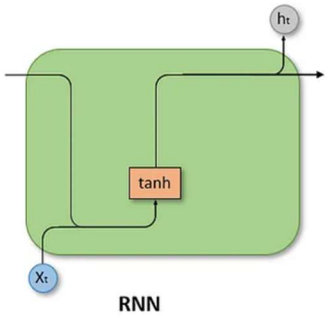

Figure 1. RNN architecture [4]

An RNN processes input data sequentially, updating its hidden state h_t at each time step t as follows:

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

where:

- W, U, b: learnable parameters
- x_t: the input at time t
- σ: non-linear activation function

The hidden state h_t encodes all previous time steps, enabling the network to model temporal relationships.

During backpropagation, gradients can become extremely small (vanishing) or excessively large (exploding). This makes it difficult for the model to learn long-term dependencies, as the gradients associated with earlier time steps diminish or explode over time. Conversely, RNNs are better at capturing short-term dependencies.

2.2. Long Short-Term Memory (LSTM) Networks

LSTMs are a specialized type of RNN designed to overcome the limitations of standard RNNs, which address the limitations of RNNs by introducing memory cells and gating mechanisms to regulate the flow of information [2].

LSTM architecture

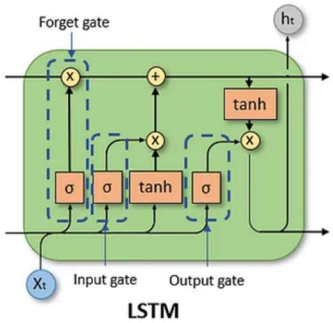

Figure 2. LSTM architecture [4]

Each LSTM cell has three primary gates:

1. Forget gate: decides which information from the previous cell state to discard.
2. Input gate: determines which new information to add to the cell state.
3. Output gate: controls which part of the updated cell state contributes to the output.

These gates allow LSTMs to selectively retain or forget information, effectively addressing the vanishing gradient problem [2]. The architecture ensures that essential information can persist across long time periods, enabling LSTMs to capture both short-term and long-term dependencies.

Approaching for Stock Price Prediction

Despite their limitations, RNNs remain a valuable tool for modeling sequential data due to their simplicity and ability to capture basic temporal dependencies. They serve as a foundation for more advanced architectures like Long Short-Term Memory (LSTM) networks.

In this study, we leverage both RNNs and LSTM networks to predict the next day's features based on historical data, thereby having deeper insights into the effectiveness of the two models in forecasting multivariate outputs.

3. Methodology

This section outlines the systematic approach undertaken for data preprocessing, model design, and training. Each step is crucial to ensure that the model learns the most relevant features from the dataset while minimizing errors in prediction.

3.1. Data preprocessing

Effective data preprocessing is one of the most crucial steps in the development of machine learning models, particularly in medical datasets where missing values, outliers, and scale discrepancies are common. In this study, preprocessing was performed to handle invalid data, to normalize the feature scales, and to ensure that the model was exposed to clean and well-structured data.

Handling invalid data

Certain historical data in the dataset contained invalid values of Closing price with double. These were replaced with half of their value to get the real value of the Close price. These invalid values likely stem from errors during data collection or measurements due to wrong multiplication with factors.

Normalizing

After handling invalid data, all features were normalized using Min-Max scaling, transforming them into a range between 0 and 1.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This step is crucial for ensuring numerical stability, especially for neural networks sensitive to feature scale differences.

Data splitting

The dataset has two subsets: training dataset and testing dataset. The training dataset was then split into two sets: a training set (80%) and a validation set (20%). The training set was used to fit the model, while the validation set allowed for hyperparameter tuning and performance monitoring during training. The test set was held back to evaluate the model's generalization ability after training.

To enable the model to learn temporal dependencies, sequences were created from the dataset. Each sequence consisted of data from a fixed number of consecutive time steps, referred to as the sequence length. For this study, a sequence length was chosen based on hyperparameter tuning.

3.2. Model design

The RNN and LSTM models used in this study consist of an input layer with 5 features, an output layer with 5 features and a number of layers selected by optimizing model. Its purpose is to predict the following day's features based on the input features.

RNN architecture

The RNN model architecture consists of:

1. Input layer: sequences of shape (30, 5) representing 30 days of 5 features each.
2. RNN layer: number of LSTM layers with 32 units.
3. Dropout layer: is to prevent overfitting.
4. Dense layer: shapes (1, 5) output units for predicting the next day's features.

LSTM architecture

The LSTM model architecture consists of:

1. Input layer: sequences of shape (30, 5) representing 30 days of 5 features each.
2. LSTM layer: number of LSTM layers with 32 units.
3. Dropout layer: is to prevent overfitting.
4. Dense layer: shapes (1, 5) output units for predicting the next day's features.

3.3. Training process

After designing the model and preparing the data, training was initiated. The RNN and LSTM models were trained over 50 epochs, where each epoch consists of passing the entire training dataset through the model. The model's weights were updated after each mini-batch of data was processed.

During training, the loss was calculated on both the training and validation sets at the end of each epoch. This allowed for monitoring the model's performance.

Three optimization algorithms were evaluated:

1. Stochastic Gradient Descent (SGD), which updates the weights based on the gradient of the loss function calculated on a single mini-batch [5].
2. Adam, an adaptive optimizer that adjusts the learning rate dynamically during training [5].
3. RMSprop, another adaptive optimizer that modifies the learning rate based on the magnitude of recent gradients [5].

Each optimizer was tested with different models, length of sequences, number of model's layers, number of units in model's layers, dropout rates and learning rates. The goal was to find the combination of hyperparameters that minimized the validation loss.

3.4. Evaluation metrics

To assess the performance of the RNN and LSTM models, we employed two widely used evaluation metrics: Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) [3]. These metrics quantify the error between the predicted and actual stock prices, providing insight into the accuracy of the models.

Mean Squared Error (MSE)

The MSE is calculated as the average of the squared differences between the predicted and actual values. It penalizes larger errors more heavily than smaller ones, making it particularly useful for highlighting significant deviations in predictions. Mathematically, the MSE is expressed as:

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

where:

- y_i: the actual value
- $\hat{y}_i$: the predicted value
- n: the total number of predictions

A lower MSE indicates better model performance.

Root Mean Squared Error (RMSE)

The RMSE is the square root of the MSE, providing an error measure in the same units as the target variable. It is particularly useful for interpreting the magnitude of errors in real-world terms. The formula for RMSE is given as:

$$RMSE = \sqrt{MSE}$$

Like MSE, a lower RMSE indicates higher accuracy of the model.

Comparison and Application

Both metrics were used to evaluate model predictions on the validation and test datasets. By comparing the MSE and RMSE across different hyperparameter configurations and model architectures, we identified the most effective settings for stock price prediction. RMSE, in particular, offered an intuitive understanding of the average prediction error, complementing the detailed error analysis provided by MSE.

These evaluation metrics allowed us to assess the RNN

and LSTM models' ability to capture complex stock market patterns, thereby validating their suitability for forecasting multivariate stock values.

4. Experimental Analysis

This section describes the experiments conducted to evaluate the performance of the RNN and LSTM models. Several hyperparameter configurations were tested, and the performance was assessed using mean square error on the validation and test datasets.

4.1. Exploratory Data Analysis (EDA)

After handling invalid data, 4 features (Open price, Low price, High price, Close price) were plotted (Figure 1).

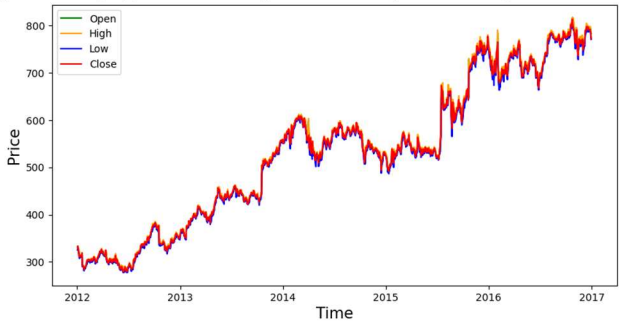

Figure 3. The stock price over time in 4 features (Open, Low, High, Close)

Overall, the stock price during the period in the training dataset was an upward trend.

4.2. Preprocessing data

The total training dataset was split into two subsets: training and validation sets. All five features were scaled with Min-Max scaling method to shift all values to the same range from 0 to 1. Afterwards, sequences of number of days based on tuning hyperparameters of the features and labels were generated for both training and validation sets. The number of input features and output labels are equal due to the purpose of this study, using the past N days data (*e.g.* $N = 30$, including all Open, High, Low, Close, Volume) to predict the next M days data (*e.g.* $M = 1, 2, 3$, including all Open, High, Low, Close, Volume).

4.3. Training and validation models

A grid search was conducted, where the following parameters were tested:

- Models: RNN, LSTM
- Length of sequences: 15, 30
- Layers: 2, 3, 4
- Units in each layer: 32, 64
- Dropout rates: 0.2, 0.3, 0.5
- Learning rates: 0.001, 0.0001

Best Model Performance

The best-performing model was identified as the LSTM model with specific hyperparameter settings: sequences of 15 days, 2 LSTM layers, 64 units per layer, a dropout rate of 0.3, a learning rate of 0.001, and the RMSprop optimizer.

	RNN	LSTM
MSE	0.00207	0.002
RMSE	0.04548	0.0447

Table 1. Mean Squared Error and Root Mean Squared Error of RNN and LSTM models

This configuration resulted in an MSE of 0. 002 and an RMSE of 0. 0447 on the validation dataset (Table 1), which outperformed the corresponding metrics of the RNN model ($MSE = 0.00207$, $RMSE = 0.04548$).

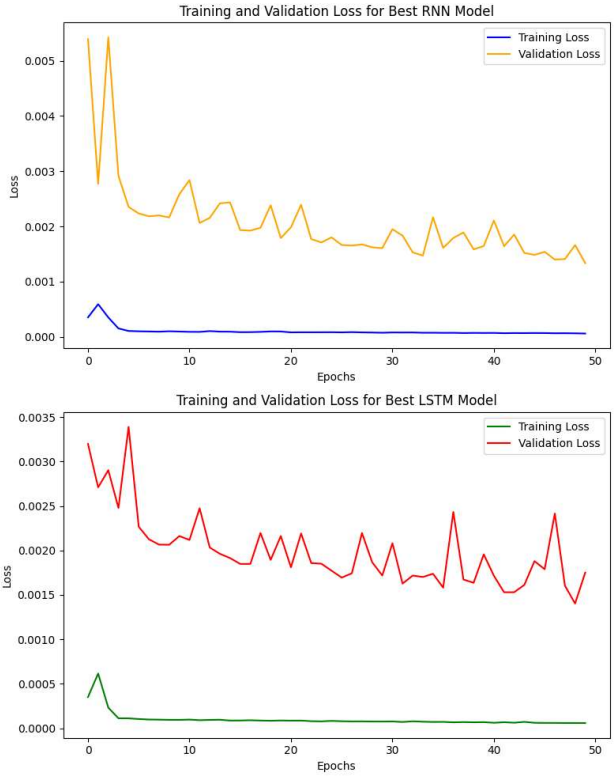

Figure 4. Training and validation loss for the best RNN model and the best LSTM model

Figure 4 compares the training and validation loss curves for the best RNN and LSTM models. Both models exhibited convergence, indicating effective training. However, the LSTM model consistently maintained slightly lower loss values during training and validation, further demonstrating its ability to capture the long-term dependencies in the stock price data. The decrease in loss for both models reflects the importance of robust preprocessing and optimal hyperparameter tuning. The slight gap between training and validation loss suggests a well-regularized model.

4.4. Final model performance

After identifying the LSTM model as the best-performing configuration, it was retrained on the entire training dataset to maximize its learning capability. When evaluated on the test set, the LSTM model achieved an MSE of 0.00711 and an RMSE of 0.08433. These results were higher than the validation results ($MSE = 0.00208$, $RMSE = 0.04566$), indicating that the model's performance decreased slightly when applied to unseen data.

5. Code availability

The full implementation of the code used for this experiment, including data preprocessing, model training, and hyperparameter tuning, can be accessed via the following link: https://github.com/Think-Nguyen/Deep-learning-fundamentals-A3.git

This repository contains all relevant Python scripts and dependencies needed to reproduce the results discussed in this report.

6. Discussion

The results indicate that both RNN and LSTM models are capable of stock price prediction, with only marginal differences in their performance. The LSTM model slightly outperformed the RNN model across validation and test datasets, reflecting its ability to handle longer-term dependencies with its gated architecture. However, the relatively small gap in performance metrics suggests that the dataset did not contain significant long-term dependencies, allowing the simpler RNN model to remain competitive.

The observed increase in error metrics on the test dataset ($MSE = 0.00711$, $RMSE = 0.08433$) compared to validation indicates that some degree of overfitting occurred, despite the use of dropout regularization and hyperparameter tuning. This highlights the challenges of achieving consistent generalization across datasets, particularly in financial forecasting tasks where unseen data often exhibit different patterns.

Challenges Encountered

The study encountered several challenges throughout the process. Hyperparameter tuning, such as determining the optimal sequence length, dropout rate, learning rate, and the number of layers, required extensive experimentation and was computationally intensive. Balancing training efficiency and model performance proved to be a persistent difficulty. Data preprocessing presented its own set of issues, including handling invalid entries like erroneous closing prices and ensuring that normalization preserved essential information while maintaining numerical stability. Selecting the appropriate sequence length was particularly challenging; shorter sequences often missed critical temporal dependencies, while longer ones introduced noise and increased computational complexity. Overfitting also emerged as a significant issue during the early stages of experimentation, requiring careful application of dropout layers and regularization techniques to ensure robust model performance.

Strengths and Limitations

The study highlights the efficacy of RNN and LSTM models for forecasting sequential data, with both architectures demonstrating robust learning on historical stock data. However, the slightly reduced performance on the test dataset suggests the need for further improvements in regularization or dataset expansion. Furthermore, the reliance on historical stock data alone may limit the robustness of the predictions.

7. Conclusion

This study investigated the application of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks for stock price prediction using historical market data. Both models demonstrated the capability to capture temporal dependencies effectively, with the LSTM model marginally outperforming the RNN. The best LSTM configuration achieved an MSE of 0.002 and an RMSE of 0.0447 on the validation dataset, and a test MSE of 0.00711 and RMSE of 0.08433, indicating good predictive performance but some generalization challenges.

While the results validate the potential of deep learning models in stock market forecasting, the differences in performance between the RNN and LSTM models were minor. This suggests that the dataset's complexity did not necessitate the LSTM's advanced architecture, and the RNN model remained competitive with its simpler structure.

Future work could focus on incorporating external features, such as news sentiment, economic indicators, and other contextual data, to enhance the models' predictive power. Exploring hybrid architectures that combine RNNs or LSTMs with attention mechanisms or transformers could further improve performance on more complex datasets. Finally, employing automated hyperparameter tuning techniques and implementing explainable AI tools could improve model efficiency and trustworthiness in financial applications.

References

[1] Chuanzhi Fan and Xiang Zhang. Stock price nowcasting and forecasting with deep learning. *J Intell Inf Syst*, 1–18, 2024.

[2] Carmina Fjellström. Long Short-Term Memory Neural Network for Financial Time Series. *arXiv preprint arXiv:2201.08218v1*, 2022.

[3] Yongqiong Zhu. Stock price prediction using the RNN model. *Journal of Physics: Conference Series*. 1650(3), 2020

[4] Murad Ahmad. Unlocking the Power of LSTM, GRU, and the Struggles of RNN in Managing Extended Sequences. *Medium*. www.medium.com/@muradatcorvit23/unlocking-the-power-of-lstm-gru-and-the-struggles-of-rnn-in-managing-extended-sequences-05879b6899d3, 2023

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861v1*, 2017.