Predicting Google Stock Prices Using Long Short-Term Memory Networks: A Comprehensive Analysis and Implementation

Hansi Cooray
The University of Adelaide
Adelaide, South Australia, Australia
hansi.cooraywijayawarnasooriya@student.adelaide.edu.au

Abstract

Recurrent Neural Networks(RNN) are advanced artificial neural networks for sequential data processing. The capability of RNNs to handle sequential data and capture temporal dependencies makes them highly useful in fields like financial markets. The challenge of predicting stock prices due to their volatile nature is huge. This study focuses on implementing a Recurrent Neural Network, specifically based on Long Short-term memory architecture to predict Google stock prices based on historical data. The model leverages sequential data to forecast future prices based on past trends.

1 Introduction

Stock prices are influenced by various factors, making it critical to employ complex models to make accurate predictions. An implementation of a recurring neural network would be the best solution for this as it captures the temporal dependencies inherited in stock market data. There are different recurrent neural network architectures. Vanilla RNN, Gated Recurrent Unit and Long Short-Term Memory(LSTM) are some of the best examples. However, the Long Short-Term RNN architecture is used during the process since it handles the long-term dependencies better than the other architectures. The model was trained using the Kaggle Google Stock Price dataset to provide accurate predictions of future stock prices. This study highlights the effectiveness of LSTM networks in stock price prediction and provides a comprehensive analysis of the model performance.

2 Recurrent Neural Network

A recurrent neural network, or RNN, is a deep neural network trained on sequential or time series data to produce a machine learning (ML) model capable of making sequential predictions or conclusions based on sequential inputs [1]. Unlike traditional deep learning networks, which assume the inputs and outputs are independent of each other, the RNN's output depends on the prior elements within the sequence. Although there are many well-developed prevailing RNN architectures, the Long Short-Term architecture is best suited for this task, as it excels at capturing long-term dependencies and also reduces vanishing gradient issues. The next section will discuss the LSTM architecture in a nutshell.

2.1 RNN Architecture

An LSTM's architecture can be viewed as a sequence of repeating "blocks" or "cells" that each include a set of interconnected nodes. The architecture consists mainly of five components, input, hidden state, cell state, gates and output [5].

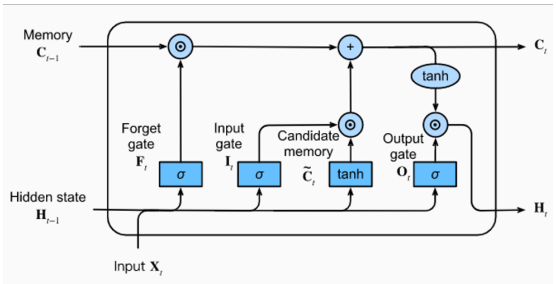

Figure 1: LSTM Architecture

The LSTM makes an input vector in each time step that indicates the current token in the sequence. Also, it maintains a hidden state as the current memory vector for each token and is initiated to zero at the very beginning of the process. The cell state is responsible for capturing and storing long-term information and gates are to control the flow of information throughout the network. Basically, there are three types of gates forget, input and output gates. At last, the LSTM outputs a vector of the network's prediction of the current time step [2].

The next two sections will compare the LSTM architecture with two other advanced RNN architectures.

2.2 Vanilla RNN

A Vanilla Recurrent Neural Network (RNN) is the most basic sort of RNN architecture used to analyze sequential data [3]. It is made up of a single hidden layer with feedback loops for long-term data retention. It uses the current input and the previous hidden state to compute the output and the new hidden state. However, unlike LSTMs, they suffer from the vanishing gradient problem, making it difficult to capture long-term dependencies in sequences [6]. This constraint occurs because gradients diminish as they are backpropagated across several layers, resulting in poor performance on memory-intensive applications over lengthy time periods.

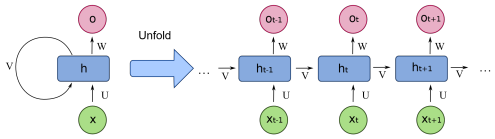

Figure 2: Vanilla Architecture

2.3 GRU RNN

A Gated Recurrent Unit (GRU) is a type of RNN that uses gating methods to solve the vanishing gradient problem [4]. These gates enable GRUs to selectively keep or discard information, allowing for improved management of long-term dependencies. It simplifies the Long Short-Term Memory (LSTM) architecture while maintaining comparable performance for many tasks. However, GRUs are more computationally efficient than LSTMs because they contain fewer parameters and a simpler structure. While GRUs perform similarly to LSTMs on many tasks, they may struggle with extremely complicated dependencies, which LSTMs can handle better [7].

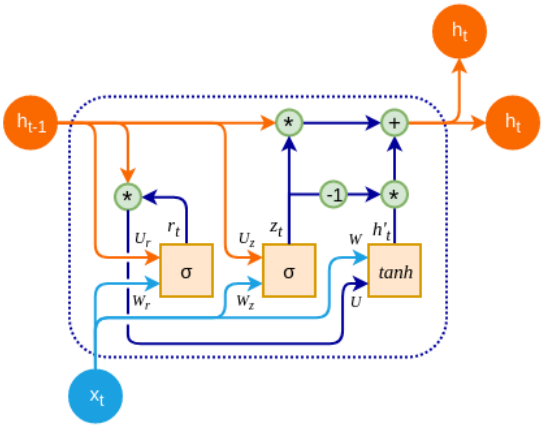

Figure 3: GRU Architecture

3 Model Pipeline

The model pipeline steps are as follows :

1. Data Acquisition and Analysis

2. Data Preparation
3. Define Model Architecture
4. Model Training
5. Evaluation on Validation and Test Sets
6. Model Inference and Results Visualization

3.1 Data Acquisition and Analysis

The first stage of the model pipeline is to select a data set that best fits the purpose. The main goal of this research work is to predict Google Stock prices based on historical data. The data set is available on the Kaggle website("https://www.kaggle.com/datasets/rahulsah06 /gooogle-stock-price"). This data collection consists of 1258 Google historical stock price data. The dataset is divided into three sets for testing, training and validating where each dataset contains 880, 127 and 251 of data respectively.

3.2 Data Preparation

The second step of the pipeline is to prepare the dataset to train the model. Specific parameters from the dataset, such as Open, High, Low, Close, and Volume, are chosen because they are the most relevant features influencing stock price prediction. These attributes are transformed into a NumPy array for further processing. Stock prices and volumes frequently fluctuate dramatically, which can have a negative impact on the performance of models such as LSTMs. Min-Max Scaling is used to normalize data into a range of 0 to 1. This ensures that all features contribute evenly during training and minimizes the possibility of gradients exploding or vanishing during backpropagation.

3.3 Define Model Architechture

The design comprises three 100-unit LSTM layers interleaved with dropout layers that minimize overfitting by randomly deactivating 30% of neurons during training. The final dense layer produces one projected value. The Adam optimizer with a learning rate of 0.005 and a mean squared error loss function was used to train the model, which efficiently balances learning speed, stability, and generalization, making it well-suited for regression applications such as stock price prediction.

3.4 Model Training

100 epochs with a batch size of 32 were used to train the model, allowing it to process the data in smaller, more manageable chunks and in each epoch, the model learned patterns from the training set while also evaluating its performance on the validation set to see how well it generalized to previously unseen data. Furthermore, the loss values for both the training and validation datasets were recorded, allowing to monitor the model's performance over time. using validation data the model was checked to make sure there is no overfitting issue and to perform well on previously unseen samples. This training technique, together with the inclusion of dropout layers in the design, helps to strike a compromise between accuracy and generalization.

3.5 Evaluation on Validation and Test Sets

The model's performance was assessed after training on both the validation and test sets. During hyperparameter tuning, the validation set was utilized to determine the optimal configuration for the model architecture and training parameters. Once the model had been trained, the test set—completely unseen during training—was utilized to assess the model's efficiency. The training and validation losses shown in Figure 4 illustrate that the model is learning effectively and generalizing well indicating a balanced training process without any overfitting or underfitting issues.

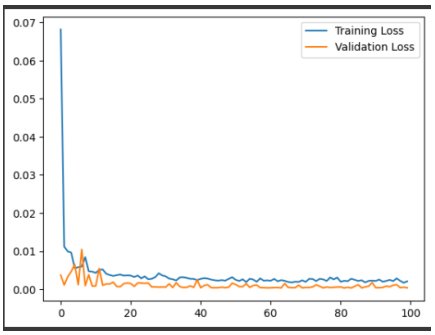

Figure 4: Training and Validation Losses

3.6 Model Inference and Results Visualization

During the testing stage, the model was applied to the testing dataset to predict Google stock prices. After that, the predicted values were transformed back to their original scale values by applying the inverse of Min-Max scaling. These transformed values are then plotted in a graph along with the actual Google stock prices to showcase the model's ability to track trends and fluctuations of the actual prices. The model is capable enough to capture temporal trends as the lines in Figure 5 show almost similar patterns and the difference of the predictions vs actuals is not very high.

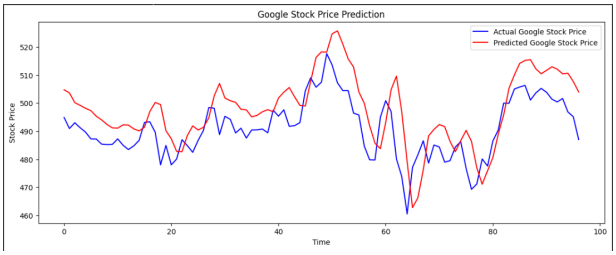

Figure 5: Actual vs Predictions

Besides the final model predictions, initially, several hyperparameter tunings were done and results were obtained. The results shown in Figure 6 below are the model predictions before adding many layers to the model.

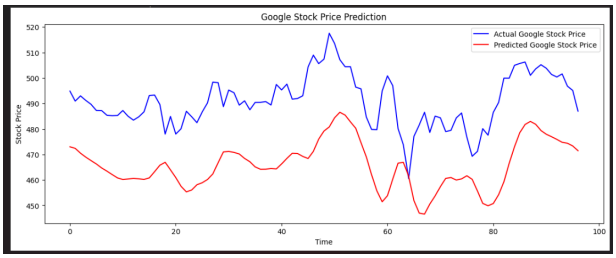

Figure 6: Model predictions with fewer layers

Here, the model was able to identify the trend and fluctuations but there is a huge difference between the actual Google stock prices and the predicted values.

Another testing was carried out to check the best architecture suitable. A new model was trained with GRU architecture and the results are as follows.

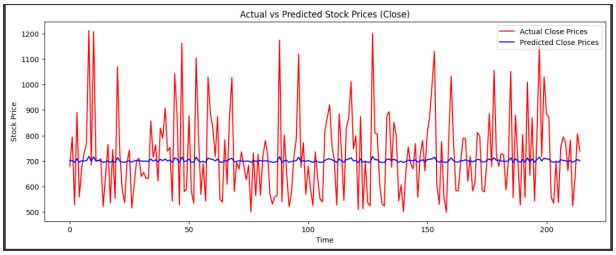

Figure 7: GRU Architecture

The GRU did not deliver improved results since the projected data has significant swings. This proved that LSTM architecture is the best approach for this course.

4 Conclusion

In this study, I investigated the use of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks to anticipate Google stock values based on historical data. The model was created to capture temporal dependencies in the data, and the LSTM layers efficiently learned long-term patterns. The training method resulted in a consistent reduction in loss for both the training and validation sets, demonstrating that the model may generalize well to new data. Visualizing the forecasts versus actual stock prices revealed that the algorithm accu-

rately identified overall patterns, however rare differences suggest areas for improvement.

5 Future Work

The model functioned reasonably well, although there is room for improvement. Future studies could involve experimenting with additional features, such as economic indicators or market sentiment data, to create a more complete picture of stock price changes. Furthermore, incorporating more complex architectures such as bidirectional LSTMs or transformer-based models[8] may enhance the model's predictive capabilities. Finally, hyperparameter optimization utilizing automated techniques such as grid search[10] or Bayesian optimization [9] may aid in fine-tuning the model for improved outcomes.

Overall, this project proves the use of deep learning approaches in financial forecasting and lays a solid platform for future research and refinement in this area.

6 Data Availability

The dataset used for this project, the Google Stock Price Dataset, is publicly available on Kaggle at this link: https://www.kaggle.com/datasets/rahulsah06/gooogle-stock-price. This dataset includes historical stock market data such as Open, High, Low, Close, and Volume prices, which were utilized for training and evaluating the model.

The complete implementation of the project, including the preprocessing pipeline, model architecture, training, and evaluation scripts, is available in this GitHub repository: https://github.com/hansi959/RNN

References

[1] IBM, "Recurrent Neural Networks", 2023, https://www.ibm.com/topics/recurrent-neural-networks. Accessed: 2023-12-04.

[2] Anish Nama, "Understanding LSTM Architecture: Pros and Cons and Implementation", 2023, https://medium.com/@anishnama20/understanding-lstm-architecture-pros-and-cons-and-implementation-3e0cca194094. Accessed: 2023-12-04.

[3] Calvin Feng, "Recurrent Neural Networks", 2023, https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent$_n eural_n etworks. Accessed$: $2023 - 12 - 05$.

[4] R. Dey and F. M. Salem, "Gate-variants of Gated Recurrent Unit (GRU) neural networks", 2017, 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, pp. 1597–1600, doi: 10.1109/MWSCAS.2017.8053243.

[5] Ottavio Calzone, "An Intuitive Explanation of LSTM", 2023, https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c.

[6] R. Qamar, R. Asif, L. Falak Naz, A. Mannan, A. Hussain, "FlightForecast: A Comparative Analysis of Stack LSTM and Vanilla LSTM Models for Flight Prediction", VFAST Transactions on Software Engineering, vol. 12, no. 1, pp. 13–24, 2024, doi: 10.21015/vtse.v12i1.1740.

[7] S. Yang, X. Yu, Y. Zhou, "LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example", 2020, International Workshop on Electronic Communication and Artificial Intelligence (IWECAI), Shanghai, China, pp. 98–101, doi: 10.1109/IWECAI50956.2020.00027.

[8] S. Karita, "A Comparative Study on Transformer vs RNN in Speech Applications", 2019, IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), Singapore, pp. 449–456, doi: 10.1109/ASRU46091.2019.9003750.

[9] J. Snoek, H. Larochelle, R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms", *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012.

[10] I. Priyadarshini and C. Cotton, "A novel LSTM–CNN–grid search-based deep neural

network for sentiment analysis," *J. Super-comput.*, vol. 77, no. 10, pp. 13911-13932, 2021, https://doi.org/10.1007/s11227-021-03838-whttps://doi.org/10.1007/s11227-021-03838-w.