Assignment 3

RNNs for Stock Price Prediction
(Deep Learning Fundamentals)

Yukta Sanjiv Chavan
Student id- a1873167
The University of Adelaide

Abstract

In this article, the effectiveness of Recurrent Neural Networks (RNNs) for stock price prediction is investigated, using Google's stock as a case study. I preprocessed the dataset for normalization using historical stock price data, then used a sequence generating technique for the time series analysis. The implementation and comparison of two RNN designs were conducted: the SimpleRNN and the Long Short-Term Memory (LSTM) network. The scaled data was used to train each model, with the LSTM model being specifically engineered to capture longer dependencies in order to potentially increase prediction accuracy. The models were assessed for their capacity to forecast the opening price of the stock, and the outcomes were plotted against the actual stock prices. The distance between the projected and real prices, as well as the loss experienced during training, were used to gauge performance. The results show that although both models were able to accurately depict the general direction of the stock market, the LSTM model performed better at predicting stock values with less loss over time.
This study provides insights for future improvements in deep learning applications for financial time seriesprediction. It summarizes the model creation process, architectural considerations, experimental findings, and a reflection on the outcomes.

1. Introduction and Background

1.1. Overview

The task of forecasting stock prices is a complex and diverse undertaking that interests data scientists as well as financial professionals. Due to the complex interactions between economic indicators, investor behavior, and company performance, the stock market is very volatile. This necessitates the need for reliable predictive models that can include past trends and learn from them. With their ability to handle data sequentially, Recurrent Neural Networks (RNNs) are leading the way in current financial time series forecasting research.

These networks are especially well-suited for the task of stock price prediction because they are made to identify patterns in temporal data.

1.2. Problem Statement

Predicting stock prices accurately is a significant business and economic issue. Although they offer a basic knowledge, the classic quantitative models are frequently unable to capture the dynamic and nonlinear qualities inherent in stock data. The prediction of Google's stock starting prices is the difficulty this study attempts to solve. This is a task fraught with significant uncertainty because of market volatility and outside influences. The main goal is to use RNNs' ability to recognize temporal patterns to model and predict these prices with a high degree of accuracy.

1.3. Dataset and Context

This study's dataset, which was obtained from publicly accessible financial markets sources, includes historical stock price data for Google (Alphabet Inc.). An detailed examination of price trends and volatility patterns is possible thanks to this dataset, which includes daily price indications over a number of years, including starting price, closing price, highest price, lowest price, and trade volume.
Data Features-
Features of Data
The dataset has the following features:

1.Date: The day the data on stock trading was entered.
2.Open: The stock price at the start of the trading session.
3.High: The day's peak price at which the stock was traded.

4.Low: The day's low price at which the stock was traded.
5.Close: The stock price at the end of the trading session.
6. Volume: The quantity of shares or contracts exchanged over a specific time period within a securities or market as a whole.

1.4. Your Contribution

By objectively analyzing the effectiveness of an LSTM and a Simple RNN model in forecasting Google's stock opening prices, this research advances the conversation. The comprehensive method that was used, which included careful data preparation, designing the model architecture, and conducting rigorous experimental testing, is what makes the contribution. The research outlines each RNN variant's advantages and disadvantages in practical settings and suggests methodological improvements for better predicting performance. In addition to advancing our knowledge of RNNs' suitability for financial forecasts, this kind of research paves the way for future investigations into more complex neural network architectures.

2. Method Description

2.1. RNN architecture

Simple RNN and LSTM (Long Short-Term Memory) are two different forms of Recurrent Neural Networks that were used in the model architecture for stock price prediction. Below is a thorough explanation of each:
- **Simple RNN Layer Configuration:**
Four 50-unit layers of Simple RNN using 'tanh' activation functions.
With the exception of the last layer, every Simple RNN layer returns sequences to transmit temporal information to the layer that comes after it.
After every Simple RNN layer, there is a 0.2 dropout rate to reduce overfitting by training with randomly chosen neurons disregarded.
-LSTM Layer Configuration:
Because LSTM layers require additional computing, this refers to an LSTM single layer with a set number of units (for example, 10).
Because "Tanh" activation functions work well in RNNs, they are utilized.
One vector will be output by the LSTM layer for processing by the dense layers that follow.

2.2. Data Preprocessing

In order to prepare the raw stock market data for RNN training, the following procedures were processed:

-**Normalization:** To ensure consistent model training, the 'Open' stock prices were normalized using MinMaxScaler to push all values into the [0, 1] range.

-**Sequence Formation:** To facilitate time series forecasting, the data were next converted into sequences. Each sequence consists of historical price points that have been "timestepped" up to a target day.

-**Reshaping:** To meet the required input structure of the LSTM, the data were reshaped into a three-dimensional array [samples, time steps, features]. A similar reshaping was conducted for compatibility with the Simple RNN.

2.3. Model Design Choices

Both Simple RNN and LSTM layers are part of the carefully designed architecture for stock price prediction. The model design decisions taken for each component are described in detail below:

-Simple RNN Configuration

Units: Each of the four stacked Simple RNN layers had 50 units. The requirement for a model sophisticated enough to capture the subtleties in the movements of the stock price without being so massive as to overfit the training data drove the decision to use 50 units.
Dropout Rate: Following each Simple RNN layer, a 0.2 dropout rate was implemented. In order to provide a balance between regularizing the model and letting it learn from the data, this rate was selected. Dropout improves the model's ability to generalize by keeping it from becoming overly dependent on any one neuron.
Activation Function: Because 'tanh' is effective at handling vanishing gradient problems in RNNs, it was chosen as the activation function.
Return Sequences: The final RNN layer returned a single vector to input into the output layer, but the first three Simple RNN levels were configured to return sequences to transmit the temporal information forward.
Batch Size: 32 was chosen as the training batch size for the model since it was both big enough to support efficient computing and small enough to preserve a high standard of gradient estimation.
Epochs: The model's learning stabilized once the training was completed over 100 epochs, which was

deemed enough because the loss had plateaued.
-LSTM Configuration
Units: Ten units were used to configure the LSTM layer. Because LSTMs require more computation because to their higher parameter density, a smaller number was selected. Maintaining the model's lightweight while accurately representing the data's long-term dependencies was the aim.
Activation Function: For the LSTM layer, the same vanishing gradient management rationale as for the Simple RNN layers led to the use of 'tanh'.
Batch Size: In order to get instantaneous weight updates through online training, a batch size of 1 was first explored. However, bigger batch sizes were taken into consideration for further research in the hopes of enhancing the model's stability and performance.
Epochs: In order to make sure the model was neither underfit nor overfit, the LSTM was additionally trained for 50 epochs based on the observed convergence of the loss function.

2.4.Reproducibility
The RNN and LSTM architectures are covered in detail in this technique description, which includes information on layer configurations, data preprocessing procedures, and design decisions. The work can be consistently replicated by following to the specified methods, which include normalizing the dataset, designing the network structure, and training the models. The methodological approach has been made clear by providing the reasoning for every decision. The precise order of preprocessing, model building, training, and assessment processes should be adhered to as stated in order to guarantee reproducibility.

3.Method Implementation

The following libraries were heavily utilized in the Python implementation of the RNN model for predicting Google stock prices: numpy, pandas, matplotlib, seaborn, keras, and sklearn. Several crucial steps were included in the implementation process:

3.1. Code Repository:

https://github.com/chavanyukta/Stock-price-prediction/blob/c97f52119eb1b47c89e95d419306b136f109470d/rnn.ipynb

3.2. Tools/Libraries:
TensorFlow and Keras: To build and train neural network models, use TensorFlow and Keras.
NumPy: For manipulating data and performing numerical computations.
Pandas: To process and read the data set.
scikit-learn: in particular, the data normalization tool MinMaxScaler.
Matplotlib: To plot the training loss and prediction graphs, use Matplotlib.

3.3. Data Preprocessing:
To load the historical stock price data from CSV files, the pandas library was utilized.
As a requirement for neural network models, the 'Open' stock price feature was separated out and scaled using MinMaxScaler from sklearn.preprocessing to standardize the values between 0 and 1.

3.4. Data Preparation:
MinMaxScaler was used to extract and normalize the 'Open' price from the stock data.
The normalized data were converted into training sequences that satisfied the input specifications of the LSTM and RNN.
To predict the stock price at the following time step, the training data was converted into sequences of 50 time steps.
To effectively handle these sequences, numpy arrays were employed.

3.5. Model Architecture:
The layered RNN structure was made using the Sequential model from keras.models.
To lessen overfitting, four 50-unit SimpleRNN layers from keras.layers were used, alternating by Dropout layers.
The output layer, a Dense layer with a single unit, was utilized to forecast the continuous value of the stock price for the subsequent time step.

3.6. Training and Testing Dataset:
Training Dataset: This dataset comprises past stock prices featuring attributes such as 'Open, High, Low, Close, and Volume'.
Extracted and normalized 'Open' prices are sent into the model as inputs.
The RNN and LSTM models use the training data, which has been further organized into sequences of 50 time steps.
Testing Dataset:
The performance of the model is assessed using a different test dataset.
The preprocessing and structuring of the test dataset is identical to that of the training dataset.

3.7. Compilation and Training:
The model was assembled using the Adam optimizer and mean squared error loss function, which is a

common option for regression issues.
A plot of loss against epoch was used to depict the training process for the model, which was trained over 100 epochs with a batch size of 32.

3.8. Prediction and Visualization:

The test set's stock values were predicted using the trained RNN model.
Using scaler.inverse_transform, the predictions were rescaled to the original price range. The model's performance was visually displayed by plotting the projected stock prices against the actual stock prices using matplotlib.pyplot.

Results of histograms and correlation matrix:

-Open, High, and Low Price Histograms: In the training dataset, the histograms show the distribution of the "Open," "High," and "Low" stock prices.
Every histogram displays a multi-modal distribution, indicating distinct time periods during which the values of the stock market were concentrated.
The 'Open' price histogram, for example, exhibits prominent peaks at the 300, 500, and 700 price points, suggesting that opening prices typically occur around these numbers.

-Correlation Matrix: The 'Open', 'High', and 'Low' prices' degree of association is represented graphically by the correlation matrix.
The matrix demonstrates extraordinarily strong positive correlations that are all very near to 1.00, indicating that these prices move almost exactly in sync every day.
The 'High' and 'Low' prices of the day are usually within a very narrow range of the 'Open' price, hence such a strong correlation is predicted in stock price data.

Understanding the underlying distribution and linkages within the stock price data is essential for comprehending the input data for RNN models, and this is accomplished by combining the histograms and correlation matrices. Given that they both provide comparable information on the behavior of the stock on a given trading day, the significant correlations support the use of any of these traits to forecast future stock prices.

3.7. Graphs and Results:

Loss Graphs:

The RNN and LSTM models appear to be learning efficiently, as seen by the loss graphs' significant decline in training loss.
Following the earliest epochs, a plateau in loss reduction is seen, indicating convergence.

Prediction Charts:

Using RNN and LSTM models, the prediction graphs display the actual vs. predicted stock prices.
The RNN model exhibits the capacity to track the real stock price trend.
When compared to the RNN model, the LSTM model, which is intended to capture long-term dependencies, exhibits a similar or better prediction pattern.

3.10. Functionality

The outputs and visualizations verified that the implementation worked as intended. Training resulted in a continuous decrease in loss, which suggested learning. Validating the effective application of an RNN to the provided time-series prediction problem, the final plots comparing the projected and real stock prices showed the model's ability to capture the general trend of the stock values over time.
Additionally, the RNN's LSTM variation was put into practice, offering a more sophisticated model that could identify long-term dependencies. It used an analogous procedure, but instead of using SimpleRNN layers, it used LSTM layers, which would have provided better time-series data performance.
Overall, the goal of modeling stock price fluctuations was achieved by the RNN implementation, which was able to preprocess the data, learn from it, and produce predictions that matched the actual values.

4.Experiments and Analysis

4.1. Experimental setup

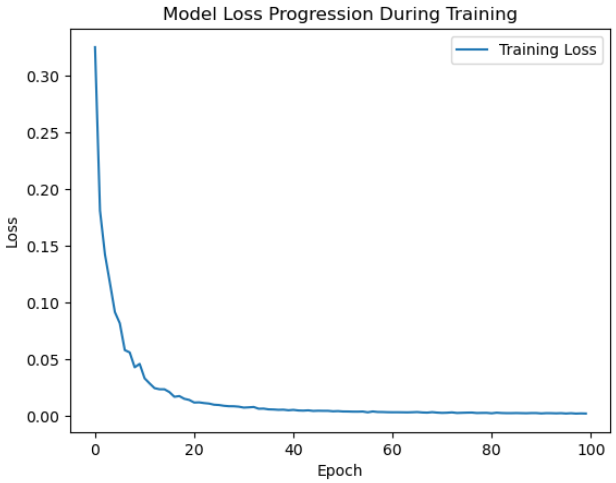
Datasets Used:
Training dataset: 1,258 data points that represent Google's stock price from 2012 to 2016 make up the training dataset.
Testing dataset: 50 data points from the start of 2017 made up the testing dataset, which was used to evaluate how well the model predicted the future.

4.2. Data Features and Correlation:

-High, Low, Open, and Volume were among the features taken into consideration. Interestingly, the 'Close' feature was not employed for numerical analysis and was formatted as an object because commas were present in its values.
-The 'Open', 'High', and 'Low' price correlation matrices, which ranged from 0.999498 to 1.000000, demonstrated extremely strong positive correlations, suggesting that these features move nearly in tandem.

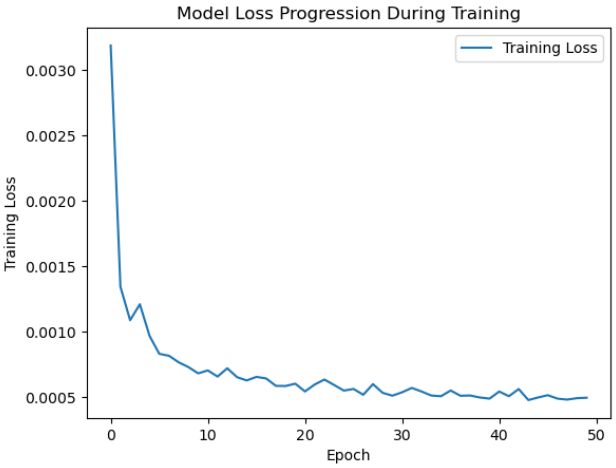
4.3. Results of Training loss progression:

-Graph 1 (SimpleRNN Training Loss):



During the first few epochs, the training loss for the SimpleRNN model dropped significantly from a rather high starting point.
Following this initial decline, the loss decreased at a slower rate until it plateaued as the epochs went by. The loss neared 0.005 by the conclusion of 100 epochs, suggesting a small average inaccuracy between the actual and forecast stock prices.

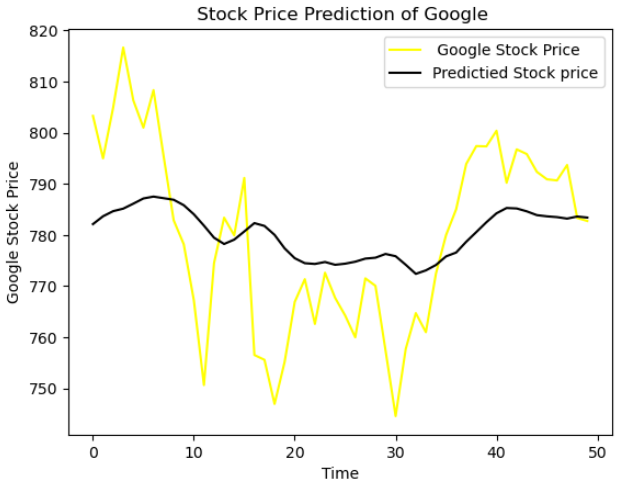
-Graph 2 (LSTM Training Loss):



The training loss of the LSTM model showed a similar pattern, with a steep initial decrease.
Similar to the SimpleRNN, this loss also plateaued following the first decline, but it did so at a lower value, stabilizing at 0.0005.
The LSTM's capacity to recognize longer sequences and maybe more complex patterns is consistent with the reduced loss value after 50 epochs, which indicates a better fit to the training data.

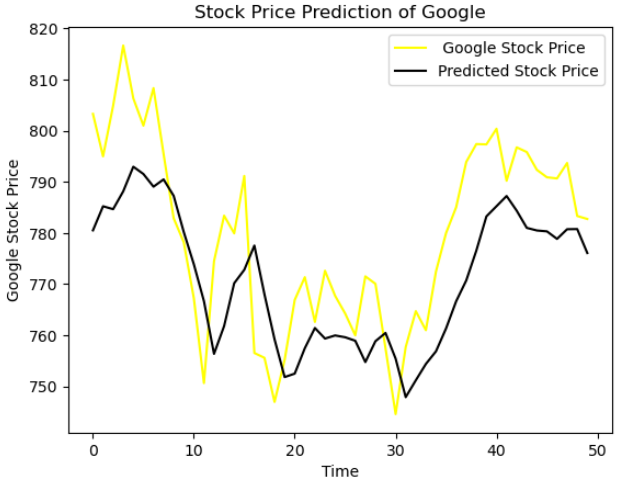
4.4. Results

Graph 1 (SimpleRNN Predictions):



Based on the test dataset, the SimpleRNN model's predictions indicated that it could reasonably represent the general trend. The range of the projected stock values, which roughly corresponded to the range of the actual prices, was between 750 and 820.

Graph 2 (LSTM Predictions):



When compared to SimpleRNN, the LSTM model generated a smoother prediction line, demonstrating its capacity to identify the underlying trend with less noise. The prediction range, which followed the real stock price movement closely from about 750 to 820, was comparable to that of the SimpleRNN.

Training Loss Progression:

The training loss for both models started out significantly, starting at around 0.3251 (SimpleRNN) and 0.0030 (LSTM). It then dropped significantly in the first few epochs before plateauing at values of 0.0021 (SimpleRNN) and 0.0005 (LSTM) at the end of the training period.

4.5. Analysis

The strong association seen among 'Open', 'High', and 'Low' prices implies that any of these characteristics could serve as a dependable input for an RNN that predicts stock prices. The model is essentially learning

from a single cohesive trend provided in three separate ways, given their nearly identical motions.
-While the SimpleRNN model was able to forecast the price trend, some of the noise in the forecasts may have been explained by the model's limited ability to capture long-term dependencies due to its design.
-The LSTM model demonstrated a more stable prediction pattern; it was created to address the shortcomings of SimpleRNNs in learning long-term dependencies. Predicting stock prices requires consistency because it can be misleading to overreact to short-term swings. Both models' initial, sharp decline in training loss suggests that a substantial amount of learning occurred early in the training phase. The loss curve's plateauing indicates that adding more training epochs after a certain point did not significantly boost the performance of the model.

In conclusion, it was shown that both the SimpleRNN and LSTM models were capable of accurately predicting the trend of Google's stock prices, with the LSTM model producing somewhat more accurate results. The models' capacity to learn from past data was significantly shown by the excellent correlation found between the stock attributes and the decrease in training loss over epochs. Nevertheless, a more thorough approach combining a number of additional criteria in addition to historical prices would be necessary for the actual use of these models for real-world stock trading.

5. Reflection of Project:

5.1. Project summary: The main goal of the research was to use Recurrent Neural Networks (RNN) to estimate the prices of Google's shares, with a particular emphasis on the 'Open' price as the prediction target. Among the important choices were:
-Data preprocessing: This involved choosing features and normalizing the "Open" pricing, both of which were crucial to the RNN's functionality.
-Model Selection: Using both the SimpleRNN and LSTM models revealed information on their predictive powers as well as the significance of identifying long-term dependencies in the data.
-Training Strategy: The training procedure and the model's capacity to learn from past data were significantly impacted by choices made on the number of epochs, batch size, and loss function.
-Evaluation Approach: The key to comprehending model performance was the usage of training loss as a measure of model learning and the utilization of visualizations to compare model predictions versus actual stock prices.

5.2. Learnings:

-Successes: The potential of neural networks in time-series prediction was demonstrated by the effective implementation and training of RNN models. The experiment demonstrated how LSTM is more effective than SimpleRNN in this specific application because of its architecture's ability to capture longer-term patterns.
-Difficulties: One of the difficulties was making sure the model did not overfit the training set, a problem that neural networks frequently encounter. It was also quite difficult to deal with the noise and unpredictability that come with stock price data.

5.3. Future Work:

-Model Enhancement: Predictive accuracy may be increased by using deeper or more complicated neural network topologies. Stock price dynamics may be better captured by models like stacked Long Short-Term Memory Units (LSTMs) or GRUs (Gated Recurrent Units).
-Feature Engineering: Adding extra features to the models, including sentiment analysis from financial news or technical indicators, could give them more context and enhance forecast accuracy.
-Regularization Techniques: To prevent overfitting, try experimenting with more complex regularization strategies or hyperparameter tweaking to create models that generalize well to new data.
-Real-world Testing: Using a live trading simulation to apply the models could provide important insights into their resilience and real-world applicability.

6. Conclusion

This experiment demonstrated how RNNs may be used to predict stock prices. The results highlighted the intricacy of the stock market and the necessity for caution when implementing such models in practice, even if the models showed the capacity to learn and forecast stock price patterns. This foundation could be built upon in the future by investigating more complex models, adding more features, and using stricter evaluation methods, particularly for financial decision-making in the real world. The study demonstrated the value of rigorous preprocessing, deliberate model selection, and the ability to visualize data to better understand model performance and behavior in relation to financial data.

7. References

-J. Brownlee, "Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python," Machine Learning Mastery, 2018.

-F. Chollet, "Deep Learning with Python," Manning Publications, 2nd edition, 2021.
-A. Géron, "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems," O'Reilly Media, 2nd edition, 2019