Stock Price Prediction

Dhasarath Dhayanidhi
University of Adelaide
a1906632@adealide.edu.au

Abstract

RNN (recurrent neural networks) and LSTM (long short-term memory networks) are being used as a default model when dealing with sequential data because of their broader practical applications and effectiveness. We will discuss the architectures of RNN and LSTM, their differences and similarities and compare them by modelling them for the provided dataset and report their performances. We will draw further conclusions based on their performance.

1. Introduction

Sequential Data are the data which has importance based on the order of the data, such as words in a sentence different ordering will mean differently than the original one. Of the available models for sequential data (Transformers, LSTMs, RNNs and GRUs), RNNs and LSTMs are used often because of their performance and effectiveness to work better on more complex data despite being less compute intensive. [1] We will start with the dataset and explain the history, architectures of the models and make an theoretical intuition on which model might perform better, then experiment using RNNs and LSTMs and check if our theoretical intuition is correct. We will then compare the Models performances and their similarity.

1.1. Dataset

The Dataset [1] [2] is a Stock Market characteristic of a particular Google Stock from the years of 2012 to 2017, containing the features of the stock price such as “Open”, “High”, “Low” and “Close” commonly known as OHLC, and the “Volume” across the weekdays of the time period. There are 1258 rows of data in the train set and 20 rows of data in the test. The Dataset had no missing entries of data.

1.2. Issues with the dataset & resulting decisions

There were some issues with the dataset, a few of them are discussed below along with their reason for making a particular resulting decision.

The Columns were of different data types upon further investigation, the entries of Close and Volume had “,” in the entries leading pandas to read it as object instead of float, this was fixed and converted to float. The Date column was converted to Datetime format. The train data was then plotted from which we can observe that the Closing price is higher than the High price, which is not expected, we can say that the data is corrupted and going forward with the dataset is not a best decision (but since no concrete info about selecting different dataset was provided we would assume that the Close price is corrupted and would try to avoid using it). Even though the data is collected on all the weekdays there were some missing dates but was assumed to be holiday and the data is sequential without any issues.

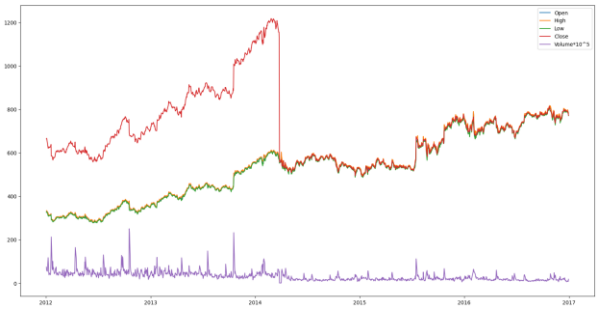

Figure 1: Plot of Training Data

1.3. Choice

The Choice of RNNs and LSTMs had to do with the fact that RNN is the most basic model when it comes to sequential data and LSTM is just a special version of RNN. Both models would be good peer models without any major differences in their architecture (which we will cover below), and by using the same conditions across the models we can compare their differences without any major factors influencing their performances.

2. History and Background

Similar to other models of Deep Learning RNN had its origin from neuroscience as well, where various biologist and neuroscientists observed and proposed neuron models with the existence of "recurrent and reciprocal connections" in the brain which enables the feedback in the brain. In 1960, Frank Rosenblatt [4]published a paper on multi layered perceptrons containing recurrent connections and showed that a cross coupled perceptron is equal to endless deep feed forward only network. Thus, the conceptual and first applications were in the 1960s.

Post the research interest spike in the field of Neural Networks around 1980s, 2 papers introduced the RNN through Jordan and Elman networks. Later RNNS became an active field of research and experimentation leading to interesting findings such as "for every recurrent network, a feedforward network with identical behavior over a finite period of time." Later in 1995 LSTMs were introduced by Hochreiter and Schmidhuber which lead to application of the model in very broad domains.

Recent Developments include bi-directional LSTMS which use 2 hidden networks to achieve information traversal from 2 sides of the data, that is from forward and backward and suitable for data where the dependencies of the data are not only on the past data but also on the future data (one such example being machine translation). Bi-directional LSTMs improved a lot in the field of recognition and introduced various applications of voice to text and text to speech synthesis.

The encoder-decoder was later introduced in 2010, whose development was motivated and driven by LSTM's architectural designs and post that seq2seq that input and output both being sequential data often used either a bidirectional LSTM or a encoder-decoder, until the development of the attention mechanisms and further the Transformer architecture which had the added advantages of not having a fixed length output vector and having the ability to operate on the input parallelly i.e. parallelizable when compared to our previous models, but this came with the added cost of increase in the complexity of the model.

3. Architecture

Similar to perceptrons and Deep Neural Networks RNNs are made of Recurrent Connections which is designed with the ability to store all previous timesteps input-output pair as internal memory (which gets modified at each step) also known as hidden states and used along with the output of a time step t being fed along with the input of timestep t+1 which is being interpreted as feedback that is to have the output of time t and comparing it with the input of t+1 (which is what we were trying to replicate). This is being done to cover all the timesteps (kind of like recursion) and this recursive nature allows it to accept variable input lengths. The architecture on the left is a compressed one which when unfolded/expanded leads to the one on the right.

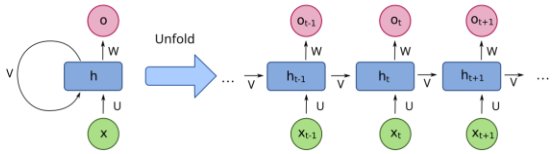

Figure 2: Architecture of RNN.

The RNNs did have a few setbacks such as unstable gradient over time leading to Vanishing or Exploding gradients, i.e. the gradients being passed over from one layer to next in backpropagation will either become too large or too small over a period which made it difficult for the model to capture long term dependencies and causing the model to not generalize properly leading to lesser performance.

Unlike the deep neural networks where the connections are linear, RNNs are made of recurrent connections so it needs a special case of backpropagation known as Backpropagation through time, which propagates the loss and gradients from time step t backwards to the initial time step t=1. For each epoch the forward pass is done, which is to compute predictions using the current biases, weights, inputs and hidden states of the model. A loss is then computed using the given loss criterion by using the prediction at each time step t with the actual output. [1] The total loss is calculated by summing the losses at each time step and then gradients are computed by using partial derivatives and then model parameters are updated using the gradients and it will be then passed to the previous step. The main difference with backpropagation being that backpropagation deals with s record/batch at a time whereas BPPT deals with the total loss in a single backward pass.

3.1. LSTM & GRU

LSTM is a special variant of RNNs which uses a set of memory cells and gates to determine which information to store namely the input gate, the forget gate and the output gate. The input gate determines how much of the input's to store and uses a tanh activation. The forget gate determines if it is needed to flush the data in the hidden states or not and the output gate determines whether to use the hidden state to calculate the prediction at current time step t or not. These three gates allow the model to achieve both short- and long-term memory and understand their dependencies. The architecture of the model is given below. [1]

With the addition of the gates LSTMs have managed to significantly reduce the effect of vanishing gradients but have failed to eradicate it completely. But by adding the cells we have introduced additional linear layers which makes LSTMs lesser efficient than RNNs and the need for higher compute hardware since training them can require additional resources and time [5]. And as same as every other model over a period our requirements started growing and over the recent time with the introduction of big data etc companies and users need higher storage capable models which LSTMs struggle to attain. [6] Also, the LSTMs are tricky to train as they tend to deviate quite a bit even for smaller changes and tend to overfits and lack any proper regularization techniques which can solve the overfitting problem. [2]

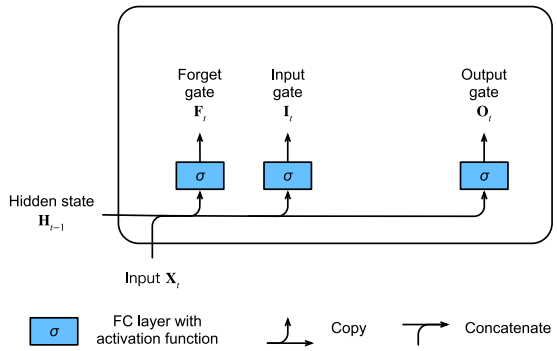

Figure 3: LSTM architecture with the gates.

Later GRUs were introduced which had update and reset gate to modify the data in hidden states. It was more efficient than LSTMs to train but lacked performance. The update gate decides on how much information to be passed over the next layer and the reset gate decides on how much information to be removed/forgotten from the hidden states. [2]

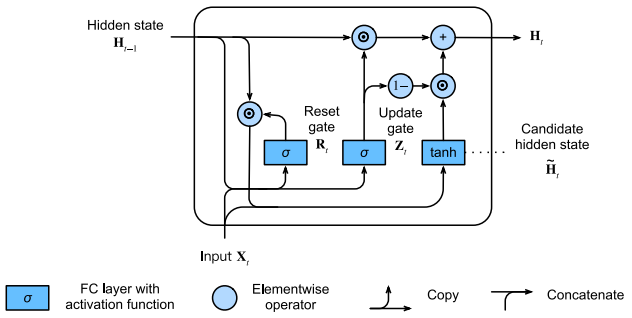

Figure 3: GRU architecture with update and reset gates.

3.2. Applications

Applications generally include Time series data. Automatic Speech Recognition such as Alexa and Siri generally transcribe the voice into text which is a time/order dependent data, even though some models can perform this RNNs, and LSTMs improved the performance significantly with lesser compute needed. Video Captioning, along with a combination of CNNs, can recognize the images in each frame but cannot store the information where RNNs and LSTMs come in and store the significant information. Anomaly detection (such as detecting fraudulent credit card transaction), other models won't be able to compute interdependence but RNNs and LSTMs can consider the dependencies between time steps to find the potential anomalies. Music generation and text generation need to understand the structure, context and grammatical correctness which all is provided by RNNs and LSTMs.

4. Experimentation

4.1. EDA

Before EDA, the model was downloaded using a CURL command from the Kaggle dataset which needs to run in a terminal to download the already splitted train and test dataset into a zip file and used python libraries to extract and read the files in the zip folder. In Exploratory data analysis, apart from the items mentioned in the dataset part we did a few basic cleanings like checking for na and found none in train and test. Made sure to modify the data types of variables after cleaning and removing any unexpected characters in the records and plotted them to observe the data to find any potential issues.

One of the issue we found (as previously said in Dataset) was the Close column had greater values than the High column which was unexpected which lead to the conclusion of not to use the Close column to the maximum extent possible, which resulted in predicting for the Open column in univariate and assuming that the Data is genuine and using the Close along with the other 4 variables (“Open”, ”Low”, ”High” and “Volume”) for multivariate models. The train data was further splitted into train and valid using a ratio. From the metrics of the describe command we knew that the scale of the data were different and MinMaxScaler with feature range of (0,1) was used to scale the data. Continuity of the data from validation to test was also confirmed.

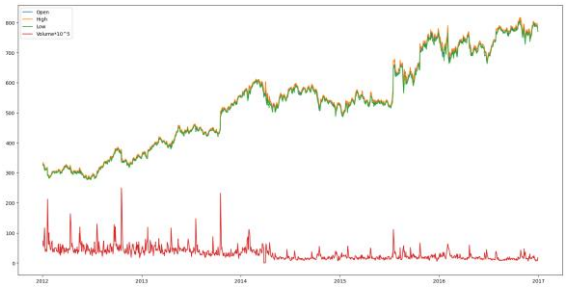

Figure 4: plotting train Data after cleaning

4.2. Experiment descriptions

The experimentation was performed with the aim of achieving the best performance (rmse) by tuning the hyperparameters. We will be doing univariate and multivariate predictions using RNNs and LSTMs. Adam was selected as the sole optimizer given it significance in the field and usually observed better and faster convergence with Adam. While Learning rate, weight decay and epochs, batch sizes, hidden size and time steps were experimented while updating one at a time and observing the loss curves and model prediction performance on train and validation datasets. The whole of the experimentation was done to take multiple time steps as input and produce a single immediate time step prediction. The performance observed with single layers was good enough so that we decided not to move on with stacked RNNs or LSTMs to avoid unnecessarily making the model more complex for far little improvement in the performance. 10-time steps made more sense as since the data was collected only on weekdays 10 steps would mean 2 weeks of data (tried with 5 the result wasn’t as promising) [1]

4.3. Results

For the Univariate models, the Open column was selected to be predicted as the Close was corrupted and Open was the next most important one in the features. The parameters used for RNN are a single layer, Adam Optimizer, MSE loss for updating the model weights and RMSE as a metric, 10-time steps (lookback), input features as 1 since univariate, hidden size of 50 (through tuning) and output size as 10, and learning rate as 0.0001, was run for 500 epochs. And when the data was scaled back to check the RMSE on the original scale, validation RMSE of **11.0292** was observed.

Figure 5: Univariate RNN prediction vs actual

The parameters used for LSTM are a single layer, Adam Optimizer, MSE loss for updating the model weights and RMSE as a metric, 10-time steps (lookback), input features as 1 since univariate, hidden size of 50 (through tuning) and output size as 10, and learning rate as 0.0001, was run for 500 epochs. And when the data was scaled back to check the RMSE on the original scale, validation RMSE of **13.4396** was observed.

Figure 6: Univariate LSTM prediction vs actual

For the Multivariate models, the Close column was decided to be included considering the dataset to be valid and genuine. The parameters used for RNN are a single layer, Adam Optimizer, MSE loss for updating the model weights and RMSE as a metric, 10-time steps (lookback), input features as 5 since using all the features, hidden size of 50 (through tuning) and output size as 5, and learning rate as 0.0001, was run for 500 epochs. And when the data was scaled back to check the RMSE on the original scale, validation RMSE is reported in the below table,

Column/Feature Name	RMSE on Original Scale
Open	7.5827
High	8.9087
Low	9.5048
Close	9.4092
Volume	689439.4969

Figure 7: Multivariate RNN results

Figure 8: Multivariate RNN prediction vs actual

The parameters used for multivariate LSTM are a single layer, Adam Optimizer, MSE loss for updating the model weights and RMSE as a metric, 10-time steps (lookback), input features as 5 since using all the features, hidden size of 50 (through tuning) and output size as 5, and learning rate as 0.0001, was run for 500 epochs. And when the data was scaled back to check the RMSE on the original scale, validation RMSE is reported in the below table,

Column/Feature Name	RMSE on Original Scale
Open	7.4923
High	8.8681
Low	9.0832
Close	8.9478
Volume	722612.4384

Figure 9: Multivariate LSTM results

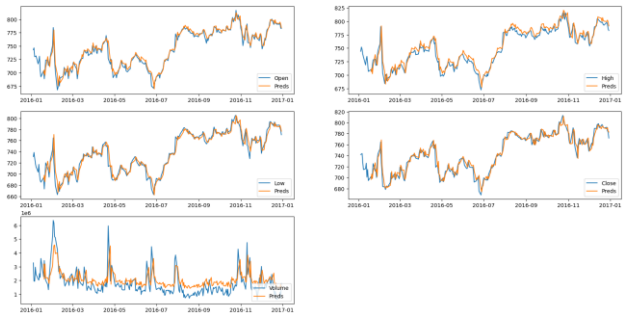

Figure 10: Multivariate LSTM prediction vs actual

For selecting the final model, the univariate performance was compared with the multivariate mean performance (of the 4 main columns namely “Open”, “High”, “Low”, “Close”). Please refer to the table below for the result of the comparison.

Model	Final Metric (RMSE)
Univariate RNN	11.0292
Univariate LSTM	13.4396
Multivariate RNN	8.8514
Multivariate LSTM	8.5979

Figure 11: Comparison of metrics among the models

Based on the above table the multivariate model performs consistently better, and we choose multivariate LSTM as the final model citing the least RMSE. Upon testing using the test dataset, we report the results as per the following table,

Column/Feature Name	RMSE on Original Scale
Open	7.2116
High	9.8659
Low	9.2837
Close	9.5974
Volume	622453.1320

Figure 12: Multivariate LSTM results on test set

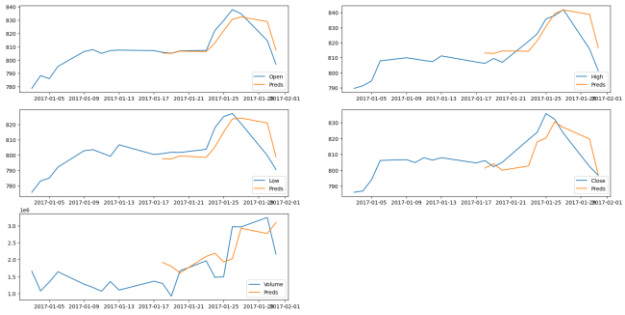

Figure 13: Multivariate LSTM prediction vs actual on test set

5. Code

The code can be found in the same repository and its available for public access. [12] [13] [14]

6. Conclusion

Upon the completion of this task, we have understood the importance and prevalence of RNN and LSTMs when it comes to dealing with sequential data. Although each model had its own advantages and disadvantages (discussed in their respective sections), we would benefit more from using them rather than doubting their performance. We do know that the transformers architecture has a good potential for beating our best model, but the transformers are much more complex, would need a lot of compute and time and wouldn't be feasible for general or low scale purposes.

Using our final model multivariate LSTM, we achieved a validation RMSE of **8.5979** and test RMSE of **8.9897,** the model is performing good considering no major difference in the metrics between test and validation. We didn't include the metrics for the feature volume as from the graph we can see that the fluctuations are a bit extreme (which is what is expected as the volume is more volatile because of the actions of intraday traders). We can experiment by using some of the commonly used metrics in technical analysis such as Moving average price of the last n days (with n=30,60,90 etc.) and maybe a CNN model, or a language model along with sentimental analysis to determine the predict the emotion of the market over the past few days or any news worth influencing any major change in the price. But for a starting point this can be an overkill unless we are developing a product for a big corporation.

We can further improve our final model by experimenting with stacked RNNs and LSTMs although at this point it is suggested against it as the current model has a good performance and the performance increase over the increase of needed computation/complexity is not good enough. But on top of all that, we can do a lot more if the dataset's issues such as the corrupted Close column (although we are assuming close is the issue and not High, since the flow of High is consistent and within meaningful range of low with the years in figure 1 and 2) and lesser sized test set were corrected.

An interesting fact to note is that as we improved our architecture from RNN to LSTM to Transformers, we can see that the only characteristic which is being changes consistently is to reduce what the model keeps in hidden states that is to focus more on important parts of the input and forget the lesser important parts.

7. References

[1] A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network," [Online]. Available: https://arxiv.org/pdf/1808.03314.

[2] "Dataset," [Online]. Available: https://www.kaggle.com/code/dpamgautam/stock-price-prediction-lstm-gru-rnn.

[3] "Dataset1," [Online]. Available: https://www.kaggle.com/datasets/rahulsah06/gooogle-stock-price?select=Google_Stock_Price_Test.csv.

[4] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network#:~:text=Early%20RNNs%20suffered%20from%20the,the%20standard%20architecture%20for%20RNN..

[5] "D2L RNN," [Online]. Available: https://classic.d2l.ai/chapter_recurrent-neural-networks/rnn.html.

[6] "D2L LSTM," [Online]. Available: https://d2l.ai/chapter_recurrent-modern/lstm.html.

[7] "GFG," [Online]. Available: https://www.geeksforgeeks.org/long-short-term-memory-networks-using-pytorch/.

[8] "GFG LSTM," [Online]. Available: https://www.geeksforgeeks.org/understanding-of-lstm-networks/.

[9] "Comparision of RNN vs LSTM vs Transformers," [Online]. Available: https://www.baeldung.com/cs/rnns-transformers-nlp.

[10] "D2L GRU," [Online]. Available: https://classic.d2l.ai/chapter_recurrent-modern/gru.html.

[11] J. Brownlee, "Machine learning Mastery," [Online]. Available: https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/.

[12] "pytorch lstm," [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[13] "pytorch Rnn," [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.RNN.html.

[14] pytorch, "pytorch," [Online]. Available: https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html.