Assignment 3: RNNs for Stock Price Prediction

Sanchit Goel
The University of Adelaide
Adelaide, South Australia, Australia
sanchit.goel@student.adelaide.edu.au

Abstract

Recurrent Neural Networks (RNNs) are sequence-to-sequence models that use neural networks for tasks that involve sequences as an input. Long-Short Term Memory (LSTMs) are a variation of RNNs that use forget, input and output gate to control the flow of information through the network and typically work better than RNNs. This project aims at using these networks to predict the stock prices of google, since stock prices are time-series data that is a form of sequence. Along with this, a loss function specific to the problem was supposed that makes sure that the model doesn't simply predict the previous days stock price to minimise loss, which is a commonly occuring problem in auto-regressive models. Experiments were done using different variations of loss and feature set. Results show that RNNs and LSTMs are not ideal models for this task. RNNs however can be used in a similar task, that is trend prediction. This is because RNNs were found to be accurate in predicting the trend, but not the price. Lastly, it is also recommended to further improve the loss function to consider various cases that might be happening when using these models for predictions and penalise/reward the model based on that.

1. Introduction

Sequence-to-sequence modelling has been a topic of interest for many years. Sequences are all around us. It essentially means using sequence as input to output another sequence, based on some intricate relation between both the sequences that is the target of the modelling task. Time-series, language, genomes, and so on are some examples of sequences. In language, sequence-to-sequence has important applications, such as machine translation, question-answering, summarising, and more. In this project, the focus is on time-series data. Specifically in time-series, this project explores the applications of AI for stock price prediction.

Historically, stock price prediction has never been an integral part of trading in firms. Most techniques used are based on standard statistical methods such as Global Minimum Variance Portfolio [1], Pairs Trading [2] and more. These are standard methods that were used back in the day, and still are, systems that are rule-based and cannot generalise well. In practice, some linear regression based approaches are used. One famous approach is rolling regression [3], where the linear model uses data from a fixed amount of past days to predict the price for the next day. Once the real world data for the next day is available, predictions are done using that for the next day, following the same procedure. Even though this is intuitive, it cannot be used to predict data for multiple days. Another famous approach is the Auto Regressive Integrated Moving Average (ARIMA) model [4]. The term autoregressive means that it regeresses on its own values using a lag factor. ARIMA is one of the most famous models used in a lot of time-series problems in a practical setting.

This project focuses more on a neural network based approach. Sequence-to-sequence models grew in popularity with the release of Recurrent Neural Networks (RNNs) [5]. RNNs are sequence-to-sequence models that use neural networks to model sequences and return either a sequence or a single output. RNNs took off because it was the first time sequential data was being modelled using neural networks, thus harnessing their power of universal approximation. LSTMs [6] are a variation of RNNs that use a gated mechanism to select the information being passed through the network and preserve key historical information that was analysed early on in the sequential input. They were a big step moving forward from RNNs and are still widely used in solving time-series based sequential problems, if not language-based problems that are currently solved mostly using Transformers [7].

Sequence-to-sequence models are an integral part of modern day Deep Learning and are worthy to gain an understanding of. The aim and the goal of this project is as follows:

1. To use RNNs and LSTMs to predict Google stock prices.

2. To propose a new loss function with respect to the problem based on Mean Squared Error (MSE) loss.

3. To compare performance improvements given additional statistical features, namely Moving Averages Convergence Divergence (MACD).

4. To compare different window sizes for the input sequence length in order to get the best performance out of the models.

5. To gain an understanding of the difference in the architectures through the results and a theoretical study.

6. To analyse the results carefully and see if these architectures are suitable for modelling stock prices or not.

2. Method Description

This section deals with the dataset setup and describes the models that were tested.

2.1. Dataset and Preprocessing

The dataset used is available on Kaggle[1]. It includes stock data of various companies. These companies belong to either the Forbes 2000, NASDAQ, New York Stock Exchange (NYSE) or the S&P500. This project uses only the Google stock prices based on the assignment specification. Additionally, it is better to do a detail analysis of the S&P500 stocks since it includes the most traded companies in the world. The dataset for these stocks consists of the stock prices from 1999 to 2021. It includes the volume traded on the day and the open, low, high and close prices. The task considered for this assignment is to predict the price change the next day based on the close price. This is called auto-regression and usually, it is done on stationary data. Price changes for stocks are almost always stationary since they are range bound as percentages.

It is foolish to predict the open, low, high and close prices all at once. This is because it would be difficult for the model to learn so many patterns in the dataset to predict these. Additionally, remember that stock prices are not something that are decided by technical factors or mathematics itself. These prices move because of economic factors and fundamentals of the company. Therefore, the value used as the response variable is the percentage change in the close price. Note that this feature had to be created in the dataset. Close price alone might not be enough information for the model. This is why open, low and high prices are used along with close price and the volume traded for the feature set.

Additionally, the goal is to test some window sizes as well. When predicting, only past data present in the window would be considered for prediction. There are certain ways in which more past data can be included for modelling. An easy way is to use Moving Average Convergence Divergence (MACD) signal. Using this signal will also incorporate more historical information in a single number. MACD essentially is a difference between two Exponential Moving Averages (EMAs) of different time periods. It is often used as a signal in trading. EMAs are nothing but weighted averages over a time period, where more weight is assigned to values that are closer to the present, thus making sure that newer information is reflected more in the signal. Finally, the features were scaled using a min-max scaler. After scaling, the dataset was split into training, validation and test sets, with training set being 80% of the data, validation set being 15% and testing set being 5%. This split was considered so that intricate analsysis of the predictions on the testing set can be done by looking only at a small set of final output.

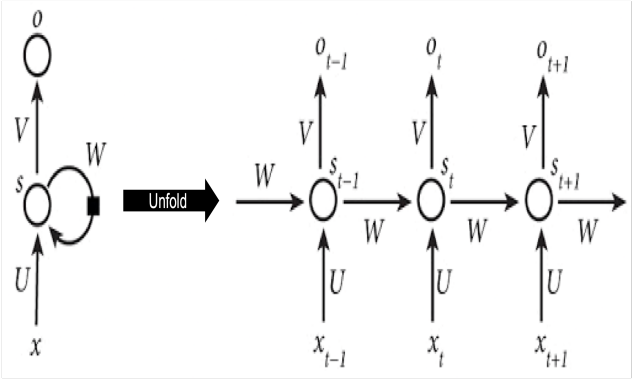


Figure 1. Standard architecture of RNNs.

2.2. Models

This section talks about the two models, namely RNNs and LSTMs used to predict the price of the stocks.

2.3. RNNs

Recurrent Neural Networks (RNNs) are sequence-to-sequence/sequence-to-one model that can take a sequence as an input and output either a sequence or a single value based on the task. Note that this value can be anything, either a continuous output or a categorical output. Figure 1 shows the architecture of RNNs. The architecture of an RNN makes use of three components:

- An input vector at time step t called x_t.
- A hidden state s_t that includes information encoded upto time t.
- The predicted output o_t for time t (o_t will exist for multiple states in sequence-to-sequence and a single state in sequence-to-one).

[1] See this link

The hidden state is maintained over time and it captures the information seen in the sequence so far, typically stressing more over the information currently seen. The hidden state is updated using the recurrence relation shown in equation 1.

$$s_t = \sigma(W s_{t-1} + U x_t) \tag{1}$$

where:

$$\begin{aligned} s_t &= \text{state at time } t \\ W_s &= \text{weight matrix for previous state} \\ x_t &= \text{input at time } t \\ U &= \text{weight matrix for input} \end{aligned}$$

Basically, the state s_t is calculated by multiplying the previous state with its weight vector W and multiplying the current input x_t with its weight vector U. In this way, we incorporate information calculated up to time $t-1$ that is present in s_{t-1} and then update it with the input x_t at time t. The output o_t can be computed at each time step using eq 2, where V is the weight vector for the state s_t.

$$o_t = V s_t \tag{2}$$

where:

$$\begin{aligned} o_t &= \text{output at time } t \\ V &= \text{weight matrix for state} \end{aligned}$$

RNNs use Backpropagation Through Time (BPTT) to update the weight for different inputs and states. This can cause a gradient vanishing and a gradient exploding issue. This means that for a state that comes early in the sequence, the gradient is going to consists of gradients of various states with respect to their previous state. If these gradients are smaller than 1, than essentially we're multiplying a lot of small values to get a value closer to 0. This means that the weight won't update since the gradient is very small and this will slow down the learning process. On the contrary, if the these gradients are greater than 1, then they'll give a large value after getting multiplied, which will cause the gradients to explode. This will result in huge updates in to the weights, thus increasing the learning time for the network. This is a very common issue in RNNs that can solved by clipping the gradients if they explode or vanish. However, LSTMs were actually designed to overcome this issue.

2.3.1 LSTMs

Long-Short Term Memory is a specialised version of RNNs that use a gated mechanism. Figure 2 shows the architecture

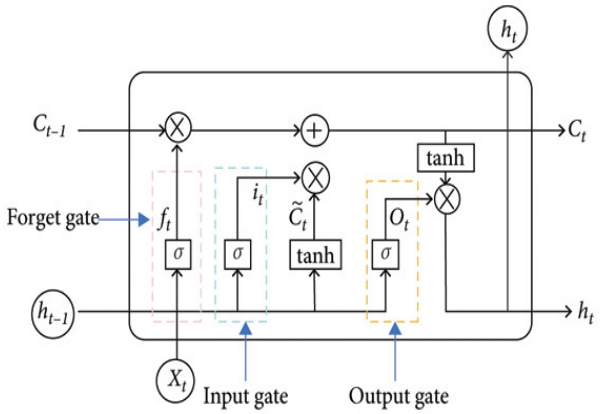


Figure 2. Architecture of an LSTM cell.

of LSTMs. LSTMs preserve two memories. One long term and one short term. At each time step, they update both the memories and decide how much of the history should be kept and how much of the new information must be stored. An LSTM cell contains the following gates:

- Forget Gate: This gate decides which information from the previous gate should be discarded. The update rule is given in eq 3
- Input Gate: This gate decides what new information should be stored in the long term memory. The update rule is given in eq 4. After this, the long term memory is updated. The update rule for this is given in equations 5 and 6
- Output Gate: This gate alters the short term memory based on the long term memory and the input. The update rule is given in equations 7 and 8

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{6}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

$$h_t = o_t \odot \tanh(C_t) \tag{8}$$

where:

$$f_t = \text{forget gate value}$$
$$i_t = \text{input gate value}$$
$$o_t = \text{output gate value}$$
$$C_t = \text{Long-term memory at time } t$$
$$h_t = \text{Short-term memory at time } t$$
$$\sigma = \text{sigmoid function}$$

LSTMs solve the vanishing and exploding gradient problems by controlling information at every step.

2.4. Loss Function

Typically, we use the Mean Squared Error (MSE) loss function when predicting continuous data. However, an issue with this problem is that even with MSE, the model might simply predict the current day's value as the next day's value since that should give a low mean squared error as well. This requires the use of a hybrid loss function that would penalise the model if the prediction is to close to the previous day's price. Equations 9 and 10 show the MSE loss function and the hybrid loss function.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{9}$$

where:

$$N = \text{Number of predictions}$$
$$y_i = \text{ith actual value}$$
$$\hat{y}_i = \text{ith prediction value}$$

$$L_{\text{Hybrid}} = L_{\text{MSE}}(y, \hat{y}) - \alpha \cdot L_{\text{MSE}}(y_{\text{prev}}, \hat{y}) \tag{10}$$

where:

$$\alpha = \text{Weight for the penalty}$$
$$y_{prev} = \text{actual value for a day before the prediction day}$$

2.5. Code Section

The code for the project can be found using this link: GitHub

3. Experiments and Results

This section deals with the experiments and the results of the project.

Table 1. Experiment Results for Different Configurations

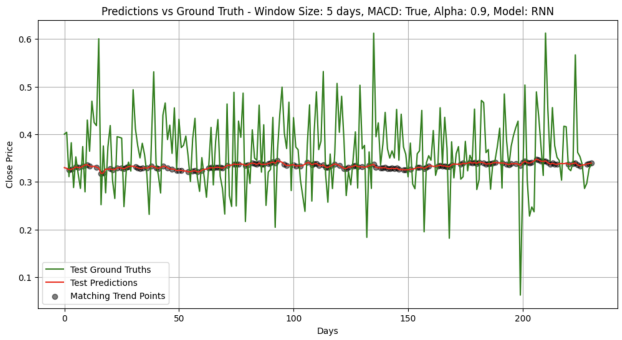
Window Size	MACD	Alpha	Model	Test RMSE	Test MAE
5	FALSE	0	RNN	0.086	0.068
5	FALSE	0	LSTM	0.103	0.085
5	FALSE	0.9	RNN	0.0782	0.060
5	FALSE	0.9	LSTM	0.0880	0.071
5	TRUE	0	RNN	0.0957	0.075
5	TRUE	0	LSTM	0.0924	0.073
5	TRUE	0.9	RNN	0.0832	0.064
5	TRUE	0.9	LSTM	0.132	0.103
50	FALSE	0	RNN	0.081	0.063
50	FALSE	0	LSTM	0.0881	0.071
50	FALSE	0.9	RNN	0.0839	0.065
50	FALSE	0.9	LSTM	0.082	0.064
50	TRUE	0	RNN	0.0993	0.079
50	TRUE	0	LSTM	0.100	0.079
50	TRUE	0.9	RNN	0.0814	0.063
50	TRUE	0.9	LSTM	0.0922	0.073
100	FALSE	0	RNN	0.0787	0.060
100	FALSE	0	LSTM	0.0808	0.063
100	FALSE	0.9	RNN	0.0811	0.063
100	FALSE	0.9	LSTM	0.0986	0.081
100	TRUE	0	RNN	0.0783	0.060
100	TRUE	0	LSTM	0.0820	0.063
100	TRUE	0.9	RNN	0.0844	0.065
100	TRUE	0.9	LSTM	0.122	0.101

3.1. Experiments

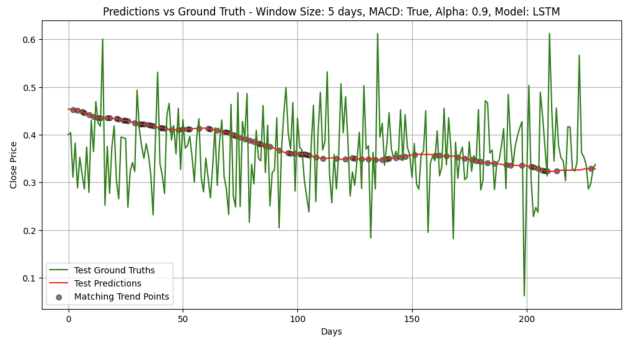
A lot of experiments were designed for this project. The data predicted is only for the day after. Experiments were done on the amount of previous data used to predict the next day's stock price. 5, 50 and 100 days were selected for experimentation. This basically means that the data from previous 5, 50 and 100 days was used to predict the stock value for the next day. Two α values for the penalty term in the hybrid loss function were explored (α is a regularising term). These values were 0.0 and 0.9. Additionally, results with and without MACD features were recorded as well. This was done for both the RNN and the LSTM model. Training was done using a learning rate of 0.00005 for 10 epochs since the model seemed to converge by that time (after looking at the validation loss). Adam optimiser [8] was used since it uses momentum and schedules the learning rate as well. Training was done on a P100 GPU using Kaggle notebooks.

3.2. Results and Analysis

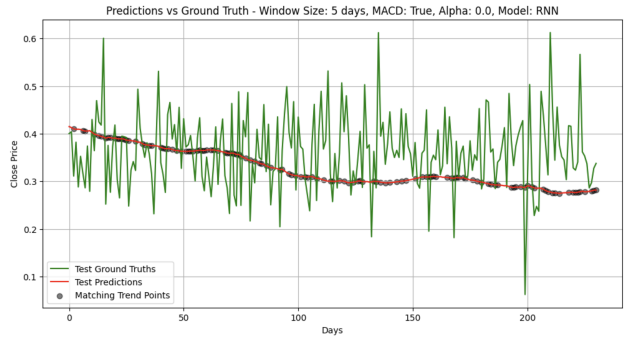
Table 1 shows the results recorded for all the experiments. Results are reported using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). If we look at the MSE, almost all of the models perform good. However, note that models that used the hybrid loss with the penalty term perform slightly better. Additionally, note that RNNs perform better than LSTMs based on the finding, since most of the RNNs reported an RMSE less than 0.1 while LSTMs reported an RMSE more than 0.1. This requires further



(a) Actuals Vs Predictions for RNNs trained with penalty and MACD features.



(b) Actuals Vs Predictions for LSTMs trained with penalty and MACD features.



(c) Actuals Vs Predictions for RNNs trained without penalty but with MACD features.

Figure 3. Actuals vs Predictions for some experiments. The black dots in the plot highlight days on which the model predicted an increase when there was an increase in actual data or the days on which the model predicted a decrease when there was a decrease in actual data. Basically, the points highlight the days on which the model predicted the stock trend correctly.

analysis.

From figure 3a, we can see that the RNN performs poorly. However, there are a lot of points where it gets the trend right (it might not be visible to the naked eye from the paper since the figures are small so please zoom in to see more variation in this figure than other figures). The prediction line for it is not smooth, unlike the LSTM model in figure 3b has a prediction line which is more smooth and not as jumpy as the real data. A possible reason for this is that LSTM is making things complex with the use of the gates. Recall that gradients were clipped for both RNNs and LSTMs to avoid the vanishing/exploding gradients problem. This is probably why RNNs are working better, since they are less complex and the main issue with them was avoided. Figure 3c shows the graph of RNN in a similar condition but without the penalty since $\alpha = 0.0$. Using only MSE as loss makes the RNN perform like the LSTM. The predictions aren't very jumpy which is what we want. They are smooth instead. This shows that the hybrid loss does help with predictions. The prediction curve gets more jumpy and better for 50 days (see figures generated by code). Internal experiments also showed that overfitting the model made it more jumpy, meaning it made it more accurate, which is what we would expect if we overfit on the selected split.

The main issue with the predictions couldn't have been seen without this analysis. MSE and MAE are low but they aren't low enough for this specific problem. The issue is regarding the scale of the predictions. RNNs seem to get the trend right, but not the scale. It is possible that this is due to the fact that not making the line move a lot, or basically producing a line of predictions that goes right through the middle of the actual prices gives the lowest error. From the results, it can be actually said that RNNs shouldn't be used for stock price prediction. They can probably be used for trend prediction, but when money is involved, RNNs/LSTMs shouldn't be used at all. Statistical methods such as ARIMA work better and are used in real world.

4. Conclusion and Future Work

This project uses RNNs and LSTMs to predict stock prices for Google. Different experiments were done by using a hybrid loss function. Stock prices were predicted using the open, low, high and close prices of previous 5, 50, and 100 days in different experiments, along with the percentage change in price from the last day. In some experiments, MACD features were used as well. Results were compared and it was found that even though all the models perform terribly, the RMSE is low because of the scale of the data. Careful analysis however revealed that RNNs are able to learn the trends in the data and predict upward/downward movement. This might be because they are simpler than LSTMs and the vanishing/exploding gradient problem was solved by clipping the gradients. It was also seen that the hybrid loss did help RNNs learn about the trends better.

Future work should focus on utilising a better loss even than the proposed hybrid loss function because even though

this loss function optimises the model in a way such that it won't predict previous day's price (this was to avoid lagged predictions which is commonly seen in regression), it will also hinder the models predictive capability when the market is indeed flat and the stock didn't move a lot from the previous day's prices. Giving a reward in such cases might help the model learn better. Additionally, simpler versions of RNNs/LSTMs should be used. Authors of [9] showed that using a much simpler version of LSTMs actually significantly improves its performance on sequence-to-sequence task, even compared to modern transformer models [7]. However, in a real world scenario, it is recommended to not use black box models and instead do trading, on your own risk using better statistical methods.

References

[1] On the estimation of the global minimum variance portfolio. 1

[2] Evan Gatev, William N. Goetzmann, and K. Geert Rouwenhorst. Pairs trading: Performance of a relative-value arbitrage rule. *Review of financial studies/The Review of financial studies*, 19(3):797–827, 2006. 1

[3] Tihana Škrinjarić. Rolling regression capm on zagreb stock exchange – can investors profit from it? *Economic review (Tuzla)*, 16(2):7–22, 2018. 1

[4] Hussan Al-Chalabi, Yamur K. Al-Douri, and Jan Lundberg. Time series forecasting using arima model: A case study of mining face drilling rig. In *ADVCOMP 2018*, pages 1–, 2018. 1

[5] Robin M Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv (Cornell University)*, 2019. 1

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. 1

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv.org*, 2023. 1, 6

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv (Cornell University)*, 2017. 4

[9] Leo Feng, Frederick Tung, Mohamed Osama Ahmed, Yoshua Bengio, and Hossein Hajimirsadegh. Were rnns all we needed? 2024. 6