Assignment 3, Deep Learning Fundamentals

Li Fu Chuen
The University of Adelaide
a1865844@adelaide.edu.au

Abstract

This paper is to predict the future stock price of QQQ with different recurrent neural networks, RNN, LSTM and GRU. To achieve optimal performance in predicting future stock prices, an exhaustive search is done to find the best parameter for the sequence length, the number of neurons in the hidden state and the number of the memory cell layer. An exploration and analysis of relations between those hyper-parameters are also visualized in different plots and graphs. The performance of the best model is mediocre and the MSE loss is higher than a simple statistical model. The mediocre performance may account for the insufficient information and the limitation of a recurrent neural network.

1. Introduction

A stock market is a system that allows buyers and sellers to exchange shares of businesses. The average volume of a major stock market exchange, such as the Nasdaq exchange, could be more than ten billion dollars daily. [1] Buying or selling a stock at the right time could be highly profitable. A common belief is that patterns of historical stock prices help determine the future stock price. For example, a stock price in a downtrend is more likely to go down than go up. Therefore, we are trying to use recurrent neural networks to make predictions for future stock prices. A recurrent neural network is somehow capable to store historical information throughout propagation. A recurrent neural network is usually used in a scenario where the data are in a sequence or a particular order with meanings. For example, the stock price is in an order of time.

This article intends to implement recurrent neural networks to predict QQQ stock prices. QQQ is an exchange-traded fund that tracks the Nasdaq-100 Index which tracks 100 different non-financial company stocks[1]. We use a look-back period of close prices of QQQ to predict tomorrow's stock price. Three common recurrent neural networks, including RNN, LSTM and GRU, are used to predict stock prices. Moreover, an exhaustive search in the sequence length, the number of neurons in the hidden state and the number of layers are done to find the optimal setting to accommodate the lowest MSE loss. The result shows that GRU performs the best on average but RNN is the best one individually. The sequence length can enhance the performance of the neural network. The increase in neurons in hidden states does help improve the performance of the memory network but there is a plateau in a further increase. The 1 layer of a memory cell is the best setting in this scenario and the number of layers is the most influential factor for determining the performance of the neural network. However, we found that the prediction result is not decent even if we implemented the best model from the experiments section. The performance of the best model is worse than a simple statistical model. The reasons for the poor performance may account for insufficient feature data and the limitation of the recurrent neural network. To address the problems, we may add more information to the feature vector or introduce new components to the recurrent neural network such as attention modules and convolutional neural networks.

2. Methodology

A recurrent neural network often works with sequential data such as languages, time-series data and DNA sequences. The sequence data are in a definite order with a meaning. Each element of the sequence is fed into the recurrent neural network one time step by one time step to generate desired outputs. For example, the stock price of yesterday is fed into the recurrent neural network and followed by the stock price of today which is in chronological order, to generate the predicted stock price of tomorrow. In this section, we would explain how three popular types of the recurrent neural network, a recurrent neural network(RNN), a long short-term memory(LSTM) and a gated recurrent unit(GRU), work.

2.1. Recurrent Neural Network

A recurrent neural network(RNN) is the origin and the foundation of all types of recurrent neural networks. RNN integrates historical information with input to generate a

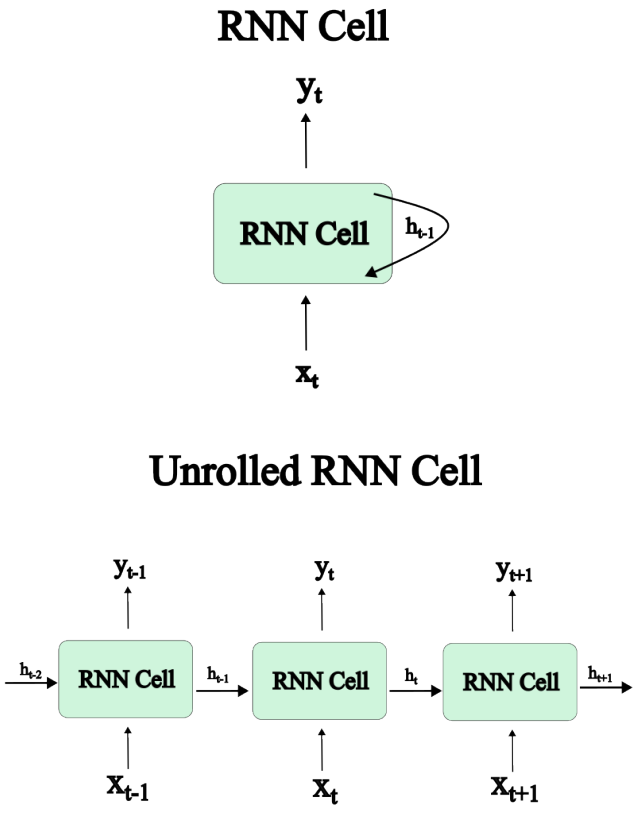

Figure 1. Graphical representation of a RNN cell and its unrolled form

piece of new historical information. RNN is not a complete feed-forward neural network but there are some connections pointing backward during forward propagation. Figure 1 shows a graphical representation of a recurrent neural network. An input x and the hidden state from the previous time step h_{t-1} are combined to generate a new output. The current hidden state will become the previous hidden state for the next time step. The bottom of Figure 1 shows an unrolled graphical representation of RNN across time steps. It shows how inputs and outputs flow through the neural network.

The inner structure and the principle of the RNN cell are presented in Figure 2. Suppose there is a historical information vector which we often name as a hidden state, h_{t-1}, and an input vector x_t. The dimension of the hidden state vectors, h_{t-1} and h_t, are in (1 x n) dimension. n is a self-defined parameter that represents the number of neurons in the hidden state. The input vector is in (1 x i) dimension. i is the number of features of the input. h_{t-1} and x_t are put into a function, $f_h(h_{t-1}, x_t)$, to generate a new hidden state h_t for next input. An output $\hat{y}_t$ will also be given by

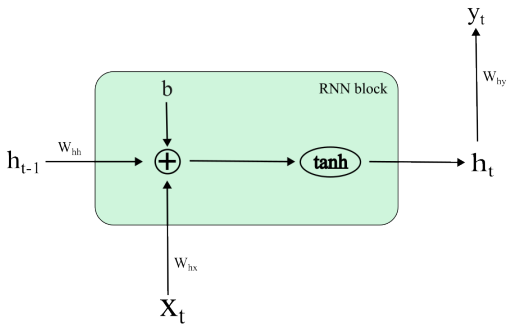

Figure 2. Recurrent neural network block

$f_y(h_t)$. $\hat{y}_t$ will be used to compare with the ground true y_t and evaluate it in a loss function, $L(y_t, \hat{y}_t)$. $L(y_t, \hat{y}_t)$ will decide the values to update the weights of the neural network and updates the weights through back-propagation. A point to note is that the initial value for the hidden state is self-defined and depends on the scenario. A common and general practice is setting zeros for the initial hidden state.

The function $f_h(h_{t-1}, x_t)$ is various which is dependent on the purpose and architecture of RNN. The typical function $f_h(h_{t-1}, x_t)$ for a vanilla RNN is shown in equation (1). W_h and W_x are weights associated with the previous hidden state and the input respectively and b is a bias for the RNN cell. W_h is with a dimension of (n x n) because W_h is connections between hidden states. W_x is with dimension of (i x n) because W_x is connections between the input and the hidden states. b is in a dimension of (1 x n) which is same as the hidden state. h_t is calculated by taking $tanh$ over the sum of $h_{t-1}W_h$, x_tW_x and b. h_t is the current hidden state so it is in a dimension of (1 x n). Because h_t is the function of $tanh$, the values are always between -1 to 1.

$$h_t = tanh(h_{t-1}W_h + x_tW_x + b) \tag{1}$$

The output y_t is further calculated by h_tW_y in equation (2). y_t is often connected to another layer or function such as a fully connected layer or a softmax function to serve different purposes. For example, a layer of a fully connected layer is added in this study to achieve the purpose of regression.

$$y_t = h_tW_y \tag{2}$$

An important thing to remind is that there is only one RNN cell in the neural network in Fig 1. Though the unrolled RNN looks like there are multiple RNN cells in the neural network, the unrolled RNN is just a computation map

only and only one RNN cell is implemented repetitively over all the time steps. Therefore, W_h and W_x are shared thorough out the whole forward propagation.

The mechanism of RNN may lead to the vanishing/exploding gradient problem. Because the weights are shared thorough out the whole forward propagation, the gradient will either become extremely large or shrink to near 0 when doing back-propagation through time. As shown in equation (3), the gradient of W_h can be determined by expanding the chain rule of $\frac{\partial L(y_t,\hat{y}_t)}{\partial W_h}$. $\frac{\partial L(y_t,\hat{y}_t)}{\partial \hat{y}_t}$ and $\frac{\partial \hat{y}_t}{\partial h_t}$ can be computed easily but $\frac{\partial h_t}{\partial W_h}$ is not as trivial. h_t is influenced by h_{t-1} and h_{t-1} is influenced by h_{t-2} so on and so on. $\frac{\partial h_t}{\partial W_h}$ involves multiple time steps so every h_{t-n} has to be considered. $\frac{\partial h_t}{\partial W_h}$ can be determined as shown in equation (4). The problem of vanishing/exploding gradient can also be seen in equation (4). The hidden states from long ago have to multiply W_h times which have a factor of W_h^{t-n}. For example, h_1 will has a factor of W_h^{t-1}. If the time steps t is large and W_h is not equal to 1, W_h^{t-1} will either be an extremely large number or shrink to near 0. Hence, $\frac{\partial L(y_t,\hat{y}_t)}{\partial W_h}$ is said to be either exploding or vanishing. When $\frac{\partial L(y_t,\hat{y}_t)}{\partial W_h}$ is exploding, optimization and convergence may not be possible. When $\frac{\partial L(y_t,\hat{y}_t)}{\partial W_h}$ is vanishing, the hidden states from long ago contribute a near 0 value to the gradient. It means RNN can only comprehend the recent hidden states. Therefore, RNN cannot handle long-term dependency tasks. A point to note that $tanh^{'}(\theta_{t-n})$ represent the derivative of $tanh$ with respective to the corresponding x_t and h_{t-n-1} in equation (4).

$$\begin{aligned}\frac{\partial L(y_t,\hat{y}_t)}{\partial W_h} &= \frac{\partial L(y_t,\hat{y}_t)}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial W_h} \\ &= \frac{\partial L(y_t,\hat{y}_t)}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial W_h}\end{aligned} \tag{3}$$

$$\begin{aligned}\frac{\partial h_t}{\partial W_h} &= \frac{\partial tanh(x_t W_x + h_{t-1}W_h + b)}{W_h} \\ &= tanh^{'}(\theta_t)\frac{\partial (h_{t-1}W_h)}{W_h}, \theta_t = x_t W_x + h_{t-1}W_h + b \\ &= tanh^{'}(\theta_t)(h_{t-1} + W_h\frac{\partial h_{t-1}}{\partial W_h}) \\ &= tanh^{'}(\theta_t)(h_{t-1} + W_h(tanh^{'}(\theta_{t-1}) \\ &(h_{t-2} + W_h(... + W_h(tanh^{'}(\theta_2)(h_1)))\end{aligned} \tag{4}$$

2.2. Long Short Term Memory

A long short term memory (LSTM) works alike a RNN but LSTM is built to handle the vanishing/exploding gradient problem. Both a RNN and a LSTM are using the previous hidden state and the current input to decide the next hidden state and hence the output. However, The architecture of a LSTM cell is different from the recurrent neural network. A LSTM cell has an extra state, which names a cell state, and it is gated. The cell state represents long term memory and the hidden state represents short term memory. As we mentioned previously, the information from time steps long ago is lost in a RNN but they are preserved in LSTM. Moreover, a LSTM cell carries 3 gates which are a forget gate, an input gate and an output gate. A graphical representation of a LSTM cell is presented in Figure 3 (left) and the mechanism of a LSTM is presented below.

2.2.1 Forget Gate

A forget gate is to determine how much long term memory, also known as the cell state, is retained. The cell state, C_{t-1}, is determined whether they are important for future prediction by the current input and the hidden state. W_{hf} and W_{xf} are weights associated with in the forget gate. The dimension of W_{hf} and W_{xf} are (n x n) and (i x n) respectively. b_f is the bias in the input gate which is in a dimension of (1 x n). A sigmoid function is taken over the sum of the weighted hidden state, the weighted input and the bias. The cell state, C_{t-1}, then multiple F_t to give an intermediate cell state, $\dot{C}_t$. Because the σ function will always return values between 0 to 1, the element-wise multiplication of F_t and C_{t-1} determines the portion of the old cell state, C_{t-1} to be retained which in term determines how much long term memory is retained or forgot. Therefore, the gate is named as forget gate.

$$F_t = \sigma(h_{t-1}W_{hf} + x_t W_{xf} + b_f) \tag{5}$$

$$\dot{C}_t = C_{t-1} \otimes F_t \tag{6}$$

2.2.2 Input Gate

The input gate determines how much the current hidden state contributes to the new cell state. W_{hi}, W_{xi}, W_{hc} and W_{xc} are the corresponding weights to the input and previous hidden state in the input gate. b_i and b_c are the corresponding bias. The dimensions of those parameters are the same as in a forget gate. The calculation of the candidate memory $\tilde{C}_t$ is the same as a vanilla RNN which $\tilde{C}_t$ determines the values of the weighted hidden state and the weighted input. I_t serves the same purpose of F_t to determine the portion of $\tilde{C}_t$ to contribute to the current cell state C_t. The current cell state C_t is the sum of $\dot{C}_t$ from the forget gate and $\tilde{C}_t$ from the input gate.

$$I_t = \sigma(h_{t-1}W_{hi} + x_t W_{xi} + b_i) \tag{7}$$

$$\tilde{C}_t = tanh(h_{t-1}W_{hc} + x_t W_{xc} + b_c) \tag{8}$$

$$C_t = \dot{C}_t + \tilde{C}_t \tag{9}$$

2.2.3 Output Gate

The output gate decides the value of the new hidden state and also the output. W_{ho} and W_{xo} are the corresponding weights to the input and previous hidden state in the output gate. b_o is the corresponding bias. The dimensions of those parameters are the same as the input gate. The current cell state C_t is put into $tanh$ to map it into -1 to 1. O_t is a sigmoid function taken over the sum of the weighted hidden state, the weighted input and the bias. It serves the purpose of deciding how much information in the cell state is transferred to a new hidden state and hence influences the calculation of the next time step.

$$O_t = \sigma(h_{t-1}W_{ho} + x_t W_{xo} + b_o) \tag{10}$$

$$h_t = tanh(C_t) \otimes O_t \tag{11}$$

2.3. Gated Recurrent Unit

A Gated recurrent unit is another type of recurrent neural network designed for solving the vanishing/exploding gradient problem. Though LSTM is proven to handle the vanishing/exploding problem somehow, LSTM may not be the best way because the complex structure of LSTM makes it hard to train especially when the training data is not sufficient. Therefore, GRU is designed to handle the vanishing/exploding problem with efficiency.

The structure of a GRU is similar to a LSTM. Both of them are built with gates to evaluate past information and update current information. However, a GRU is only built with 2 gates which are an update gate and a rest gate. A graphical representation of a GRU cell is presented in Figure 3 (right).

2.3.1 Rest Gate

A reset gate acts like a forget gate in LSTM which decides how much information in the previous hidden state should be forgotten. W_{xr} and W_{hr} are the corresponding weights to the input and previous hidden state in the reset gate. b_r is the corresponding bias. The σ function squash the values into 0 to 1. Therefore r_t is deciding what percentages of $W_{hh}h_{t-1}$ are added with the input. It means that it controls how much the previous hidden state contributes to forming the candidate hidden state $\tilde{h}_t$, by squashing them in $tahn$.

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \tag{12}$$

$$\tilde{h}_t = tanh(W_{xh}x_t + r_t \otimes W_{hh}h_{t-1} + b_h) \tag{13}$$

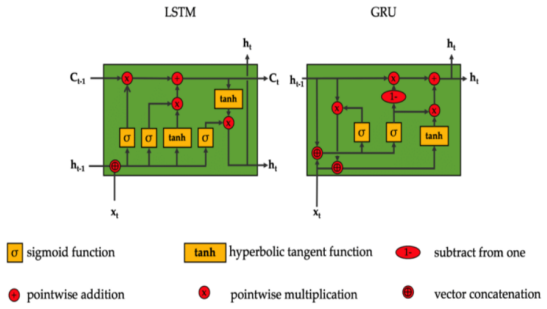

Figure 3. A LSTM cell and a GRU cell

2.3.2 Update Gate

An update gate decides how much historical information would influence and pass to the current hidden state. W_{xz} and W_{hz} are the corresponding weights to the input and previous hidden state in the update gate. b_z is the corresponding bias. The σ function squash the values into 0 to 1 in equation(15). Therefore, z_t is determining how many portion of historical information, h_{t-1}, and candidate hidden state , $\tilde{h}_t$, are contributing to the current hidden state, h_t. For example, if the current input is deemed as completely useless, z_t would be 0 so the current hidden state would completely depend on the historical information, $\tilde{h}_t$.

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \tag{14}$$

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes \tilde{h}_t \tag{15}$$

3. Experimental Analysis

This section is to apply recurrent neural networks to predicting stock prices. QQQ is the selected stock to be predicted which is one of the most commonly traded stocks in the market. The details of the dataset applied would be explained more in the corresponding section. The batch size of the dataset is set to 256. The optimizer would be SGD with a learning rate of 0.05 and the loss function would be the mean squared error. During training, the performance of the model is tracked with the validation dataset at the end of each epoch. If the performance has not improved, an early stopping would be implemented. The setting mentioned above is implemented throughout the study unless specified.

3.1. QQQ Stock Price Dataset

This study is implemented on the QQQ stock price dataset from Yahoo Finance. QQQ is an exchange-traded

fund that tracks the Nasdaq-100 Index.[1] The Index includes the 100 largest non-financial companies listed on the Nasdaq based on market cap.[1] The QQQ stock price dataset has 5952 rows and 7 columns in total originally. Each row represents a trading day. Each column is information of the QQQ stock. They are Date, Open, High, Low, Close, Adj. Close and Volume which represent the trading date, the stock price when the market opens, the highest stock price of the day, the lowest stock price of the day, the price when the market closes, the close price but include any cooperate actions and the number of shares traded respectively. The purpose of this study is to predict tomorrow's stock price so a Label column is created in the QQQ stock price dataset which is the close price of the next day.

The original data set is processed and modified to be implemented in this study. The modified data set discarded all features in the original dataset but only create a certain look-back period of close prices as the features and the labels remain tomorrow's close price. Let C_t be today's close price and n is the look-back period. The features would be $\{C_{t-n+1}, C_{t-n+2}, ..., C_t\}$ and the label would be $\{C_{t+1}\}$.

The new dataset was further split into 3 datasets, the training set, the validation set and the testing set in a ratio of 80:10:10 to serve different purposes. The training set is for training neural networks to optimize the weights, the validation set is for comparisons between neural networks to find the best one and the testing set is for evaluating how good is the neural network for the designed scenario. The new datasets also undergo normalization. Neural networks are best to handle data within a small range, such as 0 to 1, because large/small numbers or inconsistent scales of numbers may create unequal weights in the neural network. It makes the neural networks hard to train and sometimes impossible to converge to a minimum. Therefore, the close price is normalized and scaled between 0 to 1 before implementing it into neural networks in the respective dataset.

3.2. Baseline

A baseline model is built to measure how well neural networks perform. The baseline model is a simple statistical model based on the close price. The model is calculating the average of the differences between n previous trading days and adds to today's close price. The equation is shown in equation (16). C_t is today's close price and C_{t-n} is the close price n day before. The motivation for how it works is that it captures the short-term fluctuation of the stock price and predicts that tomorrow's stock price will follow the short-term fluctuation.

$$C_{t+1} = C_t + \frac{(C_t - C_{t-1}) + (C_{t-1} - C_{t-2}) + (C_{t-2} - C_{t-3})}{n} \tag{16}$$

	Mean	Standard deviation
Neural Net		
RNN	0.0342	0.0655
LSTM	0.0818	0.0790
GRU	0.0148	0.0417
Sequence Length		
2	0.0533	0.0795
16	0.0408	0.0774
64	0.0367	0.0476
Hidden Size		
2	0.0758	0.0834
32	0.0217	0.0462
128	0.0334	0.0628
Layer		
1	0.00331	0.00525
2	0.0498	0.0679
3	0.0777	0.0846

Table 1. Mean and Standard deviation of the grid search of types of model, sequence lengths, hidden sizes and the number of layers

3.3. Grid Search Exploration

This section is to explore how choices of parameters would influence the performance of the respective neural network. A grid search is performed to find the best setting from the type of recurrent neural network, the sequence length, the number of neurons in the hidden state and the number of layers. The mean and standard deviation of the parameter across all settings are calculated to evaluate how big its influence is on the performance. Some notable relations between parameters are found and presented in graphs.

3.3.1 RNN vs LSTM vs GRU

This section is to analyze which recurrent neural network, RNN or LSTM or GRU, is the best one for predicting QQQ stock price. Though RNN, LSTM and GRU are all classified as recurrent neural networks, their structure are built differently and the difference in structure can make a great deviation in performance.

The result shows that GRU is the best memory neural network among all on average, followed by RNN and the worst is LSTM. Table 1 summarizes the result. The means MSE loss of RNN, LSTM and GRU are 0.0342, 0.0818 and 0.0148 respectively. The average loss of GRU is only about 2/5 and 1/5 compared to RNN and LSTM. In addition, the standard deviations are 0.0655, 0.0790 and 0.0417 respectively. The standard deviation of GRU is also the lowest among the 3 neural networks. It means that not only GRU gives the best performance, but it also gives the most stable performance across all settings. A point to note is that though GRU has the best performance on average, the best

individual neural network is from RNN. One of the reasons that LSTM has the worst performance on average may associate with its complexity in the structure and insufficient data. The extra cell states and gates increase the complexity of the structure. Also, the QQQ stock price dataset is a small dataset that only has less than 6000 data entries. The insufficient data is not capable to train the complex LSTM which leads to poor performance.

3.4. Sequence Length

This section is to analyze the how sequence length of the input would affect the performance of neural networks. The sequence length of input determines the time step of each input and also the history of stock price. We tested sequence lengths of 2, 16 and 64.

The result shows that the sequence length has a positive relationship with the performance. The means MSE loss of sequence lengths 2, 16 and 64 are 0.0533, 0.0408 and 0.0367 respectively. The standard deviation is 0.0795, 0.0774 and 0.0476 respectively. It shows that the loss decreases with the increase of the sequence length. It makes sense that the sequence length represents the look-back period and the number of features of the feature vector to be input. The longer sequence length means more information is provided and included so the prediction would be better and hence the loss would be smaller.

Figure 4 provides insight that how different types of recurrent neural networks handle different sequence lengths. The result matches the expectation that RNN in fact could only handle short sequence data. The MSE losses of RNN remain low in shorter sequences but the loss increases dramatically when the sequence length increases to 64 from 32. A possible reason is that RNN suffers from the vanishing gradient problem. LSTM has poor performance regardless of the sequence length. The losses are high in all 2-, 16- and 64-sequence length. The complex structure of LSTM and the insufficient data may be the causes. The extra cell state and gates increase the complexity of the neural network and make it harder to train, compared with RNN and GRU. The QQQ stock price dataset is a small dataset that only has less than 6000 data entries. The insufficient data and the complexity of the LSTM cell make it has poor learning and hence lead to poor performance. In contrast, GRU has performed well in all sequence lengths. The reason may be GRU is capable to handles the vanishing gradient problem in long sequence length but it is still trainable.

3.5. Number of Neuron

This section is to analyze how performance change when the number of neuron in the hidden state alters. A change in the number of neurons also means a change in the number of weights in the neural network. When there is an increase in the number of neurons in the hidden state, there is an

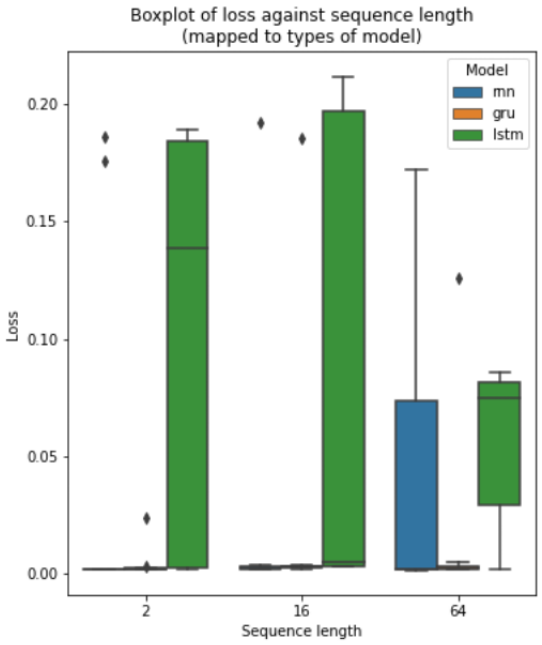

Figure 4. Boxplot of loss against sequence length

increase in the number of weights and hence increase the complexity and flexibility of the neural network, and vice versa. We tested 2, 32, 128 neurons in the hidden state to track the change in performances.

The result shows that the loss decreases with the number of neurons in the hidden state but may reach a plateau afterward. The means of MSE loss of hidden sizes 2, 32 and 128 are 0.0758, 0.0217 and 0.0367 respectively. There is a significant drop in MSE loss when the hidden size increases from 2 to 32 but drops a little when the hidden size further increases to 128. The standard deviation of hidden sizes 2, 32 and 128 are 0.0834, 0.0462 and 0.0628. A possible reason for the trend is that hidden size 2 has too few neurons in the hidden state and hence the number of weights is too little. There is not enough complexity to capture the information on stock price movement and there is not enough flexibility for the neural networks to adjust themselves to learn from the training data. However, hidden size 32 already provides enough complexity and flexibility for neural networks to capture the information needed so a further increase in the number of hidden sizes does not help in improving the performance.

3.6. Number of Layer

This section is to study the effect of changing the number of respective recurrent cell layers in the neural network on their performance. A study show that recurrent cells are

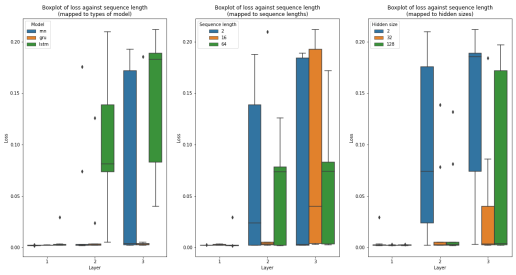

Figure 5. Boxplot of loss against layer in all parameter settings

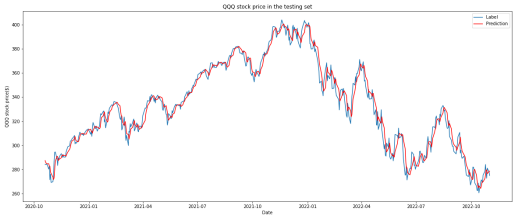

Figure 6. RNN predictions in the testing set

stacked to increase the complexity of the neural network and achieve the aim of getting better performance[2]. We tested to stack 1, 2 and 3 layers of respective memory cells in the neural networks.

The result shows that 1 layer of memory cell performs the best across all parameter setting. The means of MSE loss of 1, 2 and 3 layers are 0.00331, 0.0498 and 0.0777 respectively. The loss of 1 layer of a memory cell is only about 3/50 and 1/25 of 2 layers and 3 layers. The standard deviation are 0.00525, 0.0679 and 0.0846 respectively. The increase in the number of layers significantly increases the loss of the neural networks and decreases the performance of the neural network. From Figure 5, the loss of 1 layer is very low across all parameter settings, no matter which type of recurrent neural network is used, how long the sequence length is, and how large is the hidden size. It means that the number of layers makes the largest influence on determining the performance of the neural networks. The increase in the number of layers could suppress the performance of the neural networks. A possible reason is that an increase in the number of layers in the memory neural network highly increases the complexity of the neural network which is unnecessary. The structure and architecture of the neural network have changed and become too complex. It does not help in capturing more information but the complex architecture makes the neural networks hard to adjust their weights and learn from the training set during back-propagation through time.

3.7. Overall Performance

From the exhaustive grid search exploration, we picked the one with the lowest loss. It is a RNN model with 64-sequence length, 128-hidden size and 1-layer of RNN cells. The best model is then passed to the testing set to evaluate the performance.

The unnormalized MSE loss of the RNN model in the testing set is 34.820. Though the loss is small in number, it is not a particularly decent result. The baseline model is a simple statistical model which averages the ups and downs of the look-back period only which is described in the experimental analysis section. The unnormalized MSE loss of the baseline in the testing set is 26.620. The loss of baseline is lower than the RNN model despite its intensive computation and the complexity of the RNN.

Figure 7 shows a histogram describing the distribution of the price difference between the prediction of the RNN and the labels. It shows that the price difference is mostly normally distributed with a little right-skewed. The model neither has a tendency to underestimate nor overestimate the stock prices.

A point to note is that the predictions from the RNN are likely to copy the price patterns of yesterday. As Figure 6 shown, the patterns of the prediction are very much similar to the label but lag behind 1 day. It may imply that the whole neural network makes predictions of tomorrow's price mostly based on today's price.

There are two possible explanation account for the poor performance of the RNN. The first one may be the features of the input is insufficient and additional information is needed. Stock prices are affected by a lot of factors such as earnings, economic environment, sentiments or news headlines. Only using the historical stock price in the look-back period is insufficient to predict the stock price of tomorrow. The second reason is a recurrent neural network alone is too limited to predict the stock price movement. A recurrent neural network only uses a hidden state vector to capture all the historical information which may not be enough and a lot of information is lost during the propagation through time.

4. Code

https://github.com/fuchuenli/DeepLearningAssignment3.git

5. Conclusion

In summary, a recurrent neural network is using historical information, which is the memory, and current input to learn the pattern of the sequence data and makes predictions. There are 3 common recurrent neural networks which are RNN, LSTM and GRU. All of them have their own advantages and drawbacks. RNN has short memory but is cheap in computation and LSTM and GRU can handle

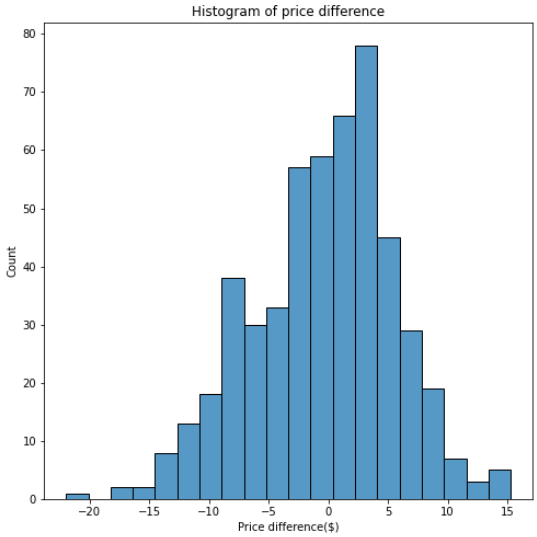

Figure 7. Distribution of price difference of RNN in testing set

a longer term memory but computationally expensive. The sequence length of the data and hyper-parameters of the recurrent neural networks can be adjusted to find the optimal setting to solve the designed scenario. In the study, to predict QQQ stock price, it is found that the GRU type of neural network has the best performance on average but a RNN has the best individual performance. The performance of the neural network increase with the sequence length. The increase in the number of neurons in the hidden states can increase the performance of neural networks generally but there is a plateau that a further increase does not help in improving the performance. The increase in the number of layers increases the complexity of the neural network which makes it harder to train with a small number of data available.

By taking the best individual model to the testing data, it was found that the result is not decent. The MSE loss is 33.652 which is higher than a simple statistical model with a MSE loss of 26.620 despite the intense computation and the complexity of the model. The poor performance may account for the insufficient feature of input or the nature of the stock market. To further improve the performance of predicting the future stock price, there may be two possible methods. The first method is to gather more information and create new features for the input. For example, a profitable company tends to have growth in its stock price and stock price tends to drop in a national/global recession. We may add profit margin data or national unemployment data into the feature vector to improve performance. The second method would be trying to replace or modify the architecture of the neural network. A recurrent neural network alone has its limitations. A study by Dai et al. found that introducing an attention module with LSTM can achieve a better performance than LSTM alone[3]. Moreover, Wu et al. show that using a convolutional neural network with leading indicators of stocks can achieve superior performance[4]. Therefore, studying different neural networks to predict stock prices could be a possible future work.

References

[1] NASDAQ, "NASDAQ's Homepage for Retail Investors," NASDAQ.com, 2019. https://www.nasdaq.com/

[2] Pascanu, R, Gulcehre, C, Cho, K Bengio, Y 2013, 'How to Construct Deep Recurrent Neural Networks'.

[3] Dai, H, Wang, W, Cao, J Wu, H 2021, 'A Deep Neural Network for Stock Price Prediction', Journal of Physics. Conference Series, vol. 1994, no. 1, p. 12029–.

[4] Wu, JM, Li, Z, Srivastava, G, Tasi, M Lin, JC 2021, 'A graph-based convolutional neural network stock price prediction with leading indicators', Software, Practice Experience, vol. 51, no. 3, pp. 628–644.