RNNs for stock price prediction

Sifan Xu
a1816684@adelaide.edu.au

Abstract

Recurrent Neural Networks(RNN) was designed to process sequence data which is difficult for traditional neural network model to handle. In this report, a simple RNN model and a LSTM RNN model were built to predict the stock price.

1. Introduction

In traditional neural network model, the layers are fully connected and the nodes between each layer are connectionless. This kind of ordinary neural network cannot handle many problems well like sequence data. For example, if a prediction is made to forecast the next word in a sentence, the preceding words are usually need to used, as the preceding and following words in a sentence are usually dependent. Therefore, Recurrent Neural Networks were designed to process sequence data. In RNN model, the connection between nodes can create a loop, allowing the output of some nodes to affect the subsequent input of the same node. [1]

In this report, a simple RNN model and a LSTM RNN model will be built to predict the stock open price. The performance of the models will be compared to discuss the advantages of the models.

2. Data

The data used in this project were provided in https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset. This dataset contains historical daily prices for all tickers currently trading on NASDAQ. The price data of each stock is stored in a CSV files. The files contain 8 columns: Date, Open, High, low, Close, Adj Close and Volume. The first column shows the change of this stock on the date, the last column is the number of shares that changed hands during a given day, and the other columns means the price of this stock in different time periods. Table 1 shows part of data in a file named 'A.csv' which is the first file in the 'stocks' folder. In this report, the data from 'A.csv' will be used to train the RNN models.

	Date	Open	High	Low	Close	Adj Close	Volume
0	1999-11-18	32.546494	35.765381	28.612303	31.473534	27.068665	62546300
1	1999-11-19	30.713520	30.758226	28.478184	28.880543	24.838577	15234100
2	1999-11-22	29.551144	31.473534	28.657009	31.473534	27.068665	6577800
3	1999-11-23	30.400572	31.205294	28.612303	28.612303	24.607880	5975600
4	1999-11-24	28.701717	29.998211	28.612303	29.372318	25.261524	4843200
...	...	...	...	...	...	...	...
5119	2020-03-26	70.000000	74.449997	69.650002	73.720001	73.532867	3267500
5120	2020-03-27	71.550003	73.209999	70.279999	70.910004	70.730003	1829800
5121	2020-03-30	71.059998	73.180000	71.059998	72.669998	72.669998	1486200
5122	2020-03-31	72.339996	72.800003	70.500000	71.620003	71.620003	1822100
5123	2020-04-01	69.470001	70.230003	68.150002	68.919998	68.919998	2173600

5124 rows × 7 columns

Table 1: part of data in A.csv

3. Method

In this project, Recurrent Neural Network (RNN) is used to predict stock price. RNNs are loops as they perform the same task on each element of the input, and the output depends on previous calculations. Theoretically, RNN can use sequence data of any length, but it usually backtracks to a fixed length in practice. Figure 1 shows an RNN is expanded into a complete network.

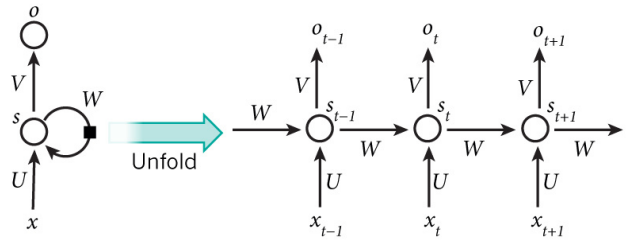

Figure 1: RNN (https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-1/)

x_t is the input at time t, and s_t is the hidden state at time t. It's the "memory" of the network. s_t is calculated based on the following formula:

$$s_t = f(Ux_t + Ws_{t-1})$$

s_t is determined by the previous hidden state and the input at the current. The function f usually is a nonlinearity such as tanh or ReLU.

o_t is the output at step t. It is obtained by s_t through some

function changes. For example, when predicting the next word of a sentence, the output is expected to be a vector composed of the probabilities of all words in our dictionary and o_t could be obtained by the following formula:

$$o_t = softmax(Vs_t)$$

In conclusion, s_t is the hidden state, it captures information at a previous point in time while it cannot capture information at all previous time points. o_t is obtained from the current time and all previous memories, it does not exist in many cases for reasons that many tasks, such as text emotion analysis, focus only on the final results. Each cell in the RNN network shares a set of parameters and this can greatly reduce the amount of calculation. [7]

3.1. Forward propagation through time

In RNN, importing sequence data into the network should accord to its time sequence, as the time sequence represents the concept of time step. In addition, the hidden state is important for reasons that hidden state is the intermediate value connecting the neurons of each hidden layer.

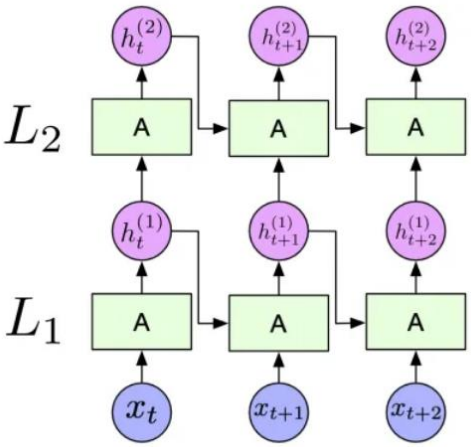

Figure 2: forward propagation of RNN(https://zhuanlan.zhihu.com/p/108276255)

As shown in Figure 2, RNN hidden layer neurons get the hidden state $h_t^{(1)}$ in the time step, and at the time step t in the first layer. At step t+1, it receives the hidden layer output $h_t^{(1)}$ from the previous time step t to get a new hidden state $h_{t+1}^{(1)}$. Each layer is also connected by the hidden state. For each layer, the output of the front step t and the input x_{t+1} will both be the input of the next step t+1 to get a new state. Therefore, RNN is a structure that can be expanded in both horizontal and vertical directions.

According to the definition of RNN, the forward propagation process of RNN are as follows:

$$h_t = g\left(Wx_t + Vh_{t-1} + b\right)$$

W, V and b are parameters that need to be learned by the model, the parameters are the same in one layer of RNN. If the RNN model has many layers, the formular is as follows:

$$h_t^{[i]} = g\left(W^{[i]}h_t^{[i-1]} + V^{[i-1]}h_{t-1}^{[i]} + b^{[i]}\right)$$

After obtaining the hidden state, if it is necessary to make linear transformation to obtain the output of each time step such as softmax function.

3.2. Back propagation through time(BPTT)

BPTT can be seen as a standard backpropagation algorithm applied in RNN to find the gradient of the cost with respect to all the network parameters.. Each time step represents a computing layer, and its parameters are shared across computing layers. For reasons that RNN shares the same parameters in all time steps, an error in a time step must reverse to all previous time steps through time, hence the name of the algorithm. When processing long sequences (hundreds of inputs), an abridged version of BPTT is often used to reduce computing costs. The truncated BPTT stops backpropagation of errors after a fixed number of steps. [3] In RNN, BTPP algorithm is to propagate the error value at time t of layer l in two directions. One direction is to transfer to the upper network, which is only related to the weight matrix V, the other direction is to transfer to the initial time along the timeline, which is only related to the weight matrix W.

3.3. Gradient dispersion and explosion of RNN

During BPTT, the reverse propagation gradient of RNN unit is as follows:

$$\Delta W_t = \left(Loss(output, y)'g(\cdot)'\prod_t^{T-t} f\left(W[h_{t-1}; x_t]^\top + b\right)'\right)[h_{t-1}; x_t]$$

The active functions usually use tanh or sigmoid function which are shown in Figure3.

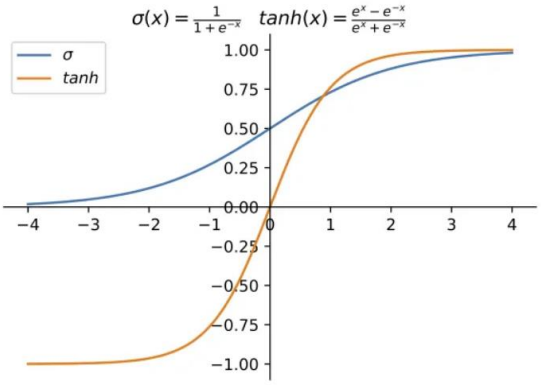

Figure 3: tanh and sigmoid function

The derivatives of tanh and sigmoid function are shown in Figure 4.

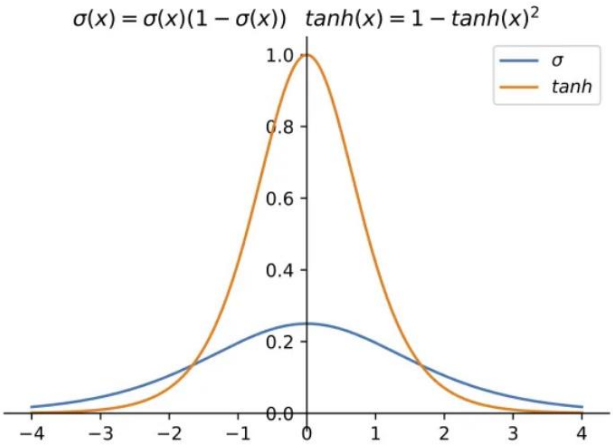

Figure 4: derivatives of tanh and sigmoid function

It can be seen from the figures that the derivative is close to 0 at both ends of the activation function. Assuming that the current time step is at the last time step t, the multiplication of multiple numbers close to 0 will result in an exponential downward trend of the gradient, and cause the gradient to disperse during the BPTT. Therefore, RNN has no long-term memory, but only short-term memory. Due to gradient dispersion, when the sequence length is very long, it is impossible to update W of the previous time step according to the gradient in the later time step. As a result, it is difficult to extract features from forward sequence because the parameters of the forward sequence cannot be modified according to the subsequent sequence. The model will not be able to obtain valid forward sequence memory information after long time step. [8]

Similarly, according to the derivation of the above gradient, the accumulation of parameters will lead to gradient explosion. If the initial parameters are large, the multiplication of large numbers will lead to gradient explosion. However, gradient explosion is easier to solve than gradient dispersion, adding gradient clipping can alleviate the problem to a certain extent.

3.4. RNN architecture

The common architecture composed of RNN are shown in Figure5. The first is a common single neural network. The second is to convert a single input into a sequential output. The third is to convert the sequence input into a single output. The fourth is to convert a sequence into a sequence. The fifth is the sequence to sequence transformation without time difference.

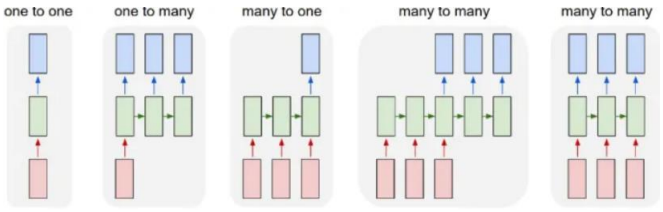

Figure 5: RNN architectures(https://stackoverflow.com/questions/43284284/many-to-one-implementation-in-keras)

3.4.1 One to many

In 1 to N structure, there are two kinds of way to input the data which are shown in Figure 6. One is to perform input calculation only at the beginning of the sequence, and the other is to take input information X as the input of each stage. This structure can handle many problems, such as image annotation. The output X is the image feature, while the output Y sequence is a sentence or generates voice or music from a category. The input X is a category, and the output Y is a string of characters.

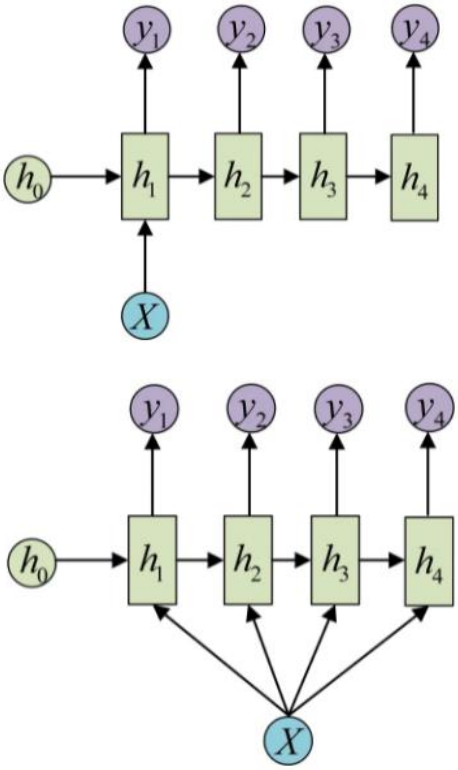

Figure 6: two kinds of 1 to N (https://zhuanlan.zhihu.com/p/34152808)

3.4.2 Many to one

The N to 1 structure is often used to deal with sequence classification problems. The input is a sequence, and the output is a single value rather than a sequence. For example, inputting a text to judge its category, inputting a sentence to judge its emotional tendency and inputting a document to judge its category.

3.4.3 *Many to many*

The many to many RNN also has two kinds. The first is N to N, the input and output sequences are of equal length. This can be used as a simple Char RNN to generate articles. The other is N to M structure, it is also called Encoder Decoder model, or Seq2Seq model. Practically, most of the sequences are unequal in length. For example, sentences in the source language and target language often do not have the same length in machine translation. The Encoder Decoder structure first encodes the input data into a context vector c, and then outputs the prediction sequence through this context vector.

3.5. Long Short Term Memory(LSTM)

The most essential reason for the gradient explosion or disappearance of RNN structure is that there are a large number of successive gradients in the process of transfer. Therefore, LSTM RNN model was proposed to remember valuable information and give up redundant memory, thus reducing the difficulty of learning. Compared with RNN, LSTM neurons add input gate i, forget gate f, output gate o and candidate memory c.[9] LSTM neurons are still calculated based on the input X and the output h of the upper hidden layer. However, the internal structure has changed which means the calculation formula of neurons has changed, while the external structure has not changed.[4] Therefore, LSTM can be used to replace various RNN structures mentioned above.

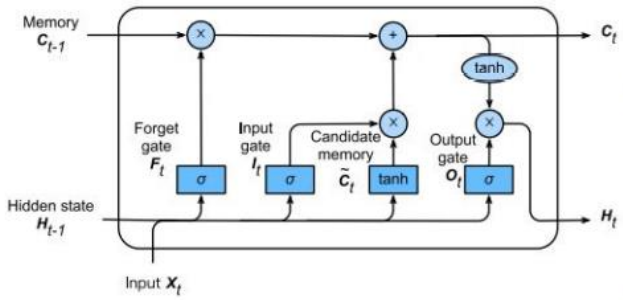

Figure 7: LSTM RNN

3.5.1 *Forget gate*

The forget gate controls the degree of forgotting input X and output h of the previous hidden layer. The formula is as followed.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

3.5.2 *Input gate*

The input gate obtaining the following formula controls the extent to which the input X and the current calculated state are updated to the memory unit.[10]

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

3.5.3 *Candidate memory*

According to the forward propagation of traditional RNN, the input should be activated after linear transformation, and the activation function usually uses tanh. Then use the element wise product, that is, the cell state of the current time, which is equal to the cell state of the previous time, filtered by the forgetting gate, and then superimposed with the information sum of the input gate. As an option to generate candidate memory c, the tanh function has an output of - 1~1, which conforms to the characteristic distribution of the center of 0 in most scenes, and the gradient (derivative) is close to 0. The convergence speed is faster than that of the sigmoid function.

$$c'_t = Tanh(W_c x_t + U_c h_{t-1})$$
$$c_t = f_t * c_{t-1} + i_t * c'_t$$

3.5.4 *Output gate*

The output gate controls input X and current output depend on the extent of the current memory unit. The formulas are as followed.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$
$$h_t = o_t * Tanh(c_t)$$

Generally, Sigmoid is selected as the excitation function mainly for gating. Because the output of the sigmoid function is 0~1, when the output is close to 0 or 1, it conforms to the physical sense of off and on.

4. Experiment

This project aims at learning RNN and building a CNN model to predict the stock open price. A simple RNN model and a LSTM RNN model were built and compared in the experiment.

4.1. Load and preprocess data

The “A.csv” file is loaded to obtain the data set. The data in “A.csv” have 5124 lines and 8 columns, each line represents the stock price at different time on that day. The first column shows the change of this stock on the date, the last column means the number of shares that changed hands during a given day, and the other columns means the price of this stock in different time periods.

Next, the data set was spilt into train set and test set. The open price data of the first 4500 days are regarded as training set to train the model, in another word, the first 4500 data in the second column were extracted. The remain data were used as test set to check the performance of the model. Figure 8 shows the plot of the training set and testing set.

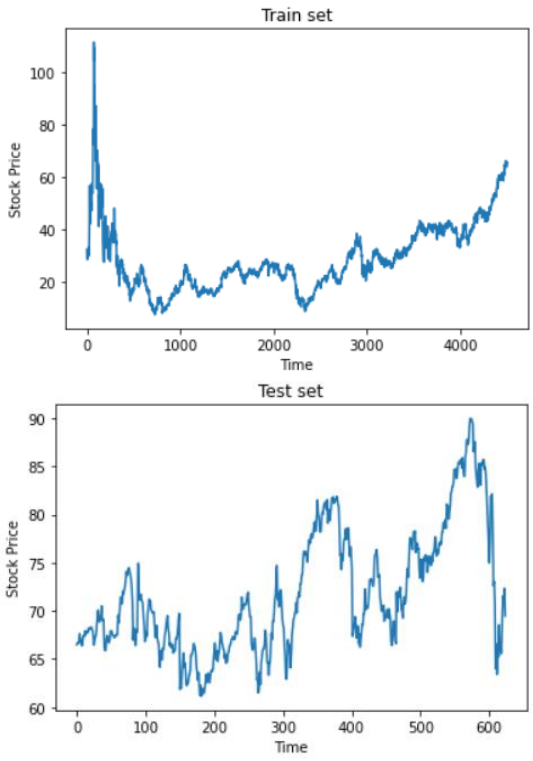

Figure 8: Plot of train set and test set

After the training set and test set have been divided, these data need to be normalized to accelerate the network training convergence.

The last step of data preprocessing is reshaping the data. In this report, the open price of the first 14 days as the input features, and the open price on the 15th day as the input label. In other words, the RNN model in this report uses stock open price in the previous 14 days to predict the stock open price at 15th day. The sliding window keep sliding and make predictions. Therefore, the features are arrays with length of 14 and the labels are the open price value corresponding to the next day.

4.2. Simple RNN model

The Keras Sequential API was used when building the simple RNN model in this report, it could add one layer at a time. Table 2 shows the architecture of the simple RNN model in this project.

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 10)	120
dense (Dense)	(None, 1)	11

Total params: 131
Trainable params: 131
Non-trainable params: 0

Table 2: simple RNN architecture

First, defining the input layer and the hidden layer with 10 neurons. Next, 1 fully-connected layers was added in the model. The first fully-connected layer contains 1 unit. Finally, defining the optimization algorithm ADMA, and use momentum and adaptive learning rate to speed up convergence. The mean squared error was used as loss. After building the simple RNN model, the batch size was set as 64 and the epochs was set as 20 when training the model.

4.2.1 Evaluate the simple RNN model

The loss of the training set and test set are shown in Figure 9. According to the picture, the training loss trends to 0 after 1 epoch and validation loss trends to 0 after 7 epochs, both training set and validation set have low loss after 20 epochs, the overfitting do not happen.

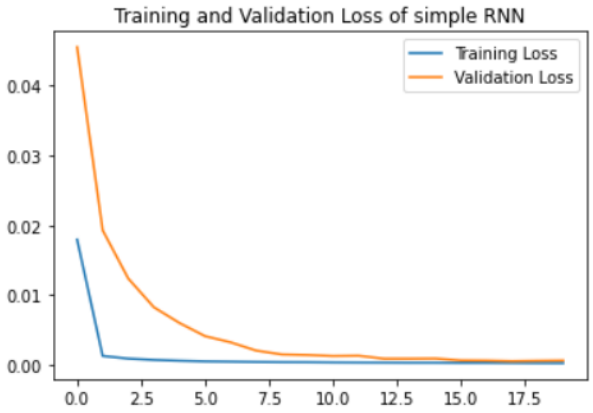

Figure 9: Loss curves of simple RNN

Figure 10 shows the predicted price of simple RNN. The red curve is the real open price since the 15th day in the test set and the blue curve is the predicted open price since the 15th day. According to the picture, simple RNN can not predict the stock price well.

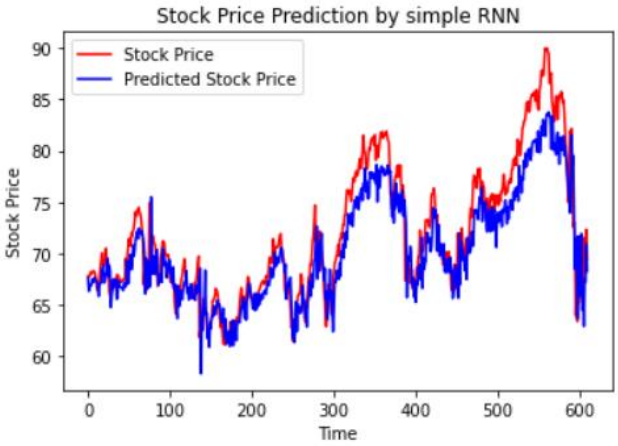

Figure 10: Stock Price Prediction by simple RNN

R^2 is also used to reflecting the performance of the model. This value was calculated by the following formular:

$$R^2 = 1 - \frac{\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{m}(y_i - \bar{y}_i)^2}$$

In this practice, the R^2 is only 0.74. In other tests, the model and the test data were controlled as same. However, the R^2 ranges from -2.4 to 0.95. Therefore, the simple RNN can not predict the stock price.

4.3. LSTM RNN model

Since the simple RNN model can not predict the stock price well, the LSTM RNN model were used to predict the open price.

The structure of LSTM is almost the same as simple RNN. Table 3 shows the architecture of the LSTM RNN model in this project. It also has the hidden layer with 10 units and 1 fully-connected layer contains 1 unit. The batch size was also set as 64 and the epochs was set as 20 when training the model.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10)	480
dense_1 (Dense)	(None, 1)	11

Total params: 491
Trainable params: 491
Non-trainable params: 0

Table 3: LSTM RNN architecture

4.3.1 Evaluate the LSTM RNN model

The loss of the training set and test set are shown in Figure 11. According to the picture, the training loss trends to 0 after 1 epoch and validation loss also trends to 0 after 1 epoch, both training set and validation set have low loss after 20 epochs which means the overfitting do not happen.

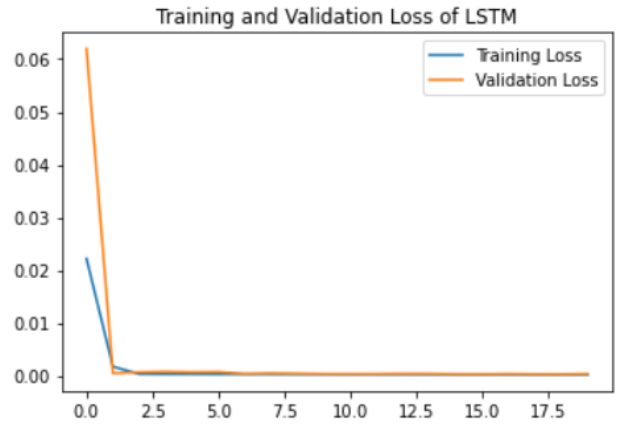

Figure 11: Loss curves of LSTM RNN

Figure 12 shows the predicted price of LSTM RNN. The red curve is the real open price since the 15th day in the test set and the blue curve is the predicted open price since the 15th day. According to the picture, the predicted curve is close the real curve, their trends are roughly the same. The LSTM RNN performs better than simple RNN in predicting the stock price.

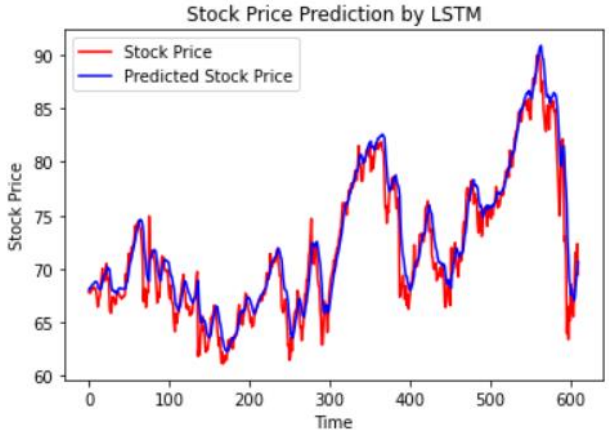

Figure 12: Stock Price Prediction by simple RNN

The R^2 is in this practice is 0.91. However, the value of R^2 also changes in other tests although the model and the test data remain the same. The R^2 ranges from 0.75 to 0.96. According to the above experiments, the prediction value of stock open price under LSTM model is more accurate and stable.

5. Code

The code could be found on github at https://github.com/a1816684/Deep-learning-assignment3/blob/main/assignment3.ipynb

6. Reflection

In this project, a simple RNN model and a LSTM RNN model were built to predict the stock price. This project provides me with a preliminary understanding of RNN model such as the principle and architecture of it. RNN was designed to predict the sequence data which could not be processed well by traditional neural network. There are many application fields of RNN, almost all problems of time sequence can be solved by RNN. RNN model has such advantages like having the ability to process the input of any length, keeping the model shape regardless of input length, considering the historical information, and sharing weight over time. However, it also has the disadvantages like calculating slowly, hard to get information at long time step, and cannot import the future state as the input of the current state.

Since RNN has defects in processing long term memory, the LSTM RNN was designed to solve the problems. LSTM neurons add input gate i, forget gate f, output gate o and candidate memory c based on RNN structure. LSTM makes the derivative of the current memory cell to the previous memory cell a constant, which alleviates the gradient loss problem caused by decimal multiplication. While the parameter quantity of LSTM is 4 times that of

RNN which could lead to overfitting and add the time of training.

In the future, I plan to implement GRU RNN as the number of GRU parameters is smaller than LSTM which reduces the risk of overfitting. GRU only uses two gating switches, which has similar performance to LSTM. The number of parameters is three times that of simple RNN.

In addition, I wish to learn a better model to predict the stock price. Although it seems that LSTM performs well on predicting stock price, it only uses a value very close to the price of the previous day as the forecast value of the next day[6].

References

[1] Dupond, Samuel (2019). "A thorough review on the current advance of neural network structures". Annual Reviews in Control. 14: 200–230.

[2] Yu, W, Gonzalez, J & Li, X 2021, 'Fast training of deep LSTM networks with guaranteed stability for nonlinear system modeling', Neurocomputing (Amsterdam), vol. 422, pp. 85–94.

[3] P. J. Werbos, "Backpropagation through time: what it does and how to do it," in Proceedings of the IEEE, vol. 78, no. 10, pp. 1550-1560, Oct. 1990, doi: 10.1109/5.58337.

[4] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[5] Abdel-Nasser, M., Mahmoud, K. Accurate photovoltaic power forecasting models using deep LSTM-RNN. Neural Comput & Applic 31, 2727–2740 (2019).

[6] Andrea Leccese 2019, Machine Learning in Finance: Why You Should Not Use LSTM's to Predict the Stock Market

[7] Fang, W., Jiang, T., Jiang, K., Zhang, F., Ding, Y., & Sheng, J. 2020. A method of automatic text summarisation based on long short-term memory. International Journal of Computational Science and Engineering, 22(1), 39-49.

[8] X. Guo, "UL-net: Fusion Spatial and Temporal Features for Bird Voice Detection," 2022 IEEE 2nd International Conference on Electronic Technology, Communication and Information (ICETCI), 2022, pp. 1009-1013, doi: 10.1109/ICETCI55101.2022.9832357.

[9] W. Xu and C. Yan, "Prediction of Lithium-ion Battery Remaining Useful Life Based on Long Short Term Memory," 2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), 2022, pp. 942-948, doi: 10.1109/AEECA55500.2022.9919036.

[10] Zhao, D., & Feng, L. 2021 A Two-Stage Machine-Learning-Based Prognostic Approach for Bearing Remaining Useful Prediction Problem. IAENG International Journal of Computer Science, 48(4).