Assignment 2: Recurrent Neural Networks for Stock Price Prediction, Deep Learning Fundamentals, 2020

Yumin Cao, a1754926
The University of Adelaide

Abstract

This assignment is to predict the stock price using Recurrent Neural Networks (RNN). The original data source comes from Google and can be downloaded from Kaggle https://www.kaggle.com/rahulsah06/gooogle-stock-price. The submission will take the Conference on Computer Vision and Pattern Recognition (CVPR) form.

1. Introduction

The Stock Price Prediction from Google is a widely-used data set for RNN beginners, it has training set and test set, with the training set consists of 1258 rows and 20 rows for the test set, and the features include Open, High, Low, Close and Volume. The assignment aims firstly at giving the background of the RNN's algorithms, especially for the Back Propagation Through Time (BPTT) algorithm. Then a serious of parameters would be turned to build a suitable model which can predict the results with low Root Mean Square Root (RMSE). Long Short Term Memory network (LSTM) is a special RNN, it was introduced by Hochreiter and Schmidhuber in 1997[1]. We used this method in this article because it's ability to study long-term dependence. However, using the past days' data to predict the future needs to confirm the sequence length of the past days (M). We optimize this problem based on the real word experience. For the CPU time problem, we use the greed search method when turning the parameters to reduce the running time. For the convergence speed problem, we also check the performance under varies of learning rates. We also compare the results based on LSTM with different layers. At last, a conclusion is given with figures to analyze the benefits and drawbacks of LSTM with its final performance.

2. Background

Before the RNN came out, scientists are suffered from data problems relating to time sequence[1]. BPTT is a gradient-based technology for training certain types of RNN, especially for those who need to consider the time series. It use the Forward Propagation to calculate the result of each step following the order of time and it use the Back propagation to transfer residuals. Although BPTT algorithm has difficulty with local optimal[2], it tends to be significantly faster for training recurrent neural networks.

There are some famous RNN nets like Vanilla-RNN, LSTM, Gated Recurrent Unit (GRU) and their variants, such as applying peephole connection to LSTM[3]. The following picture shows the cell (basic structure) of Vanilla-RNN, LSTM and GRU.

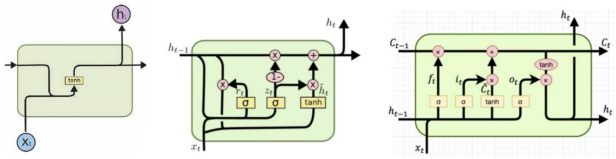


Figure 2.1 (vanilla, GRU, LSTM)[4]

Vanilla RNN is a simple RNN and can be seen as the multiple overlay structure of the same networks. It implies that RNN has a tight connection with time series, every network pass the combined information to the next network. This RNN net is simple but may face gradient vanishing or gradient exploding problem when the time series is long.

In order to improve the long-range dependence of the recurrent neural network, a gating mechanism is introduced to control the speed of information accumulation, including selectively adding new information and selectively forgetting past information. LSTM has three gates: forgotten gate, input gate and output gate, and it controls the information by two steps, one is the long-term internal state flow of the memory unit controlled by the top line c, and the other is the external output state h, which stores short-term memory information, and then forgets the memory unit again through the loop, and through the external output state h_t-1 at the previous moment to update three doors. GRU comes from LSTM, but it is more simple than LSTM, because it reduce the redundancy between input gate and forgotten gate, what's else, GRU doesn't have memory c, instead, it add the update gate and use the h to Control how much information the current state needs to retain from the historical state, and how much information is accepted from the candidate state.

As for the performance of LSTM, GRU and other variants, Greff has given a comparison between them and draw the conclusion that their performance are similar[5]. In this article, we will use LSTM, because LSTM has more parameters and has a stronger fitting ability[6], and from personal aspect, using LSTM helps me better understanding the RNN networks.

1

3. Method

How does RNN work:

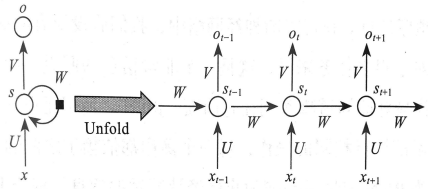


图 10-20　RNN 细节示意图

Figure 3.1[6]

The above picture is the simplest unfold structure of RNN, and it should has formulas:

$$S_t = F\left(Ux_t + Ws_{t-1}\right)$$

$$y = g\left(Vs_t\right)$$

Xt is the input at time t, St is the hidden state at time t, F is the activate function, U,V,X are the networks' parameters, g is the activate function.

However, as we have discussed, when the predict point is far from its correlated information points, then the RNN would be hard to learn the past information. So we use LSTM to handle this long-distance dependence problem.

How does LSTM work:

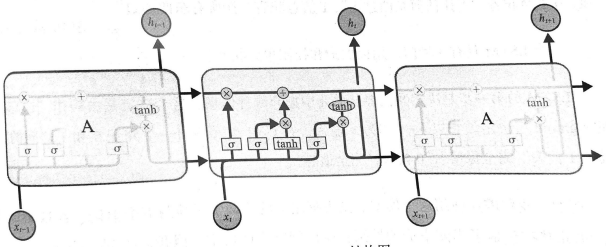


Figure 3.2[6]

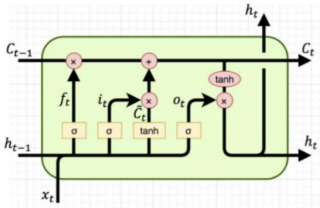


Figure 3.3[4]

From the above pictures we can see that line C links from the beginning cell to the ending cell, which makes information easy to flow through it.

The first step of LSTM is determining the information that we want to abandon, using:

$$f_t = \sigma\left(W_{if}Xt + b_{if} + W_{hf}X_{t-1} + b_{hf}\right)$$

The next step of LSTM is determining what do we want to update through input gate, and create a candidate vector Ct and update the Ct, using:

$$g_t = tanh\left(W_{ig}Xt + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right)$$

$$i_t = \sigma\left(W_{ii}X_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

The final step of LSTM is determining what do we want to output through output gate, and handle the state with tanh, using:

$$h_t = o_t \odot tanh\left(c_t\right)$$

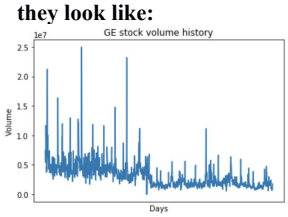
$$h_t = o_t \odot tanh\left(c_t\right)$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the layer at time t-1 or the initial hidden state at time 0, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and ⊙ is the Hadamard product.

How do we design & details of methods:

Preprocessing the data:

The format of two columns is wrong because it records the data like '1,008.64' and we transformed these string to numeric value. Then we normalized the data, because The data is not normalized and the range for each column varies, especially Volume. Normalizing data helps the algorithm in converging i.e. to find local/ global minimum efficiently. Then we give a look at the data, they look like:



Creating model:

My goal is using the past M days data to predict the next day's data (M+1), the reason of we just predict one more day is that this may has a higher accuracy and is more close to real world, because quantitative trading focus on the short period profit, the shorter the time, the less environmental factors will affect the final results. What's more, if we can use the M to get M+1, then we can also use the M+1 to get M+2.

We design two models, one is the simple layer LSTM and the other is the multiple layers LSTM to see whether the number of layers can affect the results.

The learning rate is set with four values: 0.0001, 0.005, 0.01, 0.1, we have know that 0.005 is an appropriate rate on the tutorial, the reason we design like this is because we want to see the performance under a relatively extreme small learning rate (0.0001) as well as a very big learning rate(0.1).

The epochs in this assignment is set to 40,because it is often not enough if we only train the CNN data once. And if the epoch is too big, we may face the over fitting problem and have to change the learning rate. Wei[7] mentions that the learning rate can be decrease when the epochs grows too big, like:

$$lr_t = lr_0/(1 + k * epoch)$$

The sequence length of M is important, and we set some values: 5, 15, 30, 45, 60, 90. we can use either 5 days' data to predict the 6th day's data, or 30 days. Based on the real world experience, 5 days means a week, 15 means half a month, then is one month, half and one months, two months, three months.

Because there are amount of parameters, so we choose greed algorithm to tune the parameters to reduce time and CPU. That is, fixing some parameters firstly to optimize the other parameters.

We choose the Mean Square Root as the loss function, because it is good for regression problem[8].

4. Experiment

At the beginning I want to see whether the dates are ordered without interval, then I found although there exists some dates that has interval, such as the next date after 10/10/2016 is 12/10/1026, but the dates' array is from the past to the present and even some dates have intervals, that date gap is small. So we treated them as continuous date time, which means when we use the M to get M+1, we will not consider the small internal impact brought by date.

As we mentioned at Method part, we tune the parameters and they include:

Number of layers for LSTM - 1,2,3,4

Hidden size - 64,128,256,512

Sequence number - 15,30,45,60,90

Learning rate - 0.0001, 0.005, 0.01, 0.1

Based on our experience, the simple layer LSTM can save time for running, and learning rate sets to 0.005 is fine, as well as hidden size sets to 128.

So my first step is separating the train set into train and cross set. I get two cross set, and the final loss = 1/2 * (loss(cross1) + loss(cross2)).

And my second step is, fixing the model layer(= 1), hidden size (= 128), learning rate(= 0.005) and tuning the sequence number, the result is 15:

```
temp = sorted(recording)
best_seq_len = temp[0][1]
best_seq_len
```

```
-- 0.20584889501333237
-- 0.44109421968460083
-- 0.5691082701086998
-- 0.3761592358350754
-- 0.6492492109537125
```

```
Out[41]: 15
```

Then we use the similar strategy, and the result of hidden size is 256:

```
temp = sorted(recording)
best_hidden = temp[0][1]
best_hidden
```

```
-- 0.5864179953932762
-- 0.47362641990184784
-- 0.39308302104473114
-- 0.4078792780637741
```

```
Out[44]: 256
```

find best learning rate

The best learning rate is 0.01:

```
best_lr = temp[0][1]
best_lr
```

```
-- 0.48425740748643875
-- 0.388694703578949
-- 0.3115990236401558
-- 1.2556000053882599
```

```
Out[45]: 0.01
```

single layer & multi layers ¶

Then we fix the parameters and check the layers(single best):

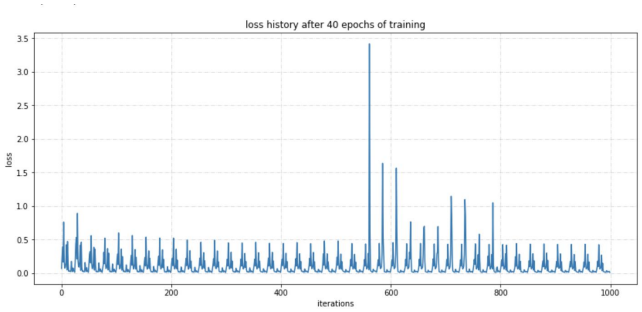
```
temp = sorted(recording)
best_layer = temp[0][1]
best_layer
```

```
-- 0.5493039265275002
-- 1.0446996241807938
-- 0.5689667016267776
-- 0.571867935359478
```

```
Out[47]: 'single'
```

Plot with train data

Till now, the parameters have been tuned, then we want to check the loss during each epoch, the average loss value is recorded after every 50 iterations.



This fluctuation is normal, because every epoch will go through the training data once. So, some features would repeat periodically.

Then for the test part, we notice that the end of training data is the end of 2016, and the test data is the start of 2017. Which means that if we use the last 15 data from train set, and get the predict value, and so on. Finally we can get a predicted list with each tensor comes from the former 15 days. Thus we can get the final results with mean absolute error from sklearn package. The error is 0.74.

```
new_data = scaler.fit_transform(new_data)
new_data = torch.FloatTensor(new_data).to(device)
```

```
final_test_seq_before = create_sequences(new_data, seq_len)
real_labels = []
predicted_labels = []
with torch.no_grad():
    for m, (seq, label) in enumerate(final_test_seq_before):
        seq = seq.to(device)
        label = label.to(device)
        real_labels.append(label[0].numpy())
        x = seq.reshape(-1,seq_len,5)
        predicted_labels.append(model_nolayer(x)[0].numpy())
#         print(len(predicted_labels),predicted_labels)
#         print(len(real_labels),real_labels)
#         print('****')
r = mean_absolute_error(real_labels, predicted_labels)
r
```

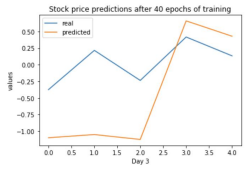
```
0.7375066
```

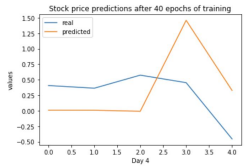
```
real_labels,predicted_labels
```

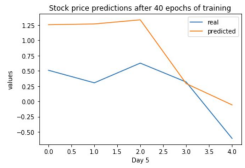
```
([array([-1.5292645 , -1.0937215 , -1.4882617 , -1.0474164 ,  0.16036312],
```

Finally, we can compare each predicted result (5 values represent open, high, low, close ,volume) with the true value.

Here we get 20 pictures. For example:







There are 10 points in each picture, the blue line represent the real values (we connected those points), and the yellow line represent the predicted values.

From the pictures we can see that, for some days, the prediction is accurate, because the values of points are similar, however for some days, there maybe some difference. But almost all of the predict values have the similar trend with the real values. It means our results follow the real world rules to some extent. The above observations means our model did get something.

5. Code

https://github.com/twofatcat/Deep-learning-Assignment3-Stock_price_prediction

6. Conclusion

For this assignment, I learned how to build RNN model using Pytorch, and reinforced my RNN knowledge, to get a better understanding of many to many/ many to one, as well as gate. Actually I found lots of sources relating to stock price predict, but they prefer to predict a single value, like open price. I wish to predict 5 values but I don't know whether I am wrong, although the results seems alright. Also, I am concerned with the date, if I follow the rules strictly, I should have separated and grouped them with the same date interval.

And, it seems that the errors of loss history has a cyclically sharp rise, this seems to be related to the similar cyclically sharp rise with volume. I tried to figure it out but I got stuck.

References

[1]Hochreiter, S. and Schmidhuber, J. (1997). ***LSTM can solve hard long time lag problems*****. In Advances in Neural Information Processing Systems 9. MIT Press, Cambridge MA. Presented at NIPS 96Alex, Ilya, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks, 2012.**

[2]M.P. Cuéllar and M. Delgado and M.C. Pegalajar (2006). ***An Application of Non-linear Programming to Train Recurrent Neural Networks in Time Series Prediction Problems.*** **Enterprise Information Systems VII. Springer Netherlands. pp. 95–102.**

[3]Gers, A. and Schmidhuber, J. (2000). ***Recurrent Nets that time and count*****. Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Volume. 3**

[4]Moghaddam, K, M. (2020). ***Recurrent Neural Nets. The University of Adelaide***

[5]Greff, K. (2015). ***LSTM: A Search Space Odyssey. TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS*****, Viewed 16/10/2020,**

[6]Xu, M. and Liu, X. (2018). ***Natural Language Processing Core Technology and Algorithm with Python.*** **Published: CIP.**

[7]XiuCan. Wei. ***The principle of Convolutional neural network and the practice of visualization.***

[8]Jha. ***A brief overview of loss function in Pytorch.*** **Viewed 15/10/2020, < https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7 >.**

References

[1]Hochreiter, S. and Schmidhuber, J. (1997). *LSTM can solve hard long time lag problems.* In Advances in Neural Information Processing Systems 9. MIT Press, Cambridge MA. Presented at NIPS 96Alex, Ilya, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks, 2012.

[2] M.P. Cuéllar and M. Delgado and M.C. Pegalajar (2006). *An Application of Non-linear Programming to Train Recurrent Neural Networks in Time Series Prediction Problems.* Enterprise Information Systems VII. Springer Netherlands. pp. 95–102.

[3]Gers, A. and Schmidhuber, J. (2000). *Recurrent Nets that time and count.* Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Volume. 3

[4]Moghaddam, K, M. (2020). *Recurrent Neural Nets.* The University of Adelaide

[5]Greff, K. (2015). *LSTM: A Search Space Odyssey.* TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, Viewed 16/10/2020,

[6]Xu, M. and Liu, X. (2018). *Natural Language Processing Core Technology and Algorithm with Python. Published: CIP.*

[7]XiuCan. Wei. *The principle of Convolutional neural network and the practice of visualization.*

[8]Jha. *A brief overview of loss function in Pytorch.* Viewed 15/10/2020, < https://medium.com/udacity-pytorch-challengers/a-brief-overview-of-loss-functions-in-pytorch-c0ddb78068f7 >.