Recurrent Neural Networks for Stock Price Prediction

Mingrui Zhao
a1717463
a1717463@student.adelaide.edu.au

Abstract

This paper is focused on how to use Recurrent Neural Networks algorithms to do Stock Price Prediction. The paper will introduce several methods of solving the problem, including vanilla RNN, LSTM and GRU. This paper will also demostrate the loss, accuracy and time efficiency for different methods by conducting several experiments, and finally, make conclusion on each of the algorithms.

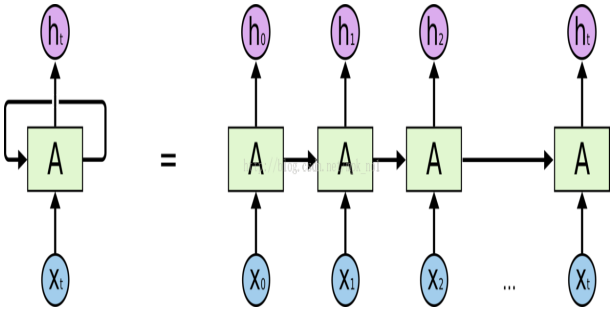

Figure 1. RNN structure

1. code

code link: https://github.com/MingruiZhao/RNN

2. Introduction

The task is to predict stock price using Recurrent Neural Networks (RNN). RNN is a special kind of neural network structure, which is based on the idea that "human cognition is based on past experience and memory". It differs from DNN and CNN in that it not only takes into account the input of the previous moment, but also gives the network a 'memory' function for the previous content. In our task, we will using vanilla RNN and LSTM as our model to predict stock price based on six variables and conduct further experiments.

3. Background

The CNN (Convolutional Neural Network) algorithm that I introduced before, we know that their output only takes into account the influence of the previous input and not the influence of the input at other moments, such as simple cats, dogs, handwritten numbers and other single-object recognition has good performance. However, these algorithms do not perform as well as they should for some time-sensitive tasks, such as predicting the next moment of a video, or predicting the content of a document. Therefore, RNN was born.

An RNN is called a recurrent neural network because the current output of a sequence is also related to the previous output. This is manifested by the fact that the network remembers the preceding information and applies it to the computation of the current output, i.e. the nodes between the hidden layers are no longer unconnected but connected, and the input of the hidden layers includes not only the output of the input layer but also the output of the hidden layer at the previous moment. There are many applications for RNN, and it can be said that RNN can be used to solve any time sequence problem. Such as: NLP, Voice recognition, machine translation etc.

One of the key points of RNNs is that they can be used to connect previous information to the current task, such as using a past video segment to infer understanding of the current segment. If RNNs can do this, they become very useful. But there are many more dependencies. Sometimes we simply need to know the prior information to perform the current task. For example, we have a language model that we use to predict the next word based on the previous word. If we try to predict the last word "the clouds are in the sky", we don't need any other context - so the next word should obviously be sky. In scenarios where the interval between the relevant information and the predicted word position is very small, the RNN can learn to use the previous information.

But there are also more complex scenarios. Suppose we try to predict the last word of the sentence "I grew up in France.... I speak fluent French". The current information

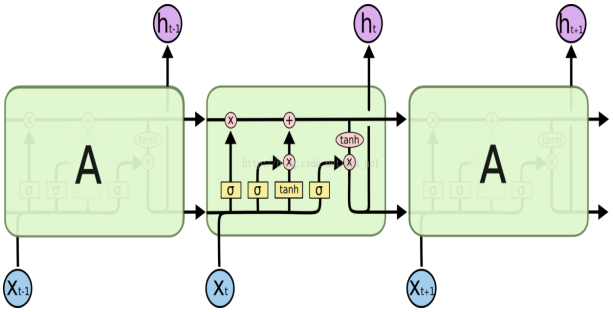

Figure 2. LSTM structure

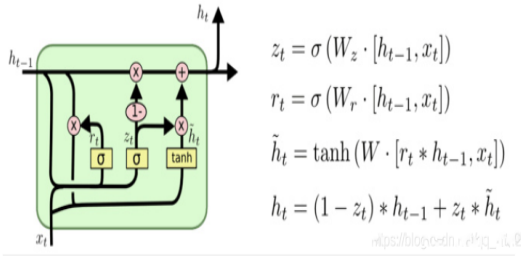

Figure 3. GRU structure

suggests that the next word might be the name of a language, but if we need to figure out what language it is, we need the context of France, which was mentioned earlier and is far away from the current position. This means that the gap between the relevant information and the current predicted location must become quite large. Unfortunately, as this gap increases, the RNN loses the ability to learn to connect such distant information. In theory, the RNN can definitely handle such long-term dependencies. One can carefully select parameters to solve the most rudimentary form of this problem, but in practice, RNNs certainly do not succeed in learning them. If the sequence is too long it can lead to problems with gradient dissipation during optimization. However, LSTM can solve this problem. I will discuss this method later.

Apart from vanilla RNN and LSTM, there is another method called GRU.

GRU (Gated Recurrent Unit) is a variant of RNN that, like LSTM, can effectively capture semantic correlations between long sequences and mitigate gradient loss or explosion phenomena. At the same time, its structure and computation is simpler than LSTM. Like the gates in LSTM, the update and reset gates are first calculated as z(t) and r(t), respectively, using a linear transformation of X(t) and h(t-1), activated by tanh. The reset gate is then applied to h(t-1), which controls how much of the information from the previous time step is available. The reset h(t-1) is then used to perform basic RNN calculations, i.e., it is spliced linearly with x(t) and then activated by tanh to obtain the

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Figure 4. equations for the forward pass of an LSTM unit with a forget gate

new h(t). This process implies that the update gate has the ability to retain the previous result, when the gate value tends to 1, the output is the new h(t), and when the gate value tends to 0, the output is the h(t) of the previous time step. (t-1)

However, in this report, I will focus on LSTM implementation.

3.1. LSTM Description

Long Short Term Memory Networks - commonly known as LSTMs - are a special type of RNN that can learn long term dependent information.lSTMs were developed by Hochreiter and Schmidhuber 1997, and has recently been refined and popularized by Alex Graves. The LSTM has been quite successful and widely used for many problems. LSTM avoids long-term dependency problems by deliberate design. Remember that long-term information is in practice the default behavior of LSTM, not a capability that costs a lot to acquire! All RNNs have a chained form of a repeating neural network module. In a standard RNN, this repeating module has only a very simple structure, such as a tanh layer. The LSTM also has the same structure, but the repeating module has a different structure. The LSTM is a special network structure with three ”gates”. LSTM relies on a number of ”gate” structures to allow information to selectively influence the state of the RNN at each moment in time. The so-called ”gate” structure is an operation that uses a sigmod neural network and a bitwise multiplication, which together is a ”gate” structure. The reason why this structure is called a gate is because the fully connected neural network layer using sigmod as activation function will output a value between 0 and 1, describing how much information can pass through this structure at the current input. 0), no information can be passed.

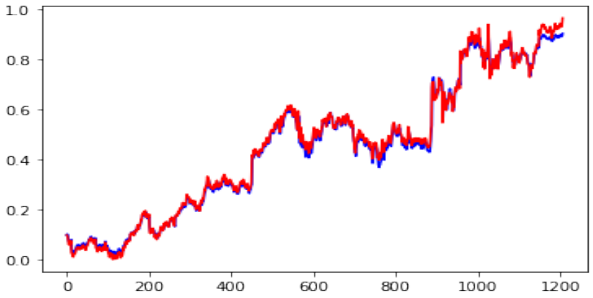

Figure 5. Predicted versus realistic for training data

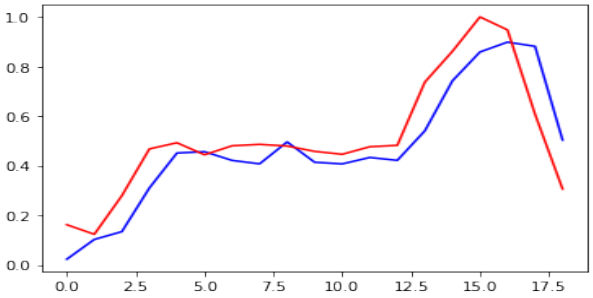

Figure 6. predicted price vs realistic for next 19 days

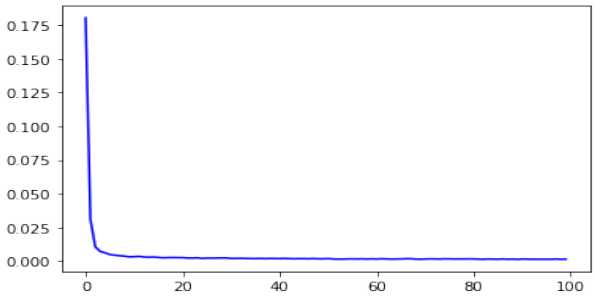

Figure 7. training loss for LSTM

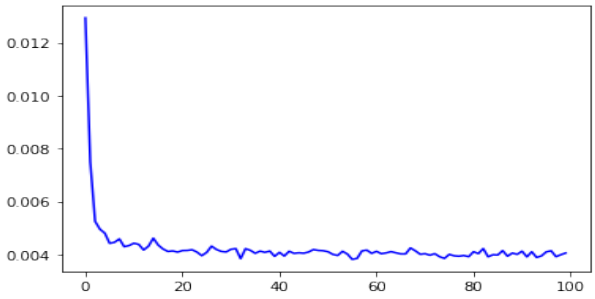

Figure 8. training loss for volume

3.2. LSTM implementation detail

When training the program, I use Keras as our library. The dataset for training is $Google_Stock_price_train.csv$, and I split the last 60 rows as our validation sets. The test dataset is called $Google_Stock_price_test.csv$. All the layer in my LSTM will have 60 units. I will use tanh as my activation function. Because we have 5 attributes, therefore the input shape is (trainX.shape[1],5). $optimizer =' adam', loss =' mean_squared_error'$.

4. Experimental Analysis

4.1. LSTM

The first experiment is to implement LSTM. The parameters for LSTM shows below:
Method: LSTM
Loss function: Mean squared error
Optimizer: Adam
Learning rate: 0.01
number of units in hidden layer: 60
batch size: 32
epoch: 100

After 100 epoches, the loss drop down from
Epoch 1/100: 38/38 [==============================]
- 0s 6ms/step - loss: 0.1813
to
Epoch 100/100 38/38 [==============================]
- 0s 6ms/step - loss: 0.0017

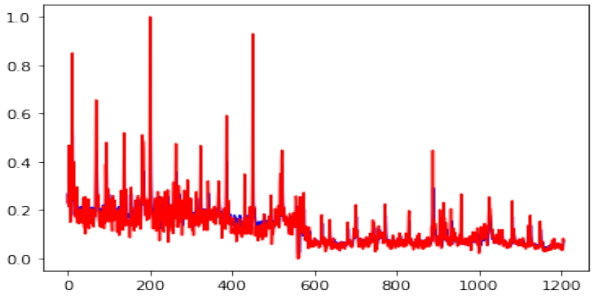

Figure 9. comparison between predicted volume value and realistic value when training

The figure 4 shows the predict value (the training set) against the ground truth. We can notice that after 100 epoches, the model was built successfully with a minor loss. Next is to predict the next 19 days of stock opening price based on the previous 1207 days. I load the model trained before and the figure 5 is the next 19 days predicted price against real price. The blue line is the predicted value and the red line is the real value. Figure 6 is the training loss for LSTM.

Figure 8 is the loss for training the Volume, figure 9 is the comparison between predicted volume value and realistic value when training, figure 10 is the predicted value and realistic value when testing.

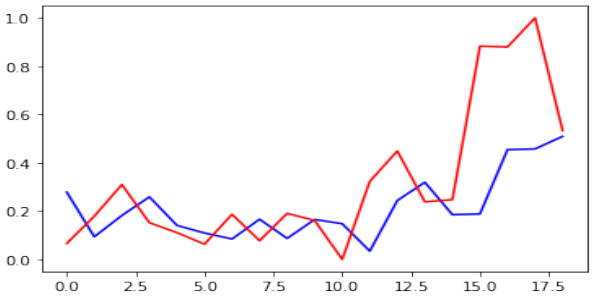

Figure 10. predicted value and realistic value when testing

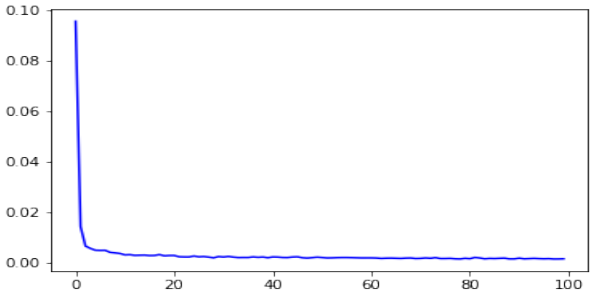

Figure 11. training loss for GRU (Opening price)

4.2. GRU

To compare with LSTM, I implement GRU method, all the variables are the same as LSTM. The purpose of this experiment is to observe which loss is lower and which model more accurate between GRU and LSTM. After 100 epoches, the loss shows in figure 6.
It is obvious that GRU has the fastest rate of loss decline as it started with a loss of 0.09 at epoch 1 wherea LSTM starts at 0.18. It proves that GRU learns faster than LSTM in this experiments. Because GRU is simpler to construct: one less gate than LSTM, which results in a few less matrix multiplications. GRU saves a lot of time when the training data is large.

5. Conclusion

Based on the previous experiments, I understand the advantages and disadvantages of different RNN structures (vanilla RNN, LSTM, GRU). To illustrate, the structure of LSTM can solve the gradient explosion, i.e., the introduction of three gates, especially the forgetting gate, which controls the inflow of the network state at the previous moment, because the tanh function is used in gating and the output is between [0,1], thus achieving the gating effect. Also, LSTM can solve the problem that vanilla RNN has, such as lost information when long distance. Furthermore, when conducting experiments on GRU, I understand that when there are more parameters, the risk of overfitting the trained network will be higher, while its training effort is costly. GRU can solve this problem by merging the original LSTM input gate and forget gate into a single gate, called the update gate. Instead of dividing the internal and external states in LSTM, it solves the problem of gradient disappearance and gradient explosion by adding a linear dependency directly between the current network state and the previous network state. When considering the computational power and time cost of the hardware, GRU is the better option