Master of Applied Mathematics Program
Professional Master's Project Report
Department of Applied Mathematics
Illinois Institute of Technology

Stock Price Prediction with LSTM model

Submitted by

Xi Sun

In partial fulfillment of the requirements of Master's candidates in Master of Applied Mathematics Program

December 2019

Abstract

This project explores the idea of how deep learning method could be used in stock price prediction. The theory of a special case of Recruit Neural Network - Long Short Term Memory will be introduced together with S&P 500 stock price data. Four training and test subset will be constructed for balancing and figuring out the batch size and epochs, and the model will be fitted separately for predicting the stock price in the test set, and the comparison in the errors for different size of subset will be analyzed. Finally, all the subsets will be compressed into one dataset and doing the prediction at once with error analyzed as well. Last but not least, some future directions will be explored which will talk about how this project could be extended for the future.

Background and Motivation

Stock is a mysterious term that can make people rich or poor. Stock price follows a random pattern that no one knows what will happen tomorrow or the day after tomorrow. Some people might get huge profit by buying stocks while most of people might not lucky enough to make profit, and even a few people might suffer a huge dive or an arbitrage in the stock market. Even so, there are still a large amount of people, who are interested in the stock market, put a huge amount of money and make few profit. The randomness in the change of stock price makes people frightened but people are still willing to take risk in order to make themselves rich.

Neural network provides an efficient way to predict such a random pattern for the future in order to make people prepared what will happen. Therefore, a neural network algorithm, or recruit neural network algorithm in this case, will be built and trained to see if the random pattern from stock price could be explained or not. If the randomness is successfully being captured, then people might also need to take risks but less frightened.

Introduction to LSTM model

Recruit Neural Network(RNN) is a deep learning method that uses previous information to predict present or future information. In order to predict the next information, RNN only needs the input to be the last information instead of a long term behavior for a particular data type(numerical, categorical, etc.). However, as artificial intelligence is developing very fast these days, in order to keep track of the usage of long behavior, traditional RNN is not that sufficient anymore. A special type of RNN - Long Short Term Memory is used to augment traditional RNN.

Long Short Term Memory(LSTM) model can be used in keeping track of information with long term dependencies. LSTM model can efficiently extract useful information from both long and short behavior and predict the future information needed. The key word here is called "memory", which the model can memorize the useful information and forget useless information from past and the provides output.

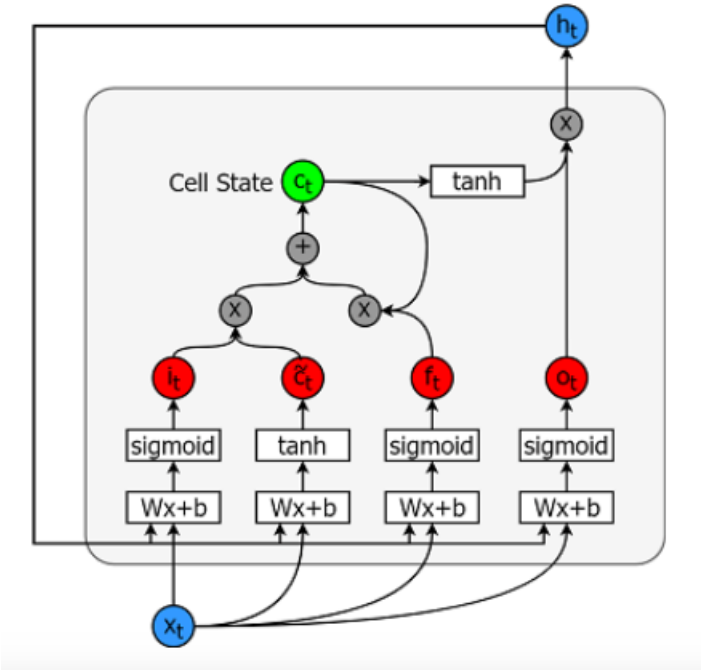

Figure 1: LSTM module

Figure 1 shows the structure of the module of LSTM. In this picture, x_t is one of the input and h_t is one of the output for the t^{th} iteration, therefore, there must be a $(t-1)^{\text{th}}$ output exists for the output in the previous iteration, together with the cell state c_t, there are three inputs in the LSTM algorithm, so the input dimension should be a 3D array. After we apply this algorithm, we know that we still have h_t and c_t as the output, so the output dimension should be a 2D array. Also, we have different kinds of layers. Sigmoid, denoted as σ is the sigmoid function defined as $\sigma(x) = \frac{e^x}{e^x+1}$, which returns the value between 0 and 1. Tanh, denoted as tanh is the hyperbolic tangent function defined as $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$, which returns the value between -1 and 1. Finally, c_t is the cell state, which remembers the feature from the past and use it for the future.

As we can see in figure 1 from bottom that the input need to go through 4 different places to do different things, which we call those(shown as red) "gates". The first gate operated by a sigmoid layer is called "input gate layer" i_t. The input x_t and the previous output h_{t-1} will be used initially to multiply a weight matrix W and add a bias term b in order to transform a 2D vector into a scalar. After that, the scalar will be applied to a sigmoid function, and it will return a value between 0 and 1, which will be i_t.

The second layer is used for balancing the output from input gate layer i_t. In order to not making the result being larger than the previous all the time, we need to determine if the new input can provide us positive information or negative information. So a tanh layer is used and the output is $\tilde{c}_t$.

The third layer is also operated by a sigmoid layer, but the meaning here is different. We call this "forget gate layer", denoted as f_t. This layer tells you how much of the information from past need to be forgot, since it will multiply together with the previous cell state c_{t-1}, the third input, which saves the information from past.

Next, we are going to update the cell state. The "+" symbol means addition and "×" symbol means multiplication. After we get the value from input gate layer i_t, balancing candidate $\tilde{c}_t$, forget gate layer f_t, and the previous cell state c_{t-1}. The new cell state c_t is defined as $c_t = i_t \times \tilde{c}_t + f_t \times c_{t-1}$. And this new cell state will not be changed anymore until the next module.

Finally, we need to consider the forth gate, which is called "output gate layer". This layer returns the output h_t for the current module, as well as being prepared for the next module. There is another sigmoid function which is same as above but the layer is denoted as o_t. After we multiply things out, we will get our output $h_t = o_t \times \tanh(c_t)$.

Data Preparation and Analysis

The dataset used in this project is the S&P 500 index from yahoo finance, from 2011-09-20 to 2019-08-30, which contains 2000 data points. The dataset contains 6 variables, but we only consider the adjusted stock price for our time series analysis. We can obtain this in R by using quantmod package. Also, we construct a structured time series dataset, so the dataset will be converted into a tibble(a special kind of data-frame in R), and this can be achieved with timetk package for converting a time series to a tibble. And here is a plot for the whole stock price by using ggplot2.(See Appendix - Data Collection)

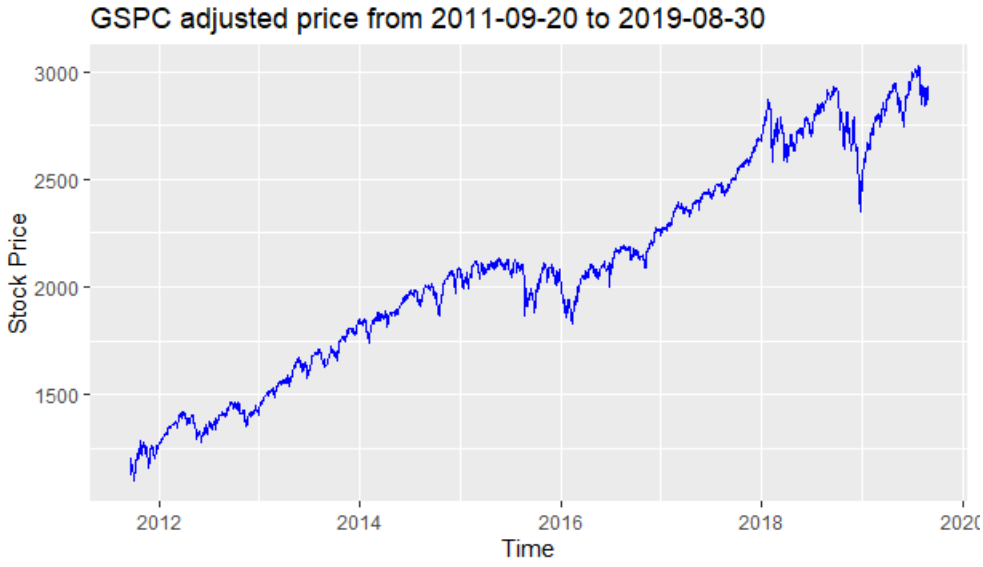

Figure 2: GSPC Adjusted Price

Now we need to prepare our data. First of all, in order to avoid large errors, the square root of the stock price will be considered. Since LSTM algorithm requires the data to be centered and scaled, therefore, this is obtained with the recipes package and bake function.(See Appendix - Normalization)

After we have the data ready, we are going to consider the training set and test set. For the whole dataset, 80/20 split will be applied. The input x in the training set contains data

from the first data to the 1600th data, the output y in the training set contains data from the second to the 1601th data. Also the input dimension needs to be a 3D array, the output dimension needs to be a 2D array.(See Appendix - Train/Test Split)

Model Construction

In order to investigate the error, the original data with 2000 data points will be split into 4 subsets and we apply the LSTM model on each subset. And since we are using 80/20 of training/test split, then each subset contains 500 data points with 400 data as training set and 100 data as test set. And for the full dataset, we have 1600 in the training set and 400 in the test set.

The hardest thing to figure out is how to balance the batch size and epochs. First of all, batch size is the total number of training samples being used in a single batch, because we cannot pass the whole training set into the module at once, so we need to divide the training set into many small pieces. Next, the number of epoch is how many times you want your entire dataset passes through the neural network. According to those theories, we know that the batch size needs to be a whole number and divisible by the number of rows in both training and test. In our case, 400 is divisible by 100, so we need to try all the following possibilities that is divisible by 100:

$$\text{batch size} \in \{1, 2, 4, 5, 10, 20, 25, 50, 100\}$$

with fixed epochs. After all the possibilities being tested, batch size = 10 provides the best prediction than all others, since the pattern or the trend of the stock price is successfully being caught. So we are going to use 10 for the batch size and finally distribute it to all the data. We notice that the best batch size is usually not changed if you are working in the same dataset.

The next thing is to figure out the number of epochs. This usually could be a hard work, but we initially set the epoch = 1000, and according to the loss we defined, which is mean squared error(MSE) in this case, goes down to approximately 0.0088 around epoch = 90 and approximately 0.0086 around epoch = 777. By considering the complexity for the algorithm and the running time for the computer, epoch = 90 will be applied and will be tested in the full dataset.(See Appendix - Model Construction and Prediction)

Result

After doing prediction all the subsets, we get the following plots for the predictions.

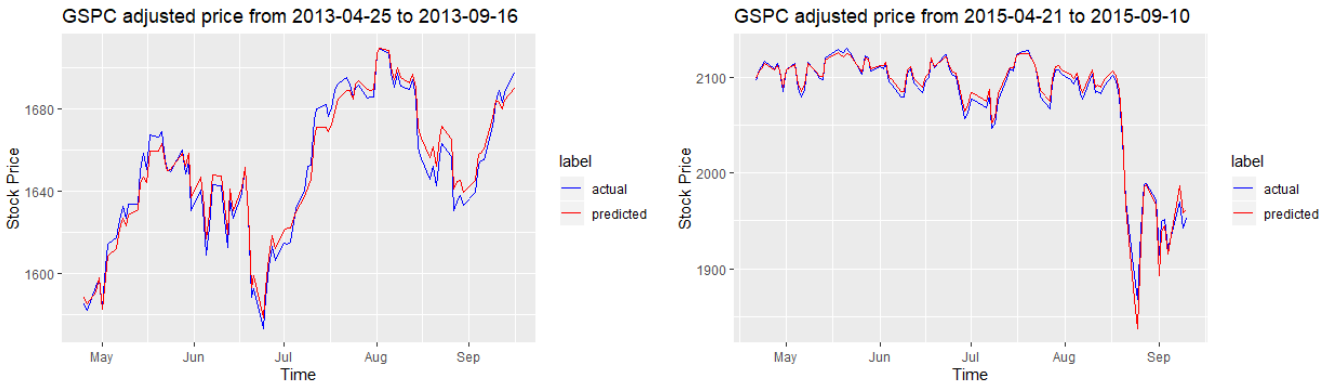

Figure 3: LSTM prediction for subsets. Left: subset 1. Right: subset 2

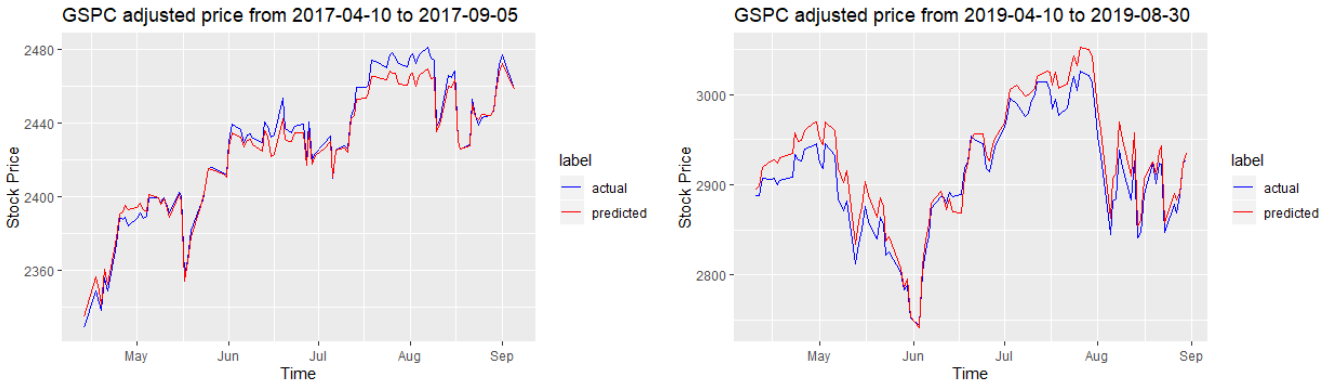

Figure 4: LSTM prediction for subsets. Left: subset 3. Right: subset 4

According to figure 3 and 4, we can see that the prediction are actually works very well, and the trend is almost correctly predicted, but we can also see that the error is not constant. For example, the error in figure 3 on the right is much smaller than on the left. This is because the errors are randomly distributed, so each time we will get a different prediction, but the trend is following the original all the time. Therefore, in order to investigate the error, we are going to change the size of our training and validation set(with same ratio of training/test split, and with the same batch size and epochs) to see how the error will behave. So we are going to pick the following number and apply the model on each one.

No.	Size of Training Set	Size of Validation Set	Time Gap
1	160	40	approx. 0.66 year
2	240	60	approx. 1 year
3	320	80	approx. 1.33 year
4	400	100	approx. 1.66 year
5	480	120	approx. 2 year
6	560	140	approx. 2.33 year

Once we determine our experiment, we will use the data starting from 2011-09-20 to the number of date after that date which are mentioned above. Then, we need to extract the

loss from our model. After finishing the extraction and prediction on all six experiments, we get the following result:(See Appendix - Error Analysis)

No.	Average Training Loss	Average Validation loss
1	0.09839216	0.04539203
2	0.06555496	0.0641022
3	0.03846245	0.118304
4	0.02532855	0.0628319
5	0.01778109	0.03892568
6	0.01427337	0.02322628

As we can see from the above chart, the training loss keeps decreasing when the sample size gets larger, and the validation loss gives the largest in the third experiment where we have 320 points in the training set and 80 points in the test set. This is because the prediction here is not easy to predict since we do not have sufficient amount of sample selected in this experiment. Figure 5 to 7 shows the plot of the loss for those 6 experiments.

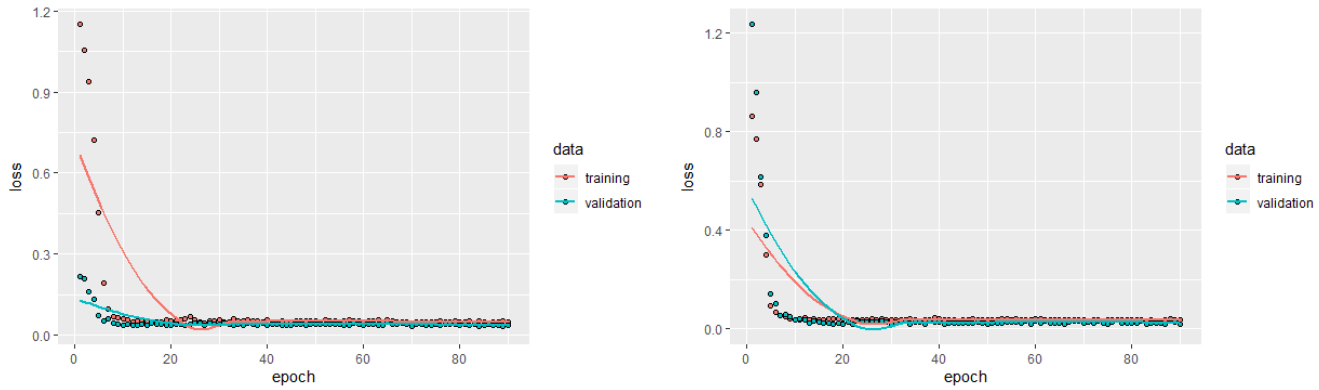

Figure 5: Loss on the training set. Left: No.1. Right: No.2

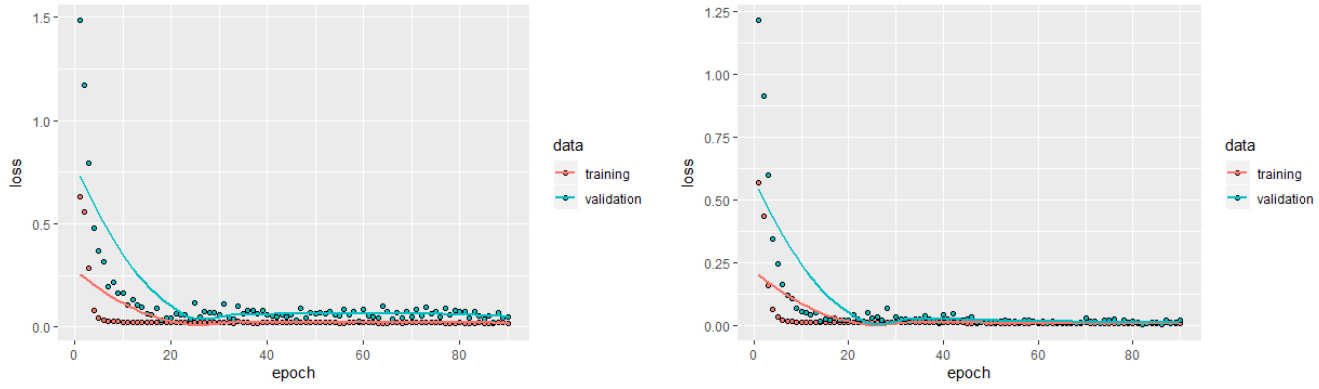

Figure 6: Loss on the training set. Left: No.3. Right: No.4

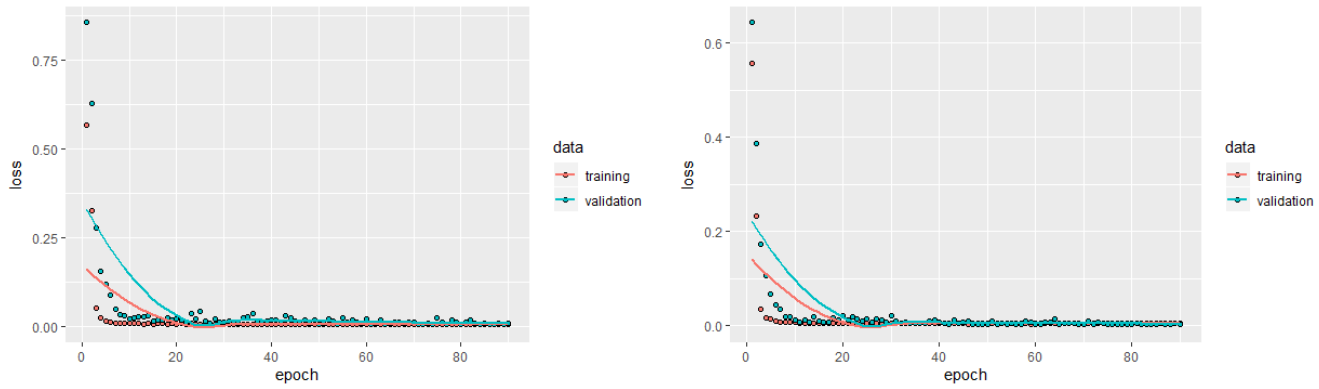

Figure 7: Loss on the training set. Left: No.5. Right: No.6

According to the 6 plots mentioned in figure 5 to figure 7, we can see that only the left graph in figure 5 has the training loss larger than the validation loss at the beginning, and all others have validation loss larger than the training loss at the beginning. This is because the sample size in the first experiment is not large enough and the stock price is not changing too much in the entire sample, therefore, it could result in a larger training error. However, according to the plot again, we can see that the lowest value for both training and validation error occurs around epochs=25. So for the future study, changing in epochs could be a appropriate idea to develop the behavior of errors.

Right now, we have finished everything for the subsets, so we can apply this model to the full dataset where we have 1600 training points and 400 test points and we get the following plot with batch size=10 and epochs=90. But before that, a 10% validation set will be set to examine the loss and accuracy for the prediction.

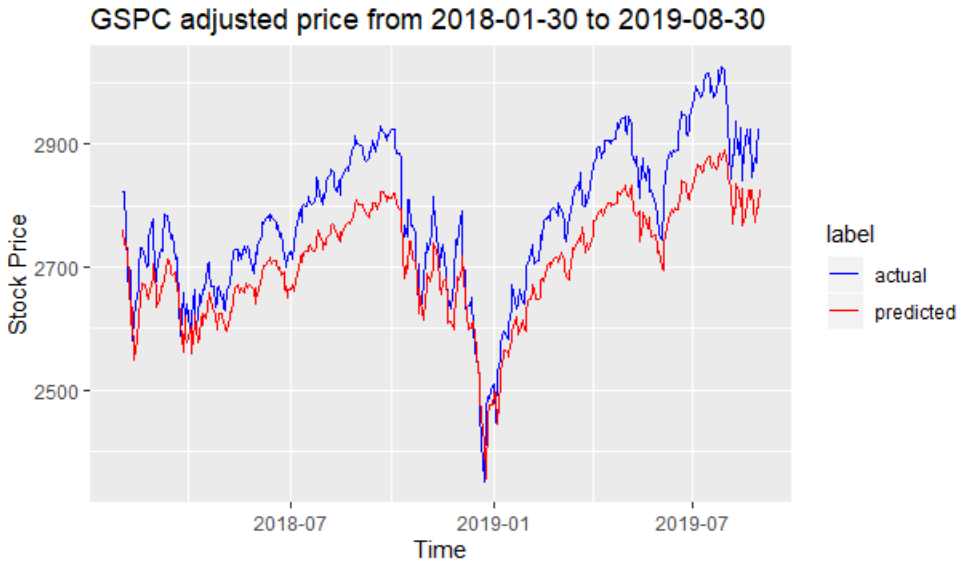

Figure 8: LSTM prediction on all test data with 160 points validation set

The model fitted in figure 8 is generated with 1440 points in the training set, 160 points

in the validation set, and 400 points in the test set. As we can see, the trend is predicted very well, but the gap between the actual curve and the predicted curve is very large. This is because the time series data is highly correlated with time, and if we use the last 160 points in the training set as the validation set, the time gap between the initial point in the test and the end point in the training set is large. Since the time series requires large dependencies on the time, so the larger error should be in our expectation. But the purpose here is still about the error in the validation set, and figure 9 shows the loss and accuracy in both training and validation set.

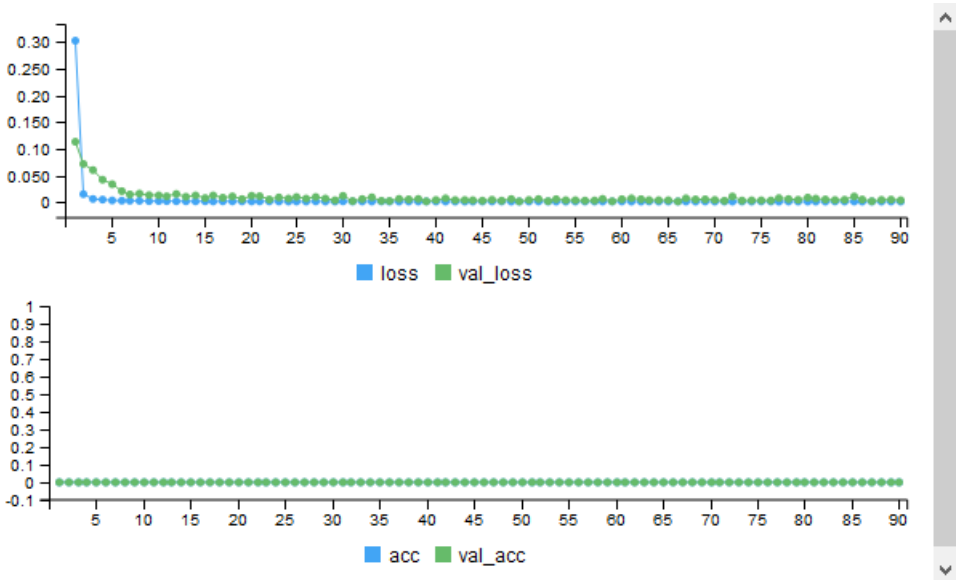

Figure 9: Loss and accuracy on training and validation set

According to figure 9, we can see that both (training) loss and validation loss are actually very small, therefore, we can conclude that this model is fitted very well for explaining the variation in the original dataset. However, according to the accuracy plot, we can see that the accuracy is straight line at 0. And this is because we can never successfully predict the exact value for the stock price. The purpose for this model is actually for predicting the trend, feature, and long-term behavior of the stock price to see if it follows the same as the actual value. By the way, the error we get from this model is 0.005186519 and 0.0108827 for average training and validation error respectively.

As we talked about in figure 8, there is a 10% validation set extracted between training and test set. So right now, we consider all 1600 points as training set and see how well we can do.

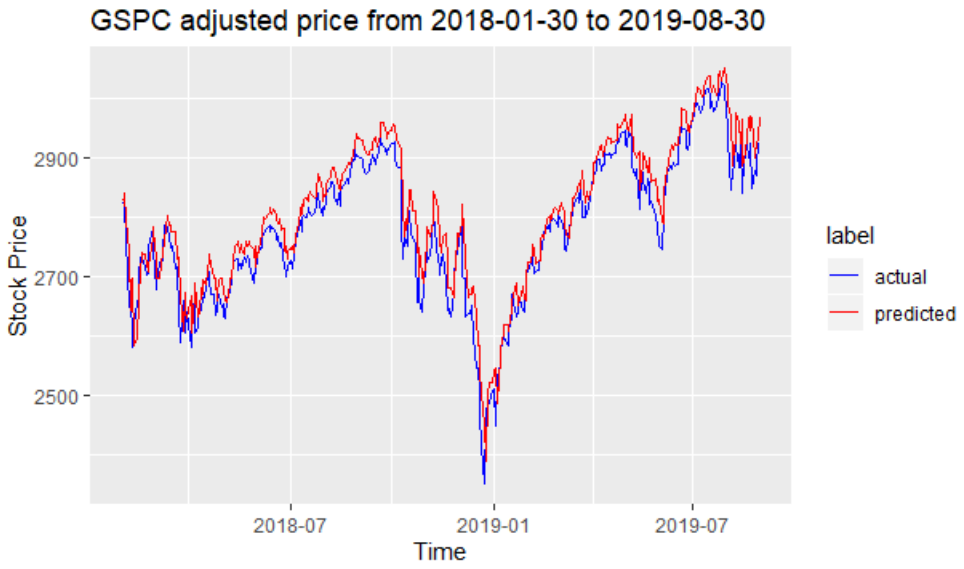

Figure 10: LSTM prediction on all test data

This is amazing, as we can see from figure 10 that the prediction is working very well. Not only the trend is successfully predicted, but also the predicted value is very close to the actual value. Since we are using the entire 1600 points in the training set, so the last point in the training set will be the first point in the test set. In time series, the data that is close to the prediction range should have more weights than the earlier-time data when doing the prediction. Thus, we can conclude that LSTM is doing an efficient work on predicting the trend of the stock price once the correct batch size and epochs are being selected.

Future Directions

There are few points that are worth considering for the extension of this project:

- At the beginning in the LSTM module, how can we estimate the weight matrix W and bias b?
- If there is an efficient way to find the best batch size and epoch more quickly?
- How does the easiness of the prediction is evaluated from the model, and how will the error changes.

As we know that this method has developed for many years but usually it cannot being used when predicting the live stock price for a long term. Remember that LSTM model uses past information to predict the next "one". So in order to get a prediction for some amount of explicit stock price or the trend of stock price, we need the information from future to predict future. Therefore, the next thing might worth considering is adding the predicted stock price from tomorrow in the training set and predict the day after tomorrow, and repeat this process, say 20 times, so we can predict the stock price for the next 20 days at today.

References

[1] Neural Networks to Predict the Market, https://towardsdatascience.com/neural-networks-to-predict-the-market-c4861b649371, Vivek Palaniappan, Sep 23, 2018. Last access: 11/09/2019.

[2] Time Series Prediction Using LSTM Deep Neural Networks, https://www.altumintelligence.com/articles/a/Time-Series-Prediction-Using-LSTM-Deep-Neural-Networks, Jakob Anugiers, Sep 1, 2018. Last access: 11/09/2019.

[3] Time Series Deep Learning: Forcasting Sunspots with Keras Stateful LSTM in R, https://www.r-bloggers.com/time-series-deep-learning-forecasting-sunspots-with-keras-stateful-lstm-in-r/, business-science.io, Apr 17, 2018. Last access: 11/09/2019.

[4] Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, Colah, Aug 27, 2015. Last access: 11/09/2019.

[5] Epoch vs Batch Size vs Iterations, https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9, Sagar Sharma, Sep 23, 2017. Last access: 11/09/2019.

[6] LSTM with Keras & Tensorflow, https://www.r-bloggers.com/lstm-with-keras-tensorflow/, R on Coding Club UC3M, Nov 26, 2018. Last access: 11/09/2019.

[7] Stock Market Predictions with LSTM in Python, https://www.datacamp.com/community/tutorials/lstm-python-stock-market, Thushan Ganegedara, May 3, 2018. Last access: 11/09/2019.

[8] Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras, https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/, Jason Brownlee, Jul 21, 2016. Last access: 11/09/2019.

[9] Time Series Analysis with LSTM using Python's Keras Library, https://stackabuse.com/time-series-analysis-with-lstm-using-pythons-keras-library/, Usman Malik, Nov 13, 2018. Last access: 11/09/2019.

Appendix - R Code

Libraries

```
library(quantmod)
library(timetk)
library(dplyr)
library(ggplot2)
library(recipes)
library(keras)
```

Data Collection

```
sp500 <- new.env()
getSymbols("^GSPC",env=sp500,src="yahoo",
           from=as.Date("2011-09-20"),to=as.Date("2019-08-31"))
gspc <- sp500$GSPC
s <- gspc$GSPC.Adjusted
colnames(s) <- "price"
s_tb <- tk_tbl(s)

s_tb %>%
  ggplot(aes(index,price)) +
  geom_line(color="blue",size=0.5) +
  ggtitle("GSPC adjusted price from 2011-09-20 to 2019-08-30") +
  xlab("Time") +
  ylab("Stock Price")
```

Normalization

```
rec <- recipe(price~.,s_tb) %>%
  step_sqrt(price) %>%
  step_center(price) %>%
  step_scale(price) %>%
  prep()
dat <- bake(rec,s_tb)
```

Train/Test Split

```
x_train_vec <- dat$price[1:1600]
x_train_arr <- array(data=x_train_vec,dim=c(length(x_train_vec),1,1))
y_train_vec <- dat$price[2:1601]
y_train_arr <- array(data=y_train_vec,dim=c(length(y_train_vec),1))

x_test_vec <- dat$price[1600:1999]
x_test_arr <- array(data=x_test_vec,dim=c(length(x_test_vec),1,1))
```

Model Construction and Prediction

```
set.seed(2000)
batch_size <- 10
epochs <- 90

lstm_model <- keras_model_sequential()
lstm_model %>% layer_lstm(units=50,input_shape=c(1,1),batch_size=batch_size,
                          return_sequences=TRUE,stateful=TRUE) %>%
  layer_lstm(units=50,return_sequences=FALSE,stateful=TRUE) %>%
  layer_dense(units=1) %>%
  compile(loss="mse",optimizer="adam")
```

```
model <- lstm_model %>% fit(x=x_train_arr, y=y_train_arr, batch_size=batch_size
                            epochs=epochs, validation_split=0.1, verbose=1)

pred <- lstm_model %>%
  predict(x_test_arr, batch_size=batch_size) %>%
  .[,1]
mean <- rec$steps[[2]]$means
sd <- rec$steps[[3]]$sds
pred_dat <- tibble(index=dat$index[1601:2000], pred_price=(pred*sd+mean)^2)

ggplot() +
  geom_line(data=s_tb[1601:2000,], aes(index, price, color="blue"), size=0.5) +
  geom_line(data=pred_dat, aes(index, pred_price, color="red"), size=0.5) +
  scale_colour_manual(name="label", values=c("blue"="blue","red"="red"),
                      labels=c("actual","predicted")) +
  ggtitle("GSPC adjusted price from 2018-01-30 to 2019-08-30") +
  xlab("Time") +
  ylab("Stock Price")
```

Error Analysis

```
plot(model)
mean(model$metrics$loss)
mean(model$metrics$val_loss)
```