**EECS6895 Milestone 2 Report**

**A-share Stock Auto Trader**

**Topic:** B9: Investment Strategy - AI Trader (CN/HK/TW/JP), transferred from A11: Reasoning (Understanding Causalities via Bayesian Network and/or Others)

**Team:** Yiwen Fang (yf2560) | Guoshiwen Han (gh2567)

**Contents**

1. Plan and achievement

2. Challenges

3. Reference

4. Next step

1. **Plan and achievement**

1). Data Collection & Procession

We collected 2000+ A-share main board stock in Shanghai/Shenzhen Stock Exchange where 1500+ were for Shanghai Stock Exchange and 500+ for Shenzhen Stock Exchange. The detailed financial data, e.g., Open-High-Low-Close (OHLC), were collected from Yahoo Finance.

The API to collect data is yahoo\_fin, which is a Python library. It is limited to thousands of calls per month. However, it is still free and has a large traffic allowance than common APIs such as Alpha Vantage. We collected the daily exchange data from the first listed day until today. Each stock attributes including date, open, high, low, close, adjclose, and volume. (**Code 1**)

# https://algotrading101.com/learn/yahoo-finance-api-guide/  
# yahoo\_fin  
  
from yahoo\_fin.stock\_info import get\_data, get\_quote\_table  
import csv  
  
# Load CSV data  
# SSE Composite Index (000001.SS), Shenzhen Component (399001.SZ)  
ticker\_list = ['000001.SS', '399001.SZ']  
  
sse\_csv = csv.reader(open('./data/SSE\_A-shares Main Board.csv', 'r', encoding='utf-8'))  
count = -1  
for row in sse\_csv:  
 count += 1  
 if count == 0:  
 continue  
 if count > 5:  
 break  
 ticker\_list.append(row[5])  
  
szse\_csv = csv.reader(open('./data/SZSE\_A-shares Main Board.csv', 'r', encoding='utf-8'))  
count = -1  
for row in szse\_csv:  
 count += 1  
 if count == 0:  
 continue  
 if count > 5:  
 break  
 ticker\_list.append(row[19])  
  
print(ticker\_list)  
print(len(ticker\_list))  
  
historical\_datas = {}  
for ticker in ticker\_list:  
 historical\_datas[ticker] = get\_data(ticker, start\_date = None, end\_date = None, index\_as\_date = False, interval = "1d")  
  
historical\_datas['600000.SS']  
  
for ticker in historical\_datas:  
 historical\_datas[ticker].to\_csv('./output/' + ticker + '.csv')  
  
quote\_table = get\_quote\_table('600000.SS', dict\_result=False)  
quote\_table

**Code 1.** Extracting stock information by the API yahoo\_fin.

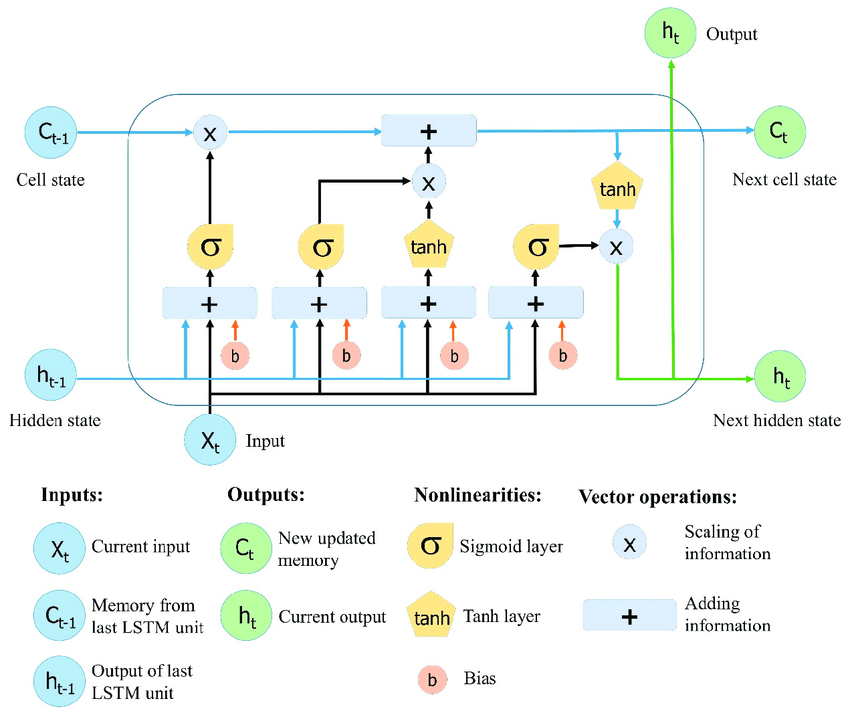
Data cleaning and mutations were continued. We eliminated any rows with empty/NA values, added a “label” column, which was the value of the next day, as the predicted value. (**Figure 1**)



**Figure 1.** Code for data cleaning and mutations.

2). LSTM Model for Stock Price Forecasting

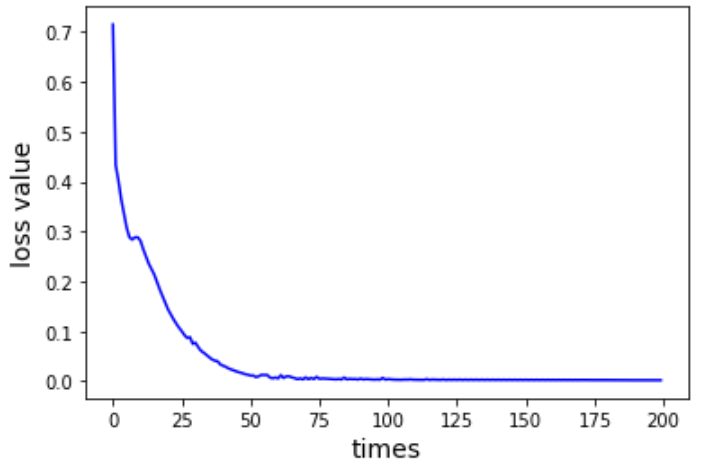
The graph (**Figure 2**) illustrates the schematic model of LSTM (Long Short-Term Memory). Stock forecasts are highly non-linear; therefore, the forecasting model should able to deal with non-linear problems. Since stocks have the characteristics of time series, cyclic neural networks are suitable to predict stocks. [2]



**Figure 2.** Schematic LSTM model for stock price forecasting. [1]

3). Training Loss

Loss function (mean-square error) is used to estimate the degree of inconsistency between the predicted value X of the network model and the true value Y. It is a non-negative real-valued function. This experiment uses a very common mean square error loss. By observing the loss function, we can find that the network model tends to be stable after the number of iterations is about 70. (**Code 2**)



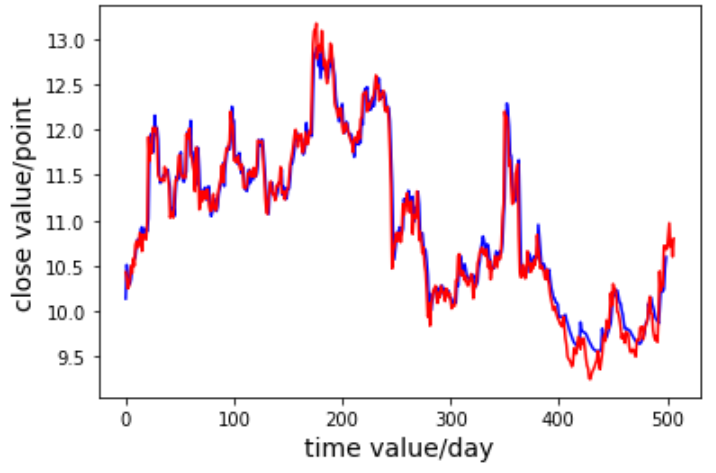
**Figure 3.** Training loss for the LSTM model by using mean-square error.

with tf.Session() as sess:  
 # initialization  
 sess.run(tf.global\_variables\_initializer())  
 theloss = []  
 # Number of iterations  
 for i in range(200):  
 for step in range(len(batch\_index) - 1):  
 # sess.run(b, feed\_dict = replace\_dict)  
 state\_, loss\_ = sess.run([train\_op, loss], feed\_dict={  
 X: train\_x[batch\_index[step]:batch\_index[step + 1]],  
 Y: train\_y[batch\_index[step]:batch\_index[step + 1]],  
 keep\_prob: 0.5})  
 print("Number of iterations:", i, " loss:", loss\_)  
 theloss.append(loss\_)  
  
 print("model\_save: ", saver.save(sess, 'model\_save2//model.ckpt'))  
 print("The train has finished")

**Code 2.** Code for the loss function estimations.

4). Model Prediction

Here we set the number of iterations = 200, forget\_bias= 1.0, the number of LSTM units = 10, and we got a LSTM model with 0.01181 relative error. (**Figure 4**) Relative error equals to (predicted value - real value) / real value \* 100%. We will adjust these parameters according to the dataset, which will help us get better results.



**Figure 4.** Model prediction results where blue indicates predicted values and red indicates real values.

1. **Challenges**

1). About model training.

There are a lot of stocks, and training models for every stock takes so long.

2). About model prediction.

By now, our model can only predict the closing price of the next day, but cannot predict the price or trend of the next few days. And every time we forecast the price of a new day, the data and model must be updated.

3). About LSTM.

LSTM can only predict the price of tomorrow based on the historical data of this one stock, but does not take into account the horizontal correlation between related stocks. We will think about and verify this in the future.

4). About training parameters.

Different stocks perform differently, so we cannot use the same set of tuning parameters, and we need to reset them for each model.

5). About training attributes.

We currently only use the opening price, closing price, highest price, and lowest price to predict the next day's price. In fact, the model can add more macroeconomic indicators and company financial reporting indicators to diversify the attributes.

1. **References**

[1] Le, X.H., Ho, H.V., Lee, G. and Jung, S., 2019. Application of long short-term memory (LSTM) neural network for flood forecasting. *Water*, *11*(7), p.1387.

[2] <https://towardsdatascience.com/predicting-stock-price-with-lstm-13af86a74944>

1. **Next step**

1). UI design.

We design three pages of our UI. Firstly, we let the user input the stock code he or she wants to invest in. Then on the second page, the user inputs the total amount of investment, expected profit, and the maximum loss he or she can accept. Then we will train the model and make a prediction, and then give out our trading suggestions.

2). Update data and model automatically.

Since every time we forecast the price of a new day, the data and model must be updated; we plan to write a trigger on the backend to update the data and retrain the model, which ensures us to get the latest and the most precise result.