# Algorithmic Trading

JiaBin Xue, Shan Kuang

2024-03-15

# *Part 1 - Background introduction*

The goal of this program is to perform the task of algorithmic trading, which aims to maximise the profit by automated stock trading. To achieve that, a model for predicting time series of financial data is needed, more specifically, the job of such models is to make predictions (forecasting) of a stock's closing price in the future (e.g. next day, next week, next month etc). After that, investors can use these predictions to decide that when to buy, sell and hold, based on designated trading rules.

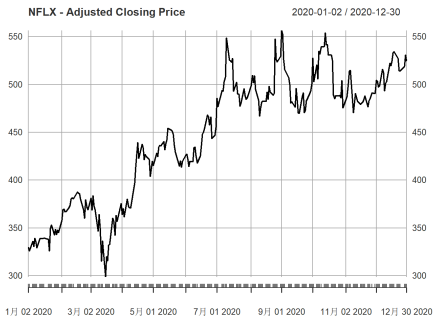
The approach we used for creating such predictive models is based on ANNs (artifical neural networks) because compared to traditional machine learning regression algorithms, ANNs can deal with linear and non-linear data, where the latter is the type of data we will use. We also introduced Genetic Algorithms for optimising parameters, window size and indicator thresholds, the former is used to slide the window of indicators and the latter is used to decide when to take trading decisions. For the Particle Swarm Optimisation which introduced during the course, considering the data we selected are high in noise which may leed to overfitting, after evaluation, we think ANN is more proporate.

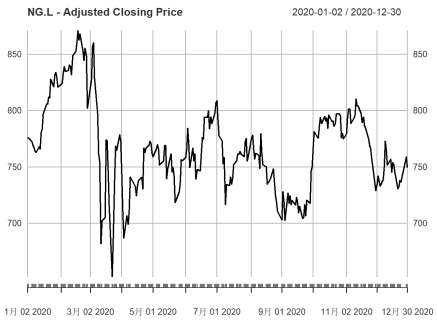
```
library(quantmod)
library(TTR)
library(GA)
library(genalg)
library(neuralnet)
```

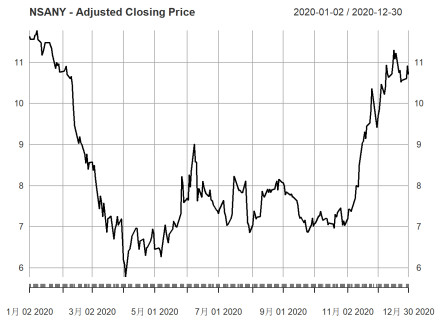
# *Part 2 - Data selection*

We decided to use three data sets chose from different sectors: Entertainment - NFLX, Cars & Transport - NSANY, Energy - NG.L, and we loaded the data during 2020-01-01 to 2020-12-31. More importantly, the trend of each asset has significant difference, and that is very useful for us to see if our proposed approach (Neural Network) is robust against such variance. #add

```
someStocks <- c("NFLX", "NSANY", "NG.L")
getSymbols(someStocks, src="yahoo", from="2020-01-01", to="2020-12-31", freq="daily")
```




From the summary below provided, we can observe some differences in the price movements of the three stocks (NFLX, NG.L, NSANY) in 2020:

In 2020, Netflix's stock showed a significant upward trend from 298.8 to 556.5 and the median and mean values are similar, indicating an even data distribution with no skewness. National Grid's stock price fluctuated between 652.6 and 871.1, indicating less volatility while the median is slightly lower than the mean, indicating a slight leftward skew (more days with lower prices). Nissan's stock price increased significantly, with a potential rise in H2 2020.

Overall, NFLX is the most volatile stock, while NSANY shows the highest growth rate. NFLX and NSANY have a relatively even price distribution, while NG.L may have a slight left skew.

```
summary(NFLX$NFLX.Adjusted)
```

```
##       Index              NFLX.Adjusted
##  Min.   :2020-01-02   Min.   :298.8
##  1st Qu.:2020-04-01   1st Qu.:380.9
##  Median :2020-07-01   Median :469.0
##  Mean   :2020-07-01   Mean   :446.5
##  3rd Qu.:2020-09-30   3rd Qu.:495.7
##  Max.   :2020-12-30   Max.   :556.5
```

```
summary(NG.L$NG.L.Adjusted)
```

```
##       Index              NG.L.Adjusted
##  Min.   :2020-01-02   Min.   :652.6
##  1st Qu.:2020-03-31   1st Qu.:739.9
##  Median :2020-07-02   Median :764.9
##  Mean   :2020-07-01   Mean   :767.3
##  3rd Qu.:2020-09-30   3rd Qu.:790.7
##  Max.   :2020-12-30   Max.   :871.1
```

```
summary(NSANY$NSANY.Adjusted)
```

```
##       Index              NSANY.Adjusted
##  Min.   :2020-01-02   Min.   : 5.780
##  1st Qu.:2020-04-01   1st Qu.: 7.122
##  Median :2020-07-01   Median : 7.720
##  Mean   :2020-07-01   Mean   : 8.233
##  3rd Qu.:2020-09-30   3rd Qu.: 9.037
##  Max.   :2020-12-30   Max.   :11.780
```

# *Part 3 - Implementation*

The implementation includes three sections:
1. A function that wraps the construction of neural networks, this function uses a window size as one of its parameters, which will further use to create RSI and WMA indicators, and they will be used as the input of created neural network. The indicator MACD is introduced, but it used fixed window size (12, 26, 9) throughout the implementation. These indicators then are scaled down to 0 -> 1 for the neural network to compute more efficiently. The architecture of the neural network we created is 3 layers fully connected network, the neurons for each layer is (4, 4, 2), and the maximum steps for the training is 1e7, by this setting, we can complete the training in an acceptable time. Training predictions, training error, test predictions, and test error are computed directly in this function.

```
# Metric, used to compute the error of predictions
rmse <- function(true, predicted){
  return (sqrt(mean((true - predicted)^2)))
}

test_Portion <- 0.2 # 20% of data for test

build_NN <- function(ws, command){
  rsi_ws <- floor(nrow(stock) * ws[1])
  wma_ws <- floor(nrow(stock) * ws[2])

  rsi <- RSI(stock[, ncol(stock)], n=rsi_ws)
  wma <- WMA(stock[, ncol(stock)], n=wma_ws)
  macd <- MACD(stock[, ncol(stock)], nFast=12, nSlow=26, nSig=9)

  mydata <- data.frame(rsi, macd, wma, stock[, ncol(stock)])
  colnames(mydata) <- c('RSI', 'MACD', 'MACD_Signal', 'WMA', "Close")
  mydata <- na.omit(mydata)
  plotIndex <- index(as.xts(mydata))

  scaled_RSI <- mydata$RSI/sum(mydata$RSI)
  scaled_MACD <- mydata$MACD/sum(mydata$MACD)
  scaled_MACD_S <- mydata$MACD_Signal/sum(mydata$MACD_Signal)
  scaled_WMA <- mydata$WMA/sum(mydata$WMA)
  scaled_Close <- mydata$Close/sum(mydata$Close)
  scaledData <- data.frame(scaled_RSI, scaled_MACD, scaled_MACD_S, scaled_WMA, scaled_Close)
  colnames(scaledData) <- c('RSI', 'MACD', 'MACD_Signal', 'WMA', "Close")

  testSize <- floor(nrow(scaledData)*test_Portion)
  train_Data <- head(scaledData, nrow(scaledData)-testSize)
  test_Data <- last(scaledData, testSize)
  X_test <- test_Data[1:ncol(test_Data)-1]
  y_test <- test_Data[ncol(test_Data)]

  set.seed(42)
  happy_nn <- neuralnet(Close ~ RSI+MACD+MACD_Signal+WMA, train_Data, hidden=c(4,4,2),
                        stepmax=1e7, threshold=0.00001,linear.output = TRUE)

  train_Pred <- unlist(happy_nn$net.result) * sum(mydata$Close)
  train_error <- rmse(train_Data$Close*sum(mydata$Close), train_Pred)
  test_Pred <- predict(happy_nn, X_test) * sum(mydata$Close)
  test_error <- rmse(y_test$Close*sum(mydata$Close), test_Pred)

  # Command 1: for returning the test error for GA
  if (command == 1){
    return (test_error)
  }
  # Command 2: for plotting the predictions
  else if (command ==2){
    plot(plotIndex, scaled_Close*sum(mydata$Close), xlab="Period", ylab="Close price",
         type="l", col="black", lwd=2)
    trainPlotIndex <- head(plotIndex, nrow(scaledData)-testSize)
    lines(trainPlotIndex, train_Pred, col="blue", lwd=2)
    testPlotIndex <- last(plotIndex, testSize)
    lines(testPlotIndex, test_Pred, col="green", lwd=2)
```

```
    return (c(round(train_error,3), round(test_error,3)))
  }
  # Command 3: for obtaining the data used to compute profits
  else {
    train_Finance <- data.frame(train_Data$Close*sum(mydata$Close),
                                head(stock[, 1], nrow(train_Data)),
                                head(scaled_RSI, nrow(train_Data))*sum(mydata$RSI),
                                head(scaled_MACD, nrow(train_Data))*sum(mydata$MACD),
                                head(scaled_MACD_S, nrow(train_Data))*sum(mydata$MACD_S),
                                train_Pred)
    test_Finance <- data.frame(y_test*sum(mydata$Close), last(stock[, 1],testSize),
                               last(scaled_RSI, testSize)*sum(mydata$RSI),
                               last(scaled_MACD, testSize)*sum(mydata$MACD),
                               last(scaled_MACD_S, testSize)*sum(mydata$MACD_S),
                               test_Pred)
    colnames(train_Finance) <- c("Closing_Price", "Opening_Price", "RSI", "MACD",
                                 "MACD_Signal", "Prediction")
    rownames(train_Finance) <- head(plotIndex, nrow(train_Data))
    colnames(test_Finance) <- c("Closing_Price", "Opening_Price", "RSI", "MACD",
                                "MACD_Signal", "Prediction")
    rownames(test_Finance) <- last(plotIndex, nrow(test_Data))
    return (list(train_Finance, test_Finance))
  }
}

model_error <- function(window_Sizes){

  error <- build_NN(window_Sizes, 1)

  return (error)
}
```

2. Use RBGA (minimisation) to find optimal window sizes (used to RSI and WMA) that minimises the prediction error, the fitness value used in here is RMSE. Since there are only two values the genetic algorithm needs to find, so the population size and iterations were set to 10 and 30 respectively. We then store the solution found by RBGA, which will be used to create the neural network in the evaluation part.

```
# Use GA to find the optimal window size that performs better RMSE for each stock
lbound <- c(rep(0.10, 2))
ubound <- c(rep(0.30, 2)) #At maximum, 30% of the data set can be used for sliding window

stock <- NFLX
set.seed(42)
nflx_GA <- rbga(stringMin = lbound, stringMax = ubound, popSize = 10,
                iters = 30, evalFunc = model_error)
nflx_ws <- nflx_GA$population[which.min(nflx_GA$evaluations),]

stock <- NG.L
set.seed(42)
ngl_GA <- rbga(stringMin = lbound, stringMax = ubound, popSize = 10,
               iters = 30, evalFunc = model_error)
ngl_ws <- ngl_GA$population[which.min(ngl_GA$evaluations),]

stock <- NSANY
set.seed(42)
nsany_GA <- rbga(stringMin = lbound, stringMax = ubound, popSize = 10,
                 iters = 30, evalFunc = model_error)
nsany_ws <- nsany_GA$population[which.min(nsany_GA$evaluations),]
```

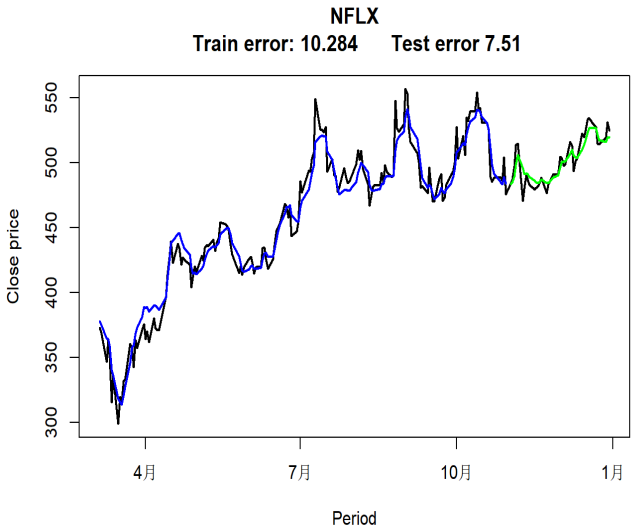
3. A function that contains trading strategies that will be used for part 4 and part 5, and it will be used to calculate the profit depending on which strategy is invoked. The profit calculated by this function is used as the fitness value of GAs in part 4 to find the threshold of indicators that maximise the profit of trading.

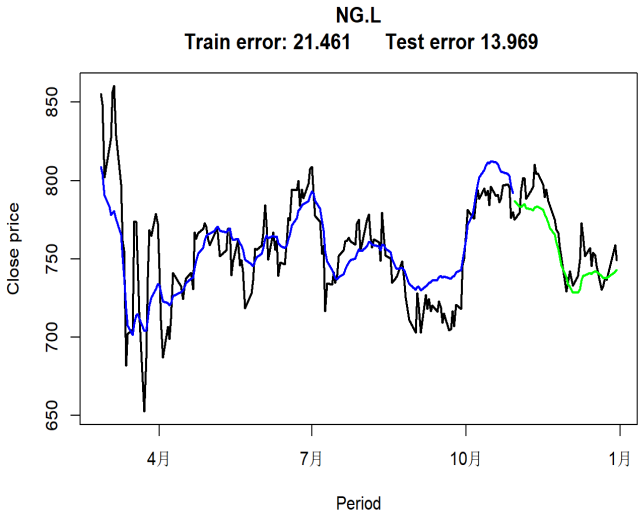
```
more_profit <- function(thresholds, strategy){
  buy_Boundary <- happyFinance$Closing_Price * thresholds[1]
  sell_Boundary <- happyFinance$Closing_Price * thresholds[2]
  decision <- c()

  # Trading rule 1
  if (strategy == 1){
    for (i in 1:nrow(happyFinance)){
      if (happyFinance$Prediction[i] >= buy_Boundary[i]){
        decision[i] <- 1
      }
      else if (happyFinance$Prediction[i] <= sell_Boundary[i]){
        decision[i] <- -1
      }
      else{
        decision[i] <- 0
      }
    }
  }
  # Trading rule 2
  if (strategy == 2){
    for (i in 1:nrow(happyFinance)){
      if (happyFinance$Prediction[i] >= buy_Boundary[i] &&
         (happyFinance$RSI[i] > thresholds[3] ||
         (happyFinance$MACD[i] > happyFinance$MACD_Signal[i]))){
        decision[i] <- 1
      }
      else if (happyFinance$Prediction[i] <= sell_Boundary[i] &&
              (happyFinance$RSI[i] < thresholds[4] ||
              (happyFinance$MACD[i] < happyFinance$MACD_Signal[i]))){
        decision[i] <- -1
      }
      else{
        decision[i] <- 0
      }
    }
  }

  happyFinance <- cbind(happyFinance, decision)
  hold_Stock <- FALSE
  buy <- 0
  sell <- 0
  profit <- 0

  # Calculate profit
  for (i in 1:(nrow(happyFinance)-1)){
    if (decision[i] == 1 && hold_Stock == FALSE){
      buy <- happyFinance$Opening_Price[i+1]
      hold_Stock <- TRUE
    }
    else if (decision[i] == -1 && hold_Stock == TRUE){
      sell <- happyFinance$Opening_Price[i+1]
      hold_Stock <- FALSE
      profit <- profit + sell - buy
    }
```

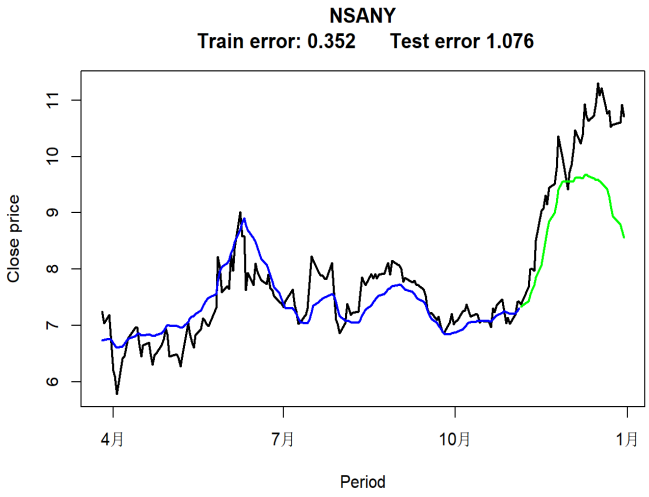
```
  }

  return (profit)
}

strategy_1 <- function(th){
  return (more_profit(th, 1))
}
```

# *Part 4 - Evaluation*

This part composed of evaluation on training data and evaluation on test data, with respect to fitness value (RMSE in this case) and profit returned. Below illustrates the RMSE of each data set on training data and test data, blue line represents training data and green line represents test data, note that the RMSE value is rescaled already. We can see that the neural network we trained can handle the first stock 'NFLX' fairly well, then the second one its performance is dropping down, where the third one has more tricky trend and the model basically failed on predicting future data.



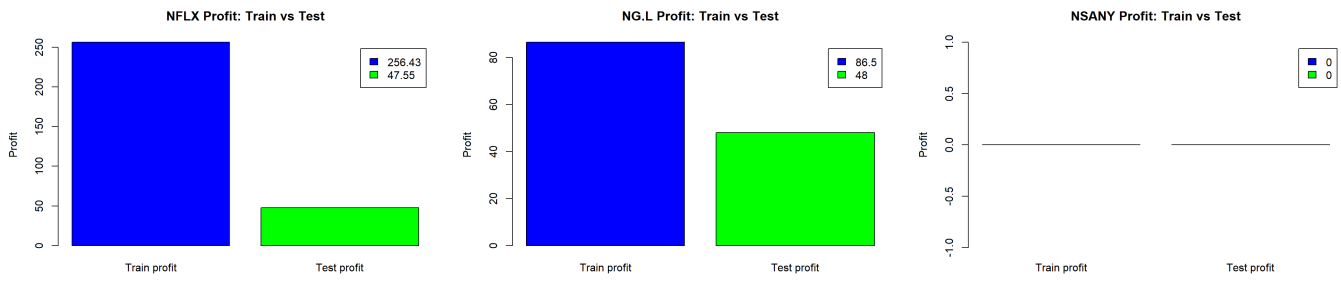




Now we have got a list of predictions, we can use it to evaluate the profit of each stock based on some trading rules. The first trading strategy we introduced is based on a basic price prediction-based trading. In our approach, the trading decision (buy, sell, hold) is triggered by variables **buy_boundary** and **sell_boundary**, where they are computed by increasing/decreasing the current day's closing price by a certain of amount(e.g. 10%). Hence, we used **Genetic algorithm** to find such amounts that maximise the profit.

```
indicators_lbound <- c(1.00, 0.01)
indicators_ubound <- c(1.25, 0.99)

nflx_train <- as.data.frame(nflx_data[1])
nflx_test <- as.data.frame(nflx_data[2])
happyFinance <- nflx_train
nflx_profit_GA <- ga(type="real-valued", fitness = strategy_1, lower = indicators_lbound,
                upper = indicators_ubound, selection='gareal_tourSelection',
                crossover='gareal_spCrossover', mutation='gareal_raMutation',
                popSize = 50, maxiter = 300, seed = 42)
nflx_thresh <- nflx_profit_GA@population[which.max(nflx_profit_GA@fitnessValue),]
happyFinance <- nflx_test
nflx_profit <- more_profit(nflx_thresh, 1)

ngl_train <- as.data.frame(ngl_data[1])
ngl_test <- as.data.frame(ngl_data[2])
happyFinance <- ngl_train
ngl_profit_GA <- ga(type="real-valued", fitness = strategy_1, lower = indicators_lbound,
                upper = indicators_ubound, selection='gareal_tourSelection',
                crossover='gareal_spCrossover',mutation='gareal_raMutation',
                popSize = 50, maxiter = 300, seed = 42)
ngl_thresh <- ngl_profit_GA@population[which.max(ngl_profit_GA@fitnessValue),]
happyFinance <- ngl_test
ngl_profit <- more_profit(ngl_thresh, 1)

nsany_train <- as.data.frame(nsany_data[1])
nsany_test <- as.data.frame(nsany_data[2])
happyFinance <- nsany_train
nsany_profit_GA <- ga(type="real-valued", fitness = strategy_1, lower = indicators_lbound,
                upper = indicators_ubound, selection='gareal_tourSelection',
                crossover='gareal_spCrossover', mutation='gareal_raMutation',
                popSize = 50, maxiter = 300, seed = 42)
nsany_thresh <- nsany_profit_GA@population[which.max(nsany_profit_GA@fitnessValue),]
happyFinance <- nsany_test
nsany_profit <- more_profit(nsany_thresh, 1)
```

For training period of NFLX, the GA above found that when **buy_boundary = Closing price x 1.000589** and **sell_boundary = Closing price x 0.978005**, this will return the highest profit **256.43**. We then apply these two thresholds on test period, and obtained **47.55**. Same principle to other two stocks. As mentioned earlier, the third stock NSANY is difficult to predict, the current trading strategy cannot make appropriate trading decisions with it (i.e. never buy/sell, or only buy/sell), hence it has got **0** profit for both periods.



# *Part 5 - Comparisons*

Introduce: Generally RSI < 30 means oversold, RSI > 70 means overbought. MACD < MACD_Signal means bearish, MACD > MACD_Signal means bullish. The comparison is accomplished by employing another trading strategy, and comparing the profit returned by each stock for both training period and test period. The new trading strategy is based on the previous one, but has introduced two more indicators: RSI and MACD (MACD & MACD Signal). The algorithm now should buy stocks when it observes that the **RSI > 50** OR **MACD rises above the MACD Signal**, where when **RSI<30** OR **MACD falls below the MACD Signal** it should sell.

```
nflx_thresholds <- c(nflx_thresh, 50, 30)
happyFinance <- nflx_train
nflx_trainProfit_tr2 <- more_profit(nflx_thresholds, 2)
happyFinance <- nflx_test
nflx_testProfit_tr2 <- more_profit(nflx_thresholds, 2)

ngl_thresholds <- c(ngl_thresh, 50, 30)
happyFinance <- ngl_train
ngl_trainProfit_tr2 <- more_profit(ngl_thresholds, 2)
happyFinance <- ngl_test
ngl_testProfit_tr2 <- more_profit(ngl_thresholds, 2)

nsany_thresholds <- c(nsany_thresh, 50, 30)
happyFinance <- nsany_train
nsany_trainProfit_tr2 <- more_profit(nsany_thresholds, 2)
happyFinance <- nsany_test
nsany_testProfit_tr2 <- more_profit(nsany_thresholds, 2)
```

```
tr1_result <- c(round(c(max(nflx_profit_GA@fitness), nflx_profit),2),
                round(c(max(ngl_profit_GA@fitness), ngl_profit),2),
                round(c(max(nsany_profit_GA@fitness), nsany_profit),2))
tr2_result <- c(round(c(nflx_trainProfit_tr2, nflx_testProfit_tr2),2),
                round(c(ngl_trainProfit_tr2, ngl_testProfit_tr2),2),
                round(c(nsany_trainProfit_tr2, nsany_testProfit_tr2),2))
comparison <- t(data.frame(tr1_result, tr2_result))
colnames(comparison) <- c("NFLX_Train", "NFLX_Test", "NG.L_Train",
                          "NG.L_Test", "NSANY_Train", "NSANY_Test")
rownames(comparison) <- c("Trading Strategy 1", "Trading Strategy 2")
comparison
```

```
##                    NFLX_Train NFLX_Test NG.L_Train NG.L_Test NSANY_Train
## Trading Strategy 1     256.43     47.55       86.5        48           0
## Trading Strategy 2     277.17      0.00        0.0        23           0
##                    NSANY_Test
## Trading Strategy 1          0
## Trading Strategy 2          0
```

# *Part 6 - Challenges and Limitations with Potential Improvements*

In our assignment, the challenges and limitations we faced are mainly focused on improving model performance, optimizing trading rules to achieve more profits, and managing model training time trade-offs.

## *1. Challenges and limitations:*

### *Model performance and trading rule optimization:*

- Our model performed poorly in predicting the third stock, which directly affected the profitability of the trading strategy, resulting in zero actual returns on that stock.
- The model's prediction of a specific stock's movement is inaccurate and the prediction curve does not match the actual movement satisfactory. This may be due to the model failing to capture all factors affecting price fluctuations.

### *The trade-off between neural network structure and training time:*

- We found that changing the number of layers and neurons of the network, we also think that introducing back-propagation and adjusting the learning rate may help improve the accuracy of the model, but those methods will significantly increase the training time.
- As students, we have constraints on computing power, which means that increasing model complexity results in significant increases in training time.

## *2. Potential Improvements:*

### *Iterative optimization of models and trading rules:*

- We could Develop more trading rules based on the real market and make decisions based on the prediction results of the model. Currently, our buying and selling rules are simple but safe for novice stock traders.
- Experiments with different feature combinations to improve predictions of complex trends. In addition, exploring other types of machine learning models, such as ensemble learning or time series analysis methods maybe better suited for specific data sets. However this is also time consuming.

### *Neural network tuning and computing resource management:*

- When training the model, we could use regularization techniques such as dropout to prevent over-fitting, and determine the optimal number of layers and neurons through grid search or random search. If available, we can utilize better computing resources or cloud services for training more complex models.

### *Prediction accuracy improvement:*

- For a more accurate prediction model, it's possible add more market data and some macroeconomic factors that may affect stock prices into the model, such as interest rate changes, policy changes, daily news, etc.
- Many trading strategies use advanced time series analysis methods like ARIMA or GARCH models, are specifically designed for handling volatility and non-linearity of time series data, which could be used to improve our prediction.