Application of Machine Learning In Algorithm Trading:

An Example of Recurrent Reinforcement Learning

Kartik Shetty

Yupu Song

Contents

1 Abstract

Emerging from 1980s, algorithm trading has been playing an increasingly important role in financial market. As scientists crowd into financial industry, many sophisticated technologies have been used in algorithm trading, and machine learning is one of the most remarkable approaches. Besides, in recent years, we have much more information and data of the market, which also become more complicated than before. Compared to the traditional mathematical models, machine learning methods are more likely to find promising approaches under such intricate conditions, and many financial companies have been trying to develop machine learning trading agent. Inspired by this industry trend, we intended to explore the application of machine learning in algorithm trading. Our project started with a brief introduction of machine learning and algorithm trading, then we analyzed the features of market data. Next, we explored some promising machine learning algorithms, including Hidden Markov Model and Reinforcement Learning, and we finally decided to implement the Recurrent Reinforcement Learning method on U.S. stocks market and China futures market. The results showed a convincing performance on both stocks and futures.

2 Introduction

2.1 Background

2.1.1 Machine Learning

Machine learning is one of the most exciting branches of artificial intelligence. Basically, it is defined as the field of study that looks into computers' ability to learn without being explicitly programmed. Based on its learning style, generally they can be categorized as supervised learning, unsupervised learning and semi-supervised learning. Nowadays, machine learning is being applied in different facets of applications. For example, it has been used in social networking sites like Facebook, where the system can segment out individual objects from photos and label them. Also web search engines, like Google, developed some algorithms which enables Google to rank web pages. Many scientists believe that machine learning will impact our life further in the future.

2.1.2 Algorithm Trading

In algorithm trading, the trading strategies are programmed and implemented by computers. In the past decades, predictive mathematical models, statistical arbitrage method, and event-driven system have been commonly used in algorithm trading. For manual traders, one of the most important issues is the tendency for them to be undisciplined. For instance, manual traders are likely to continue trading after suffering a big loss, because the more money they have lost, the more they are desperate for getting them back. Unlike manual traders, algorithms which set explicit rules will faithfully implement them, thus avoiding the man-caused interruptions and improving the trading performance.

2.1.3 Machine Learning in Algorithm Trading

In the financial market, the price of the assets changes over time very frequently. There are large uncertainties in the market, so even if there might exist a pattern for a short period of time, the pattern will tend to collapse in the long run. A human will find it difficult to go through loads of dynamic data and recognize different patterns to predict the future state. However, an appropriate machine learning algorithm, using considerable computing power, might be able to help one find such dynamic patterns. These dynamic models will be changing over time, but in most cases we only have to update certain parameters, which can be done automatically. So once the machine learning model is built, the future maintenance requires only routine work. In the financial industry, many companies have been trying to use smart machine learning algorithms plus powerful calculating devices in trading. Not only hedge funds like Bridgewater, Two Sigma, but also investment banks like J.P. Morgan are using machine learning in trading. They hire machine learning experts in academia to develop the most advanced trading system, and upgrade their trading facilities to get the fastest calculating speed and the most stable environment. It is like an arms race, a competition of both technology and capital.

2.2 Related Work

Although machine learning is a promising method in trading, we can find only a few related research studies in the applicability of machine learning algorithms on financial trading. One reason is probably that most of the experts in this field are working in industry, so even though there are a few reputed hedge fund companies that use machine learning in trading, they would not disclose their algorithms because it will benefit their competitors. One of the related research study by Kim, 2003 focused on Vector Support Machines (SVM's) to forecast financial time series. Besides, considering the Markov property of stock price, there are also some researches regarding Hidden Markov Model. Idvall and Jonsson, 2008 implemented Hidden Markov Model on high frequency foreign exchange data, however the performance is not very satisfying and unstable. Moody and Saffell, 2001 proposed a new approach to self-learn a trading strategy using reinforcement learning method, which takes into account the commission fees by calculating the derivative of the trading position over the weight of the neuro network. Based on that, Dempster and Leemans, 2006 further introduced risk control mechanism and implemented it on foreign exchange market, which showed a pretty good performance on test data. In our project, we used the mathematical principles in the paper of Moody and Saffell, 2001, then introduced our own risk-control mechanism, and implemented our algorithm on both U.S stocks market (day-level trading) and China Futures market (minute-level trading).

3 Algorithm Research

3.1 Research Scheme

To explore the feasible machine learning algorithm, we first dig into the market data. Inspired by some certain features, we tried to learn and evaluate the feasibility of implementing certain algorithms. We first explored the Hidden Markov Model, however afterwards we found it too complicated to be finished given limited time. So we switched to the reinforcement learning method instead. We then coded our program, evaluated the performance, and improved its efficiency. In the end, we drew our conclusion, and came up with some potential future work.

3.2 Market Data

We started our project with looking into the data. In mathematics, the price of the stocks, also many other assets like the futures, options and foreign exchanges, are usually modeled as Markov Processes, which involves stochastic items like Brownian Motion.

Date	Time	Open	High	Low	Close	Volume	Open_Int
2014-04-01	9:15:35	2147.2	2147.2	2147	2147	18	71809
2014-04-01	9:15:36	2147	2147	2146.8	2146.8	26	71817
2014-04-01	9:15:37	2147.2	2147.2	2147.2	2147.2	49	71847
2014-04-01	9:15:38	2147.4	2147.4	2147.2	2147.2	68	71861
2014-04-01	9:15:39	2146.8	2147	2146.8	2147	19	71867
2014-04-01	9:15:40	2147.2	2147.2	2147.2	2147.2	44	71868
2014-04-01	9:15:41	2146.8	2146.8	2146.6	2146.6	42	71895
2014-04-01	9:15:42	2146.6	2146.6	2146.4	2146.4	16	71903

Figure 1: Second-granularity Data of China SSE50 Stock Index Futures

Generally, the price of these assets is very volatile, which changes very frequently, but has some certain trend in small time period. Besides, the data we can get is in high dimensions. For example, in Figure 1, we have the price of China SSE50 Stock Index Futures in second-granularity, which involves date, time, open price, highest price, lowest price, close price, volume and open interest. There is also a recurrence property for the assets prices: history repeats somehow in a similar way, but will not be exactly the same.

3.3 Recurrent Reinforcement Learning

In the traditional reinforcement learning method, the model components include environment states $\{ S_i \}_n$, which describe the current states; Actions $\{ A_i \}_n$, which is the action to take at certain state; Transition Model P, which determines how the state changes from the previous step to the next step when taking a specific action; Reward $\{ R_i \}_n$, which gives the immediate step reward; Utility, which is a goal function regarding total step rewards. In our recurrent reinforcement learning (RRL) method, there will be also some corresponding components, which will be showed later. To explain the mathematical principle of RRL, we start by looking into the sample data as

Date	R_t
2014/9/1	10740
2014/9/2	10735
2014/9/3	10725
2014/9/4	10730
2014/9/5	10735

Table 1: Sample Data

This is a simplified version of data, with the first column as date and second one as the close price. We assumed here that we only allowed to trade one share per transaction, and we made some denotations as follows:

P_t: Price at date t. e.g. at date 2014/9/1, $P_t = 10740$

r_t: Price increment at date t, $r_t = P_t - P_{t-1}$

N: Dimension of observations, or number of bars
w: Weights vector of our neuro network, $w \in R^{N+2}$
T: Length of time period. e.g. In the sample data, T = 5
u: Commission fees per trade
Based on that, we introduced the components in our model:

- States X_t. We assumed that the current state involves the previous N days price information (by using price increment vector), the position in previous day, and noises. This is defined as a vector:

$$\begin{bmatrix} 1 & r_t & r_{t-1} & r_{t-2} & \cdots & r_{N-1} & F_{t-1} \end{bmatrix} \tag{1}$$

  In which the first element 1 stands for a combination of all noises, whose coefficient will be determined after training; r_t to r_{t-1} are the price increments, and F_{t-1} is our position in the previous day. For example, at 2014/9/5 , if N = 4, X_t = [1, 5, 5, -10, -5, 1] as showed in the sample data with a noise element 1, and a previous long position denoted as F_{t-1}, which we assumed to be 1 on 2014/9/4.

- Actions/Positions F_t: The position on date t. F_t can be 1 as a long position, 0 as no position, -1 as a short position.

- Immediate Rewards: $R_t = F_{t-1} * r_t$ – u * abs(F_t – F_{t-1}) For example, if we hold a long position yesterday F_{t-1}, and the increment from yesterday to today is r_t, then we can get a return of $F_{t-1} * r_t$. Besides, if we change our position at the end of the day, we will also have to pay an amount of u for commission fee, thus we subtract it from our subtotal return.

- Utility:

$$S_T = \frac{E[R_t]}{Std(R_t)} = \frac{E[R_t]}{\sqrt{E[R_t^2] - E[R_t]^2}} \tag{2}$$

  This is similar to the famous financial indicator, the Sharp Ratio. It evaluates the "profit per unit risk". When using this benchmark, we not only considered the total profit we can make, but also scaled it by the variance. If the variance is very high, the sharp ratio will be small, which indicates that our daily return is changing everyday dramatically, resulting into unstable return and potential big risk. Therefore, a high sharp ratio indicates both high and stable return. If we Denote A = E[R_t], B = E[R_t^2], then $S_T = \frac{A}{\sqrt{B \hat{} A^2}}$. We will use it in our program.

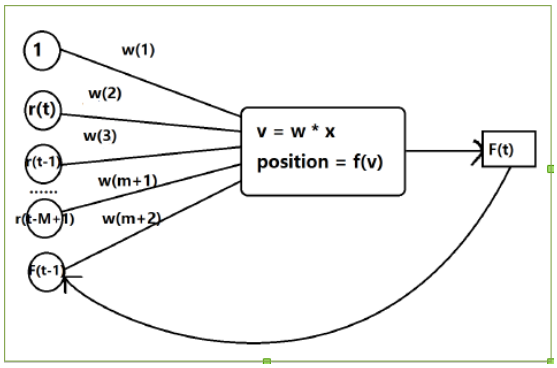
Now we construct our neuro structure: In a specific step of training process, our weight vector remains the same until the training is finished. For each date, we calculate :

$$v = w^T x = w_1 + w_2 * r_t + w_3 * r_{t-1} + \cdots + w_{M+1} * r_{t-M+1} + w_{M+2} * F_{t-1} \tag{3}$$

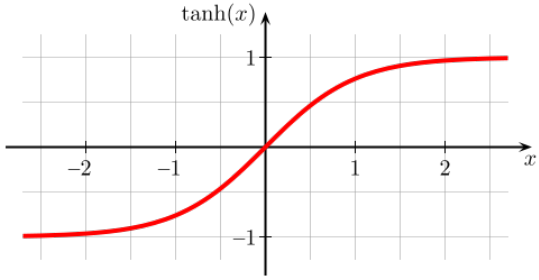
We use the hyperbolic tangent function to switch the value of v into a bounded interval (-1, 1) using the following function:

$$f(v) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4}$$

When all of this is done, we get an initial value of the position for the specific date, and we further set a threshold k>0. When the absolute value of f(v) is smaller than k, we judge that the trading signal is not significant enough, and we will keep an empty position, else for example if k<f(v)<1, we will take a long position at the end of the day, and if -1<f(v)<-k, we will take a short position at the end of the day.



(a) Neuro structure of the algorithm



(b) Hyperbolic tangent function

We update our weight vector using gradient ascent method:

$$w^{i+1} = w^i + \alpha * \frac{dS_T}{dw_i} \tag{5}$$

Core part of recurrent reinforcement learning method is implemented using the following calculation $\frac{dS_T}{dw_i}$:

$$\frac{dS_T}{dw} = \frac{dA}{dw\{\sqrt{B - A^2}\}} \tag{6}$$

where:

$$A = E[R_t] = \frac{\sum_{t=1}^{T} R_t}{T} B = E[R_t^2] = \frac{\sum_{t=1}^{T} R_t^2}{T} \tag{7}$$

$$B = E[R_t^2] = \frac{\sum_{t=1}^{T} R_t}{T} B = E[R_t^2] = \frac{\sum_{t=1}^{T} R_t^2}{T} \tag{8}$$

Equation 6 can be expanded as :

$$\frac{dS_T}{dw} = (\frac{dS_T}{dA} * \frac{dA}{dw} + \frac{dS_T}{dB} * \frac{dB}{dw}) \tag{9}$$

$$= \sum_{t=1}^{T} (\frac{dS_T}{dA} * \frac{dA}{dR_t} + \frac{dS_T}{dB} * \frac{dB}{dR_t}) * \frac{dR_t}{dw}$$

$$= \sum_{t=1}^{T} (\frac{dS_T}{dA} * \frac{dA}{dR_t} + \frac{dS_T}{dB} * \frac{dB}{dR_t}) * (\frac{dR_t}{dF_t} * \frac{dF_t}{dw} + \frac{dR_{t-1}}{dF_{t-1}} * \frac{dF_{t-1}}{dw}) \tag{10}$$

where $\frac{dS_T}{dA}, \frac{dS_T}{dB}, \frac{dA}{dR_t}, \frac{dB}{dR_t}$ and at each step and then we have the following conditions:

$$\frac{dR_t}{dF_t} = -u.sign(F_t - F_{t-1}) \tag{11}$$

$$\frac{dR_t}{dF_{t-1}} = r_t + u * sign(F_t - F_{t-1}) \tag{12}$$

$$\frac{dF_t}{dw} = \frac{\{tanh(w^T x_t)\}}{w} \tag{13}$$

$$= (1 - tanh^2(w^T x_t)) * (x_t + \frac{dF_{t-1}}{dw} * w_{N+2}) \tag{14}$$

$\frac{dF_t}{dw}$ is recurrent and is depend on the previous $\frac{dF_{t-1}}{dw}$,which can be iteratively programmed. Therefore we can $\frac{dS_T}{dw}$ and update the weight after each iteration.

4 Implementation

4.1 Backtest Method

In backtest, we split historical data into training and test dataset, and used training data to train our algorithm and determined the optimal weights of our neuro network. The benchmark we used here is the Sharp Ratio. In the training process, the performance of our algorithm gradually got better, because the Sharp Ratio kept going up, and even resulted into an unrealistic excellent result, which possibly overfitted the training data. So we have to apply the predetermined weights on test data and evaluate the performance on this dataset of the future. Here we assumed that we only trade one share per transaction, and we can at most have a long or short position of one.

```
D:\python34\python.exe C:/Users/acer/Desktop/Presentation/rrl_stock.py
Please enter the initial weight coefficient (0.01-1.00): 0.1
Please enter the learning Speed (0.1-5.0): 2
Reinforcement Learning 1 th step, Sharp Ratio: -0.015538848940336492
Reinforcement Learning 2 th step, Sharp Ratio: 0.06929876556275599
Reinforcement Learning 3 th step, Sharp Ratio: 0.1287925043522688
Reinforcement Learning 4 th step, Sharp Ratio: 0.12907035149680088
Reinforcement Learning 5 th step, Sharp Ratio: 0.13037694126475244
Reinforcement Learning 6 th step, Sharp Ratio: 0.14789008367970302
Reinforcement Learning 7 th step, Sharp Ratio: 0.15682476764671893
Reinforcement Learning 8 th step, Sharp Ratio: 0.1506964686043493
Reinforcement Learning 9 th step, Sharp Ratio: 0.14426021998371022
Reinforcement Learning 10 th step, Sharp Ratio: 0.15453325780698657
Reinforcement Learning 11 th step, Sharp Ratio: 0.1680635929509651
Reinforcement Learning 12 th step, Sharp Ratio: 0.16540300311604716
Reinforcement Learning 13 th step, Sharp Ratio: 0.17892174648168402
Reinforcement Learning 14 th step, Sharp Ratio: 0.17956181701849072
Reinforcement Learning 15 th step, Sharp Ratio: 0.18498154040414663
```

Figure 3: The training process on AAPL stock

4.2 Implementation on U.S Stocks

We implemented our algorithm on the stocks of AAPL (Apple) and AMZN (Amazon) under day-granularity. We used the simplified data which only involved close price. We assumed that the commission fee is 0.05 per share, much higher that the practical case. Besides, we didn't set any loss control limit in the case of stocks.

For AMZN, we used 21-day price increment vector, and our training data is the price of AMZN from 2013/1/2 to 2015/5/29, totally 606 days. We trained our algorithm with an initial weight factor 0.02, learning speed 0.3, and training steps 1000. The training cost us less than 1 minute. In addition, the natural net return, which is the profit we can make by simply taking a long position at the beginning date, on training data is $171.08. The test data is from 2015/6/1 to 2015/9/30, totally 86 days, and the natural net return is $80.97. The results on training dataset gives us net return of around $850, almost 500% of the original natural net return. The results on test dataset also shows pretty good performance with net return of around $220, which is 270% of the natural net return. However, here we can trade by both taking long and short positions. If we restrict our algorithm to only long positions, we can also get net return of $140 on test data, around 170% of the natural net return.

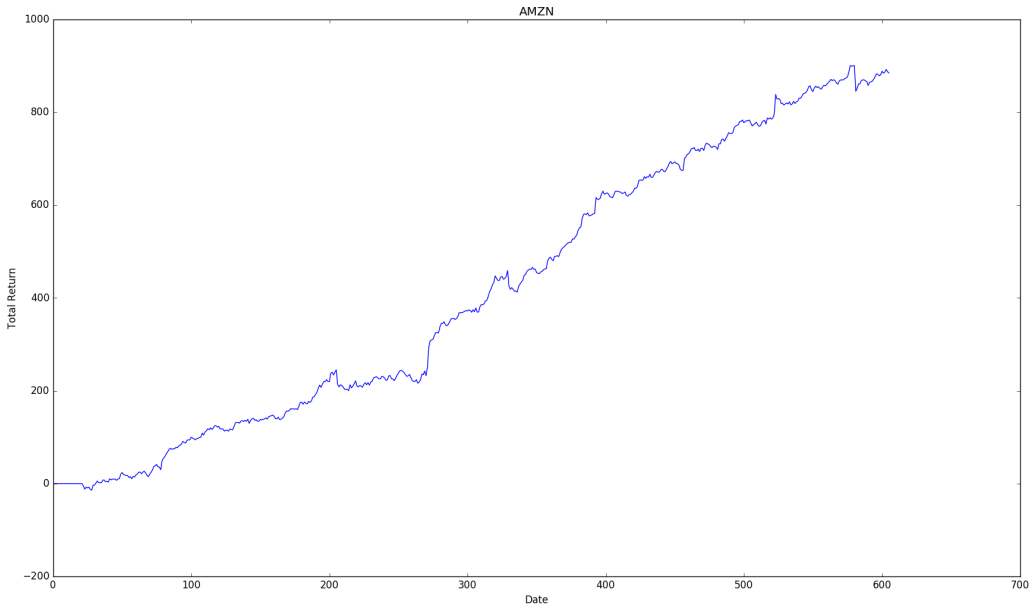


Figure 4: RRL Performance on AMZN training dataset

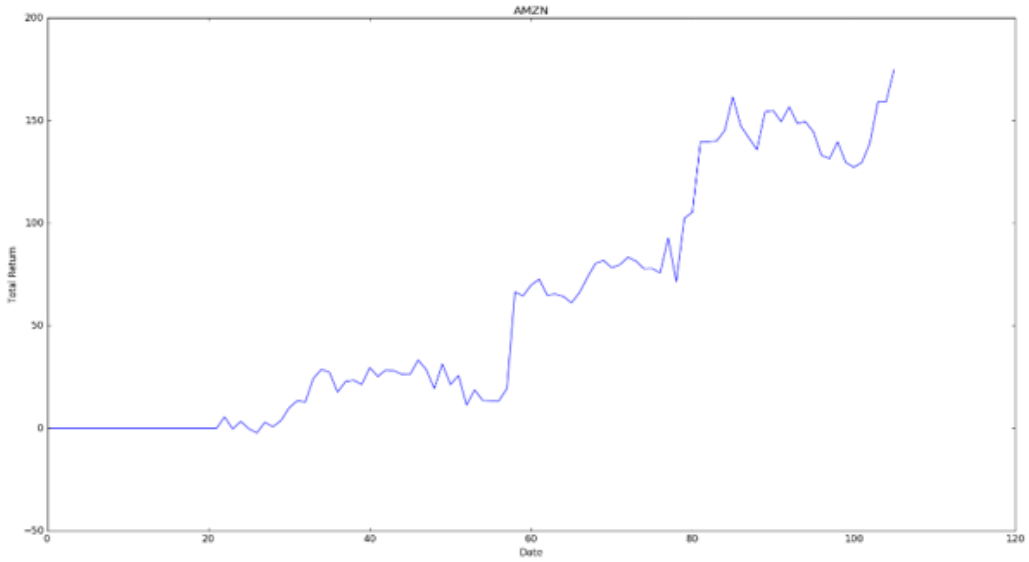


Figure 5: RRL Performance on AMZN test dataset

But this methods is not always working. We also implemented it on AAPL under day-granularity. We still used 21-day price increment vector, and our training data is from 2012/7/2 to 2015/1/29, totally 859 days. This time we use the initial weight factor 0.1, learning speed 1.0, and training steps 1000. The natural net return on training data is \$197. The test data is from 2015/12/1 to 2016/3/31, totally 83 days, and the natural net return (loss) is \$-12.24. After training, we get a considerably excellent result which gives us net return of \$197, almost 600% of the natural net return. However on the test data, we suffer a loss of \$25, which is twice larger than the original natural loss.

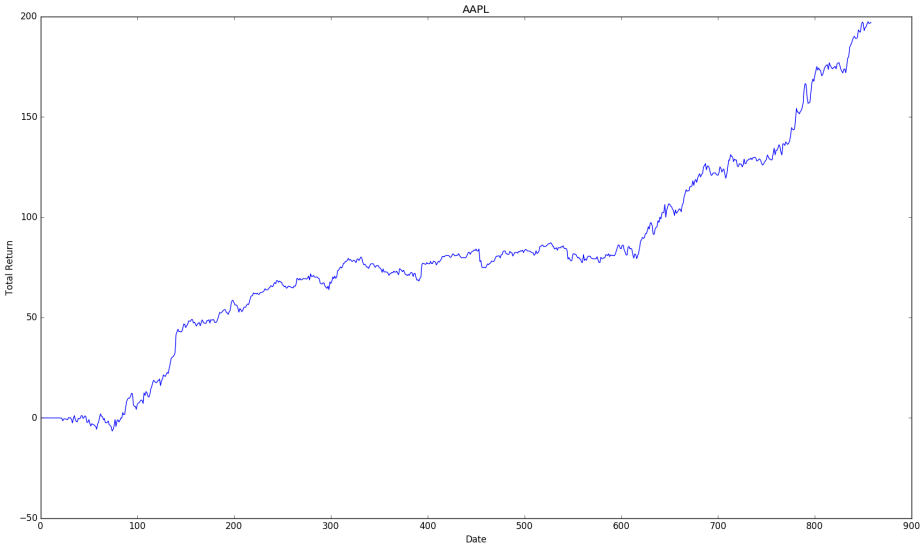


Figure 6: RRL Performance on AAPL training dataset

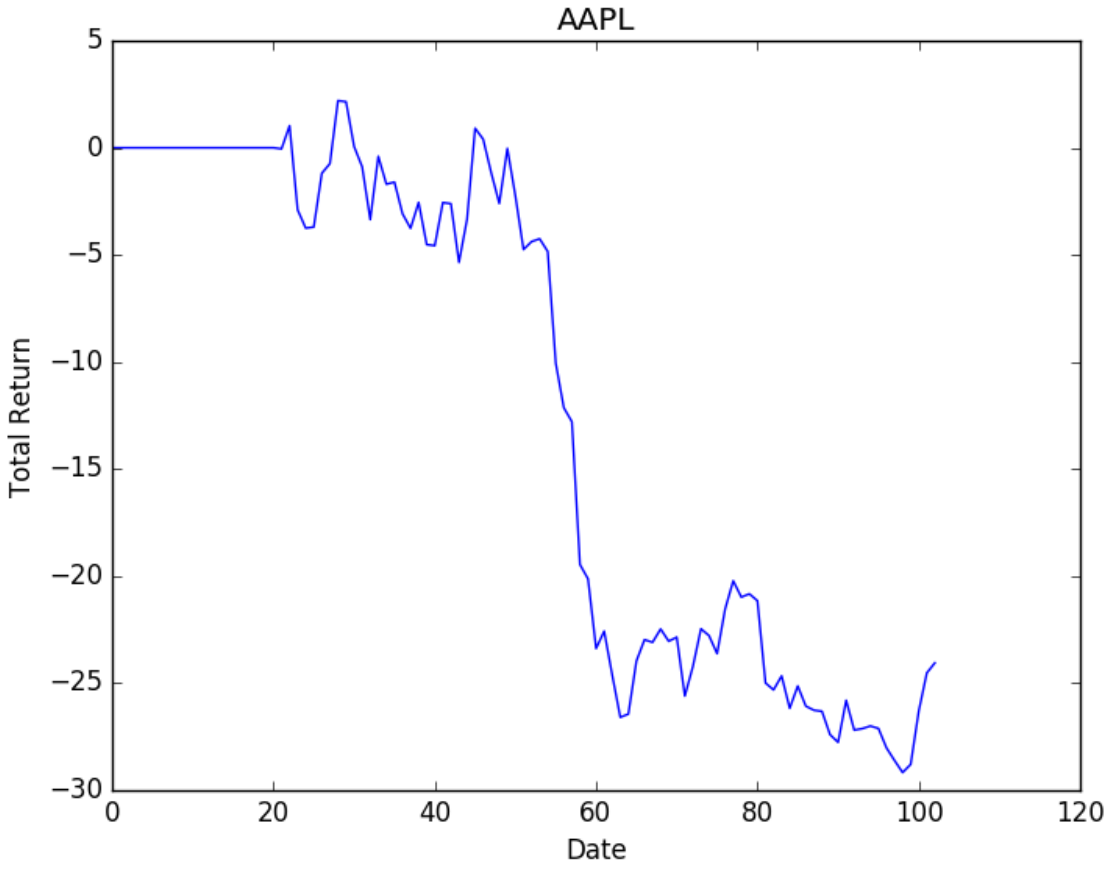


Figure 7: RRL Performance on AAPL test dataset

The reason why it doesn't work well may be concerned with the price movement style in test data, and also the lack of loss limit control. The trading strategy generated by RRL is likely to be a trend-following strategy. In the test result, the biggest drawback happened at the second half of Jan, 2016, when the stock price was extremely bumpy. Thus the strategy generated by RRL will perform good during the period with a significant trend, but will lose money in a bumpy period.

4.3 Implementation on Futures

We also implemented our RRL algorithm on China futures market. We took the example of PE(polyethylene) futures under minute-granularity. Here the point value (the corresponding amount of money for unit point) is 5 yuan, and the commission fee is around 3 yuan per share (a total amount for building and covering the position). Here we assumed no slippage (the loss that trader may suffer due to minute price changes in very small time period), however we can take them into account by raising commission fee in our program. We still used the simplified data involving only close price.

Our training data is the close price of the PE futures in every minute from 2013/1/4 to 2014/8/29, totally 90224 tuples, and our test data is from 2014/9/1 to 2014/11/28, totally 13275 tuples. Here we introduced risk control mechanism. Generally, we decrease the risk exposure by lowering trading times and forbid overnight positions, and we also cut the loss by setting absolute loss limit. In detail, we only trade during 9:30 A.M. - 2:30 P.M. every trading day, and we cover our position and stop trading for certain minutes when the sum loss in certain bars is greater than a threshold. To lower the trading frequency, we also keep holding the same position for at least 5 bars. These restrictions can help avoid overfitting and is corresponding to the expectation of real trading. We trained our program under 3 different conditions using different input of parameters. In the first example, we trained our program using initial weight 0.03, learning speed 0.5, number of bars 21, and training steps 200. It took about 2 hours to finish training. We used the price increment vector of previous 21 bars, and set a loss limit that when the sum loss in the last three bars exceeds 200 yuan we would covered our position and stop trading for 5 minutes. The training results gave us net profit of 36409.0 yuan, around 820% of the original margin for one share, and the number of trading times was 9114, which indicated that we made about 24 transactions every day. On the test set we got net profit of 3702.0, around 86% of the original margin, with trading times of 1357, around 20 transactions per day.

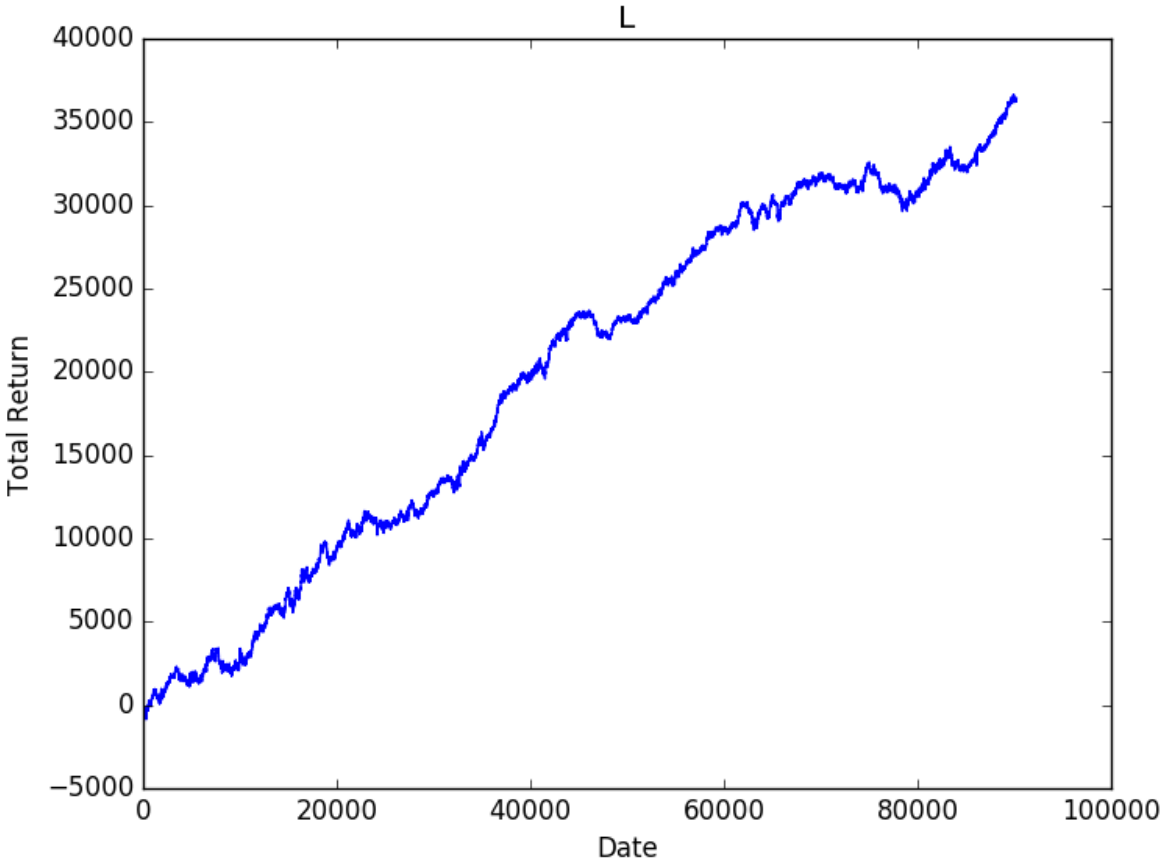


Figure 8: RRL performance on training dataset for PE futures example1

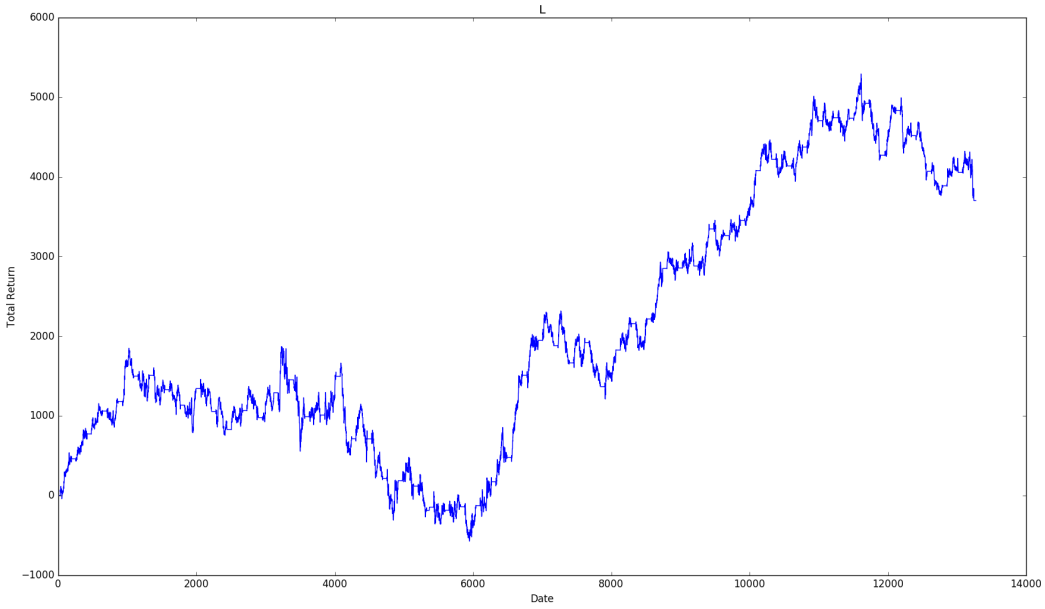


Figure 9: RRL performance on test dataset for PE futures example1

In the second example, we used an initial weight factor of 0.02,learning speed 0.8, number

of bars 30, and loss limit 175. The others remained the same. We got net profit of 28783.0 yuan on training dataset, around 650% of the original margin for one share, with trading times 9427, around 25 times per day.

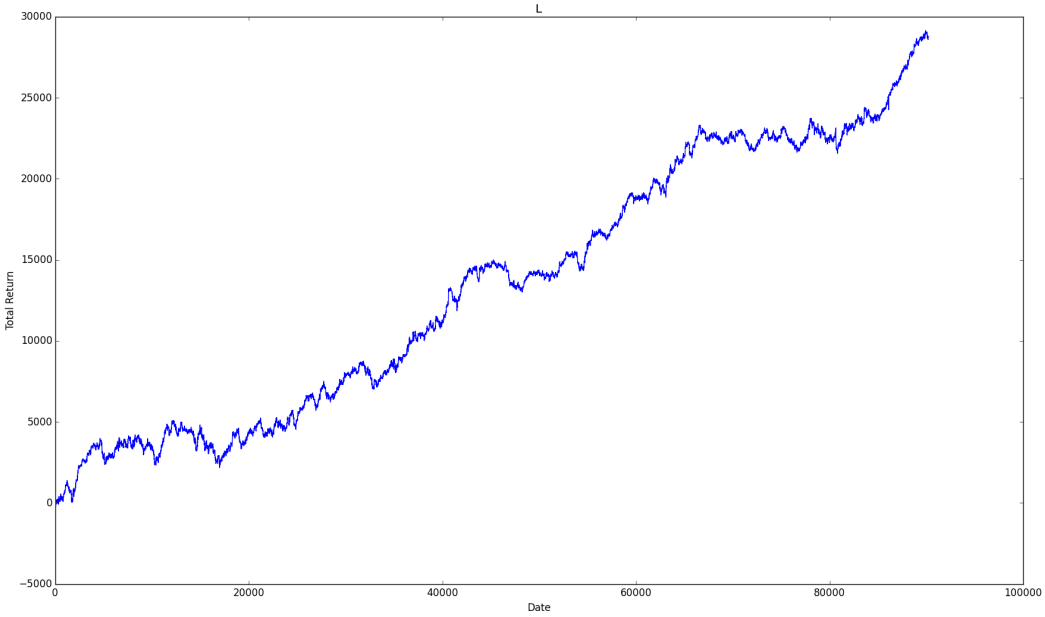


Figure 10: RRL performance on training dataset for PE futures example2

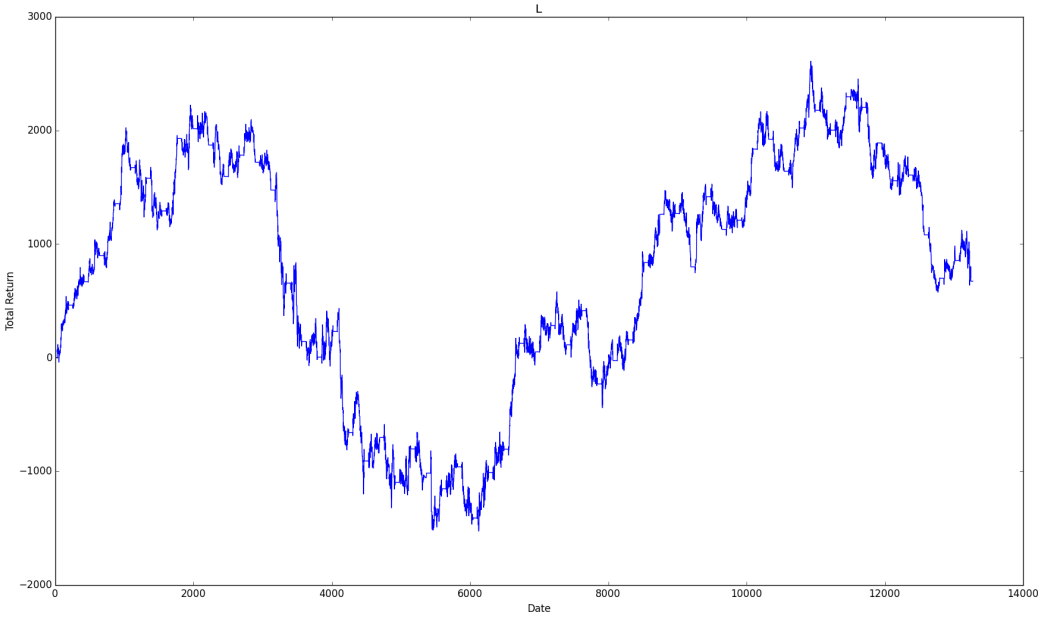


Figure 11: RRL performance on test dataset for PE futures example2

On test dataset, we only got net profit of 673.0 yuan, around 15% of the original margin, and we suffered a severe drawback. We traded for 1392 times, around 20 transactions per day.

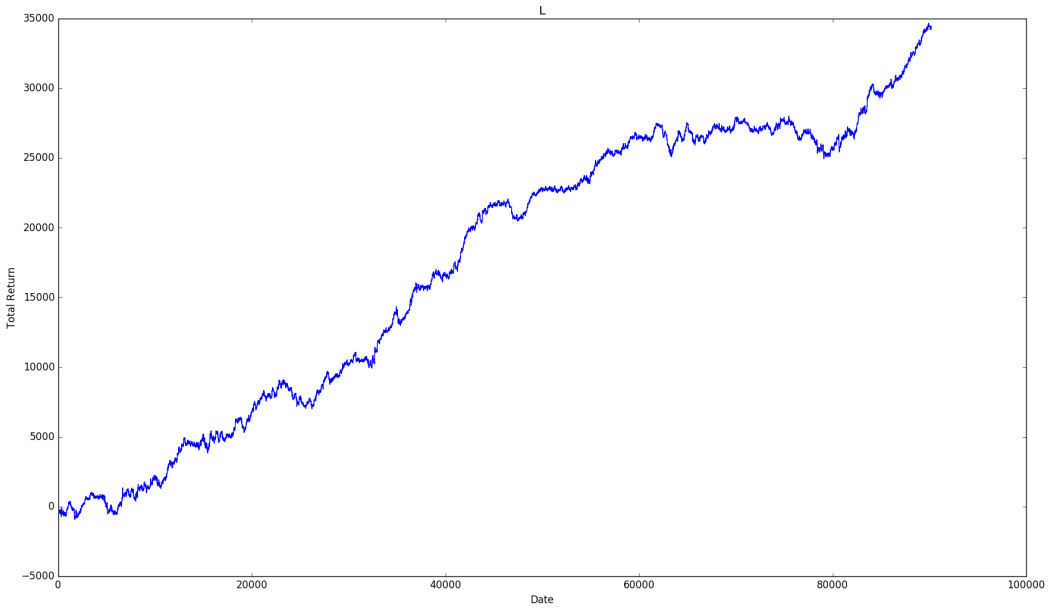


Figure 12: RRL performance on training dataset for PE futures example3

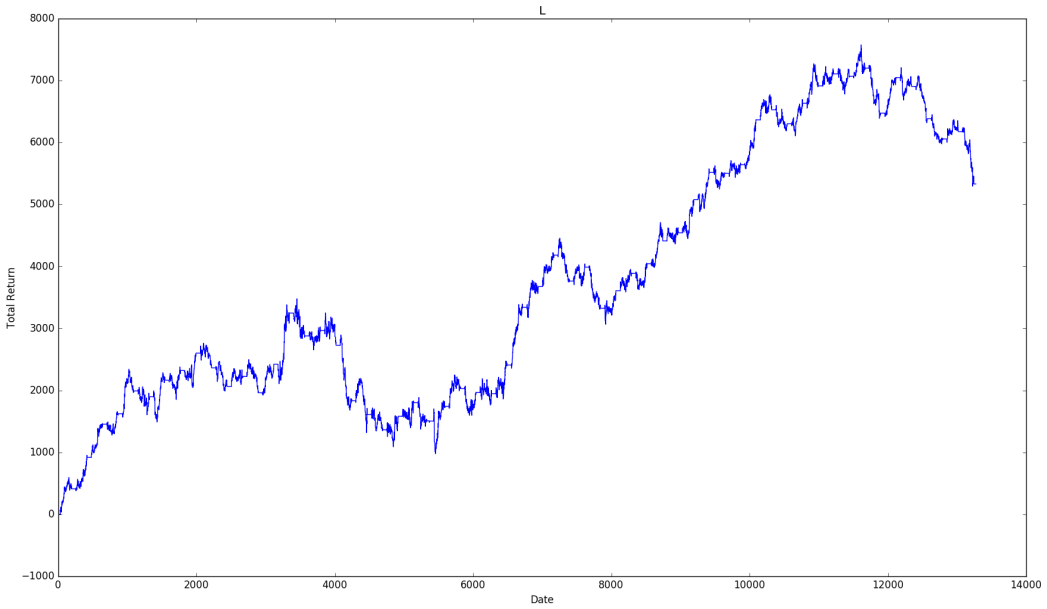


Figure 13: RRL performance on test dataset for PE futures example3

In the third example, we trained by using an initial weight factor 0.1, learning speed 1.0, steps 300, number of bars 14, and loss limit 150. Others remained the same as they were in case 1. We got net profit of 34424.0, around 790 of the original margin, with trading times 9096, around 24 transactions per day. On the test dataset we got net profit of 5331.0, around 124% of the original margin, and number of trading was 1349, around 20 transactions per day.

According to the different examples above, we can see that the last one which considered less time period and smaller loss limit performs better. This indicates that the trading strategy trained by RRL may perform better using relatively less previous information. Our algorithm trades the assets in a style like hit-and-run. Based on that, we also guess that RRL may perform even better on foreign exchange tick-level (milliseconds) high frequency trading, because under tick-level, the price of foreign exchange can be modeled as a less order Markov Process than stocks and futures.

5 Conclusion

According to the results of implementation, we have the following conclusions:

I. Trained by RRL, the utility function (S_T) can be optimized efficiently.

II. The raw algorithm shows promising result on test data, generally it will perform better than the natural return if the market is not too bumpy.

III. The strategy developed by RRL is likely to be a trend following strategy, for it performs worse when it is bumpy, and much better when there is a good trend.

IV. The style of the strategy on higher frequency data is like hit-and-run, very swift and sensitive. Under minute-granularity, we have to filter the trading signals to lower the trading frequency, thus avoiding slippage.

V. The strategy with such trading style may perform better on even higher frequency data, e.g. High frequency trading on Forex under tick-level.

6 Future Work

Although our algorithm has showed a pretty good performance, it remains to be improved for practical trading. For example, we used a very simple loss limit control mechanism in our program, while we can improve it by using some other advanced loss limit control scheme like trailing stop loss limit control. Besides, how to determine the initial weight factor and significance threshold remain to be explored. Furthermore, for a complete algorithm trading agent, it should consist of both backend and frontend. In our project, actually we only developed the backend. To implement our trading algorithm, we should connect to the API

of the brokers such like Interactive Brokers, read the real-time data, calculate trading signals in the backend and then send it to the front end to complete transactions. We are going to do this in the future.

7 Work Division

Yupu Song:

- Research work in the background of algorithm trading
- Analyzed papers of Hidden Markov Model and Reinforcement Learning
- Designed mathematical model and architecture of RRL algorithm
- Designed and wrote the raw Python codes which gave reasonable results.
- Introduced a risk control mechanism

Kartik Shetty:

- Research in the background of machine learning,
- Analyzed papers of Hidden Markov Model
- Got historical data
- Analysis of code for code optimization

A Pseudocode

```
#   Load Data

        #   Get the price increment series for the whole dataset
def increment(list):

#   Calculate increment vector of length n
#   n-number of bars to be used
def incVector(length, index, list):

#   Calculate the value of position for a single day
def position(list, weight, prepos):

#   Calculate the positions vector for the whole dataset
def posVector(wt, length):
        invoke postion(), incVector()


#   Calculate the sum of return
def sumReturn(pos, inc):

# Calculate the sharp ratio
def sharpRatio(weight, length,posVec):
        invoke sumReturn()

#   Calculate the gradient of position vector (dFt/dw)
#   For a while iteration step
def posGradient(weight, length,posVec):
        invoke posVector(), incVector, postion()

#   Calculate the gradient of Sharp Ratio vector
#   for a whole iteration step
def sharpGradient(weight, length, posVec):
        invoke posGradient(), sharpRatio()

#    Main function, the learning process
def learning(length, step):

#    Plot and output
```

The whole program is written in 140 lines, which actually can be optimized and compressed using more matrix calculation by using Numpy in Python. The complicated part is to understand the mathematical principles in RRL, to do matrix calculation in high dimension, and to figure out the correct form of output of each function. This involves intensive mathematical thinking, and we have debugged each of our functions to make sure every one of them gives reasonable output.

References

Dempster, Michael AH and Vasco Leemans (2006). "An automated FX trading system using adaptive reinforcement learning". In: *Expert Systems with Applications* 30.3, pp. 543–552.

Idvall, Patrik and Conny Jonsson (2008). "Algorithmic trading: hidden markov models on foreign exchange data". In:

Kim, Kyoung-jae (2003). "Financial time series forecasting using support vector machines". In: *Neurocomputing* 55.1, pp. 307–319.

Moody, John and Matthew Saffell (2001). "Learning to trade via direct reinforcement". In: *Neural Networks, IEEE Transactions on* 12.4, pp. 875–889.