Deep Reinforcement Learning applied to trading

Thesis submitted at the Polytechnic University of Madrid
in partial fulfilment of the requirements for
the degree of Master in Signal Theory and Communications

by

Borja Gómez Solórzano
Quant at Darwinex

Advisor: Eduardo López

July 2018

Contents

Abstract

The goal of the paper is to present an authomatic trading strategy over a single asset based on a reinforcement learning approach. I follow Direct Reinforcement Learning to solve it, the difference between DRL and the commonly Q-Learning is the learning phase, the first tries to train the model optimizing the inmediate rewards and the second the long term rewards. In the case of price series DRL is more suitable, cause the high noise and the continuity in the signal. The target is to maximize an utility function, in this paper I will introduce some of them, like Sharpe Ratio. The trader's action is done by a decision function, this function is the output of a neural network with a deep architecture, the input of the network is the price series and some side information like other price series and indicators, this side information is needed for filtering the trading actions over the main asset traded. In this implementation I will have to face, in the training phase, with some problems of the backpropagation algorithm like named vanishing gradient, to solve this problem I implement a solution based on LSTM networks. Finally, I include graphics to show the execution of the algorithm over the time, the most important are: The equity curve, the trader's action and the evolution of the utility function. I will add also some trading based statistics to measure the performance of the strategy like drawdowns, p/l.

Introduction

Traditional trading algorithms

Traditionally, trading algortithms are build using technical analysis techniques based on asset price indicators like moving averages or oscillators like RSI and fixed parameters. This parameters are selected running different backtest over the history and are very sensitive to overfitting. The principal drawback of this approach is the difficult to adapt to different market regimes, Reinforcement Learning techniques have the capability to learn from interactions with the environment.

Reinforcement Learning

[1]Recurrent Reinforcement Learning, uses immediate rewards to train the trading systems, while the second (Q-Learning (Watkins 1989)) approximates discounted future rewards. [4] Q-Learning is in the context of action spaces and discrete state. In many situations, this will suffer the "curse of dimensionality" When Q-Learning is extended to function approximators, researchers had shown it fail to converge using a simple Markov Decision Process. Brittleness which means small change in the value function can produce large changes in the policy. In the trading signal world, the data can be in presence of large amounts of noise and nonstationarity in the datasets, which could cause severe problems for a value function approach. [3]RL is a prevalent self-taught learning paradigm that has been developed to solve the Markov decision problem. According to different learning objectives, typical RL can be generally categorized into two types as critic-based (learning value functions) and actor-based (learning actions) methods. Critic-based algorithms directly estimate the value functions that are perhaps the mostly used RL frameworks in the filed. These value-function-based methods, e.g., TD-learning or Q-learning [15] are always applied to solve the optimization problems defined in a discrete space. The optimizations of value functions can always be solved by dynamic programming [16]. While the value-function-based methods (also known as critic-based method) perform well for a number of problems, it is not a good paradigm for the trading problem, as indicated in [17] and [18]. This is because the trading environment is too complex to be approximated in a

discrete space. On the other hand, in typical Q-learning, the definition of value function always involves a term recoding the future discounted returns [17]. The nature of trading requires to count the profits in an online manner. Not any kind of future market information is allowed in either the sensory part or policy making part of a trading system. While value-function-based methods are plausible for the offline scheduler problems [15], they are not ideal for dynamic online trading [17], [19]. Accordingly, rather than learning the value functions, a pioneering work [17] suggests learning the actions directly that falls into the actor-based framework. The actor-based RL defines a spectrum of continuous actions directly from a parameterized family of policies. In typical value-function-based method, the optimization always relies on some complicated dynamic programming to derive optimal actions on each state. The optimization of actor-based learning is much simpler that only requires a differentiable objective function with latent parameters. In addition, rather than describing diverse market conditions with some discrete states (in Q-learning), the actor-based method learns the policy directly from the continuous sensory data (market features). In conclusion, the actor-based method exhibits two advantages: 1) flexible objective for optimization and 2) continuous descriptions of market condition. Therefore, it is a better framework for trading than the Q-learning approaches. In [17] and [19], the actor-based learning is termed DRL and we will also use DRL here for consistency

Deep Reinforcement Learning

One of the most important steps in a the reinforcement learning is to select the best features to take the best actions in order to obtain the best rewards. Adding some layers to the deep architecture of the network finds more complex relationship between the transformed features passing through the different layers. DL is an emerging technique [28] that allows robust feature learning from big data. To implement feature learning, in this paper, we introduce the prevalent DL into DRL for simultaneously feature learning and dynamic trading. DL is a very powerful feature learning framework whose potentials have been extensively demonstrated in a number of machine learning problems. In detail, DL constructs a DNN to hierarchically transform the information from layer to layer. Such deep representation encourages much informative feature representations for a specific learning task.

Chapter 1

Basic market concepts

Asset: Is a contract to buy some valuable thing. Some times you buy physically the object (a share of a company) and some time this contract is a derivative over an underlying asset (an option over the same company). The place (physical or electronic) where you can buy an asset is called the market. There are different tipes of markets

Order: Is an electronic contract with all the information needed to buy an asset.

The order elements are: the asset name, the commission charged to the market provider services, the order type is the side of the market you want to bet, (buy/long) if you bet with the market or (sell/short) if you go against it. The contract express the quantity in terms of number of assets you want to buy, so if you buy a contract

Trade: Is a closed order.

In this chapter we will introduce the problem of the trader agent. Our agent only can trade one asset.

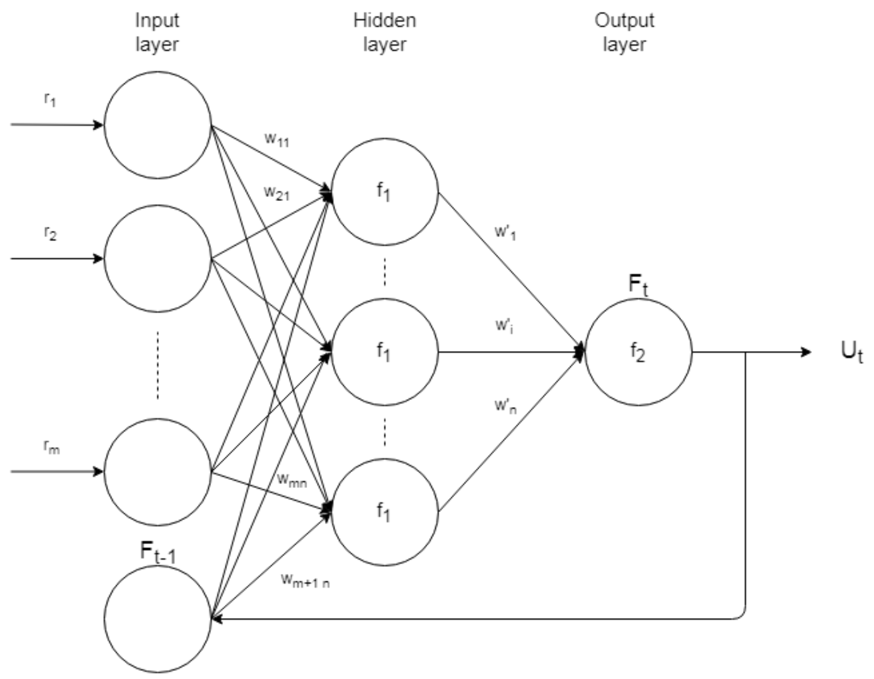
Chapter 2

Recurrent Reinforcement Learning

[1] Rather, the system takes actions (makes trades), receives feedback on its performance (an evaluation signal) and then adjusts its internal parameters to increase its future rewards. With this approach, an ultimate measure of trading performance , such as profit, utility or risk-adjusted return is optimized directly.

2.1 The recurrent model

Here we consider the following first order recurrent model with one hidden layer, as in [5], we can extend it also two second order, third order, etc. depending of the number of terms F_{t-i}



the equations for each one of the layers are

$$F_t = f_2(\sum_{i=1}^{n} w_i' h_i + b')$$

$$h_i = f_1(\sum_{j=1}^{m} w_{ji} r_{t-i} + w_{m+1,i} F_{t-1} + b)$$

Algorithm 2.1 Direct Reinforcement

```
1: Input: network architecture params
2: Output: $\pi_\theta$ policy, u=max utility
3: for (for each episode) do
4:     Initialize $\theta$ randomly
5:     for (for each step in the episode) do
6:         take action $a'$ $\pi_\theta(.|s)$ and observe state $s'$ and $r$
7:         $r = reward(s', a, a')$
8:         $a = a'$
9:     end for
10:    $u(s, a; \theta) = utility(rewards)$
11:    $\pi_\theta = \text{argmax}_{a' \in A} u(s, a; \theta)$
12: end for
13: return $u, \pi_\theta$
```

2.2 Neural network trader agent

I_t is all the information available at time t, including the prices and external information history. $F_t = F(\theta; F_{t-1}, I_t) \in [-1, 1]$ is the trader action (the percentage of asset bought), we supose in this first approach presented that the asset is infinitelly divisible, we use for this the tanh function. $U_t = U(R_1, R_2, ..., R_T)$ is the utility function that measures the goodness of the actions the trader takes and the one that we want to maximize. The learning algorithm is based on the gradient ascent with updating formula:

$$\frac{dU_t(\theta)}{d\theta} = \sum_{t=1}^{T} \frac{dU_t}{dR_t} \left\{ \frac{dR_t}{dF_t} \frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}} \frac{dF_t}{d\theta} \right\} \tag{2.1}$$

which is path dependent because the derivatives $\frac{dF_t}{d\theta}$ depends on the previous trades. The optimization of the recurrent network weights is done by an algorithm called Back Propagation Through Time (BPTT). It could be possible an online training using stochastic gradient as in [2], but is out of the scope of the thesis and is not implemented in the examples.

2.3 Utility function: Sharpe ratio

The rewards are computed as

$$R_t = F_{t-1} r_t - c|F_t - F_{t-1}| \tag{2.2}$$

where c are the commissions per order and r_t the price differences between times t and t-1. We define the following performance measure

$$S = \frac{\mu(R_t)}{\sigma(R_t)} \tag{2.3}$$

(where μ is the average and σ is the standard deviation) known as Sharpe ratio, which is a risk-adjusted measure of the strategy.

Chapter 3

Optimization drawbacks

[4] The fundamental issue was that gradients propagated over many stages tend to either vanish or explode. In a traditional recurrent neural network, during the gradient back-propagation phase, the gradient signal can end up being multiplied by humongous times perhaps as many times as of the timesteps by the weight matrix associated with the connections between the neurons of the recurrent hidden layer. In other words, the magnitude of weights in the transition matrix can have a large impact on the learning process. If the weights in this matrix are small it leads to vanishing gradients where the gradient signal gets so small that learning either becomes very slow or stops working altogether. In defiance of this, if the weights in this matrix are large where the gradient signal is large, where we often refer this as exploding gradients. Problems with the unfolded BPTT algorithm (Gradient Ascent). [6] Evolutionary algorithms, for instance particle swarm optimization prevens to get stuck in a local minimum using emerging intelligence, each swar particle tries to optimize locally and then with the information provided by other swarms tries to reach a global optimum solution. Other more complex architectures like LSTM networks instead our dense model [4]have the hability to “forget” past results, the explanation of this techniques is out of the scope of the thesis.

Chapter 4

Experiments

First we are going to present the problem for the experiment. Our trader is an agent based trading limited to a single asset, is a Deep Recurrent Reinforcement Learning model with one hidden layer and ... neurons, the network is dense meaning that each layer is fully conected to the next layer, and the utility function selected is the Sharpe Ratio. Experiments are done with Gold asset (XAUUSD ticker), backtested over the period ... and tested over the period, in a ... timeframe, we tested some commissions value in order to see the efect over the performance. As side data I have used two indicators a moving average and a RSI over the asset price timeseries.

4.1 Performance measures

For neural network trading MSE loss, for trading results we consider as performance metric the sharpe ratio (the utility function we want to maximize), both for trading and testing data.

4.2 Results

TODO: Include graphics (The graphics contains the evolution of the: profits and losses of the strategy, the trader agent actions, and the Sharpe Ratio evolution) and tables with the results changing the parameters.

Chapter 5

Conclusions

As we can see in the graphics showed...

Appendix 1.

Bibliography

[1] John Moody and Matthew Saffell, Reinforcement Learning for Trading, *Advances in neural information processing systems, July 1999*

[2] John Moody, Lizhong Wu, Yuansong Liao & Matthew Saffell, Performance functions and Reinforcement Learning for trading systems and portfolios*, Journal of Forecasting, Volume 17, Pages 441-470, 1998.*

[3] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai, Deep Direct Reinforcement Learning for Financial Signal Representation and Trading, *IEEE Transactions on neural networks and learning systems*, *2016.*

[4] David W. Lu, Agent Inspired Trading Using Recurrent Reinforcement Learning and LSTM Neural Networks, *arXiv:1707.07338 [q-fin.CP]. July 2017.*

[5] Carl Gold, FX Trading via Recurrent Reinforcement Learning, *Conference: Computational Intelligence for Financial Engineering. January 20*03.

[6] J. Kennedy, R. Eberhart, Swarm Intelligence. *Morgan Kaufmann, San Francisco, CA, 2001*