March 31, 2019

1 Motivation

Reinforcement Learning(RL) is considered as a third paradigm alongside unsupervised and supervised learning. But nevertheless, unlike the other approaches RL considers the whole problem of goal-directed agent that interacts in an uncertain environment. This involves learning what is necessary to do in order to maximize a numerical reward signal.

The most important distinguishing features of a RL problem are: 1. they behave as closed-loop problems given that its learned actions influence later inputs, 2. learners must try different operations to discover which strategy yields the most reward, and 3. actions may affect next situations and all subsequent rewards.[1]

Among its main applications are: resources management in computer clusters, games, traffic light control, robotics, etc. The aim of this project is to apply the RL techniques to make decisions in the stock market given that it involves the interaction of an active agent that has to make decisions based in imperfect information environment while is interacting with other market participants. Some previous findings indicate that RL can be successfully applied to the portfolio problem and its performance exceeds the supervised learning approach[2] and Q-learning algorithm operates better than kernel-based methods[3].

The approach in this paper to the stock market prediction problem is to apply the Q-learning algorithm that is a RL method that uses the principles of Dynamic Programming. The paper is organized with the following sections: first Data, the explanation of stock market data utilized for the purpose of analysis; follow by Methodology, an overview of what

[1] Sutton and Barto, 2014
[2] Neuneier, 1996
[3] Bertolutzzo and Corazza, 2012

is Reinforcement Learning, Q-learning and formulation of the problem; next section Results, talks about the implementation of RL to the stock market problem and obtained results; and the last section Conclusion, explains findings and conclusions of the project.

2 Methodology

2.1 Overview

Before delving into the specifics of employing reinforcement learning to the problem of automated trading, it will be informative to discuss the general theory and its underlying principles.

Reinforcement learning aims to maximise a given reward signal by undertaking certain actions (in a restricted space). In this framework, an agent must take the state of the environment as input and take actions to alter the future state. A measurable goal related to the environment is also necessary in the problem formulation. Beyond this, each reinforcement learning problem contains four sub-elements: a *policy*, a *reward signal* and a *value function*.[4]

The policy defines the agent's actions in different environment states. The reward signal defines the goal and should be maximised over the course of the learning process. The value function maps the current state to a value so the agent can make optimal longer run decisions. It can be seen as the expected total future reward that can be obtained beginning from that state. Most of the challenges associated with the implementation of reinforcement learning derive from the estimation of the value function.

Formally, we construct a Markov Decision Process (MDP). In an ideal situation, we would have access to the value function directly in tabular form when we have a tractable action and state space.

At a sequence of discrete time steps $t = 0, 1, 2, 3...$, the agent interacts with the environment. At each step t, the agent receives state information $S_t \in \mathcal{S}$ and performs an action $A_t \in \mathcal{A}(s)$. As a consequence of the action, the agent receives a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and transitions to a new state S_{t+1}.

[4]sutton book

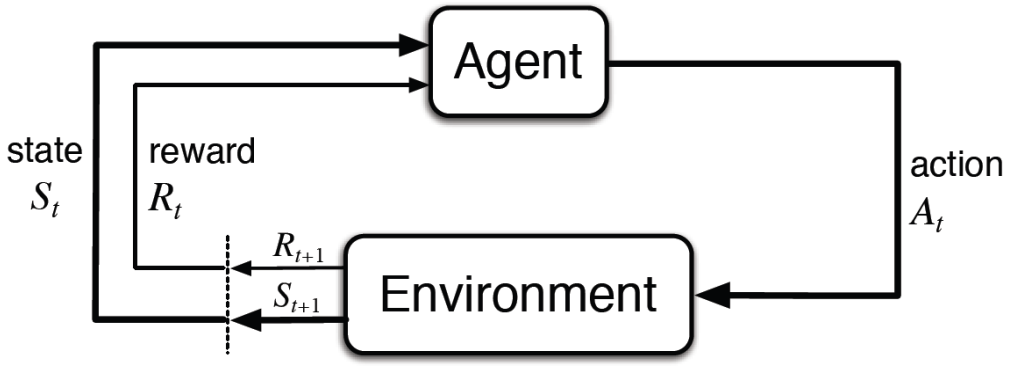

Figure 1: Agent and Environment Interaction

In the context of a *Markov* decision process, the future rewards (R_t) and states (S_t) only depend on the previous state and action.

The general reinforcement learning paradigm involves find an optimal policy π to maximise the expected discounted return. The discount factor is required to ensure that rewards in the distant future are less valuable than current rewards.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Value and action-value functions allow the actions of the agent to be assessed under the implementation of a certain policy. The value function and action-value functions respectively are defined below:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t | S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \text{ for all } s \in \mathcal{S}$$

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t | S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$$

Ideally, the value function is decomposed into the following (known as the *Bellman's Equation*):

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi\left[G_t | S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma G_{t+1} | S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p\left(s', r | s, a\right)\left[r + \gamma \mathbb{E}_\pi\left[G_{t+1} | S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p\left(s', r | s, a\right)\left[r + \gamma v_\pi\left(s'\right)\right], \qquad \text{for all } s \in S
\end{aligned}
$$

Both expressions relate to a specific state and action taken at any time t.

A reinforcement learning problem principally involves pursuing the optimal policy π which is is said to maximise the value and action-value functions:

$$
\begin{aligned}
v_*(s) &\doteq \max_\pi v_\pi(s) \\
q_*(s, a) &\doteq \max_\pi q_\pi(s, a)
\end{aligned}
$$

In the most simple cases where the value and action-value functions are specified, dynamic programming can be used to derive the optimal policy π.

In the algorithmic trading problem, the value function cannot be ascertained easily in this way. To deal with these situations and arbitrarily large state space, approximate solution methods must be used. This is known as a partially observable markov decision process as the state is only observed indirectly and cannot be fully known (we cannot know the trading behaviour of other agents for example) and we do not have access to the transition probabilities between states.

Q-Learning is a technique whereby the value functions are repeatedly estimated based on the rewards of our actions and assumes no prior model specification.

2.2 Q-Learning

Q-learning attempts to estimate q_* (optimal action-value function) without any regard for the policy followed.From a high level perspective, The Q-Learning algorithm proceeds by randomly initialising q, perform actions, measure reward and update q accordingly. The final output after a training period should be a stable approximation of the q_* table.

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$

Initialize $Q(s, a)$, for all $s \in \delta^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$

Loop for each episode:

Initialize S

Loop for each step of episode:

Choose A from S using policy derived from Q(e.g. , ε -greedy)

Take action A, observe R, S'

$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q\left(S', a\right) - Q(S, A)\right]$

$S \leftarrow S'$

Notice the Bellman equation appearing in the update phase of the algorithm.

2.2.1 Deep Q Learning

An extension of this idea (which we aim to employ) is to use neural networks to approximate the Q-function.

A neural network is an appropriate tool for our use case due to the infinite nature of the state space. Estimating a table for each possible state would be excessive in regards to memory requirements.

To train the neural network on the state space, we must define a loss function. As The Bellman Equation defines the optimal result, we can use this to calculate our loss as follows:

$$\hat{Q}(s, a) = R(s, a) + \gamma \max_{a' \in A} Q(s, a)$$

$$\text{Loss} = \|Q - \hat{Q}\|_2$$

In general, a partition of the data is used to train the neural network and approximate the q_* function, then this is used as our action-value function for deciding the optimal policy.

A neural network with 3 hidden layers was implemented in Tensorflow in our case.

2.3 Problem formulation - Algorithmic Trading

Now we must formualate the trading problem as a Markov Decision Process and define the states, actions and rewards.

- State: $\mathcal{S} = \{prices, holdings, balance\}$ where *prices* refers to the current prices of all the stocks in our portfolio, *holdings* the quantity of each stock held and *balance* as the total portfolio value. In our simple variant, we are only trading one stock but include a history of prices also (default 200)
- Actions: $\mathcal{A} = \{buy, sell, hold\}$ For simplicity and computational tractability, we restrict our action space to buy 1 stock, sell 1 stock and hold.
- Rewards: $R_t \in \mathcal{R}$ can be defined as the change in the portfolio value due to an action A_t. Our Reward was defined as $R_t = balance_t - balance_{t-1}$
- Policy: π which is governs the trading strategy at state S. We converge on the optimal policy by approximating the q_* function.
- Action-value function: $q_\pi(s, a)$ as defined above. The expected reward we obtain by following policy π, choosing action A while in state S. The action-value function is approximated by a neural network.

Initially we do not consider any transaction costs and only trade with a single stock (AAPL). Furthermore, a negative portfolio is not permissible ($balance_t \geq 0$ for all t).

In our primary model, the initial $balance_0$ is set to \$1000 and $price_0$ to the price of the stock at our chosen start time t_0

3 Results

Our neural network was set-up to learn from a 200 length rolling window (history) and as such, each state S_t is a $200 + 2$ dimensional vector containing the history, the current budget ($balance_t$) and the current number of stocks held ($holdings_t$).

To ensure that the Q function was being learned in the correct way and trending towards an optimal policy, we train the neural network over multiple epochs. From the plot below, the final portfolio value achieved continues to increase, signalling convergence towards an optimal policy. The hyper-parameters were tweaked to derive maximum gain. (PLOT)

To evaluate the performance of the algorithm, we compared it to a standard *Buy and Hold* strategy and to a randomised action strategy. The plots of the portfolio value over time below clearly demonstrate the superiority of Q-Learning, achieving a final portfolio profit of XXXX.

Despite the algorithm performing well, the *Buy and Hold* strategy also performs similarly as AAPL is a strong stock with a stable upward trend.

To highlight the strength of Q-learning more effectively, we compared the algorithms behaviour on a more volatile stock, where *Buy and Hold* should be less effective.

One of the noticeable strengths of the Q-Learning approach is its control of the volatility (SEE TABLE), in a practical setting, this would be an attractive feature for a potential investor.

3.1 Weaker stock comparison

To illustrate how Q-Learning can out perform a *Buy and Hold* strategy, we selected a stock with a less favourable trend over the past 10 years (WWL - Whiting Petroleum Corp.). Notably, the Q-learning algorithm clearly learns to buy low and sell high from its training, and out-performs *Buy and Hold* dramatically.

4 Conclusion

The aim of the investigation was to implement reinforcement learning in the context of stock trading. A deep Q-Learning approach was used to approximate the Q-function (action-value function) and develop an trading policy. Through our experiments with AAPL and WLL stocks, we have managed to show that with minimal training and hyper-parameter tuning, our agent can outperform random actions and a *Buy and Hold* strategy. However, the margin of improvement is variable depending on the presence of an upward trend in the stock price.

We have also demonstrated the agent's ability to manage volatility of the portfolio effectively. There are many possible future developments that could be explored such as using multiple stocks, modifying the reward function, performing online-learning and doing further hyper-parameter tuning.