Reinforcement Learning for Portfolio Optimization

Angad Singh, Nikhil Krishnan, Xiaotian Zhang & Zhi Ren

June 2, 2018

1 Introduction

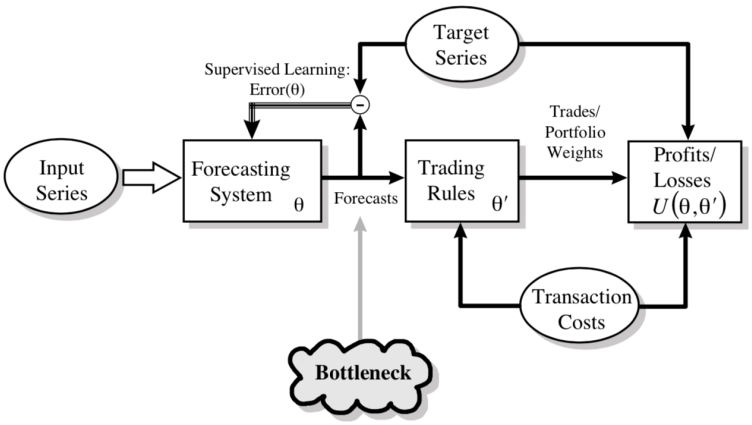
Portfolio optimization is a classical problem in finance with a long history of research, for example by Markowitz in the 50's and Merton in the 70's. The basic premise is that an investor attempts to optimally balance her money between stocks and bonds. The stocks are risky, in that their future values are unknown at the time of purchase. The bonds are risk-free, in that their payoffs are known with certainty at the time of purchase. The investor is able to re-balance her portfolio multiple times and has access to a variety of information when making her decisions. The investor's overall objective is to maximize some utility function.

The theoretical finance literature typically takes an analytical approach to the problem. Assumptions are made about the stochastic dynamics of prices and information, and then optimal portfolios are found analytically using techniques from stochastic control. A key assumption in these problems is that the market is competitive, which means that market dynamics are unaffected by the investor's actions.

In practice, professional investors take a data driven approach to the portfolio problem, and in recent years machine learning has become the tool of choice. The standard approach is to use supervised learning to forecast the time series of returns. One typically takes as features a time series of financial indicators and as labels the forward returns on the market. So the at time t the features would consist of various forms of currently available financial data, and the label would be the return on the stock market over the interval $[t, t+1]$. The features should contain predictive signals on the forward returns, and by using supervised learning over a historical sample we should be able to capture this predictability. These predictions are later used to form a trading strategy, the simplest one being to go long when a gain is predicted and go short when a loss is predicted. A variety of such algorithms are presented in a recent paper from JP Morgan [KK17].

There are some clear drawbacks to the supervised learning based forecasting approach. The true objective in the portfolio problem is obtain a profitable a trading strategy; a trading decision rule based on available information. However, the learning system is not optimized with this objective in mind. Instead it is optimized to minimize forecasting error, which is an intermediary quantity. Furthermore, even after the forecasts are generated, a second system is required to turn these forecasts into a trading strategy. This typically involves assessing the reliability of the forecasting

system while also taking into account transaction costs from frequent trading. Aside from it's complexity, this two system approach also presents an informational bottleneck as shown in the following diagram from [KK17]:



In this project we attempt to circumvent this bottleneck by directly learning how to trade using reinforcement learning. We learn a policy that maps features to trades, thereby combining the two systems above into one. Our training methodology is model-free and similar to a policy gradient approach. We test our trading agent by designing a long-short trading strategy for the S&P 500 index.

The rest of the paper is organized as follows. In section two we discuss related literature on which this project is based. In section 3 we go over the basic background and mathematical setup involved in a portfolio optimization problem. In section four we formulate the problem as a Markov Decision Process and in section five we present our learning algorithm and derive the policy update rule with respect to our reward functions. Section six discusses experimental results.

2 Previous Work

Non-reinforcement learning techniques have been summarized in a recent whitepaper by JP Morgan [KK17], which includes methods such as XGBoost, using random forest techniques, lasso regression, etc. on financial data features, either to predict the next day's return, or to classify option returns into categories. The authors provide many examples of success, but reproduction of their results is difficult without access to their proprietary database. Therefore, in this paper, we will attempt to use some financial features together with their XGBoost technique to compare its performance versus our reinforcement learning technique.

There is relatively few work in the literature on the use of reinforcement learning in finance. In [MW97], the authors provide a basic framework for the use of RL in portfolio selection. In a trading system, we are trying to maximize function U that measures any values that are of interset

to investors, for example, profit, Sharpe Ratio, etc. In the framework, traders are assumed to only take long, short or neutral positions. Let F_t be the position at time t, then $F_t \in [-1, 1]$. Taking into account of all the transaction costs and market information, we have the function: $F_t = F(\theta_t, F_{t-1}, I_t)$, where I_t is the information from the market and θ_t is the parameter of the trading system that we want to optimize and F_{t-1} is our position at the previous time step. Here, F can be as simple as a lienar model, or as complex as a neural network.

In this framework, the reward function we are considering U_t is a function of F_t, the positions of the trading system. Then, to find the optimal value for parameter θ that maximizes U, we can use gradient ascent (or batch gradient ascent), where the gradient is given by $\frac{dU}{d\theta} = \frac{dU}{dF}\frac{dF}{d\theta}$. Due to the recurrent nature of the problem, backpropagation in time may be used to calculate $\frac{dF_t}{d\theta}$.

Based on the framework established by Moody and Saffel in [MW97], several other research projects have been conducted. In [MS98], it was demonstrated that using recurrent reinforcement learning on S&P 500 index outperformed the buy and hold strategy. Moreover, the authors found that RRL method also outperformed Q-learning. Also, in [DL04], the authors obtained very good results on historical high frequency forex data.

Our project will mostly be based on the framework proposed by Moody and Saffel in [MW97]. However, we will be using different reward functions and we will apply the algorithm to different datasets. In the paper, we will derive the policy-update rules based on our specific reward functions. Morever, we will also explore some supervised learning techniques by combining supervised-learning-generated trading signals in our features fed to the reinforcement learning algorithm. At the end, we will study how our reinforcement learning algorithm compares with the traditional algorithms used in finance.

3 Background on Portfolio Optimization

There are three main components of the optimal portfolio problem: the market data, the trading strategies, and the performance measure. The problem is to choose a trading strategy to maximize the performance measure given the market states. The entire problem takes place on a discrete finite time horizon $t = 0, \cdots, T$. We enter the market at time 0 and exit at time T. We consider a particularly simple version of the problem where there is only one asset to invest in and the trading decision is simply to go long or short.

3.1 The Market Data

The market is given by three sequences: $X_0, \cdots, X_T$, $r_1, \cdots, r_T$, and $r_1^f, \cdots r_T^f$. $X_0, \cdots, X_T \in \mathbb{R}^D$, where $\mathbb{R}^D$ is the **information state space**. This means that X_t contains all the information about the market that we have available a time t. $r_1, \cdots, r_T \in \mathbb{R}$ are the **market returns**. This means that if you invested w dollar in the market at time $t-1$ then you would have $w \cdot r_t$ dollars at time t. $r_1^f, \cdots, r_T^f \in \mathbb{R}$ are the **risk-free returns**. This means that if you put w dollar in the bank at time $t-1$ then you would have $w \cdot r_t^f$ dollars at time t.

Basically one should imagine consecutive trading intervals $[0,1), \cdots, [T-1,T)$. At the beginning of interval $[t-1,t)$ we receive the information X_{t-1}. The returns r_t and r_t^f refer to returns that occur over this interval. Namely you invest the money at the beginning of the interval (either in the bank or the market) and you harvest the return at the end of the interval.

Another important point is that the information state vector X_t can (and should) contain information on past returns on both the market and the risk-free asset. This vector can contain *any* information we like that is available at time t. The only requirement is that the dimensionality of our information (D) is the same at all times t.

3.2 Trading Strategies

A **trading strategy** is a sequence $\pi_0, \cdots, \pi_T \in \{+1, -1\}$, where π_t is our portfolio at time t. If $\pi_t = +1$ then we are long the market and hold no position in the risk-free asset. If $\pi_t = -1$ then we are short the market and put all proceeds in the risk-free asset. Associated with each strategy π is a sequence of **strategy returns** $R_1^\pi, \cdots, R_T^\pi$ defined by

$$R_t^\pi = \Big[r_t^f + \pi_t(r_t - r_t^f)\Big] \cdot \Big[1 - \lambda|\pi_t - \pi_{t-1}|\Big] \tag{1}$$

The rough idea is that if we invested w dollars at time $t-1$ and used the trading strategy π, then we would have $w \cdot R_t^\pi$ dollars at time t. More generally, if we start with w dollars at time 0 and use the trading strategy π, then we will finish with $w \cdot \prod_{t=1}^T R_t^\pi$ at time T. Note that when considering the return on $[t-1,t)$ we also include the transaction cost incurred when moving out position from π_{t-1} to π_t. This is the reason for the second bracketed term in (1).

More precisely, the sequence of events on the time interval $[t-1,t)$ unfolds as follows. We enter the time interval $[t-1,t)$ with w dollars held in the position π_{t-1}.Next we will move our money from the position π_{t-1} to the position π_t. We will incur a transaction cost for this. Finally at the end of the trading interval we will receive a return based on π_t, r_t^f, and r_t.

We will assume that transaction constants are proportional to value of the trade, with constant of proportionality 2λ. Thus if we move w dollars from a long position to a short position (or vice versa), then we pay a transaction cost of $2\lambda w$. A good rule of thumb is $2\lambda \approx .0001$, corresponding to .01% transaction cost. Thus if we move w dollars from π_{t-1} to π_t then after transaction costs the value of our position π_t is $\Big[1 - \lambda|\pi_t - \pi_{t-1}|\Big]w$.

Finally, over the interval $[t-1,t)$, the return on our new position π_t is $\Big[r_t^f + \pi_t(r_t - r_t^f)\Big]$. Indeed if $\pi_t = 1$ then this reads r_t and if $\pi_t = -1$ then this reads $2r_t^f - r_t$. Thus if we entered the trading interval with w dollars in position π_{t-1} then we exit the trading interval with

$$\Big[r_t^f + \pi_t(r_t - r_t^f)\Big] \cdot \Big[1 - \lambda|\pi_t - \pi_{t-1}|\Big]w$$

dollars held in the position π_t. So it follows that the return on the strategy π over the interval $[t-1,t)$ is R_t^π.

3.3 The Performance Measure

We now need to define a performance measure $J(\pi)$. This measure will quantify what we mean by one trading strategy being better than another: strategy π_1 is better than strategy π_2 if and only if $J(\pi_1) \geq J(\pi_2)$. We will work with the general form of a **utility function defined over return sequences**. This means that we take a function $U : \mathbb{R}^T \to \mathbb{R}$, and for any sequence of returns $R_1, \cdots R_T$ we assign to it the utility value $U(R_1, \cdots, R_T)$. This utility value measures how happy we would be if our money grew according to the return sequence $R_1, \cdots R_T$. Given such a utility function, we define out **performance measure** as

$$J(\pi) = U(R_1^\pi, \cdots, R_T^\pi) \tag{2}$$

and the investors goal is to find the strategy that **maximizes the performance measure**.

It important to note that U is defined over the *entire sequence* of returns, and not just, for example, the total return $\prod_{t=1}^T R_t$. This is necessary because it allows us to account for the notion of *inter-temporal risk*. Indeed consider the following two sequences of returns over two days: $(.5, 4)$ vs. $(\sqrt{2}, \sqrt{2})$. In the first sequence you lose half your money on the first day and then quadruple your money on the second day, leading to a 2-day return of doubling your money. In the second sequence you multiply your money by $\sqrt{2}$ on both days, leading to a 2-day return of doubling your money. It should be clear that the second sequence is far superior to the first. For example, if you are investing on the behalf of others, they might panic after the first day's losses. This could lead them to fire you as their investor, never even allowing you to experience the quadruple profit on the second day. Distinguishing between return sequences of this kind is very important for investors, and the only way to account for it is to define the utility function over the entire sequence.

For our purposes we will work with only one utility function, the **Sharpe Ratio**:

$$\begin{aligned} Sharpe(R_1, \cdots, R_T) &= \frac{mean\big[R_1 - r_1^f, \cdots, R_T - r_T^f\big]}{std\big[R_1 - r_1^f, \cdots, R_T - r_T^f\big]} \\ &= \frac{\frac{1}{T}\sum_{t=1}^T R_t - r_t^f}{\left[\frac{1}{T}\sum_{t=1}^T (R_t - r_t^f)^2 - \left(\frac{1}{T}\sum_{t=1}^T R_t - r_t^f\right)^2\right]^{\frac{1}{2}}} \end{aligned}$$

and the performance measure for out investor is

$$J(\pi) = Sharpe(R_1^\pi, \cdots, R_T^\pi). \tag{3}$$

The intuition behind the Sharpe ratio ratio is that it rewards sequences with a high average return but it penalizes sequences with a high variance. Hence sequences with consistent good returns are preferred to those with large fluctuating profits and losses. The Sharpe ratio is one of the standard performance measures used by investment professionals.

4 MDP Formulation

The previous section outlined the structure and goals of the portfolio optimization problem. In this section we formulate the problem as a Markov Decision Process. This will allow us to attack

the problem with techniques from reinforcement learning. We need to discuss the three main components of an MDP: states, actions, and rewards.

4.1 State Space

The **state space** is $\mathcal{S} = \mathbb{R}^D \times \{+1, -1\}$ and the state at time t is (X_t, π_t). X_t is the **informations state** at time t. It represents all the economic information the investor has at time t. π_t is the **portfolio state** at time t. It is the portfolio held by the investor at time t. It is necessary to include the portfolio as part of the current state when considering transaction costs. This will be clearer below.

Note that we do not include any returns as part of the state. The risk free return r_{t+1}^f will be included in the information state X_t. The stock returns are not included in the state space whatsoever. They will only be a component of the reward function. Also note that we are implicitly assuming that the information states are generated by a stationary Markov process.

4.2 Action Space

The **action space** is $\mathcal{A} = \{+1, -1\}$ and the action at time t is a_t. The action is supposed to be the trade we make at time t, so it corresponds to deciding whether we change our portfolio or not. This is equivalent to making the action at time t the choice of portfolio at time $t + 1$. This is the formulation we use, so part of the state dynamics is that $\pi_{t+1} = a_t$. The best way to think of a_t is as the trading position held on the interval $(t, t + 1]$.

4.3 Reward Function

The reward function presents a bit of a challenge when considering the portfolio optimization problem. In reinforcement learning, the total reward over an episode is typically the sum of reward received at each instant in time. For trading, the most natural candidate for the reward received at each time would be the return of the trading strategy on the current interval. If we use this as out reinforcement learning reward, then our total reward for reinforcement learning is

$$\sum_{t=1}^{T} R_t^\pi.$$

However this is quite far from out true trading objective given by the Sharpe ratio (3).

In order to deal with this we will have to be more creative with the rewards that we give to our trading agent. In general we will work with a total reward which has the functional form

$$d(R_1^\pi, \cdots, R_T^\pi).$$

One should continue to think of R_{t+1}^π as the reward obtained by taking an action in the state at time t. And d should be thought of as some way to intermediate between summing up rewards and taking their Sharpe ratio. During optimization we adjust the policy at time t by looking at

the total reward it would have earned up to time t. This creates a recurrence which we sidestep by using a stochastic optimization algorithm. This is outlined in detail in the fifth section.

In order to think of R_{t+1}^{π} as a reward we need it to be a function of the state and action at time t. To accomplish this we assume that there exists a function $r : \mathbb{R}^D \to \mathbb{R}$ such that

$$r_{t+1} = r(X_t) + \epsilon_{t+1}$$

where $\epsilon_1, \cdots \epsilon_T$ are noise terms independent of the information process. Note that this is essentially the same assumption made when using supervised learning to predict the time series r_t using the time series X_t. In the supervised approach the function r is what we are trying to learn. However here the function r is part of our environment. We do not try to learn the function r, as our approach is model-free. Instead we learn directly how to act in an environment driven by the function r.

With this in mind we can write R_{t+1}^{π} as:

$$\begin{aligned} R_{t+1}^{\pi} &= \Big[r_{t+1}^{f} + \pi_{t+1}(r_{t+1} - r_{t+1}^{f})\Big] \cdot \Big[1 - \lambda|\pi_{t+1} - \pi_t|\Big] \\ &= \Big[r_{t+1}^{f} + a_t(r(X_t) + \epsilon_{t+1} - r_{t+1}^{f})\Big] \cdot \Big[1 - \lambda|a_t - \pi_t|\Big]. \end{aligned}$$

This shows that the return R_{t+1}^{π} is indeed a function of state and action at time t, so it can be thought of as a reward received at time t. Furthermore, this also shows why we need to include the current portfolio as part of our state: we need it to compute the return/reward from taking an action.

In our experiments we shall work with two choices for the total reward function d. The mean-variance reward:

$$d(R_1^{\pi}, \cdots, R_T^{\pi}) = \sum_{t=1}^{T} R_t^{\pi} - \gamma \sum_{t=1}^{T} {R_t^{\pi}}^2$$

and the Sharpe ratio reward:

$$d(R_1^{\pi}, \cdots, R_T^{\pi}) = Sharpe(R_1^{\pi}, \cdots, R_T^{\pi}).$$

The motivation for the mean-variance reward is that it can be thought of as proxy for the objective of maximizing the mean returns subject to the constraint that their variance is below some threshold. γ can be thought of as the Lagrange multiplier for this constraint, though we treat it as a hyper-parameter during optimization.

5 The Recurrent Reinforcement Learning (RRL) Approach

In this section, we describe in detail the Recurrent Reinforcement Learning algorithm and specifically how it can be used to solve the Markov Decision Problem as described above. Let X_t be our features at time step t, where each X_t is a D dimensional vector. π_t^{θ} is our portfolio at time step t. a is the action function and it is defined as follows: $a^{\theta} : \mathbb{R}^D \times \mathbb{R} \to [-1, 1]$, $a(X, \pi) = \tanh(\theta_0 + \theta_1 * X1 + ... + \theta_D * X_D + \theta_{D+1} * \pi)$. Here θ determines our policy and is the

parameter we try to learn in RRL and we put it in the superscript to emphasize that the actions are dependent on this parameter. a_t^θ and π_t^θ are related in the following way:

$$a_t^\theta = a(X_t, \pi_t)$$

$$\pi_{t+1}^\theta = a_t^\theta$$

After we do the action and generate the portfolio, we can calculate the return on the portfolio:

$$R_{t+1}^\theta = (r_{t+1}^f + a * (r_{t+1} - r_{t+1}^f))(1 - \lambda * (a_t^\theta - \pi_t^\theta))$$

where λ is a hyper-parameter that controls transaction cost.

5.1 Sharpe Ratio Reward

The total reward we consider here is the Sharpe Ratio reward, which is defined as follows:

$$J(\theta) = Sharpe(\{R_t^\theta\}_{t=0}^T) = \frac{mean(\{R_t^\theta - r_t^f\}_{t=0}^T)}{std(\{R_t^\theta - r_t^f\}_{t=0}^T)}.$$

We take Sharpe Ratio as the reward function here because Sharpe Ratio is a great measure of risk-adjusted return as described in the previous section and essentially that is what we are trying to maximize. The goal of our RRL algorithm is to choose a policy θ that maximizes this reward.

The technique we use to maximize $J(\theta)$ with respect to θ is the standard gradient ascent approach and the derivation is the follows:

$$J(\theta) = \frac{mean(\{R_t^\theta - r_t^f\}_{t=0}^T)}{std(\{R_t^\theta - r_t^f\}_{t=0}^T)} = \frac{U_t^\theta}{V_t^\theta - (U_t^\theta)^2},$$

where $U_t^\theta = \frac{1}{t}\sum_{s=1}^t (R_s^\theta - r_s^f)$ and $V_t^\theta = \frac{1}{t}\sum_{s=1}^t (R_s^\theta - r_s^f)^2$

Then we can compute the gradient with respect to θ as follows:

$$\nabla J_t(\theta) = \frac{\partial J_t(\theta)}{\partial U_t(\theta)}\frac{\partial U_t(\theta)}{\partial \theta} + \frac{\partial J_t(\theta)}{\partial V_t(\theta)}\frac{\partial V_t(\theta)}{\partial \theta}$$

Note that we have

$$\frac{\partial U_t(\theta)}{\partial \theta} = \sum_{s=1}^t \frac{\partial U_t(\theta)}{\partial R_s(\theta)}\frac{\partial R_s(\theta)}{\partial \theta},$$

and

$$\frac{\partial V_t(\theta)}{\partial \theta} = \sum_{s=1}^t \frac{\partial V_t(\theta)}{\partial R_s(\theta)}\frac{\partial R_s(\theta)}{\partial \theta}.$$

Then we can approximate the gradient by:

$$\nabla J_t(\theta) = \big(\frac{\partial J_t(\theta)}{\partial U_t(\theta)}\frac{\partial U_t(\theta)}{\partial R_t(\theta)} + \frac{\partial J_t(\theta)}{\partial V_t(\theta)}\frac{\partial V_t(\theta)}{\partial R_t(\theta)}\big)\frac{\partial R_t(\theta)}{\partial \theta}.$$

Note that we use proxies to compute $\frac{\partial U_t(\theta)}{\partial \theta}$ and $\frac{\partial V_t(\theta)}{\partial \theta}$ because if we use the eaxct formula for the computation, we need to go through the whole data set from $t = 0$ to the current time step, which can be very computationally expensive. Therefore, we choose to do the online proxy approach.

We can further decompose $\frac{\partial R_t(\theta)}{\partial \theta}$ using the chain rule to get the follows:

$$\frac{\partial R_t(\theta)}{\partial \theta} = \frac{\partial R_t(\theta)}{\partial a_{t-1}(\theta)} \frac{\partial a_{t-1}(\theta)}{\partial \theta} + \frac{\partial R_t(\theta)}{\partial \pi_{t-1}(\theta)} \frac{\partial \pi_{t-1}(\theta)}{\partial \theta}.$$

We now write

$$B_t = \frac{\partial R_t(\theta)}{\partial a_{t-1}(\theta)},$$
$$C_t = \frac{\partial a_{t-1}(\theta)}{\partial \theta},$$
$$D_t = \frac{\partial R_t(\theta)}{\partial \pi_{t-1}(\theta)},$$
$$E_t = \frac{\partial \pi_{t-1}(\theta)}{\partial \theta},$$

and we let

$$A_t = \frac{\partial J_t(\theta)}{\partial U_t(\theta)} \frac{\partial U_t(\theta)}{\partial R_t(\theta)} + \frac{\partial J_t(\theta)}{\partial V_t(\theta)} \frac{\partial V_t(\theta)}{\partial R_t(\theta)}$$

and compute

$$A_t = \frac{1}{t}[(V_t - U_t^2)^{-\frac{1}{2}} + U_t(U_t - R_t + r_t^f)(V_t - U_t^2)^{-\frac{3}{2}}].$$

Utilizing all the intermediate derivatives we calculate above, we get the update rule for the gradient ascent algorithm.

- Initialize to some random θ
- Initialize $\Delta_{prev} = \vec{0} \in \mathbb{R}^{D+2}$
- For $t = 0, 1, ..., T-1$
  - Use θ to compute

$$\pi_t^\theta = a^\theta(X_{t-1}, \pi_{t-1})$$
$$a_t^\theta = a^\theta(X_t, \pi_t^\theta)$$
$$R_{t+1}^\theta = [R_{t+1}^f + a_t^\theta(r_{t+1} - r_{t+1}^f)][1 - \lambda|a_t^\theta - \pi_t^\theta|]$$

  -

$$\Delta_{current} = (1 - a_t^\theta)^2 \left(\begin{bmatrix} 1 \\ X_t \\ \pi_t^\theta \end{bmatrix} + \theta_{D+1} * \Delta_{prev} \right) \tag{4}$$

  - Update

$$\theta \leftarrow \theta + \eta A_{t+1}(B_{t+1} * \Delta_{current} + D_{t+1} * \Delta_{prev}).$$

  - Set $\Delta_{current}$ to be Δ_{prev}

5.2 Other Reward Functions

If we use other reward functions other than Sharpe Ratio, the derivation for the derivatives can be conducted in a similar way and we can still use gradient ascent to update the policy. Specifically, one other reward function that we considered is the $g(R_t) = R_t - \gamma R_t^2$. We consider this reward function because we want to maximize return but also want to keep the variance of returns low, which is the reason why the $-\gamma R_t^2$ regularization term comes in. Then, the total reward becomes $J(\theta) = \sum_{t=0}^{T} g^{\theta}(R_t)$ Using proxy for the gradient, we have: $\nabla J_t(\theta) = \nabla g_t(R_t^{\theta}) = g'(R_t^{\theta})\big(\frac{\partial R_t(\theta)}{\partial a_{t-1}(\theta)}\frac{\partial a_{t-1}(\theta)}{\partial \theta} + \frac{\partial R_t(\theta)}{\partial \pi_{t-1}(\theta)}\frac{\partial \pi_{t-1}(\theta)}{\partial \theta}\big)$.

We now write

$$A_t = g'(R_t^{\theta}),$$
$$B_t = \frac{\partial R_t(\theta)}{\partial a_{t-1}(\theta)},$$
$$C_t = \frac{\partial a_{t-1}(\theta)}{\partial \theta},$$
$$D_t = \frac{\partial R_t(\theta)}{\partial \pi_{t-1}(\theta)},$$
$$E_t = \frac{\partial \pi_{t-1}(\theta)}{\partial \theta}.$$

Following a similar derivation, we have:

$$A_t = 1 - \gamma R_{t-1}$$

$$B_t = (r_t - r_t^f)[1 - \lambda|a_{t-1}^{\theta} - \pi_{t-1}^{\theta}|] + [r_t^f + a_{t-1}^{\theta}(r_t - r_{t-1}^f)][-\lambda sign(a_{t-1}^{\theta} - \pi_{t-1}^{\theta})]$$
$$D = [r_t^f + a_{t-1}^{\theta}(r_t - r_t^f)][\lambda sign(a_{t-1}^{\theta} - \pi_{t-1}].$$

For the calculations of C and D, we will be using proxies from previous iterations in the loop, which are exactly the $\Delta_{current}$ and Δ_{prev} as described in the previous parts of the paper.

6 Experiments on the S&P 500 Index

We have attempted to create a trading strategy using multiple variations on the algorithm and reward functions. We will discuss three primary experiments:

1. Online learning: plain reward vs. Sharpe reward
2. Gradient Boosting (Batch) vs. Sharpe reward (Online)
3. Sanity check: using tomorrow's return as a feature in the online strategy

6.1 Data

Our data sources are Bloomberg and CRSP, with 8 features from Bloomberg: CL1 Comdty, XAU BGN Curncy, DXY Curncy, TY1 Comdty, CESIUSD Index, USYC2Y10 Index, CDX IG CDSI GEN 5Y Corp, CDX HY CDSI GEN 5Y SPRD Corp. The S&P index price data is from CRSP. Our data frequency is daily and spans 2006-01-04 to 2017-12-29 for a total of 3022 trading days.

6.2 Cross Validation

It is worth noting that due to the nature of time series data, we can not randomly shuffle data points to do cross validation for hyper-parameter tuning. Instead, we adopt an approach named rolling fit. First, we take the first 500 data points as the training set and we further split into a group of first 250 points and a group of the last 250 points. We train the model on the first group, and then use grid search to find the combination of hyper-parameters that produce the best results on the second group. Then, use this combination of hyper-parameters, we train the model on the whole 500 points. After having trained a relatively good model this way, we test for the next 10 data points (so the 500-510 points). Then we roll the 500 data points training window by 10 data points forward and do the training and validation again, after which we test on the next 10 points. Then, we continue this process until we go to the end of the time series.

6.3 Online learning variations

6.3.1 Plain Reward

Our first attempt was to create a trading strategy for the S&P 500 using the plain reward (section 5.2) of

$$g = R_t - \gamma R_t^2, \tag{5}$$

where R_t is defined as above in section 5.1, r_f is the risk-free rate of return, approximately 1.00012 (daily yield of treasury bond), $a_t \in [-1, 1]$ is the action taken by the policy, π_t is the portfolio size, and $\lambda = 0.0001$ is the transaction cost penalty.

Our training window was 500 trading days with the first 400 days for training θ and the last 100 days for validation. On day 1, we assumed the portfolio size was +1 (long) and initialized θ to a uniform random $\mathbb{R}^{8+2}$ vector between -0.001 and +0.001. We ran the training window for all combinations of η and γ in our hyperparameter inventory, and picked out the combination of η and γ resulting in the best Sharpe ratio over the training window to use in our test window of 10 days. We generated a set of 10 actions in the test window according to the policy with θ generated from the full training window, then moved the window by 10 days, repeating the train-validate-test process until we predicted a trade for the last day in our dataset.

6.3.2 Sharpe Reward

Instead of our plain reward, we also tried to implement a Sharpe reward to compare reward function performance differences. Our window for training was 500 days, and we did not need to search for hyperparameters so there was no validation period. We also found that using a rolling test window of 100 days instead of 10 days improved performance. We initialized $\theta \in \mathbb{R}^{8+2}$ to all 0.1.

6.3.3 Results

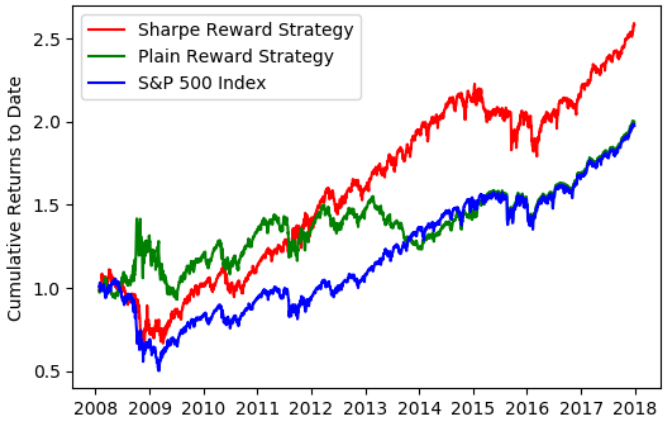


Figure 1: Cumulative returns to date for the online learning variants using Sharpe reward and plain reward. A multiplier of 1 is set at 2008-01-28, the first date in the series, since 500 trading days were discarded for training. The annualized Sharpe ratio during this time period for the Sharpe reward, plain reward, and buy-and-hold strategies were 0.422, 0.295, and 0.290, respectively.

Figure 2: Portfolio sizes for both the Sharpe reward algorithm and the plain reward algorithm. +1 corresponds to buying, -1 corresponds to shorting. Note that portfolio size cannot be anything in between, since we take the sign of a_t when computing the action.

6.4 Batch vs Online learning

6.4.1 Gradient Boost Batch Strategy

Following past experience with financial predictions and the JPM paper [KK17], we attempted to use XGBoost and scikit-learn's Gradient Boosting Regressor on the 8 features that we had, in order to predict the next day's return. For these regressors, we tried default parameters with a moving window of 252 training days + 20 testing days, without any validation window (we don't pick hyperparameters, since we don't in the Sharpe reward online strategy). Scikit-learn's regressor performed much better at default parameters, so we took the sign of the prediction as the action (buy or sell) and calculated the returns of the strategy. This batch learning method assumes that trading is a trajectory-independent process: it ignores the transaction cost and all past portfolios. Whether this faulty assumption creates a penalty is of great interest because trajectory dependent policies are a key motivation for using reinforcement learning in a trading system.

6.4.2 Comparison to the Sharpe Reward Online Strategy

We first added the batch learning predictions into the set of 8 features (for a total of 9 features) to use in the Sharpe reward strategy, but noticed that the performance was significantly worse. We suspect that the signal from this batch-learned feature may be confusing our simple policy during gradient ascent (it must now balance between the 9th feature and the original 8, which produced a decent signal already), so we took out the feature during comparison. In addition, the batch-learned feature isn't stellar when it comes to predictions, though it does add some diversity.

6.4.3 Results

Figure 3: Cumulative returns using the batch learning trading strategy, compared to the Sharpe reward online learning strategy. The batch learning strategy has a Sharpe of 0.152, significantly worse than the online strategy (0.422), and cannot outperform the S&P 500 (0.290). A noteworthy time period for the batch learning method is when it somewhat successfully avoids a crash in 2009, when the online learning strategies could not. Overall, it is still not that great.

6.5 Sanity check using "cheated" data

To check whether our online learning would work in an optimal case, we produced the forward returns "cheated data" of the S&P 500 index and took them as a 9th feature in our Sharpe reward strategy. In theory, this strategy should significantly outperform anything else, since all it has to do is learn that the feature corresponding to tomorrow's return is extremely important. If this strategy does not provide major benefits, then something is wrong with the algorithm or the model. We observed a performance improvement when we initialized the θ corresponding to the future data to a larger-than-0.1 value.

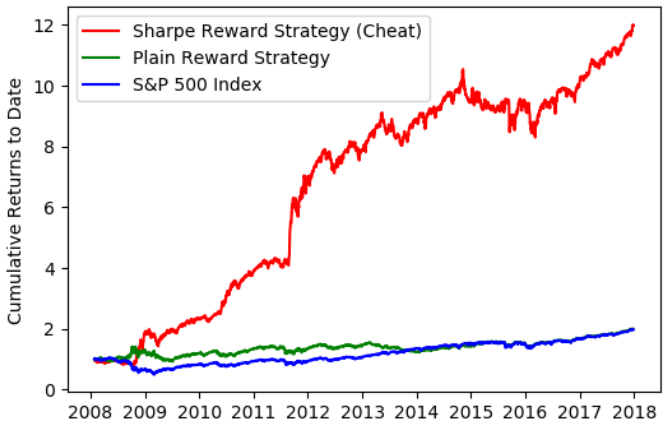


Figure 4: Cumulative returns using cheated data and an initial $\theta_{fr} = 1$, with fr standing for "future returns". Initializing to a larger value results in significantly larger performance improvements.

It is worth noting that there are a total of 11 θ's to update with 10 of them as "noise" when we have tomorrow's return, so the gradient ascent may be unable to pick out that the θ_{fr} corresponding to the future return is the most important predictor of improving reward.

References

[DL04] MAH Dempster and V Leemans. An automated fx trading system using adaptive reinforcement learning. *Research Paper in Management Studies*, 2004.

[KK17] Marko Kolanovic and Rajesh Krishnamachari. Big data and ai strategies machine learning and alternative data approach to investing. *J.P. Morgan*, May 2017.

[MS98] John Moody and Matthew Safell. Reinforcement learning for trading systems and portfolios. *Decision Technologies for Computational Finance*, 1998.

[MW97] John Moody and Lizhong Wu. Optimization of trading systems and portfolios. *CIFEr*, 1997.