Final report: Dynamic Asset Allocation Using Reinforcement Learning

Enguerrand Horel, Rahul Sarkar, Victor Storchan

December 16, 2016

1 Introduction and objective

The role of portfolio managers is to optimize the allocation of capital across various financial assets such as stocks, bonds or funds to maximize a chosen performance metric like expected returns or risk-adjusted returns.

This task is traditionally performed in two steps as described in [4]. First, it is necessary to predict the expected returns, and the covariance matrix of the returns of the assets considered. Then a quadratic programming problem can be solved to either minimize the risk of the portfolio for a constrained return or maximize its return for a given risk.

Finding an optimal way of allocating assets in a portfolio in order to outperform the market is still a very active research area in finance as recently reviewed in [8].

Reinforcement learning, as a global procedure to maximize cumulative reward from a random underlying environment, fits perfectly with the problem of asset allocation [6], [5]. Hence we need to model our problem as a Markov Decision process (MDP). Finding the optimal policy from this model can then be solved using model based reinforcement learning [3] or Q-learning [9], [7].

2 Models

2.1 Outline

We consider the traditional capital allocation problem that is based on a portfolio of two assets: a risk-free asset (fixed-income security like a treasury bond) and a risky asset (such as an individual stock). We define the portfolio by two weights w_B and w_S that represent the proportion of the portfolio invested in the bond and in the stock respectively. We further assume that all the portfolio's money is invested which is formulated as: $w_B + w_S = 1$.

In this framework, our objective is to dynamically determine the optimal w_S that would optimize the overall return of the portfolio.

In the MDP framework, we consider states s_t for $1 \leq t \leq T$, T representing our final investment date, that combines both the stock $r_S(t)$ and bond $r_B(t)$ returns. We have decided to consider two different models that gradually make our problem representation more complex but also more realistic. We consider one simple model with few discrete state and action spaces and a more advance model with a larger number of states and actions.

We describe more formally these two different models in the next subsections.

2.2 First model

- States: $s_t = \tilde{r}_S(t)$. The stock returns are discretized into positive and negative returns, i.e., $\forall t$, $\tilde{r}_S(t) = \begin{cases} -1 & r_S(t) \le 0 \\ 1 & 0 < r_S(t) \end{cases}$. Since the return of the bond is constant over time, it doesn't bring much information and this is why it is not included into the state.

- Actions: a_t is the weight w_S the investor assign to the stock. They are discretized such that $a_t \in \{0, 1\}$. Hence, in this model, the capital is fully allocated in the stock or fully invested into the bond.

- Rewards: R_t defined as the instant return: $R_t = a_t * r_S(t+1) + (1 - a_t) * r_B(t+1)$.

- IsEnd$(s_t) = \mathbf{1}_{t=T}$, investments are done until the final investment date.

- Discount factor: $\gamma = 1$

2.3 Second model

Since we believe from a financial point of view that it is the evolution of the returns over several days that will contain the most information to predict future returns, we decided to introduce a hyper parameter *mem* which encodes the amount of relevant past information to include into the current state.

- States: $s_t = (\tilde{r}_S(t - mem), ..., \tilde{r}_S(t-1), \tilde{r}_S(t))$. The states are discretized into three buckets based on the two terciles $q_{33\%}$ and $q_{66\%}$ that divide the stock return values into three equal parts, i.e. $\forall t$, $\tilde{r}_S(t) = \begin{cases} -1 & r_S(t) \le q_{33\%} \\ 0 & q_{33\%} < r_S(t) \le q_{66\%} \\ 1 & q_{66\%} < r_S(t) \end{cases}$

- Actions: $a_t = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$

- Rewards: same as in first model

- IsEnd(s_t): same as in first model

- Discount factor: same as in first model

Although the transition probabilities are unknown, we assume they satisfy the following relation:

$$T(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1}|s_t, a_t) = \mathbb{P}(s_{t+1}|s_t)$$

which means that the investor's action has no impact on the market.

2.4 Baseline and Oracle

2.4.1 Baseline

The baseline chosen here is simply the bond performance that represents a 2% annual return. The policy of the baseline is to keep investing the whole portfolio on the bond at each time period. The goal of this project is then to find a policy that would beat the risk-free asset performance by including a risky asset with higher returns.

2.4.2 Oracle

Our oracle makes use of future returns data, one week ahead. It is computed as follow: for each week, we compute the average return of the stock and the bond over the week. The oracle allocates 100% of the portfolio on the better performing asset during that week.

3 Data

The data used in this project are the daily returns of a bond and a stock. We decided to use a virtual 2% annual return bond and the Walmart stock. Walmart has been chosen among other stocks because it satisfies the following criteria: it is a very volatile stock that contrasts well with the risk-free bond, it does not follow any upward or downward trend over the period considered and its final cumulative return is roughly the same as the bond. Hence, a strategy that can outperform the baseline would really be able to benefit from the extra return of Walmart during the upward periods.

The returns are calculated from the daily closing price downloaded from Google Finance. The training period spans from 2000 to 2008 included and the testing period spans from 2009 to November 2016.

The cumulative returns of the baseline and Walmart stock over the training and testing period can be seen in Figure 1.

4 Algorithms

4.1 First model implementation

Let's recall that our input data are daily returns of Apple stock and Dow Jones index from 2010 to 2016 that are stored in arrays r_a and r_m of size T. For the simple model we implement an online model based reinforcement learning algorithm. As explained before, we only have four states : {(1,1), (1,-1), (-1,1), (-1,-1)}, and two actions {0,1}. The model based reinforcement learning idea is that given a state s_t and time t, we define the best action as:

$$\bar{a}_t = \max_a \hat{R}(s_t, a) + \sum_{s'} T(s_t, a, s')\hat{V}_{opt}(s') = \max_a \hat{R}(s_t, a) + \sum_{s'} \mathbb{P}(s'|s_t)\hat{V}_{opt}(s') = \max_a \hat{R}(s_t, a) \tag{1}$$

In the above equation, $\hat{R}(s_t, a)$ is an estimate of the average reward obtained when action a is taken from the state s_t. $\hat{V}_{opt}(s')$ is an estimate of the expected reward of the state s',

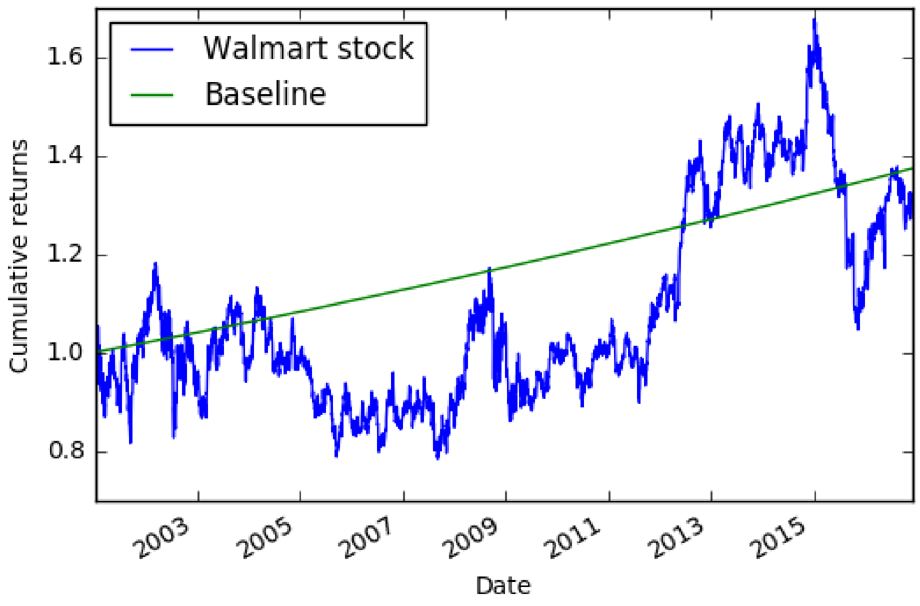

Figure 1: Cumulative returns of the bond and Walmart stock throughout the training and testing period

under the optimal policy. But due to the assumption that $\mathbb{P}(s_{t+1}|s_t, a_t) = \mathbb{P}(s_{t+1}|s_t)$, we get the final simplification.

The algorithm is implemented by maintaining two global lists $N(s, a)$ and $\rho(s, a)$, for all possible pairs of states s and actions a. At any time t, $N(s, a)$ stores the count of the number of times the action a was taken from the state s, while $\rho(s, a)$ stores the cumulative sum of the previous rewards obtained every time the action a was taken from the state s. Hence, at time t, for any state action pair (s_t, a), we estimate the average reward as $\hat{R}(s_t, a) = \rho(s_t, a)/N(s_t, a)$. Finally, the best action $\bar{a}_t$ can now be computed using equation (1). We adopt an ϵ-greedy strategy to get the final action a_t to take at time t, where $0 \leq \epsilon \leq 1$. In case of the simple model, we choose a fixed value of $\epsilon = 0.001$. We generate a random number q using an uniform distribution in $[0, 1]$. If $q \geq \epsilon$, we set $a_t = \bar{a}_t$, otherwise we choose $a_t \in \{0, 1\}$ randomly.

The final step is to update the global lists $N(s, a)$ and $\rho(s, a)$. $N(s, a)$ is updated as $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$, and $\rho(s, a)$ is updated as $\rho(s_t, a_t) \leftarrow \rho(s_t, a_t) + r_t$. The algorithm then proceeds to the next time $t + 1$ and the process is repeated.

4.2 Second model implementation

The second model implementation attempts to refine the first model based learning. This refinement is done by considering a state space that takes into account the history of the stock and that classify the returns in a larger number of classes. The three classes are defined by the two terciles of the return values. More precisely, the 81 states considered are $(\tilde{r}_S(t-3), \tilde{r}_S(t-2), \tilde{r}_S(t-1), \tilde{r}_S(t))$, where each $\tilde{r}_S(t) \in \{-1, 0, 1\}$ and the 6 actions are: $0, 0.2, 0.4, 0.6, 0.8, 1$. As one can see, a memory length of 4 days is used, i.e. the algorithm learns at date t using the returns at times t, $t-1$, $t-2$, $t-3$.

Dealing with **Q-learning** implementation, we used the Watkins and Dayan algorithm. On each (s_t,a,r,s_{t+1}):

$$\hat{Q}_{opt}(s_t, a) = (1-\eta)\hat{Q}_{opt}(s_t, a) + \eta(r + \gamma \hat{V}_{opt}(s_{t+1}))$$

where

$$\hat{V}_{opt}(s_{t+1}) = \max_{a \in Actions(s_{t+1})} \hat{Q}_{opt}(s_{t+1}, a)$$

Additionally, an **ϵ-greedy policy** is used for the sake of exploration. $\epsilon = 0.4$ during training and $\epsilon = 0.01$ during testing as we expect to have a quasi-deterministic policy at testing.

The hyperparameters ϵ and mem, the memory length have been set doing a grid-search on a validation set.

5 Results and discussions

Figure 2 and Figure 3 show the cumulative returns of method 1 and method 2 respectively. The annualized returns, volatility and Sharpe ratio are display in Table 1. The Sharpe ratio is defined as: $SR = \dfrac{R - R_f}{\sqrt{var(R)}}$, where R is the annualized return of the strategy considered and R_f is the risk-free annualized return.

	Annualized returns	**Annualized volatility**	**Sharpe ratio**
Baseline	2 %		
Walmart stock	4.8 %	17 %	16 %
First method	8.3 %	13 %	48 %
Second method	5.2 %	9.4 %	34 %
Oracle	15.8 %	4.4 %	313 %

Table 1: Annualized returns, volatility and Sharpe ratio of the baseline, the oracle, Walmart stock and the two models

It can be seen that our two models achieve to capture local trend of stocks and market to learn an efficient investment strategy. It is interesting to notice that the two methods achieve different kind of investment strategies:

- the model based learning method outperforms significantly both the baseline and the stocks in terms of cumulative returns but with a rather high volatility,

- in comparison, the Q-Learning algorithm is very efficient at decreasing the volatility of the portfolio while keeping satisfying returns. This is certainly due to a larger action space that allows more flexibility in the investment strategy.

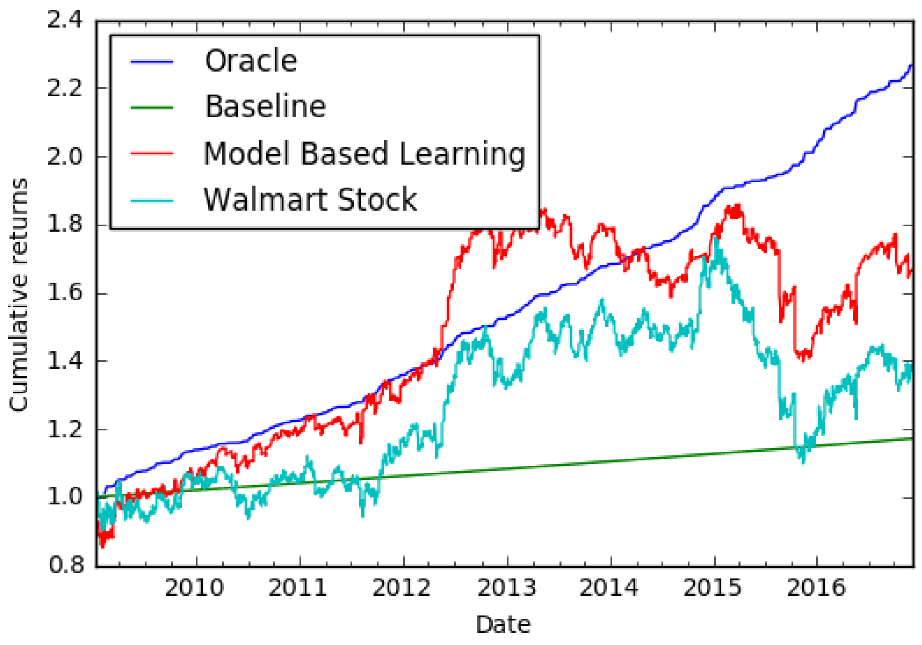

Figure 2: Comparison of cumulative returns of the baseline, the oracle, Walmart stock and the first model

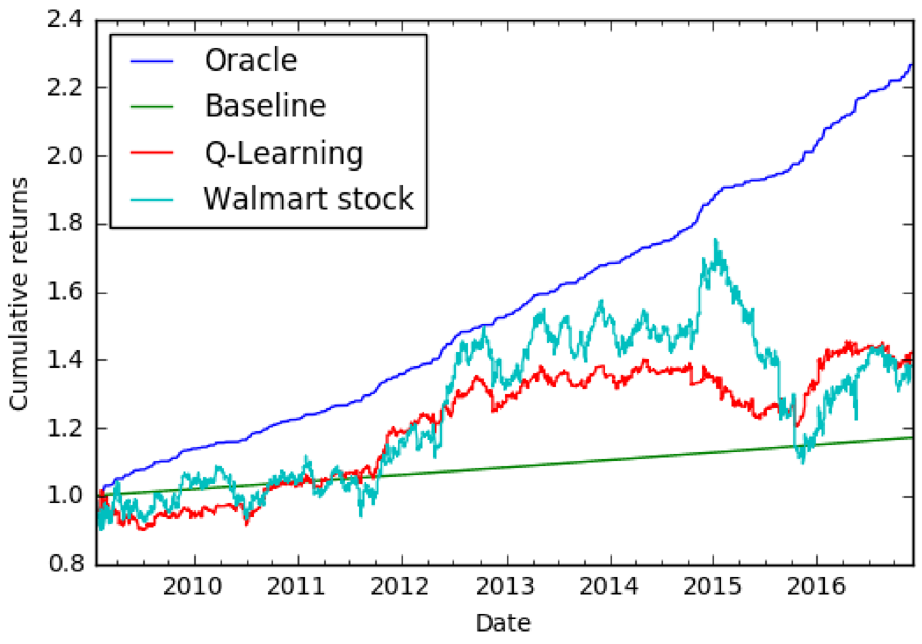

Figure 3: Comparison of cumulative returns of the baseline, the oracle, Walmart stock and the second model

It is worth mentioning that even though both methods use anϵ-greedy policy during the training phase, they do not present the same variance in their results. The model based learning is very consistent across different simulations but the Q-learning results show much more variance between runs.

We would like to add that we also implemented a Q-learning algorithm with function approximation. Indeed, it is clear that a continuous state space is a better model for continuous return values. We considered a linear function and a neural network for the approximation. However, in both cases, the weights corresponding to the model parameters failed to converge. Because of that, no satisfying results could be obtained. We experimented across a large space of learning rate parameters to control the evolution of the parameters but in all cases the weights diverged. This may be due to a lack of normalization of the states.

6 Next steps

Here are some further research directions that could be interesting to consider as next steps:

- One assumption we made is that the investor has the possibility of reallocating weights across assets without any trading fees. To make it more realistic, we would like to take into account trading fees. Because of that we can expect a decrease of the performance and fewer reallocation over time.
- We would like to think more about the trade-off between the exploration of the states and the exploitation of a deterministic investment strategy. Indeed, in this project, we decided to first train the model with a large exploration rate and then we exploit it with almost no randomness. However, an online-learning strategy could be more relevant to allow the investment strategy to be more flexible to react to market changes.

References

[1] David Barber. *Bayesian reasoning and machine learning.* Cambridge University Press, 2012.

[2] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

[3] D Kuvayev and Richard S Sutton. Model-based reinforcement learning. Technical report, Citeseer, 1997.

[4] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[5] John Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks*, 12(4):875–889, 2001.

[6] John Moody, Lizhong Wu, Yuansong Liao, and Matthew Saffell. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(56):441–470, 1998.

[7] Ralph Neuneier et al. Enhancing q-learning for optimal asset allocation. In *NIPS*, pages 936–942, 1997.

[8] Jessica Wachter. Asset allocation. Technical report, National Bureau of Economic Research, 2010.

[9] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.