Portfolio Management using Reinforcement Learning

Olivier Jin
Stanford University
ojin@stanford.edu

Hamza El-Saawy
Stanford University
helsaawy@stanford.edu

Abstract

In this project, we use deep Q-learning to train a neural network to manage a stock portfolio of two stocks. In most cases the neural networks performed on par with benchmarks, although some models did significantly better according in terms of raw returns.

1. Introduction

Accurate stock market predictions can lead to lucrative results, which is no wonder why investors are turning toward machine learning applications to analyze financial markets. However, one of the inherent difficulties with this approach is producing an accurate model of the current market and predicting future stock behaviors. In fact, one school of thought argues that, given the efficient market hypothesis (EMH), it is impossible for any agent to truly 'beat the market' by exceeding benchmark predictions.

We attempt to evaluate this challenge by utilizing artificial neural networks (ANN), due to their ability to model nonlinear relationships between variables, as well as their lower need for formal statistical training. In addition, we use Q-learning since it is a model-free algorithm relying only on Q-factors without attempting to model the environment (which, in the case of the stock market, would be entirely unfeasible). Q-learning provides the added benefit of balancing between 'exploration' and 'exploitation' in order to provide the most optimal outcome.

The input to our system is a portfolio containing one high-volatility stock and one low-volatility stock. Since most stock portfolios consist of any combination of high-volatility and low-volatility stocks, these two-stock portfolios would represent a reduced model of an actual portfolio. We then feed the input portfolio to our neural network to produce a recommended action: either buying more low-volatility stocks and selling more high-volatility stocks, or vice versa. Our states and action space will be discussed more in-depth in a later section. Finally, we compared results to two benchmarks to evaluate results.

2. Related Work

The use of neural networks to manage stock portfolios is not a novel concept, although we were unable to find any related works which also used Q-learning as a training method. Indeed, backpropagation was by far the dominant method as evidenced by Zimmermann et al[22] and Costantino et al[9], given its simplicity, efficiency, and compatability with stochastic gradient descent.

However, beyond any similarities in training methods, each paper adopts a different approach when constructing the algorithm. For example, Fernandez et al adopt the Markowitz mean-variance model when selecting their portfolio[10], whereas Toulson et al pursue an orthogonal approach by using neural network 'ensembles', essentially multiple independently-trained neural networks which work together to estimate future returns and risks[20]. Each approach had its strengths and weaknesses, and we carefully considered each aspect when deciding upon our own algorithm.

Since the results produced by Fernandez did not result in noticeable improvements over preset benchmarks, we decided against adopting this heavily-mathematical model. On the other hand, while the technique proposed by Toulson produced returns which were greater than or equal to the FTSE-100, we felt that implementing and training multiple neural networks would exceed the timeframe of this course.

As a result, our approach was most similar to the methods adopted by Franke and Klein[11]. Although they populate their portfolio using currencies rather than stocks, we follow their experimental approach while implementing our own methods. For example, we use tau to model weight delay to prevent overfitting, and rely on the Sharpe return ratio to calculate risk premiums. We felt that this approach provided a good overall balance between performance and complexity.

3. Dataset and Features

We trained our neural network using historical stock data gathered from Google Finance's API using the Python li-

brary `Pandas.DataReader` to automatically download the stock histories [6]. Our data was the daily closing price of 20 stocks ranging from July 2001 to July 2016. Notably, the data include the 2008 stock market crash in order to train our model on real world data fluctuations.

Stock riskiness was quantified using the 'beta' index, or a security's tendency to respond to market swings. Beta > 1 indicates a stock is more volatile than the market; whereas less-volatile stocks have a beta < 1 [1]. We chose ten stocks from S&P 500's high-beta index fund [7], and ten low-beta stocks from two online editorial recommendations [21, 8]. (See Figure 1 for stock histories; beta values in parentheses [2]).

From our stock choices, we generated all 100 possible combinations of low and high beta stocks, and trained our model on 80 randomly chosen combinations (using `sklearn.model_selection.train_test_split`), while leaving the remaining 20 for testing.

No datat pre-processing was performed, although some pricing data did require date alignment (since some stocks did not have prices listed for all the days the stock markets were open). The `Pandas` library allowed us to join price histories using the date as an index and drop any days where stocks were missing values.

4. Methods

We used Deep Q-learning when training our ANN. In this section, we will discuss further details of the algorithm including the design constraints, reward functions, and performance benchmarks.

4.1. Design Environment

Rather than starting from scratch, we used the Python library `Keras` to build and train our models. The `Keras` library builds on top of either `Theano` or `TensorFlow`, which are mathematics libraries for efficient multi-dimensional calculations [4]. We chose `Theano` because it was easier to install on a Windows machine [3].

Besides Keras, `Pandas` was used to efficiently store our stock data and the resulting portfolios as `DataFrames`, which greatly simplified saving, comparing, and plotting the data.

4.2. Design Constraints

Our initial design constraint was to use reinforcement learning to build an agent that controls a portfolio of only two stocks, with one stock being significantly more volatile than the other. For each state, our neural networks received the stock histories for both stocks over a set number of days (either 2, 7, or 30), the number of shared owned in each stock, the total portfolio value, and the left-over cash (usually less than the cost of the cheapest of the two stocks at time t). Given the need to track portfolio history to calculate reward (Section 4.3) and compare performance (Section 5), portfolios were modeled as a `pandas.DataFrame`, with each row, indexed at time t, containing the cost of the two stocks, the number of shares owned in each, the total value, and the left over cash.

Instead of a continuous action space, where the agent chooses what percentage of the portfolio each stock should constitute (*e.g.* stock A should constitute 35% of the portfolio's total value), the agent was given 7 actions: $a_t \in [-0.25, -0.1, 0.05, 0, 0.05, 0.1, 0.25]$. For each action, a_t, the portfolio sells $a_t \times \text{total}_t$ of the low-beta stock and buys the corresponding amount of the high-beta stock (and vice-versa for $a_t < 0$). This discrete action-space, alongside the simplified state-space, helps make the problem tractable.

In addition, a small transaction cost per transaction ($\$0.001$) was used to encapsulate the various trading fees [5]. Finally, to avoid issues when stock prices where too large to allow an action to achieve its desired result (*e.g.* a stock costs 10% of the portfolio value, so selling 5% is impossible), all portfolios and benchmarks started with $\$1,000,000$ initial cash.

4.3. Deep Q-Learning Algorithm

Previous work used a neural network to trade between T-bills and the S&P-500 stock index, or currency markets, choose actions using softmax (and a time-dependent Boltzman temperature), and gradually increased the discount factor γ [15, 16, 12]. Furthermore, they compared the performance of two difference reward functions: the current portfolio return, $R_t = v_t - v_{t-1}$, where v_t is the portfolio's current value at time; and the Sharpe Ratio: $S_T = \text{mean}(R_t)/\text{std}(R_t);\ \forall t \in [1, T]$ [15].

Building on their work, we also trained neural networks to approximate the Q value of portfolio states. However, we modified the portfolio return reward to include a penalty for volatility: $P_T = R_T - \lambda\ \text{std}(R_t);\ \forall t \in [1, T]$. Furthermore, we based our system off of more recent architectures, such as the AlphaGo architecture [17]. Namely, we use an ϵ-greedy exploration strategy, where the agent chooses a random action with probability $1 - \epsilon$. However, since the state was a tuple with 8, 18, or 64 (if the state contained 2, 7, or 30 days of stock history, respectively, see Section 4.2) and neither highly dimensional nor very large (like an image or Go Board would be), we used simple fully-connected, feed-forward layers instead of convolution and pooling layers. To remove the correlation between successive samples, we used an experience replay, where, for each iteration, the network approximates the Q-values for a randomly-selected minibatch (of size 8) of portfolios, the maximizing actions (with probability $1 - \epsilon$) are taken, the reward is observed, and the network is trained on the desired output $Q(s, a) = r_t + \gamma \max_{a'} Q(s', a')$ for using

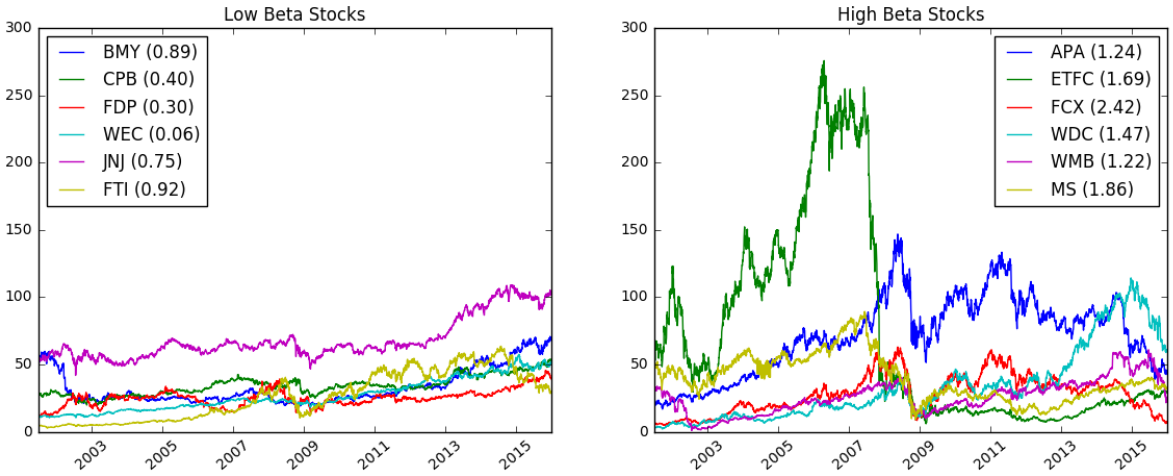

Figure 1. Stock histories for the low and high volatility stocks. Beta values, an indicator of volatility, are in parentheses.

only those 8 Q values [18, 14].

However, $Q(s', a')$ was approximated with a target network instead of the network currently being trained. After updating the main network's weights (θ_M) with stochastic gradient descent, the target network's weights (θ_T) are updated gradually with $\theta_T = (1 - \tau)\theta_T + \tau\theta_M$; $\tau \ll 1$ [18, 13].

4.4. Benchmarks

The model's test data performance was compared against two benchmarks: The first, the do-nothing benchmark, allocates half of its starting value to each stock and then does nothing. This benchmark acted as a very crude approximation of the market since it represents the raw performance of the two stocks.

The second, the rebalance benchmark, reevaluates its holdings every so many market days (30 in our simulations) and buys or sells stock to ensure the total portfolio value is split 50-50 between the two stocks. It is important to note that it maintains a proportion of stock values, not stock shares.

5. Results and Discussion

The performance of our models varied greatly across multiple reward functions and history lengths.[1] (See Figure 2 and Figures 3–9.) All models trained using a penalized reward with $\lambda = 0.5$ — and not the Sharpe ratio reward — consistently had the highest average Sharpe ratio. Moreover, except for the two models trained using 7 days of input and either penalized reward with $\lambda = 0.5$ or the Sharpe ratio, the models displayed much less variance is portfolio value. Finally, the models trained with 30 days of data had more variance than the other models. This was especially apparent for the model trained with 30 days of input and the Sharpe ratio reward, which had one of the lowest Sharpe ratios of all the models.

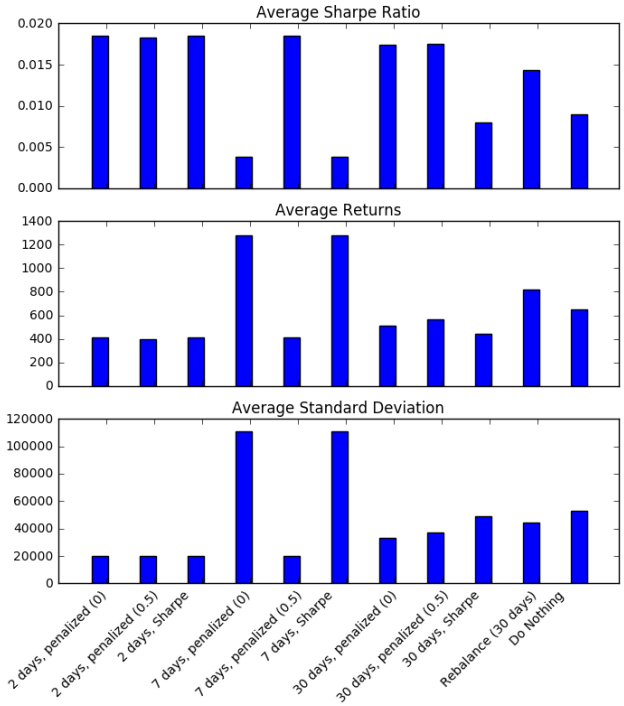

Figure 2. Model performance across trained models and benchmarks).

In terms of returns, the two benchmarks outperformed all but the two previously mentioned models. However, the cost of higher returns was greater variance in the portfolio's value, which could make them more volatile. Figure 5 shows a particularly bad portfolio result.

In our investigations, we found that evaluating portfolio

[1] Our results differ greatly from the poster session because of an off-by-one bug in our performance evaluation code.

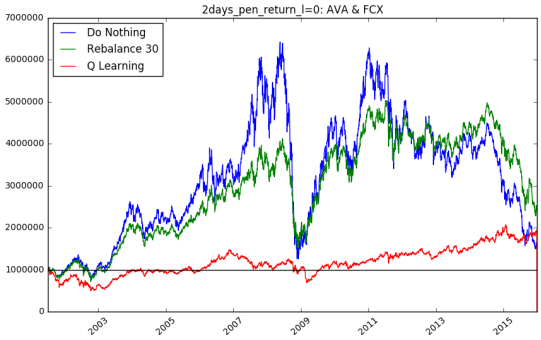

Figure 3. Model performance on the stocks AVA and FCK, using two days of data and a penalized reward ($\lambda = 0$).

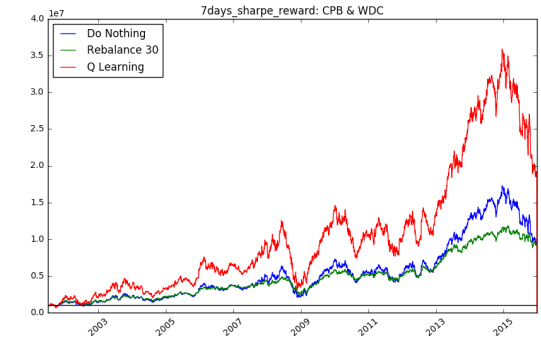

Figure 4. Model performance on stocks CPB and WDC, using 7 days of data and the Sharpe ratio reward.

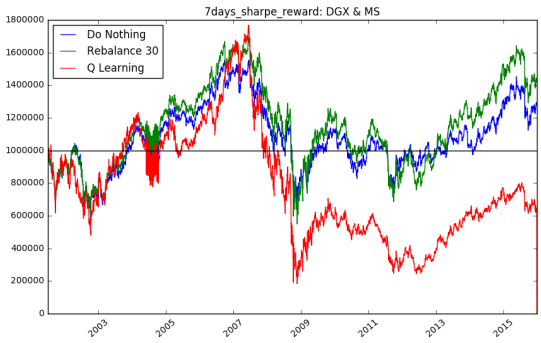

Figure 5. Model performance on stocks DGX and MS, using 7 days of data and the Sharpe ratio reward.

performance is more complicated than simply looking at the metrics used in Figure 2. Figure 3 shows an example where the model's portfolio was significantly less volatile, but still ended at approxiamtly the same value as the two benchmarks. Moreover, in Figures 3 and 7, the do nothing benchmark results in higher net portfolio value over time, but as Figure 2 shows, the rebalance benchmark consistently displayed higher returns and less variance.

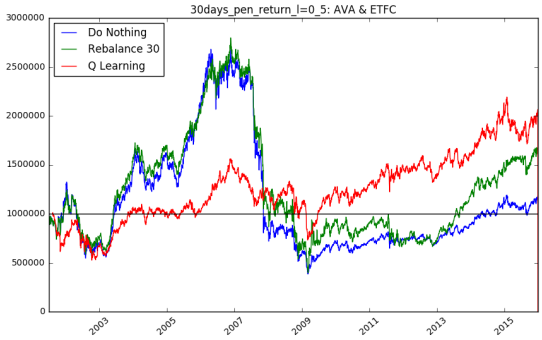

Figure 6. Model performance on stocks AVA and ETFC, using 30 days of data and a penalized reward ($\lambda = 0$).

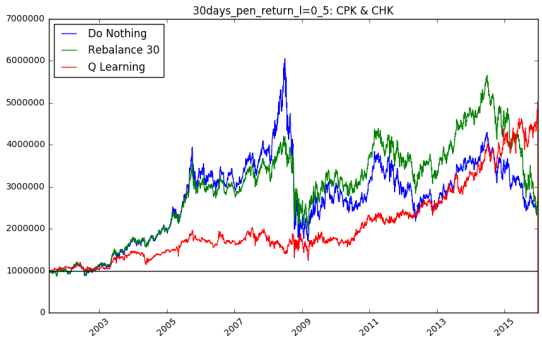

Figure 7. Model performance on stocks CPK and CHK, using 30 days of data and a penalized reward ($\lambda = 0.5$).

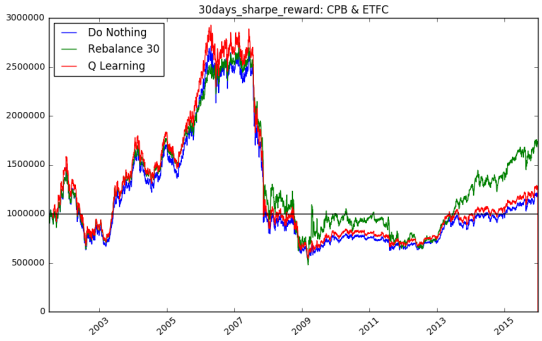

Figure 8. Model performance on stocks CPB and ETFC, using 30 days of data and the Sharpe ratio reward.

Overall, our models had a much higher Sharpe ratio and significantly less variance (standard deviation) than the benchmarks. While these results are significant, they are average values and not consistent across portfolios. However, the models' results demonstrate that training a neural network to manage a portfolio is feasible, and the network does not resort to taking random actions.

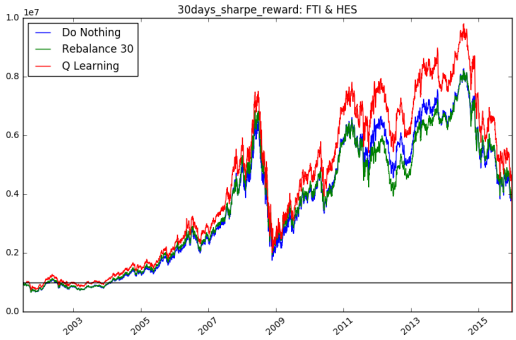

Figure 9. Model performance on stocks FTI and HES, using 30 days of data and the Sharpe ratio reward.

5.1. Future Work

Our work was successful as a proof of concept, and future work could result in stronger and more consistent model performance, possibly on par with modern actively-managed funds. Specifically, our efforts focused on prototyping models with different state spaces and reward functions, but we were unable to explore the effect of different hyperparameters on model training and performance. We chose four hidden layers with 100 neurons per layer as our model architecture, with the reasoning that it would be small enough to train quickly yet robust enough to adequately approximate the Q-function However, it is likely that this architecture was not flexible enough, and that convolution layers tailored to looking at differences between successive stock prices could perform significantly better.

Furthermore, we chose $\epsilon = 0.15$ for our ϵ-greedy exploration strategy using the values from previous works[14, 13]. However, other papers had ϵ values which were half of ours[19], and it is likely that too much exploration may have interfered with the training processes. This is compounded by the fact that our action space is significantly smaller than those in the works which we based our values on.

Another avenue could be investigating the effect of using the weekly average stock price, or some other preprocessing technique to reduce the resolution and therefore variance in stock prices. A downside is that, by preprocessing input data, we run the risk of losing any sense of real market behavior. For example, if our sample interval is too long, we lose the ability to accurately predict future behavior by making too many 'coarse' assumptions. Nevertheless, data pre-processing would be a valuable tool when training the initial behavior of the ANN.

Finally, our states only relied on historical stock data as well as total value and various other auxiliary parameters. This is a rather simplified assumption, since the stock market would behave rather independently of past performance. Indeed, actual stocks would rely more on the economy and the companies themselves, which we could capture by parsing through headlines or qualitative economic forecasts. By making our states more complex, we could potentially increase the accuracy of simulating the stock market environment.

6. Conclusion

In this project, we utilized ANNs to manage a two-stock portfolio with the goal of maximizing returns while minimizing risk. By investigating various reward functions and hyperparameters, we successfully implemented an algorithm which performed on-par if not better than preset performance benchmarks, according to the different metrics. If given more time, we would like to increase the complexity of our model while fine-tuning our hyperparameters to further optimize performance.

References

[1] Beta Index. Accessed: 2016-12-14.

[2] Google Finance. Accessed: 2016-12-14.

[3] How to install Theano on Anaconda Python 2.7 x64 on Windows? Accessed: 2016-12-14.

[4] Keras: Deep Learning Library for Theano and TensorFlow. Accessed: 2016-12-14.

[5] NYSE Trading Fees. Accessed: 2016-12-14.

[6] pandas-datareader. Accessed: 2016-12-14.

[7] S&P 500 High Beta Index Fund. Accessed: 2016-12-14.

[8] S. Bajaj. Add These Low-Beta Names to Your Portfolio to Escape Market Volatility, Jan 2016. Accessed: 2016-12-14.

[9] F. Costantino, G. D. Gravio, and F. Nonino. Project selection in project portfolio management: An artificial neural network model based on critical success factors. *International Journal of Project Management*, 33(8):1744 – 1754, 2015.

[10] A. Fernndez and S. Gmez. Portfolio selection using neural networks. *Computers and Operations Research*, 34(4):1177 – 1191, 2007.

[11] J. Franke and M. Klein. Optimal portfolio management using neural networks - a case study. 1999.

[12] X. Gao and L. Chan. An Algorithm for Trading and Portfolio Management using Q-Learning and Sharpe Ratio Maximization. *Proceedings of the International Conference on Neural Information Processing*, 2000.

[13] B. Lau. Using Keras and Deep Deterministic Policy Gradient to play TORCS, Oct 2106. Accessed: 2016-12-14.

[14] B. Lau. Using Keras and Deep Q-Network to Play Flappy-Bird, Jul 2106. Accessed: 2016-12-14.

[15] J. Moody and M. Saffell. Reinforcement Learning for Trading Systems and Portfolios. *Advances in Computational Management Science*, 2:129–140, 1998.

[16] J. Moody and M. Saffell. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks*, 12(4):875–889, 2001.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the Game of Go with Deep Neural Networks and Tree Search.

[18] Silver, David. Deep Reinforcement Learning. Accessed: 2016-12-14.

[19] M. Tokic. Adaptive epsilon-greedy exploration in reinforcement learning based on value differences. In *Proceedings of the 33rd Annual German Conference on Advances in Artificial Intelligence*, KI'10, pages 203–210, Berlin, Heidelberg, 2010. Springer-Verlag.

[20] S. Toulson. Use of neural network ensembles for portfolio selection and risk management, 1996.

[21] Zacks Equity Research. 5 Low Beta Stocks to Withstand Market Volatility, July 2016. Accessed: 2016-12-14.

[22] H. G. Zimmermann, R. Neuneier, and R. Grothmann. Active portfolio-management based on error correction neural networks. In *in: Advances in Neural Information Processing Systems (NIPS 2001*, page forthcoming., 2001.