EXPLORING DEEP REINFORCEMENT LEARNING FOR PRACTICAL FINANCIAL PORTFOLIO OPTIMIZATION

Bryan Brzycki
bbrzycki@berkeley.edu

ABSTRACT

We explore deep reinforcement learning (DRL) as a strategy for equity trading and portfolio optimization. Much of the machine learning work in the literature on equity trading is performed under very idealistic assumptions and rarely for the noisy retail trading environment. In reality, whether it is for a high profile firm or for a retail trader, there are complications to an idealistic trading study, such as transaction or brokerage fees and delays between transactions (resulting in slippage, in which the price of an equity changes during the execution of a trade). These effects can add up, especially for retail traders, depending on their preferred trading platform.

We use value-based deep reinforcement learning methods such as DQN and DDQN to train a portfolio optimization trading policy that attempts to these real world effects into account. In particular, we build off of the methods in Li et al. 2019, which used DRL to develop trading strategies on single equities at a time with promising results.

We create custom environments based on OpenAI Gym[1], using historical daily Open-High-Close-Low (OHCL) data to construct observations. We use a discrete action space, representing the possible number of shares held by the trading agent. We specifically create environments which trade only either one or two equities. In our experiments, we focus on \$SPY and \$GLD as ETFs that represent broader market sentiment and are not directly correlated in general, so as to create a complex interplay in the two equity environments.

We use two main neural network models to estimate Q functions; the first just uses fully connected (FC) layers, while the second uses an LSTM module before passing outputs to FC layers. To evaluate trained models, we divide our historical data into train and test splits, and compare DQN/DDQN performance on test data with buy-and-hold strategies for the appropriate ETF.

The vast majority of experiments perform worse than buy-and-holding the maximum allowable number of shares (which we hold at 3, to limit the action space for this work). Even the LSTM-based models do not consistently perform better than the FC models, even though they are explicitly intended to learn temporal features and trends.

We also find that extending the number of timesteps of data seen in agent observation do not necessarily correspond in an increase in performance on test data. Our results demonstrate the difficulty in deriving effective active portfolio trading strategies, especially when using historical daily quotes. Nevertheless, we observe that agents are capable of learning complex trading strategies, especially in our custom two equity RL environment, albeit not necessarily stable. They are also capable of learning buy-and-hold strategies themselves, either right from the beginning or as episodes progress.

There are many further possible directions, but in this work, we lay out the groundwork for future investigations using custom environments and slightly different approaches towards the decision-making problem (e.g. using actor-critic and continuous action spaces).

1 Introduction

Algorithmic trading is very important for both institutional and retail investors alike. For high frequency trad-

[1] https://gym.openai.com/

ing and large institutional investing, it becomes vital to use algorithms to place trades with appropriate risk levels and other specific parameters. For retail investors, as data mining and wrangling tools become more and more mainstream, getting into automated trading is likewise more accessible.

The stock market is notoriously difficult to predict. Much of the literature are conducted in idealistic environments and sometimes with more data than is usually available to most people (particularly real-time data). Especially for the retail investing environment and for new traders, restrictions a la brokerage fees and pattern day trading rules just further the divide between trading in reality and using theoretical trading strategies. A potential answer is to use historical financial data with coarse enough resolution that real-time data *is* attainable, e.g. daily quotes, to derive a portfolio optimization algorithm to balance one's portfolio.

The strengths of this approach is that one may be able to take advantage of more than one equity, rather than using algorithms to attempt predicted each one individually and in turn combing those results somehow. Ideally, a single agent learning potential dependencies and trends both within a single equity's stock performance and between multiple equities might be able to learn to divide capital between different equities, rebalancing daily, so as to beat the market in the long term.

Deep reinforcement learning (DRL) is a very appealing strategy for learning this behavior. Reinforcement learning in general focuses on learning how to make the best decisions in an environment, given a set of possible actions. Portfolio optimization and rebalancing is a complex decision making problem – what is the best way to divide funds among a set of potential investments? Training a neural network-based agent to find the dependencies and patterns in the data, and allowing it to play through in a virtual environment created from historical data, could serve as an algorithmic approach to financial decision making.

An important work that our method draws from is Li et al. 2019, which attempted to tackle algorithmic trading under practical conditions. The DRL architectures used are DQN (Deep Q Network) and A3C (Advantage Actor-Critic), and the reward function used takes into account transaction costs and slippage rate (which are different for each equity tracked). However, the work limited its study to training trading strategies on a single equity at a time (as seen in Table 2 & 3 in Li et al. 2019), though it did a comprehensive comparison. Nevertheless, they obtained reasonable results and returns after training on 1-minute granularity data from multiple equities. In this work, we investigate how well similar DRL methods perform if we only use 1-day granularity data. We also seek to build on their work by extending it to trade with two equities at the same time, instead of just one.

Other related works include Jiang et al. 2017, which used policy gradients for portfolio management using cryptocurrencies, and Liang et al. 2018, which compares a few RL algorithms for portfolio management using Chinese securities. Both have interesting ideas regarding a policy gradient approach towards trading with DRL agents, but the discussion on rewards vs. costs is more fleshed out in Li et al. 2019.

In this work, we describe the custom trading framework and environments created for learning practical retail trading. We explore a set of experiments focusing on two ETFs: $SPY, which tracks the S&P 500, and $GLD, which tracks the value of gold. These ETFs are not precisely positively correlated, but both represent a unique market sentiment. This makes them interesting components of a single portfolio for a DRL trading agent. We present results on test data, compared to a simple buy-and-hold strategy, and discuss future steps for practical financial portfolio optimization.

2 Deep RL Framework

This section details the framework we create to support a Deep RL trading agent with historical daily data.

2.1 Historical Data

The dataset we have consists of historical daily Open-High-Close-Low (OHCL) data for over 700 different equities spanning a few decades, depending on the equity. For each day, we also have the trade volume, for a total of 5 features per day. For instance, for $AAPL, the historical data ranges from September 7, 1984 to November 10, 2017, while for $SPY, the range is from February 25, 2005 to November 10, 2017. Corporate changes to stock price, such as dividends and stock splits, have been accounted for by making appropriate adjustments to historical prices.

Daily changes in equity price are affected by all kinds of factors, from company news to manipulation by large investors to overarching movements in the stock market. These effects make it very tough to come up with reliable swing trading strategies for retail traders. For example, it is harder to find patterns in price movements from day to day than it is to find patterns in minute to minute data over many days. That being said, it is also harder to get access to such high quality high time resolution equity data, for which institutional investors pay a premium to obtain. Unfortunately, we do not have access to this level of historical data, but on the upside, it automatically frames the DRL problem as relevant and practical to retail investors.

2.2 Environments

We design custom RL environments based on historical data. For this work, we focus on two types of environments: trading with a single equity and with two equities. For both of these, we can vary different parameters, such as the maximum position, whether or not shorting stocks is allowed (i.e. a negative position to be bought back later), and whether transaction costs are included.

We set some default environment properties to facilitate comparison between experiments. We start with a balance of $10,000 in all environments. We also set a maximum position of 3 in any given equity for the sake of comparison and for limiting the action space to a reasonably amount for exploration.

2.2.1 Observations

Our observations are modeled off of the historical data we have. Each episode begins at a random day in the training dataset. We define the "lookback time" n_{lb} as the number of days past the current day that we include in our observational space. For each date, we have the 5 features: Open, High, Close, Low, Volume. Including the current timestep, this can be represented as a $5 \times (n_{lb} + 1)$ array for a single equity environment. For a two equity environment, we include all the same values for both equities (e.g. 10 values per day rather than just 5). We also include a few auxiliary values that describe our positions, such as the remaining monetary balance, the number of shares held, and the cost basis for the shares held.

One of the dangers in applying ML algorithms to stock data is that stock prices do not have maximums, so neural network input normalization can be tricky. We choose to normalize prices and volume by large enough values that realistically won't be obtained anytime soon, if ever, but this is unfortunately quite heuristic. Another option is to normalize by the maximum price obtained over the last $n_{lb} + 1$ days, but the problem with this is that there are really 4 prices per day that are important, and normalizing horizontally could wash out dependencies between Open, Close, High, and Low values for the same day.

2.2.2 Action Spaces

Usually, actions in RL formulation correspond directly to specific operations. For example, these could be to buy or sell x stocks. But as mentioned in Li et al. 2019, if we introduce the ability to short stocks, then the action space needs to account for additional complexity in the decision making. We instead choose to take a range of possible positions as our action space, as $\{0, \ldots, x\}$ for a maximum position of holding x shares (and ranging down to $-x$ for shorting).

We set a maximum position of 3 in all cases, so that if no shorting is allowed, the action space is $\{0, 1, 2, 3\}$ of allowable positions. With shorting enabled, the action space becomes $\{-3, -2, -1, 0, 1, 2, 3\}$. We choose 3 to keep the action space relatively small for our experiments. When trading with two equities without shorting, the action space goes up to $4 \times 4 = 16$ possible aggregate positions.

2.2.3 Taking Actions

After the agent sees the observation and chooses an action, we step the environment in the following manner. From the historical data, we have Open, High, Close, Low data, so we sample a price uniformly between the low and high equity prices. Since the selected action tells us our position in the equity directly, the environment calculates and saves changes in net worth, available balance, and other relevant quantities. By sampling prices between these known historical peaks and troughs, we essentially offer some regularization into the learning, so our models are less likely to overfit.

2.2.4 Rewards

For our reward function, we choose

$$r_t = \Delta b - c = (b_t - b_{t-1}) - c, \qquad (1)$$

where b_t is the net worth at time t, c is the transaction cost for purchasing and selling equities (most brokerages charge a flat fee per trade, not per share). When we ignore transaction costs, we have $c = 0$. According to this article[2], the average transaction cost is about $c = 9$.

2.3 DRL Algorithms and Networks

Since our action space in these environments is discrete, we chose to use the DQN and DDQN algorithms. We use these value-based methods with a policy that tries to select actions that maximize the expected change in net worth over each episode.

Our baseline Q function models use two 64-node fully connected (FC) layers connected to an output dense layer with a dense network to predict Q values for each action. This architecture performs well on standard OpenAI Gym tasks, so it is a good starting point.

We also implement an LSTM-based Q function model to try incorporating time series relationships into our agent. We incorporate an LSTM with 128 hidden units, applied to the OHLC time series with $n_{lb} + 1$ timesteps, and the outputs from each step are fed into two 64-node fully connected layers, which are in turn connected to a linear output layer.

During training, we set an "episode length" for each experiment, which defines how many timesteps each training episode goes for. Intuitively, the shorter the episode length, the less temporal information is processed by the agent, so that longer term trends should be better captured with higher episode lengths.

[2]`https://www.valuepenguin.com/average-cost-online-brokerage-trading`

3 Experiments

3.1 Training and Evaluation Approach

For many conventional RL tasks, doing well in the training phase is exactly the end goal. However, to be useful in many real world tasks, the learned agent must be scalable and have some generality. This is the case for our trading agents; we have to train on historical data, but we need to maintain good performance on unseen data to continue generating profit.

For single equity environments, we divide historical data into a 90-10 split for training and testing. For two equity environments, since there may be a mismatch in how much historical data we have for the two, we first isolate all the dates that are shared (this amount to only going back to the oldest date shared by the two, since all of the data goes to November 10, 2017), then split into 90-10.

While we sample potential prices uniformly from historical data during training, for the test set, we limit ourselves to the close prices, so that there is a single point of comparison across different experiments.

We also compare the results of DQN performance to a buy and hold strategy, assuming the maximum possible position of 3 shares. For the two equity case, we compare to holding the maximum position in each individually, and in having 3 shares of each at the same time. This is very relevant to retail trading, where the conventional wisdom is that “time in the market is better than timing the market” – just having and holding stock works out in the long term better than consistently trying to make well-timed trades. After all, if an algorithmic approach cannot beat buy and hold strategies reliably, they are not too useful.

3.2 Choice in equities

In this work, we choose to focus on two ETFs, $SPY and $GLD. $SPY tracks the performance of the overall stock market, and $GLD tracks the performance of gold. These track larger movements in the market and are not generally correlated closely, so it provides an interesting choice for optimizing a portfolio that contains the two. At test times, $SPY mostly goes up, while $GLD is much more variable.

Figures 1 and 2 show the prices of each ETF over our dataset, highlighting the train-test split.

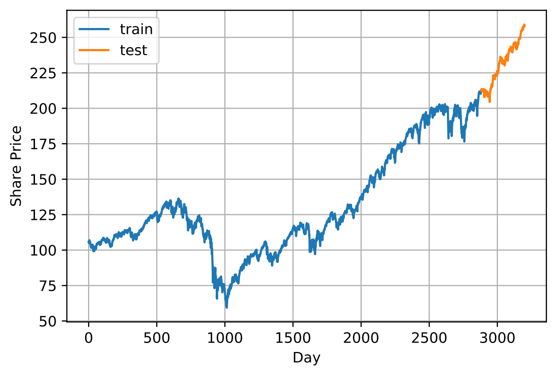

Figure 1: $SPY price over available historical data.

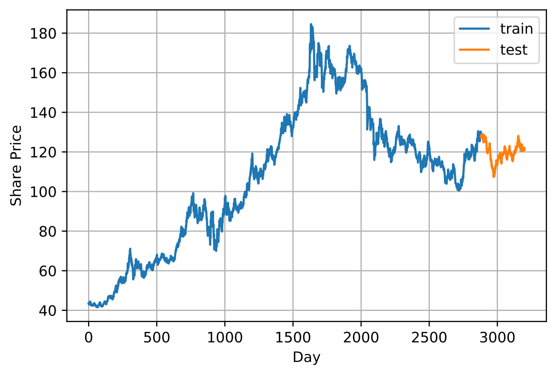

Figure 2: $GLD price over available historical data.

3.3 One equity experiments

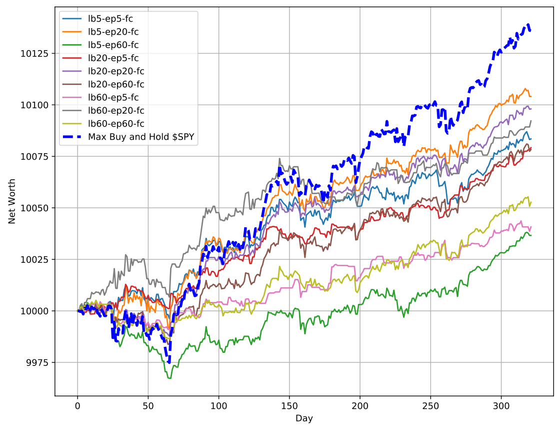

Figure 3: DQN performance on $SPY test data using the single equity FC model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

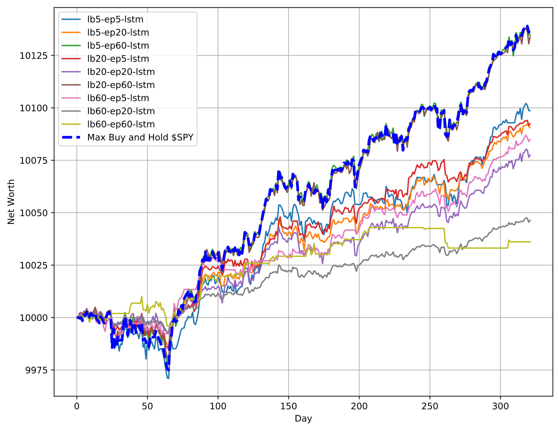

Figure 4: DQN performance on $SPY test data using the single equity LSTM model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

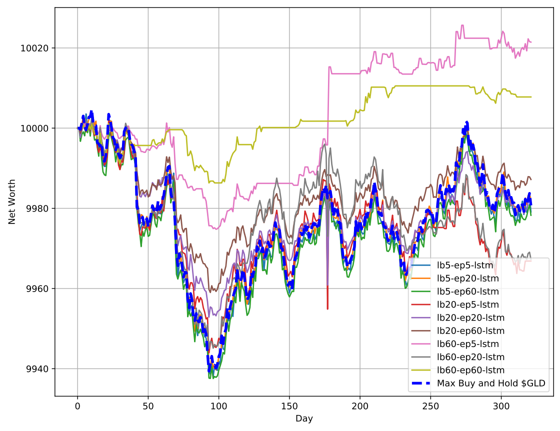

Figure 6: DQN performance on $GLD test data using the single equity LSTM model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

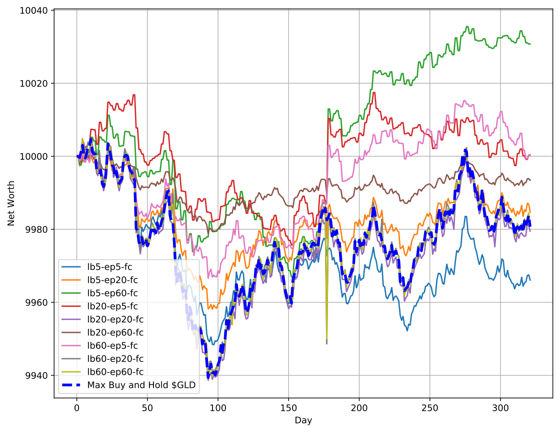

Figure 5: DQN performance on $GLD test data using the single equity FC model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

Figures 3-6 show the DQN performance for each ETF using the FC and LSTM models over a range of lookback times and episode lengths. The bold, dotted lines show the result of buying and holding 3 shares, for comparison.

For $SPY, we notice that the overall trend in the test data is positive. For both models, most of the experiments do not perform as well as the buy and hold strategy. At least in the beginning, one of the FC strategies performs better on the initial downturn in the equity's price, but falls off after around 150 days. On the other hand, more of the LSTM strategies at least match the buy and hold strategy, but most do not perform as well.

On the other hand, for $GLD, most of the strategies actually tend to perform better than a maximum position. Note that this is in large part because the buy and hold position is largely in the rest in the test data. A few strategies actually manage to turn a profit for both the FC and LSTM models. Unfortunately, the learned strategies that perform the best on the test data do not quite match in lookback time and episode length for training. Nevertheless, it shows that DQN can learn strategies that can protect against large dips in equity price (to varying effectiveness, of course).

3.3.1 DDQN

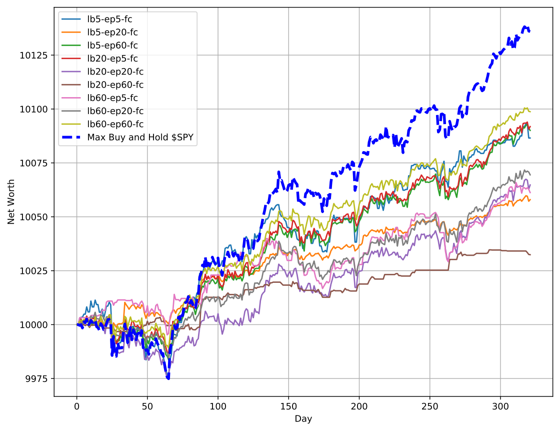

Figure 7: DDQN performance on $SPY test data using the single equity FC model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

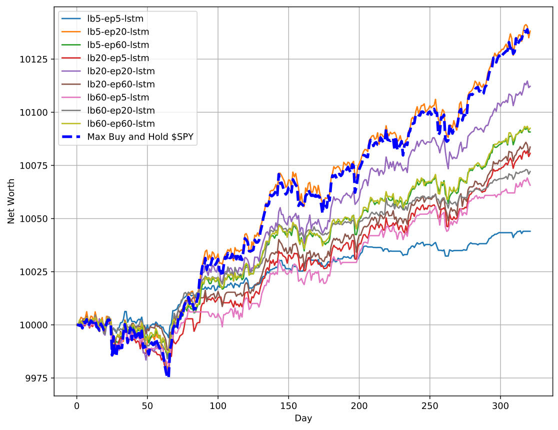

Figure 8: DDQN performance on $SPY test data using the single equity LSTM model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

We use DDQN instead of vanilla DQN in Figures 7-8. Besides a few of the LSTM runs performing better, we do not observe particularly higher returns compared to the maximum buy and hold strategy. The distribution of returns are quite similar, however, which is expected since DDQN typically achieves higher stability than DQN.

3.3.2 Including shorting shares

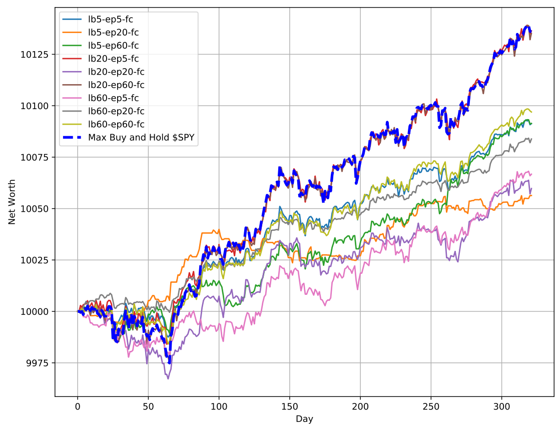

Figure 9: DQN performance on $SPY test data using the single equity FC model for the Q function, with shorting equities allowed. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

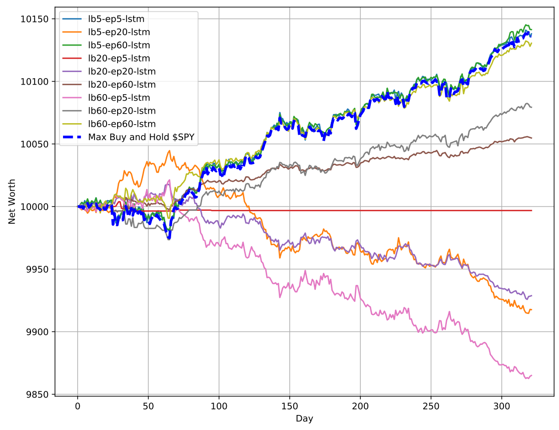

Figure 10: DQN performance on $SPY test data using the single equity LSTM model for the Q function, with shorting equities allowed. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

Here, we enable shorting, so that positions can range from -3 to 3. Comparing Figure 9 with Figure 3 shows that the learned strategies are much closer together in performance, but nevertheless do not beat buy and hold in the long term.

The situation is even worse for the LSTM runs (Figure 10), which somehow "learn" to hold a negative position even while the share price keeps rising. This is particularly interesting, since one of the runs (`lb5-ep20`) actually starts out beating buy and hold, but deteriorates over time.

3.3.3 Scaling reward by last share price

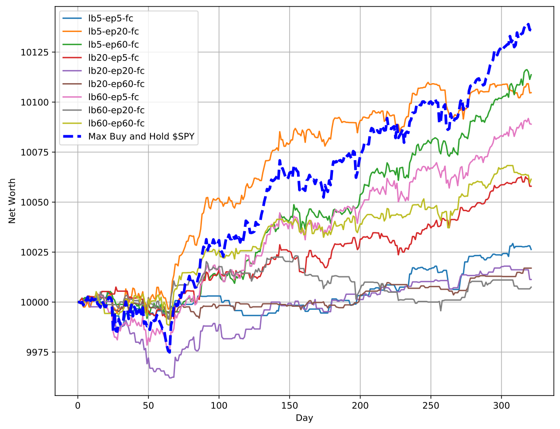

Figure 11: DQN performance on $SPY test data using the single equity FC model for the Q function, using a reward scaled by the last share price $r_t \rightarrow r_t/p_{t-1}$. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

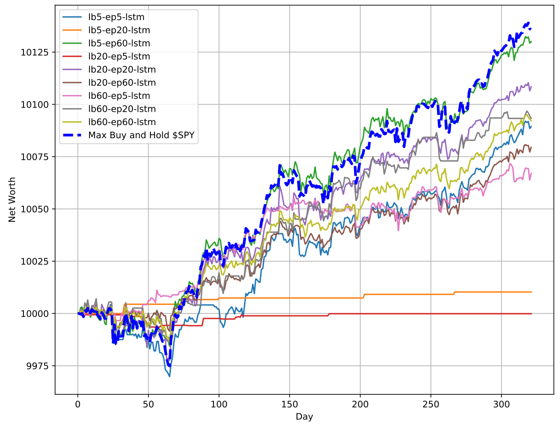

Figure 12: DQN performance on $SPY test data using the single equity LSTM model for the Q function, using a reward scaled by the last share price $r_t \rightarrow r_t/p_{t-1}$. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares are shown in the bold, dotted curve.

Since episodes are calculated over a period of years, share prices can varying by a few factors between episodes. To test whether this is a limitation in training our models, we adjust our reward function by scaling by the last share price: $r_t \rightarrow r_t/p_{t-1}$. Since we are limited by a maximum share position of 3, this tries to standardize performance at each time regardless of the actual share price.

The effect this has is also complex. For the FC models (Figure 11), one of the runs actually beats buy and hold appreciably for a majority of the test run. However, other runs are much less effective than when they used the default reward function. For the LSTM models (Figure 12), most of the runs are grouped towards the buy and hold strategy, and a couple also show poor performance.

3.4 Two equity experiments

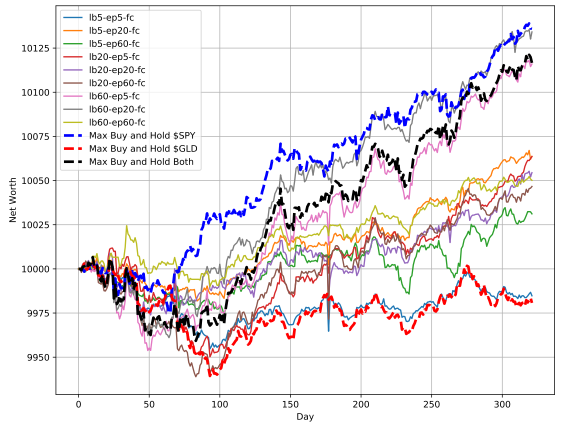

Figure 13: DQN performance on $SPY and $GLD test data using the two equity FC model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares for each ETF individually and together (i.e. 3 shares of both) are shown in the bold, dotted curves.

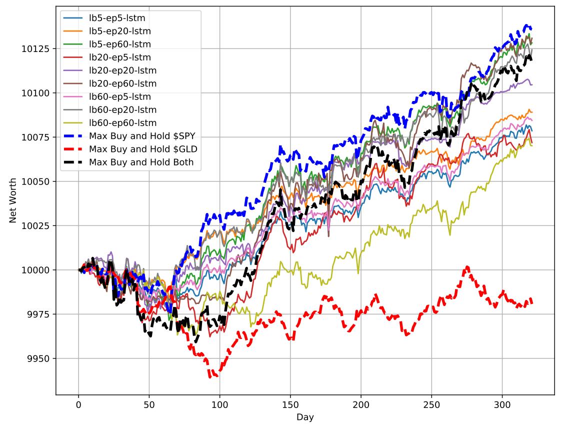

Figure 14: DQN performance on $SPY test data using the two equity LSTM model for the Q function. Curves are shown for all combinations of lookback times in (5, 20, 60) and episode lengths in (5, 20, 60). Results from a buy and hold strategy of 3 shares for each ETF individually and together (i.e. 3 shares of both) are shown in the bold, dotted curves.

Figures 13-14 show the results of experiments trading two equities simultaneously. We also show the buy and hold strategies for each ETF independently and together.

For FC runs, most perform relatively poorly, falling between the buy and holding $GLD and both simultaneously. On the other hand, most of the LSTM runs perform better, avoiding being over leveraged in $GLD and focusing on $SPY for most of the test run.

We observe that we do not just obtain simple linear combinations of buy and hold strategies in each equity; the agents actually make trades during the course of the test run. So, the behavior we see is actually some form of decision-making on which stocks are a better investment at that point, which is very encouraging.

4 Other Parameter Space Exploration and Failures

Initially, we normalized observational quantities like prices and volume by much too large of a constant factor, and the agents would "learn" to just default to a buy and hold strategy, of 1-3 stocks.

We noticed that the longer the lookback time, the more iterations it took to update our models. Even after allowing the models to train for many more iterations, we did not observe an increase in performance over a lookback time or episode length of 60. For example, we tried going up to 240 (near the number of trading days in a year). However, considering the performance of each run selected lookback time and episode length from (5, 20, 60), performance seems quite variable and does not directly vary as expected, and varies differently for different equities.

In experimentation with environment with transaction costs, much of the time the agent decided never to buy a share in the first place. This was particularly true in using $9 as the cost, and basically just lowered as we continued to lower the transaction cost to $0. This is probably such a concern since $9 is a large amount compared to the daily returns from holding at most 3 shares. With a larger action space, including a transaction cost would likely result in more nuanced differences in trading strategy. But with the trend of more brokerages adopting a no commission policy for stock trades, transaction costs are no longer *necessarily* a bottleneck for algorithmic trading.

5 Summary and Future Work

We found that the vast majority of experiments perform worse than buy-and-holding the maximum allowable number of shares (which we hold at 3, to limit the action space for this work). Even the LSTM-based models did not consistently perform better than the FC models, even though they are explicitly intended to learn temporal features and trends. We also find that extending the number of timesteps of data seen in agent observation do not necessarily correspond in an increase in performance on test data.

Our results demonstrate the difficulty in deriving effective active portfolio trading strategies, especially when using historical daily quotes. Nevertheless, we observe that agents are capable of learning complex trading strategies, especially in our custom two equity RL environment, albeit not necessarily stable. They are also capable of learning buy-and-hold strategies themselves, either right from the beginning or as episodes progress.

There are many interesting future steps for this work. The most straightforward is more parameter space exploration and experimentation with Q function models.

Making more general environments that can scale to n equities at once is an important extension. Even moving to account for 2 equities introduces a lot of complexity to DRL action selection (though the action space can easily blow up as a result).

An exciting idea that we would like to explore in the future is to use a continuous action space instead of a discrete one. There is actually a convenient way to define the portions of a portfolio in each equity that lends itself to softmax activations on neural network outputs, particularly by summing to 1. In the one equity case, this would be a tuple of the fraction of the current portfolio in case and the fraction of the portfolio invested in the equity – summing to 1. It is then relatively simple to scale this up to n equities, with an continuous action space of dimension $n + 1$. This formulation readily lends itself to actor-critic methods, which would predict the best fractional allocation for each asset. These fractions could then be converted to the appropriate integer positions in each asset (since fractional shares are only supported for a handful of brokerages, which likely do not allow for retail algorithmic trading). This method would enforce more of a trade-off in investing in one equity over another, an important aspect of picking stocks as a retail investor. Accounting for extensions like shorting stocks adds a bit more complexity, but the trade off seems to be well worth the constrained action space.

Our work shows that it is quite difficult to learn a reliable swing trading strategy using daily OHLC data alone. Over the experiments we conducted, even the LSTM method was not noticeably more effective than simply using fully connected layers for defining our Q functions.

Using higher resolution financial data could certainly improve our results, and perhaps re-contextualize the problem for a retail trader. Any DRL trading strategy should take multiple equities into account, since that is where DRL can really shine over other ML techniques. However, to be most practical, historical data used to train a DRL strategy must be identical to data that can be obtained real-time by retail traders, whether by a free or paid service. Daily quotes are the simplest form of accessible financial information, so making effective use of them is of particular interest.

References

[1] Jiang, Z., Xu, D., Liang, J. A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem. *arXiv:1706.10059*, 2017.

[2] Li, Y., Zheng, W., Zheng, Z. Deep Robust Reinforcement Learning for Practical Algorithmic Trading. In *IEEE Access*, 2019.

[3] Liang, Z., Chen, H., Zhu, J., Jiang, K., Li, Y. Adversarial Deep Reinforcement Learning in Portfolio Management. *arXiv:1808.09940*, 2018.