Deep Q Networks in Algorithmic Trading

Omkar Kharkar, Hyunsu Kim, Alexandrea Ramnarine, Joshua Salit

School of Professional Studies, Northwestern University

MSDS 464: Intelligent Systems and Robotics

Professor Shreenidhi Bharadwaj

March 19, 2021

Abstract

The application of artificial intelligence (AI) to quantitative finance, particularly in stock trading optimization, is well-established yet broadly unsuccessful in predicting and acting on market price fluctuations over time. Sophisticated machine learning (ML) methods show promise of high accuracy predictions, as in the case of reinforcement learning (RL). The volume and velocity of available stock data is especially advantageous for deep "quality" RL models, where an agent learns to achieve the trading execution goal based on reward and loss function updates as it traverses market states in an environment. This study leverages three types of Deep Q Networks (DQNs) in both low and high dimensional state spaces to perform buy, sell, or hold actions for single stock trading, where it is rewarded for profiting. Two advanced methods, the prioritized replay buffer and an N-step rollout, are utilized to enhance model performance. We demonstrate that the advanced DQNs are not able to substantially outperform the baseline models, and given the natural imbalance and noise associated with temporal stock data, RL is not currently an optimal solution in determining a policy for successful stock trading.

Keywords: Quantitative finance, reinforcement learning, deep Q networks, stock trading

Introduction

Over the past decades, the field of artificial intelligence has captured not only massive public attention, but also practical appeal to many practitioners across various industries. In particular, the field of finance, specifically quantitative finance, is one of the few areas swiftly adapting and applying ML and AI to many domain-specific topics, ranging from option pricing to signal processing to market predictions. Many adaptations of AI, particularly RL, permeated through the sub-field of algorithmic trading strategy and execution development. The adaptation of RL in the algorithmic trading domain seemed to be a natural progression with AI development as an AI agent trained on a particular reward function in mind can achieve its goal in a rather autonomous fashion – minimizing rule-based logic embedded in classic algorithmic trading strategies and alleviating unnecessary human interventions in trading executions.

Throughout the first part of this paper, we demonstrate how the DQN, Double Deep Q Networks (DDQN), Dueling Double Deep Q Networks (Dueling DDQN) can be used in single stock trading by taking one of three actions (buy, sell, and hold) available in discrete action space, utilizing low dimensional state space. In the second part, we substantiate the reward function and the state space by incorporating additional state information available in the real markets that we believe might help RL agents better achieve different objectives in rather higher dimensional state space. In the third part, we introduce the prioritized replay buffer to improve sampling efficiencies and potentially alleviate the imbalanced nature of financial data. Lastly, we experiment with the three baseline agents using a 2-step rollout overlay for potentially better convergence.

Literature Review

Minh et al. 2013 describe, in their seminal paper, the foundational concepts of using Deep Reinforcement Learning to play games, namely Atari 2600 games. Their work was a foundation for understanding the basics behind how Deep Reinforcement Learning fundamentally works at a mathematical level. Van Hasselt et al. 2015 formulated the idea for Double Q-learning in their seminal work, which was another important resource for this work. Schaul et al. 2016 demonstrated the important concept of the prioritized replay buffer, which helped to understand how algorithms such as Q-learning can benefit from the use of prioritized experience replay. Specifically, their work utilizes the concept of stochastic prioritization in order to correct many of the issues highlighted in TD-error (p.4). Wang et al. 2016 expanded on the concepts presented in Van Hasselt et al. 2015 by presenting their idea of the Dueling Q-network architecture (p.1). In their book Machine Learning in Finance, Chapter 10 (Dixon et al. 2020, pp. 347 - 419) describe real-world applications of reinforcement to trading environments. Specifically, their work builds on Black-Scholes model using a Q-learning methodology, and utilizes a modified version of Q-learning called G-learning (p.380), which takes into account the probabilistic nature of options trading. Chakole & Kurhekar (2019) describe in their paper how the Q-learning algorithm can be used to make trading decisions, an important foundation for understanding how this methodology can be applied to this domain (p.1). Li et al. 2019 introduce the idea of using deep learning models in a trading environment. Their work builds off previous research in this space by adapting multiple DQN models, such as Double DQN and Dueling DQN models (p. 7).

Data

The data used for this analysis was daily price data of the S&P 500 retrieved from Yahoo Finance as a timeseries of history going back to 1950-01-03. Baseline rewards were calculated using price change based on the difference of the price of each position in the agent's portfolio and the price of the S&P 500 in the current state. Baseline state space was the history of today's price change calculated as the difference between today's price and yesterday's price. Each observation contains a history of the last 90 days and used to make an action within discrete action space to buy, hold, or sell in the current state.

The multivariate analysis explored the use of an expanded state space with six other price indexes. These other price indexes are Volatility Index (VIX), Nikkei 225, Russell 2000, Gold, Dollar Index, 10-year interest rates. Consistent with the other experiments, observations represent daily price changes of each index for the last 90 days.

To partition training and testing data, the history was separated based on a fixed date. Training data ranges from 1950-01-03 to 2010-01-05 and test data as 2010-01-05 to present. The multivariate analysis used a later date as the split point to provide more data available for training. Acceptable dates require all seven price indexes to have a value for that date so any missing values result in removing the day's data. The resulting history for this analysis used training data from 1990-01-05 to 2013-01-05 and test data from 2013-03-05 to present.

Part I - Baseline

In order to experimentally compare different reward functions, expansion of state space, prioritized replay buffer, and N-step methods, it is necessary to establish baseline performance of the three DQNs on both training and test data.

Methodology

The stock data, consisting of opening, high, low, closing prices, and volume, from 1950 to 2021 were read into a Pandas data frame. A rigid train-test split was established using a date index of 01/04/2010, as depicted in Figure 1. Python implementation of NumPy and Chainer rectified linear unit activation function, linear links, and Adam optimization function were utilized to build the deep networks. Python implementation of Plotly was utilized to visualize metrics and both training and test results.

DQN

Q-value iteration-based reinforcement learning method utilizes deep neural networks as its universal approximation function. In comparison to a standard tabular Q-learning approach, it is considered to be computationally more efficient since it natively deals with the situations where the count of the observable set of states is very large. In that regard, instead of keeping track of every single state information, DQN uses DNNs to approximate some form of non-linear representations mapping between states, actions and values. The loss formulation ultimately becomes:

$$Loss = (Q(s,a) - \left(r + \gamma \max_{a' \in A} Q_{s',a'}\right))^2$$

Double DQN

Following the same methods as DQN, an adjustment on overestimations of Q values by DQN is established by choosing actions for the next state (a', s') via the trained network, but taking values of Q from the target network (Q'). The loss formulation ultimately becomes:

$$Loss = (Q(s,a) - (r + \gamma \max_{a} Q'(s_{t+1}, argmax_a\big(Q(s,a)\big))))^2$$

Dueling Double DQN

In a similar principle as DQN and DDQN, the network tries to approximate Q values by dividing it into two different quantities; value of the state (V(s)) and advantage of actions in that state (A(s,a)). Then, heuristically Q(s, a) = V(s) + A(s, a). The ;oss formulation ultimately becomes:

$$Loss = (Q(s,a) - (r + \gamma \max_a Q'(s_{t+1}, argmax_a(Q(s,a))))^2$$
$$, where\ Q(s,a) = V(s) + A(s,a) - \frac{1}{N}\sum_k A(s,k)$$

Experimental Setup

Environment

The environment involved historical tracking of time steps, state information, and rewards. Since the data provided were tracked daily over time, the time step was set to 90 days, and the data were used as state information to update the history. At each step, there are three available actions: stay, buy, or sell. Buying updated the position with the state's closing price, short selling resulted in a negative reward, and selling as a result of profits without commission resulted in a reward equal to that profit. To stabilize training, reward clipping was implemented as follows: if the reward was greater than zero, the reward equals one, and if the reward was less than zero, the reward equals negative one. Each step concludes by setting up the next time forward, where the change in closing price from the previous step is tracked in history.

DQN

The Q Network is composed of three linear chains, the first being an input layer of input size equal to the history at time *t*+1, a hidden layer of size equal to 100, and an output layer of size equal to three. Table 1 indicates the network hyperparameters, as well as epsilon and gamma to mitigate overfitting and set importance of the previous state for rewards respectively.

Table 1

Q Network Parameters

Epochs	50
Maximum Step	Environment size - 1
Memory Size	200
Batch Size	20
Epsilon	1.0
Epsilon Decay	0.001
Minimum Epsilon	0.1
Step to Start Reducing Epsilon	200
Training Frequency	10
Update Q Frequency	20
Gamma	0.97
Log Frequency	5

Memory, total rewards, and total losses are stored over the epochs. A greedy action is randomly initialized using reset environment data while the step is less than the maximum step. Q is trained by randomly shuffling the memory to obtain batched observations, actions, and rewards. Each batch instance reward is added to the product of gamma and maximum observation. This is used to obtain the mean squared error loss, which is propagated backward, and the optimizer is updated. This will occur if the training frequency divided by the total step leaves no remainder, otherwise Q will update if the updated Q frequency divided by the total step leaves no remainder. Epsilon is then decreased if it is larger than the minimum defined epsilon and if the total step is greater than the step indicated to start reducing epsilon. The next step is then initialized by taking one step forward and updating both the total reward and observation.

Double DQN

The DDQN followed the same experimental set up as the DQN, however each batch instance reward is added to the product of gamma and the maximum Q observation with the maximum index of an estimated Q-value of the target network.

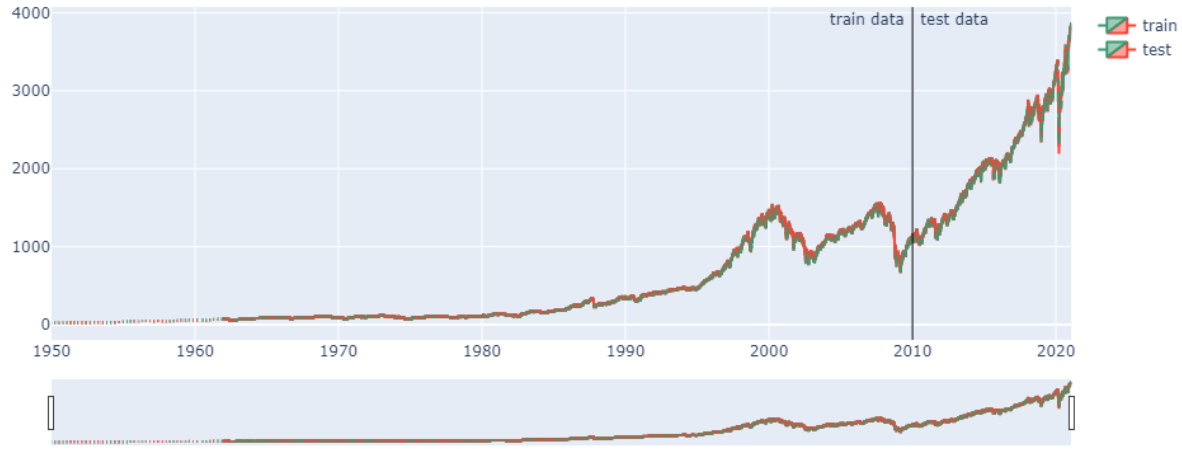
Dueling Double DQN

Six linear chain links are implemented in the network, the first being an input layer, the second using hidden size outputs similar to the other DQNs tested, the third and fourth using half the hidden size outputs as the second layer, and the fifth and sixth representing the separation of output into two destinations of a dueling DDQN, state value with an output size of 1 and advantage value with an output size set in the same manner as the other DQNs. Additionally, calculations for the advantage and Q values are added to the network's call function.

Results

Figure 1

Opening, Closing, High, Low Prices versus Year for Training and Test data Split



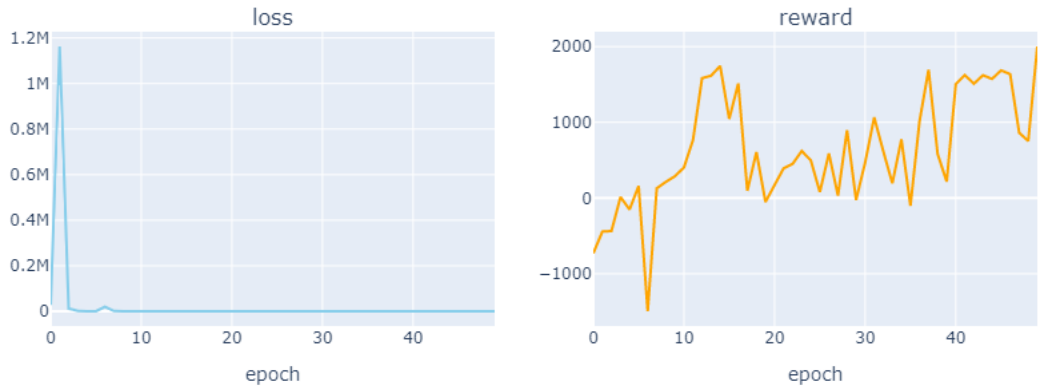
Note. Green indicates dates where the closing price was greater than opening price. Red indicates dates where the closing price was lower than opening price.

DQN

Table 2
Training Results for DQN

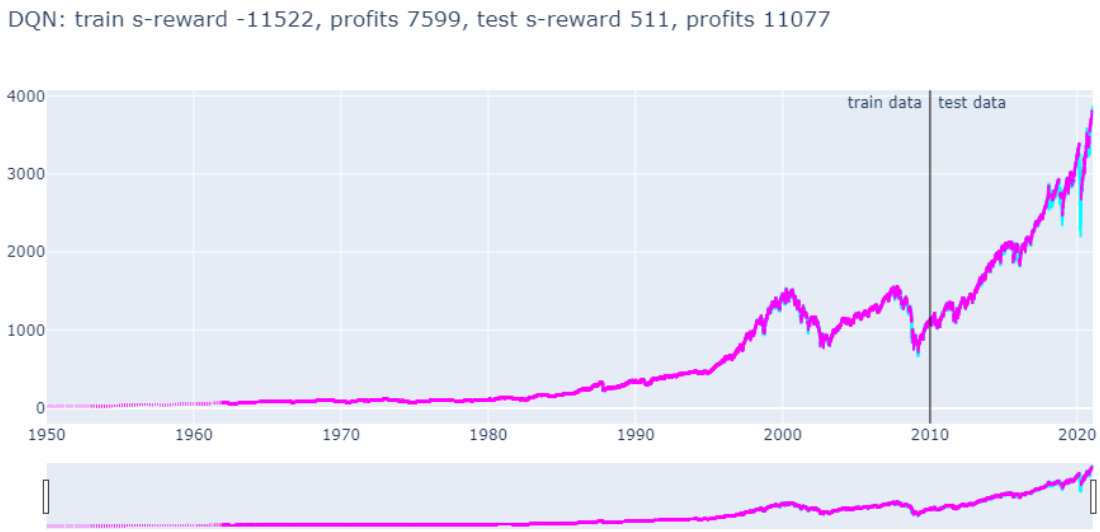
Epoch+1	Epsilon	Total Step	logReward	logLoss	Elapsed Time (s)
5	0.0999999999999992	75485	-349.4	241240.76383258976	281.66751289367676
10	0.0999999999999992	150970	-141.8	4714.858016387856	295.4078242778778
15	0.0999999999999992	226455	1222.6	1237.318213738699	272.735821723938
20	0.0999999999999992	301940	642.0	722.1539171342738	278.78027844429016
25	0.0999999999999992	377425	426.8	558.447739397455	292.4360806941986
30	0.0999999999999992	452910	314.4	542.9916924540419	316.29938316345215
35	0.0999999999999992	528395	624.4	514.4390936629003	305.05794191360474
40	0.0999999999999992	603880	680.8	485.72001587145644	309.65519881248474
45	0.0999999999999992	679365	1566.4	736.7257471117191	282.13106417655945
50	0.0999999999999992	754850	1387.8	768.3685168989701	281.44928669929504

Figure 2
Total Monetary Loss and Reward over Training Epochs for DQN



Loss significantly dropped after the first two to three epochs, finally plateauing close to zero. Total reward was negative over the first few epochs before sharply spiking above zero and fluctuating in the positive region with a general increasing trend over epochs.

Figure 3
Actions Taken Over Time by DQN



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

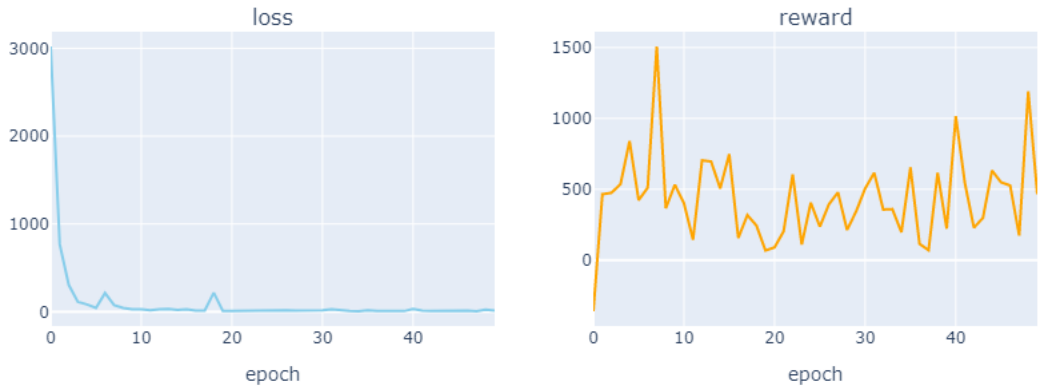
The DQN chose selling actions over most of the training and test data with a few instances of buying, where buying increased with the latest time periods around 2020. Test profits were much greater than training.

Double DQN

Table 3
Training Results for DDQN

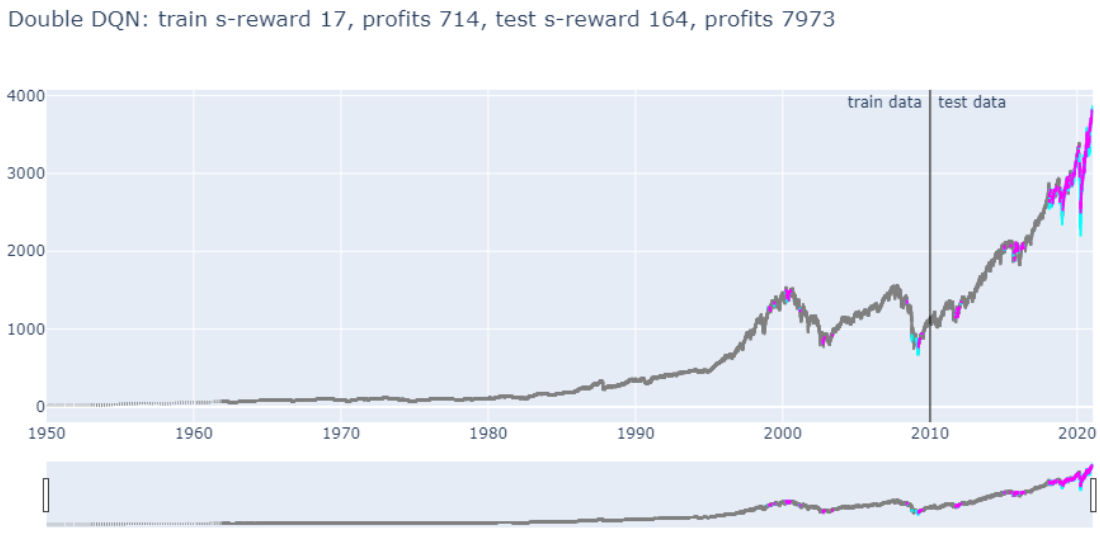
Epoch+1	Epsilon	Total Step	logReward	logLoss	Elapsed Time (s)
5	0.0999999999999992	75485	391.0	858.491977963593	204.28008103370667
10	0.0999999999999992	150970	668.0	81.14851778572789	190.35887694358826
15	0.0999999999999992	226455	490.4	24.99293798435773	163.48699951171875
20	0.0999999999999992	301940	306.8	56.425646797981265	159.85518765449524
25	0.0999999999999992	377425	282.6	13.505374020339236	159.1351490020752
30	0.0999999999999992	452910	332.8	13.273634661575347	160.28518533706665
35	0.0999999999999992	528395	407.2	14.641262539194418	160.97613406181335
40	0.0999999999999992	603880	335.6	10.208690613294348	159.522634267807
45	0.0999999999999992	679365	544.2	15.745150236345983	160.0468373298645
50	0.0999999999999992	754850	581.4	15.124426054307065	162.1545331478119

Figure 4
Total Monetary Loss and Reward over Training Epochs for DDQN



Much like the DQN, the DDQN experienced a sharp drop in total loss over training epochs after the first few epochs. Total reward immediately spiked above zero, fluctuating in the positive region around an average between $500 and $1000.

Figure 5
Actions Taken Over Time by DDQN



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

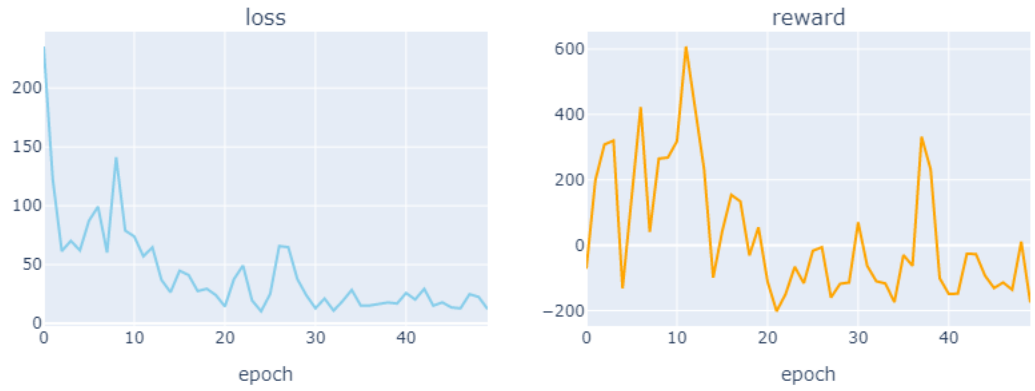
For much of the training instance, the DDQN chose stay actions with a few sporadic sell actions until right before 2010 where selling actions increased. The network thus chose to hold for much of the test data prior to 2015 with two occurrences of selling periods, until just before 2020 where the model chose many sell actions daily with some buying actions interspersed closer to 2020. Test profits were much greater than training profits by almost 11-fold.

Dueling Double DQN

Table 4
Training Results for Dueling DDQN

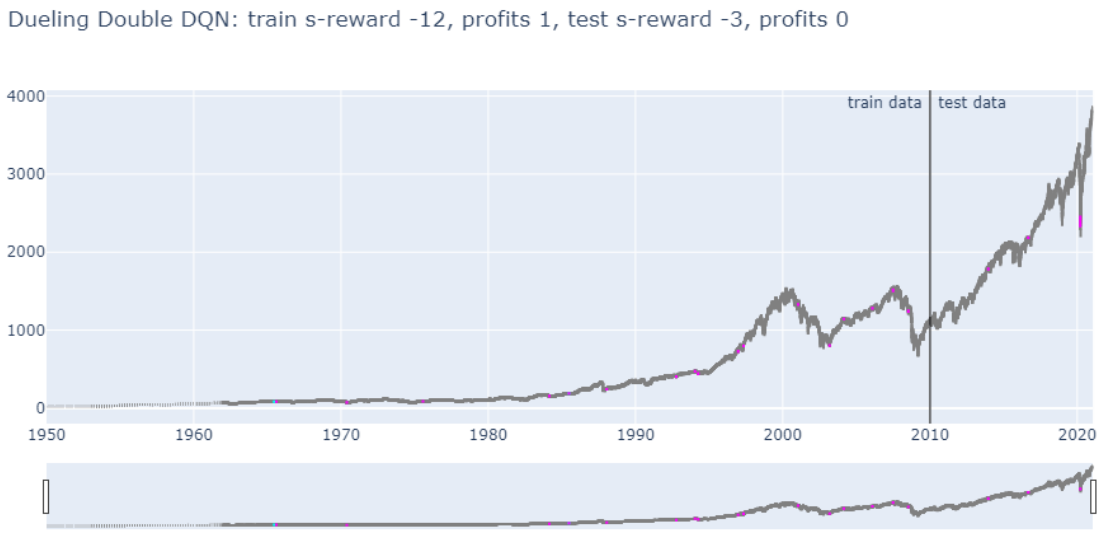
Epoch+1	Epsilon	Total Step	logReward	logLoss	Elapsed Time (s)
5	0.0999999999999992	75485	124.6	110.38291590210046	301.8617022037506
10	0.0999999999999992	150970	229.6	93.39705792363365	303.6508870124817
15	0.0999999999999992	226455	295.4	51.82594048878242	288.54467725753784
20	0.0999999999999992	301940	71.0	33.366876501808825	280.5574254989624
25	0.0999999999999992	377425	-128.8	26.20483373081243	277.5099663734436
30	0.0999999999999992	452910	-83.0	43.40282096080616	283.4147753715515
35	0.0999999999999992	528395	-78.8	18.478823908782562	274.64373683929443
40	0.0999999999999992	603880	74.0	16.36544979719969	275.8953468799591
45	0.0999999999999992	679365	-88.6	21.69070617660398	274.7770850658417
50	0.0999999999999992	754850	-109.0	17.174235925254997	273.93863129615784

Figure 6
Total Monetary Loss and Reward over Training Epochs for Dueling DDQN



Total loss decreases over epochs, however loss regularly spiked upwards from near zero during the middle of training. However, total reward displayed erratic spiking behavior across epochs fluctuating around zero reward, trending towards and ending with a negative reward overall.

Figure 7
Actions Taken Over Time by Dueling DDQN



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

The Dueling DDQN chose to hold at each closing period, with a few sporadic instances of selling in training and test; there is virtually no buying actions taken by this network in either training or test scenarios. Unlike the other deep Q networks, test profit was lower than the negligible training profit, seeing negative rewards in both cases.

Discussion

A baseline was established for the DQN, DDQN, and Dueling DDQN. The standard DQN took the most actions in training and test, primarily selling stocks, while the Dueling DDQN took the least actions choosing to hold off on buying or selling across training and test

data. Interestingly, the DDQN spent a majority of the time holding, however choosing sell actions at the later time points from 2015 onwards, a region where the data displayed higher closing prices than opening prices that correlate to an increase in total potential in gaining profit (Fig. 1). While the DQN maximized total reward better than the DDQN, training losses are comparable between the two networks and therefore it may be advantageous to implement the DDQN which still managed to accomplish a highly positive test reward but demonstrated intelligent decision-making in holding over periods where the stock closing price was lower than the opening price.

Part II – Complex Reward Functions

Methodology

Determining an optimal reward function is one of the major decisions made when building a reinforcement learning model. This reward function is one of the foundational building blocks for helping the agent optimize its policy to determine the next action. The baseline model used a simple reward function however it does not completely capture the decision and conviction behind each decision. Each action was scaled up or clipped at discrete values of 1, 0, and -1 in the baseline model. Financial markets' variable nature means no sense of magnitude was fed to the policy when training to influence the action taken. The methodology employed for these experiments was to remove the clipping and scaling resulting in continuous values and then add other scalers to develop a more sophisticated reward function.

Several experiments were tested to evaluate the benefit of adding more complex reward functions. The primary broad theme applied to the reward function was managing the volatility of positions in the agent's portfolio. In the financial markets, reward per unit of risk is one of the

foundational concepts that is desired. Achieving a high return but taking a lot of risks is not as successful as a high return with a low amount of risk. Risk is often measured by standard deviation and referred to as volatility. One of the more popular metrics factoring this is the Sharpe ratio calculated as return subtracted by some risk-free rate and then divided by the standard deviation (volatility).

Three approaches were explored in this analysis centered around this volatility concept - return minus volatility, return divided by volatility (ie. Sharpe ratio), and return divided by the volatility of negative performing positions. This last test was influenced by the concept of only penalizing performance when the agent experienced negative returns. Ideally, the agent would minimize the volatility when they lose money but not restrict the upside of positive return.

Experimental Setup

Three primary tests were performed to evaluate the benefit of using a more complex reward function - subtracting volatility, dividing by volatility, and dividing by negative returning volatility. These were all performed using the continuous reward structure and not the clipping and scaling of rewards to discrete 1, 0, and -1. Each test required additional calculations to identify this other scalar to apply to the reward. An important factor for running these tests was ensuring an appropriate scale for the contribution of this penalizing factor. This is even more relevant with an unbounded reward, driven by the primary component of the analysis - price performance.

Each experiment contained a calculation based on the positions that the agent held at that point in time (the agent's portfolio). This could range from having 0 positions to over 100 positions in the portfolio at each step. At each step, a return was calculated by subtracting the current price of the asset to the positions in the portfolio. Volatility was calculated as the standard

deviation of these returns. The volatility of negative performing positions is simply a filter of only the positions with negative returns. Both subtracting this calculated volatility figure and the division of it were tested to determine the most appropriate way to influence reward. Multiple approaches were driven by the desire to optimize a policy with a reward that can reach global minimum loss functions.

Results

Results of the complex reward functions were not as compelling as expected pointing to the challenge of training a policy to converge. Subtracting volatility and percentage change in the number of positions from the reward could not achieve positive profits and rewards. Dividing by volatility and volatility of negative performing positions were able to achieve some success but less than the baseline model on test performance. Scaled by negative returning positions performed well on the training data with a reward of 49,808 and profits of 48,495 however on the test data, it was less compelling with rewards of -1,010 and profits of 135. The number of positions was lower for this experiment pointing to the model wanting to minimize volatility.

Figure 8

Total Monetary Loss and Reward over Training Epochs for Continuous Reward Function Over Negative Returns Volatility

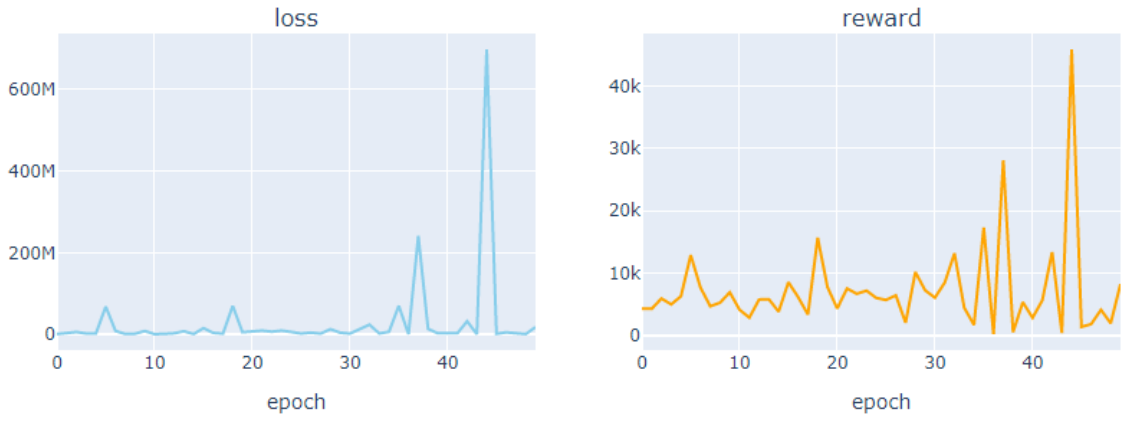
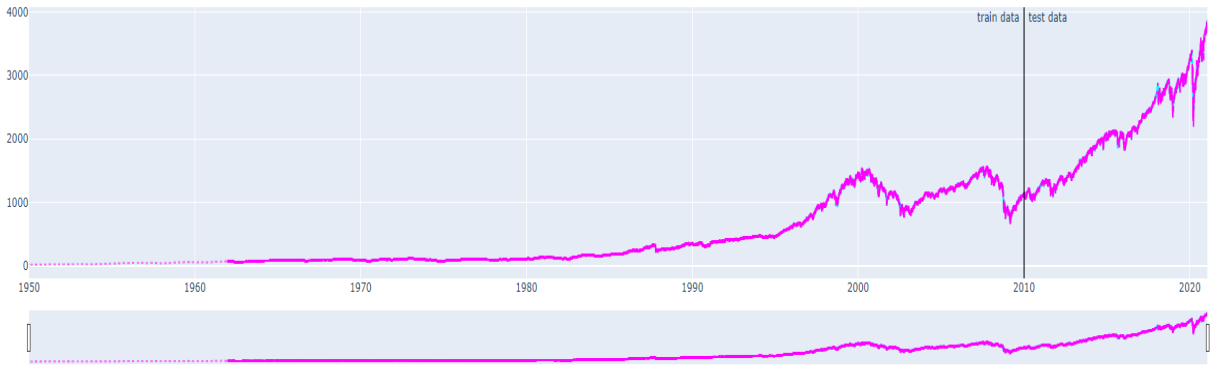


Figure 9

Actions Taken Over Time by Continuous Reward Function Over Negative Returns Volatility



Scaled by normal volatility achieved moderate performance with training generating rewards of 3091 and profits of 9647 while test performance generated 1764 reward and 10157 profits. The baseline model had issues converging making more complex reward functions struggle with reaching an optimal policy. Activation values continued to increase through each epoch with fewer parameters contributing overwhelmingly more to the decision of what action to take. Scaling by volatility often resulted in buy and hold type strategies where positions purchased in the beginning were held with lower fluctuations in positions through time.

Figure 10

Total Monetary Loss and Reward over Training Epochs for Continuous Reward Function Over Volatility

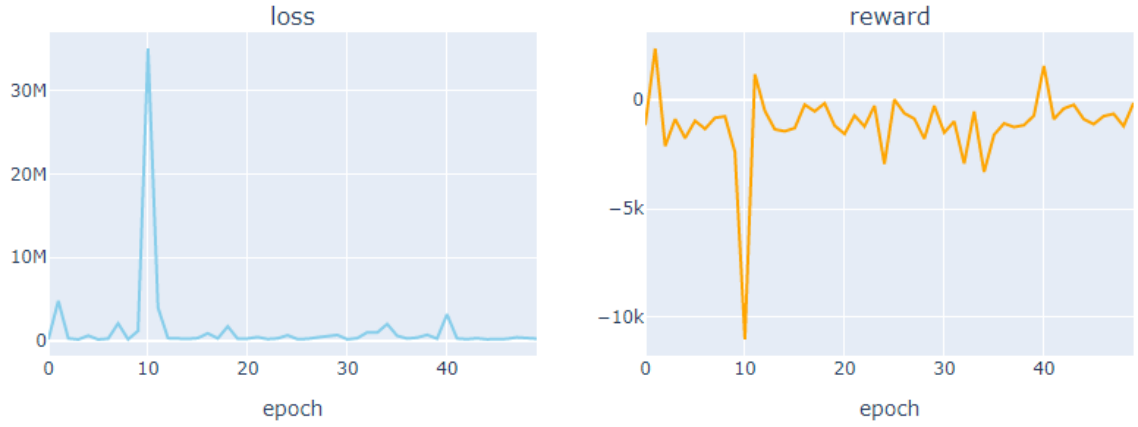
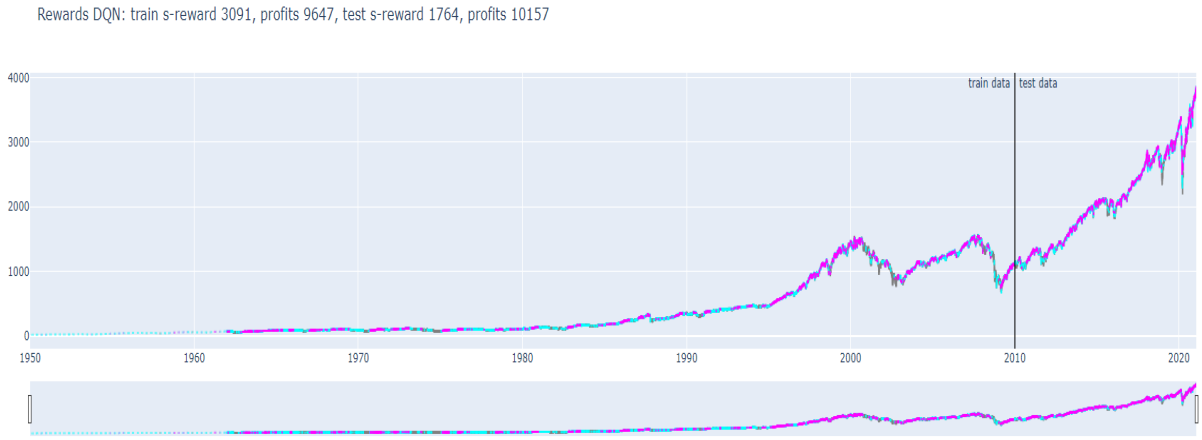


Figure 11

Actions Taken Over Time by Continuous Reward Function Over Volatility



Discussion

Complex reward functions were not successful in developing a better model than the baseline method. A couple of factors can be attributed to the weaker performance of these models including an incentive to sell positions, going against the grain of upward sloping price performance, and too much variation in rewards. Minimizing the volatility of positions made the policy choose to sell positions more often than the other models. This especially detracts from performance when there is a positive slope in the underlying index to predict. The best model would buy a lot of positions because the S&P 500 index has a strong trend of going up and to the right in the graph. Inhibiting the natural trend would prevent the maximization of reward and profit. Poorer performance on test data would be expected because of the strong bull market experienced over the last decade making trading strategies that tried to time and sell out of the market struggle versus strategies with more bets.

Another large component to the challenge of complex reward functions converging to reach an optimal policy was the volatility of the rewards themselves. The baseline model utilized clipping and scaling data to discrete -1, 0, and 1. The complex reward functions removed this

making the resulting reward continuous. Combining this with the large changes in the number of positions caused the reward to change greatly between each trade. While intuitively it would seem that this would promote trading strategies to incorporate a level of magnitude into trades, it actually made it more challenging for the policy to reach optimal values. Policies can be very sensitive and this was clearly shown with activation values increasing through time and unable to appropriately predict actions. To more appropriately reflect scale in trading, either some normalization technique should be applied or removing the scale of the number of positions.

Part III – Multivariate State Space

Methodology

Financial markets represent a complex environment where many factors influence the performance of an asset. Predicting actions to be taken based on one index is limiting especially when working with only price data which is closely followed. Expanding the state space to other indexes could make the policy more robust with additional information. In this experiment, we tested a multivariate state space adding the Volatility Index (VIX), Nikkei 225, Russell 2000, Gold, Dollar Index, 10-year interest rates, and the existing S&P 500 index. An expanded environment opens up more signals to be used to influence decisions.

Experimental Setup

A multivariate analysis using an expanded environment required adjusting the state space to reflect the price performance of these other six assets. At each step, a new set of price changes are added and the oldest is removed. The change to the observations passed through to the model was the last 90 days of price changes for each index. This increased the time to train each step of the agent because the inputs to the network are now six times larger than the standard method.

However, these seven assets (the new six including the existing S&P 500 data) had a shortened history of data reducing the number of timesteps of the data. The difference was significantly reducing the traditional analysis number of steps from 17,881 to 6,255. The reason for this smaller environment was because of the need to ensure a value was available for all seven assets. This meant removing any timestep that did not have price data for a day. The input of this joined dataset was then standardized taking the mean and dividing by the standard deviation of the percentage price change of each index. Standardizing the data was required given the different volatility of each asset. One change with the multivariate analysis was the use of the Sigmoid as opposed to the ReLU activation function. This helped prevent intermediate activation values from reaching extremes.

Results

Multivariate state spaces were able to achieve positive performance and outperform the baseline model in the test data but unable to in the training data. Overall training reward was 178 with profits of 6,279 and test performance of 37 reward and 22,136 profits. Despite the expectation for more data to intuitively enable the agent to leverage more information to improve performance, this additional information adversely affected training results. The number of positions for this analysis was the most steady not making large bets. Q-values continued to increase through time and were unable to converge.

Figure 12

Total Monetary Loss and Reward over Training Epochs for a Multivariate State Space

Discussion

Multivariate state spaces could not develop a policy to outperform the baseline model in the training data pointing to the difficulty of training a reinforcement learning model. Improved performance in the out of bound data could have been due to not overfitting to the S&P 500 data. This was interesting because performance of the S&P 500 was much better in the history of the test data relative to the training data. Position counts were also less so this model was able to achieve better performance with less risk. Poor profits on the training data is difficult to explain as we would have expected more information to better inform the model. However, the inability for the policy to converge could attribute to most of this issue. The S&P 500 was also one of the better performing indexes, especially when compared with other time series such as the Dollar and VIX so that information may detract from a policy incentivized to purchase positions.

Part IV – Prioritized Replay Buffer

Methodology

With the same three baseline agents (DQN/DDQN/Dueling DDQN), in this part of the paper, we introduce an important sampling technique called prioritized replay buffer during the

training process. The prioritized replay buffer is a way of improving sampling efficiencies further by sampling based on weights calculated off of a training loss of each sample – some form of importance sampling.

This technique follows a rather simple mathematical framework detailed below:

$$P(i) = \frac{p_i^\alpha}{\sum_i p_i^\alpha}$$

$$, where\ p_i\ is\ defined\ as\ a\ weight\ on\ i^{th}\ sample$$

$$\alpha\ is\ a\ scalar\ importance\ adjustment\ factor$$

Intuitively, can be interpreted as a hyper-parameter that provides an additional utility of a degree of importance on the weight. As an example, on the one end of , the entire weighting scheme becomes uniform. On the other end of the spectrum, samples with higher weights will be even further emphasized during this weighted sampling process.

There are possibly many different ways of expressing the weights for efficient sampling. Among many, the most simplistic and intuitive is the one that is proportional to the loss for each sample during the training:

$$p(i) \propto\ c * loss(i)$$

With that, an additional adjustment is applied to each initial weight to compensate for the implicit bias introduced by this weighted sampling scheme:

$$w(i) = (N * P(i))^{-\beta}$$

$$, where\ N\ is\ the\ total\ number\ of\ samples\ in\ a\ buffer$$

$$\beta\ is\ defined\ as\ a\ scalar\ bias\ adjustment\ factor$$

This is particularly important in case of any batch learning with any form of stochastic gradient descent (SGD) optimizers since it assumes each training batch follows the independent and identically distributed (i.i.d.) property – so that a given SGD optimizer does not end up getting stuck in one of those local optima.

An additional important note in using this importance sampling technique is ensuring the balance between "usual" and "unusual" samples during the training process. As can be expected intuitively, any serious degree of imbalance between those two groups can lead to a potential overfit to small subsets of the given data. Throughout this experiment, the importance sampling is applied every 1000 steps of each epoch.

Experimental Setup

The following table contains a hyper-parameter setting for two additional hyper-parameters for the prioritized replay buffer introduced in this section.

Table 5
Prioritized Replay Buffer Special Hyperparameters

α (importance adjustment factor)	0.6
β (bias adjustment factor)	0.4

Results

DQN

Table 6
DQN - Total Rewards/Profits for Train and Test Data Set

	Train	**Test**
Total Rewards	-10944	374
Total Profits	4220	-3722

Figure 13

DQN - Losses (Left) and Rewards (Right) throughout Epochs

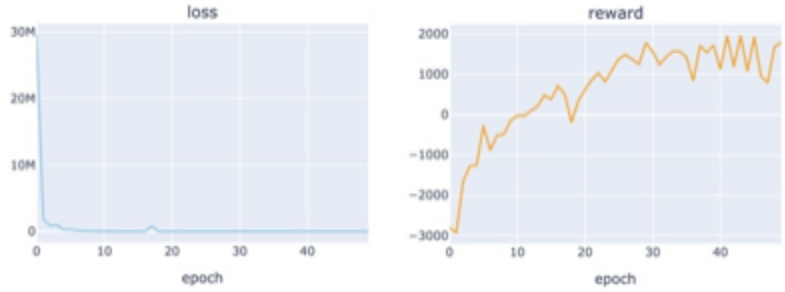
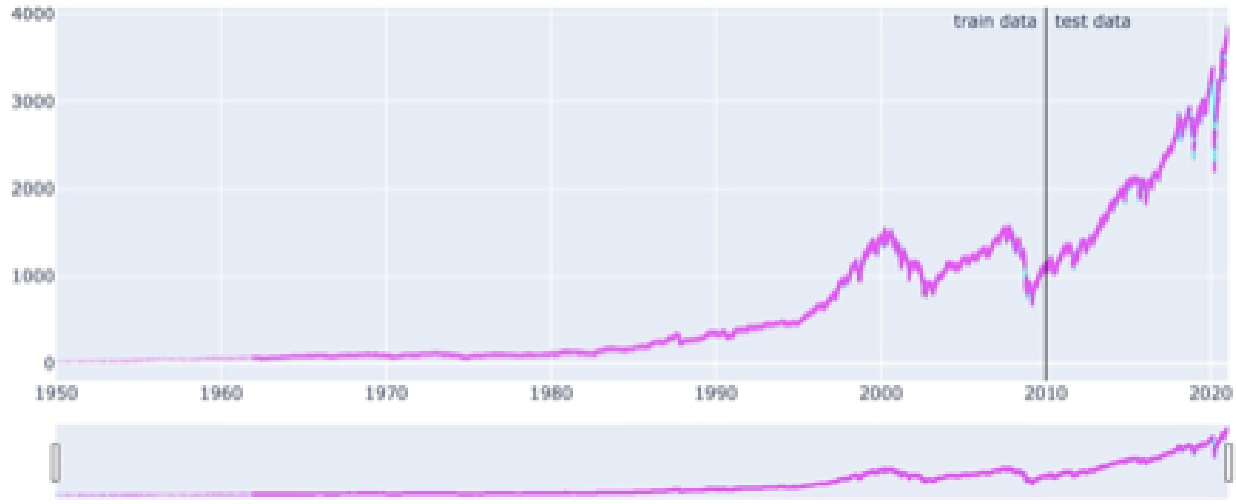


Figure 14

DQN - Choice of Actions across Time-horizon



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

DDQN

Table 7

DDQN - Total Rewards/Profits for Train and Test Data Set

	Train	**Test**
Total Rewards	-195	147
Total Profits	246	6584

Figure 15

DDQN - Losses (Left) and Rewards (Right) throughout Epochs

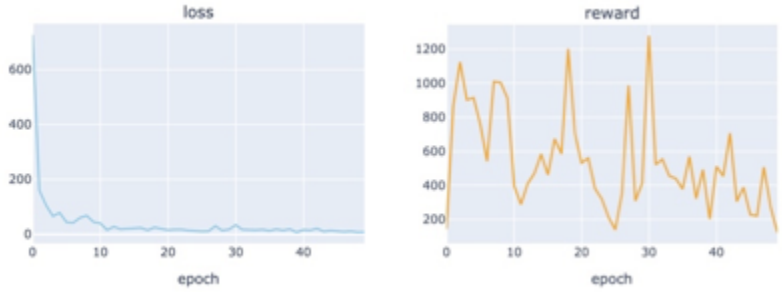
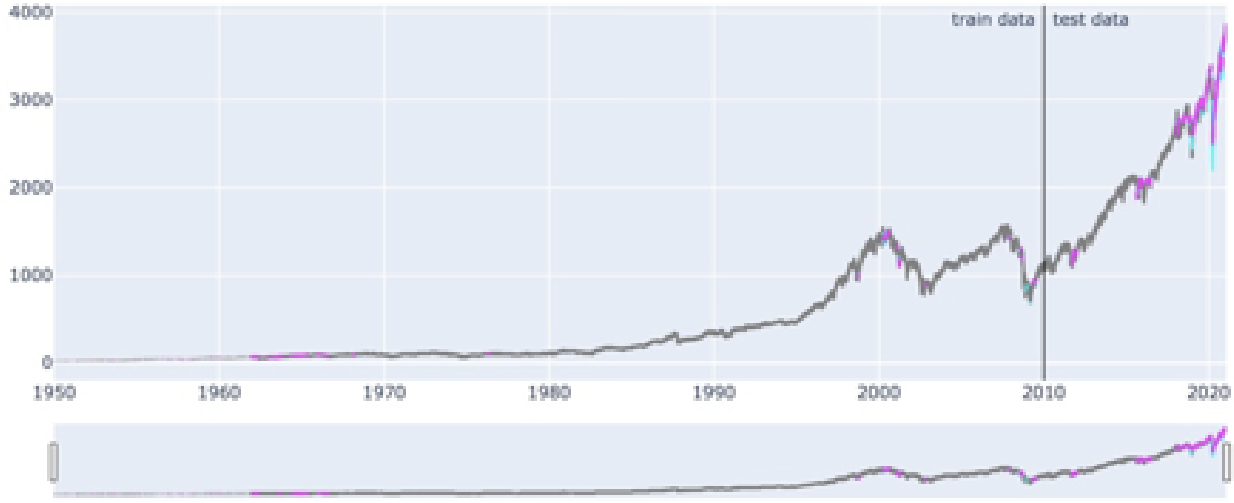


Figure 16

DDQN - Choice of Actions across Time-horizon



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

Dueling DDQN

Table 8

Dueling DDQN - Total Rewards/Profits for Train and Test Data Set

	Train	**Test**
Total Rewards	2391	579
Total Profits	31077	28185

Figure 17

Dueling DDQN - Losses (Left) and Rewards (Right) throughout Epochs

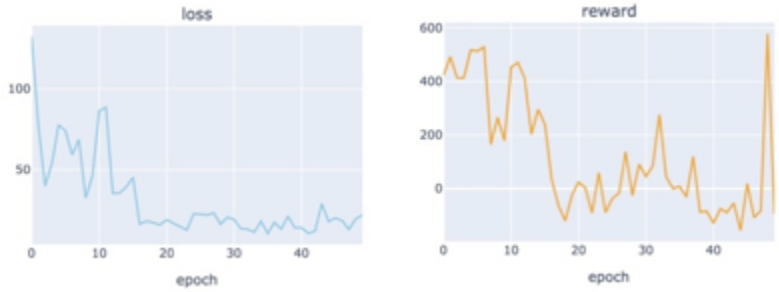
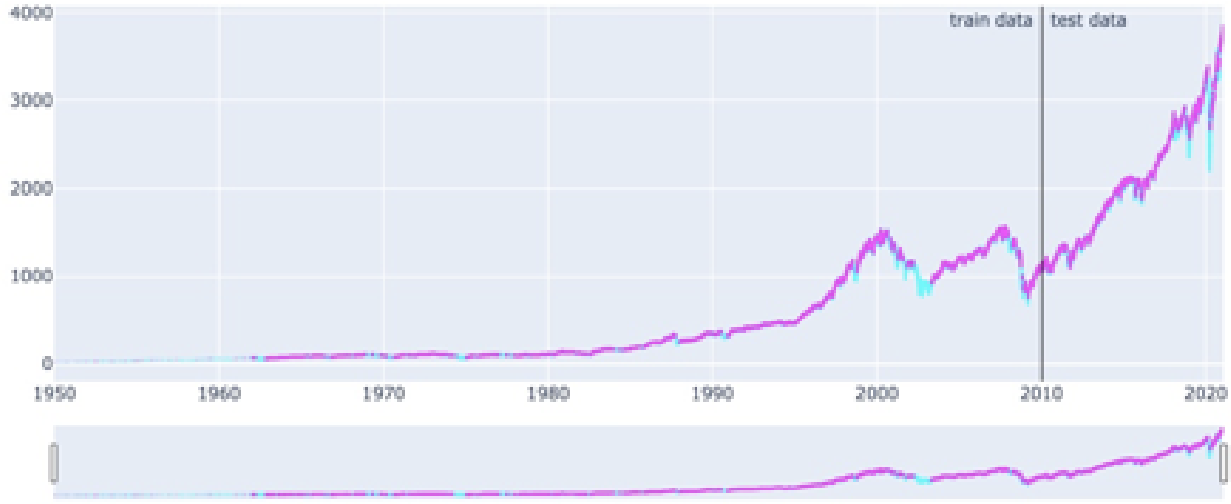


Figure 18

Dueling DDQN - Choice of Actions across Time-horizon



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

Discussion

For the DQN agent, looking at Table 6 and Figure 13, loss and reward values throughout epochs implied that potential overfitting might be of concern in this case – a steep drop in the loss in early stages of training and evolution of rewards only improved without showing much of noise – or with much exploration involved. This is a quite different behavior from the baseline DQN agent – the one without the prioritized replay buffer where we did not observe this degree of overfitting. We believe that this may be largely due to the very nature of the prioritized replay

buffer in terms of the possibility of getting stuck in local optima as briefly discussed in the method section above. In particular, this can be noticeably seen from some of the actions the agent took with the testing data set – Figure 14. Even when market signals started to drop, the agent with the prioritized replay buffer tends to stick with the selling action by following its locally optimized policy.

For the DDQN agent, looking at Table 7 and Figure 15, it seemed to be generalized better than the DQN agent above under the presence of the prioritized replay buffer. However, the overall performance was not still satisfactory compared to the baseline DDQN agent. Looking closely at its actions taken across time-horizon – Figure 16, it is rather evident that most of the profits with the testing data set stemmed from taking the holding action at the beginning of the testing period by carrying out the learned action throughout the vast majority of the training period – the holding action. Putting these together, we do believe that the agent is still locally optimized so that it tries to achieve the learning objective by following a sub-optimal policy.

However, looking at Table 8 and Figure 17, the dueling DDQN agent seemed to be regularized and generalized much better than the baselines and the other two agents under the presence of the prioritized replay buffer. The dynamic of loss values across epochs indicates a relatively sound convergence was made without much steep gradient movement. Also, the dynamic of rewards shows an appropriate amount of exploration has been made throughout epochs. This relatively efficient learning process was, in turn, reflected in the total rewards and profits the agent achieved in Figure 18. Looking closely, when the market went through those "trough" regions, the agent with the prioritized replay buffer tends to take rather immediate action by buying more. On the other hand, when heading to those local "peak" regions, the agent took the selling action accordingly. However, the agent still at times made suboptimal decisions

by buying when the market continued to decline even further, indicating that there is still more room to improve the current policy for better decision making of the agent. In particular, once we start incorporating transaction costs in reward function construction, this rather immediate action taken by the agent can be since more frequent transactions based on a "locally" optimized policy will not be able to achieve the global objective.

Part V - N-Step DQN/DDQN/Dueling DQN

Methodology

N-Step DQN's utilize a combination of Deep Q-learning, combined with a n-step rollout, in order to improve the convergence of training. This method, first popularized by Sutton (1988), allows reinforcement learning models to learn using n-step look-ahead, by weighting the value function as well as the reward defined (para. 1). The equation shown below (Lapan, 2020, Chapter 8) is the exact equation that was implemented in this case, for the various DQN network architectures considered.

$$Q(s_t, a_t) = r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s_{t+2}, a')$$

The experimental setup for the N-step DQN, N-Step Double DQN, and Dueling DQNs were exactly similar to the baseline models in Part 1, as described above, in an effort to compare the improvement of the baseline Q-learning algorithms vs. the N-step versions. However, there are several changes that are reflected in the implementation to make this change. First, Q-learning classes kept track of the most recent batch, while an N-step network requires additional data, as seen in the t+1 and t+2 variables in the equation. Therefore, the code was modified to include the t+1 timestep reward function, along with the modified estimate of Q

values. This step was repeated for both Double and Dueling DQNs as well. All 3 networks were tested using a 2-step (N = 2) rollout.

Experimental Setup

The experimental setup for the 2-step DQN models was exactly the same as the setup for the baseline models, to ensure consistency (Please refer to Part 1 - Baseline), with main modification being the n-step equation shown above.

Results

The results for the 2-step DQNs demonstrated that the 2-step DQN and 2-step Double DQN vastly underperformed the baseline results for both their respective models. It should be noted that for both these models, the loss decreased rapidly, while the reward was initially high (> 200), but then decreased sharply and plateaued at approximately 0. The 2-step Dueling DQN, however, greatly outperformed the benchmark, earning a profit of $43,644 in the test set, compared to a $0 profit to its 1-step counterpart. This was mainly due to the high amount of buying and selling that occurred following the 2008 financial crisis, greatly increasing profits in the 2010 - 2020 time period. The Figures below demonstrate the performance for each of the models.

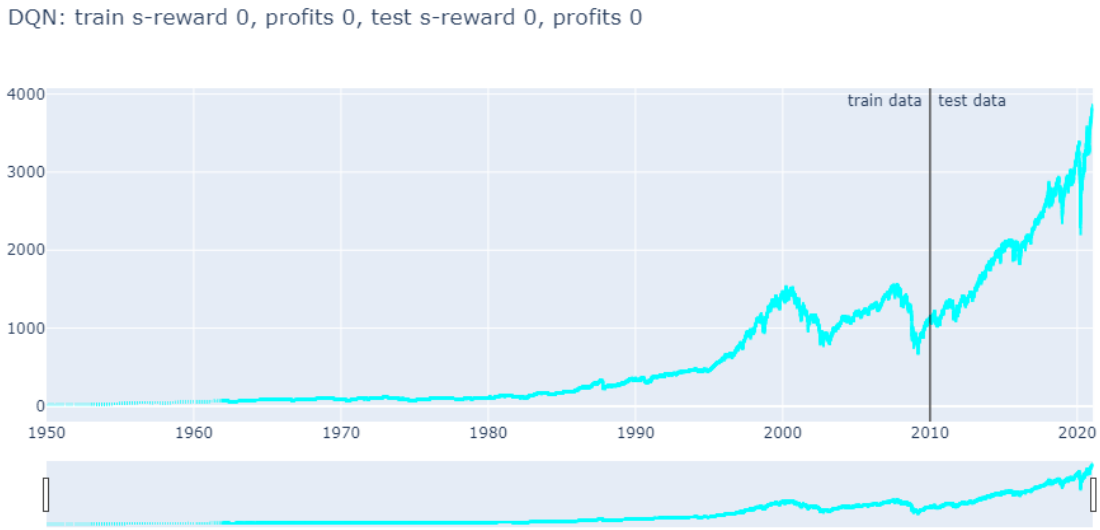
Table 9

2-Step DQN Results

Epoch+1	Epsilon	Total Step	Log (Reward)	LogLoss	Elapsed Time (s)
5	0.09999999999999920	75485	142.4	2317.395719	328.3833444
10	0.09999999999999920	150970	-10	413.4354554	447.3319972
15	0.09999999999999920	226455	-25.2	396.4653115	439.3473337
20	0.09999999999999920	301940	-29.8	391.9982624	436.431329
25	0.09999999999999920	377425	-23.4	409.4248027	447.1110008

30	0.09999999999999920	452910	-23.4	388.4055137	450.757544
35	0.09999999999999920	528395	-26.6	372.03282	461.6767323
40	0.09999999999999920	603880	-30.4	381.5658286	448.0211041
45	0.09999999999999920	679365	-38.6	377.4591421	450.3803256
50	0.09999999999999920	754850	-26	403.571843	437.9663653

Figure 19
2-step DQN Choice of Actions across Time-Horizon



Note: Cyan indicates buy action, magenta indicates sell action, and gray indicates stay option.

Figure 20
2-Step DQN - Losses (Left) and Rewards (Right) throughout Epochs

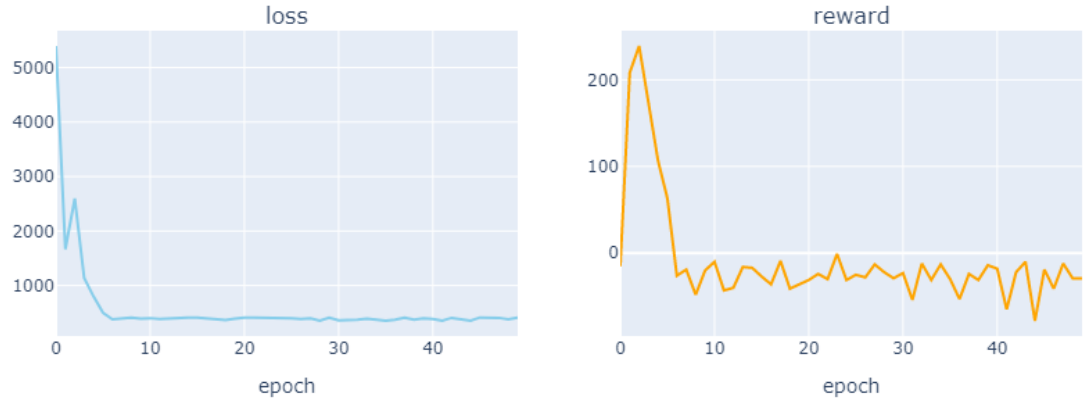


Table 10
2-Step Double DQN Results

Epoch+1	Epsilon	Total Step	Log (Reward)	LogLoss	Elapsed Time (s)
5	0.09999999999999920	75485	313.2	518.3178438	208.3444831
10	0.09999999999999920	150970	117.2	211.5079254	325.5810256
15	0.09999999999999920	226455	-13.8	164.9810496	372.7104537
20	0.09999999999999920	301940	-19.2	153.950322	370.8479385
25	0.09999999999999920	377425	-23	147.6152731	368.1765821
30	0.09999999999999920	452910	-13.2	160.0796866	358.7660577
35	0.09999999999999920	528395	-10.2	160.2989112	374.8074288
40	0.09999999999999920	603880	-14	153.921854	374.2628131
45	0.09999999999999920	679365	-12	159.510028	353.7344165
50	0.09999999999999920	754850	-13.4	157.3946832	360.8499038

Figure 21
2-step Double DQN Choice of Actions across Time-Horizon

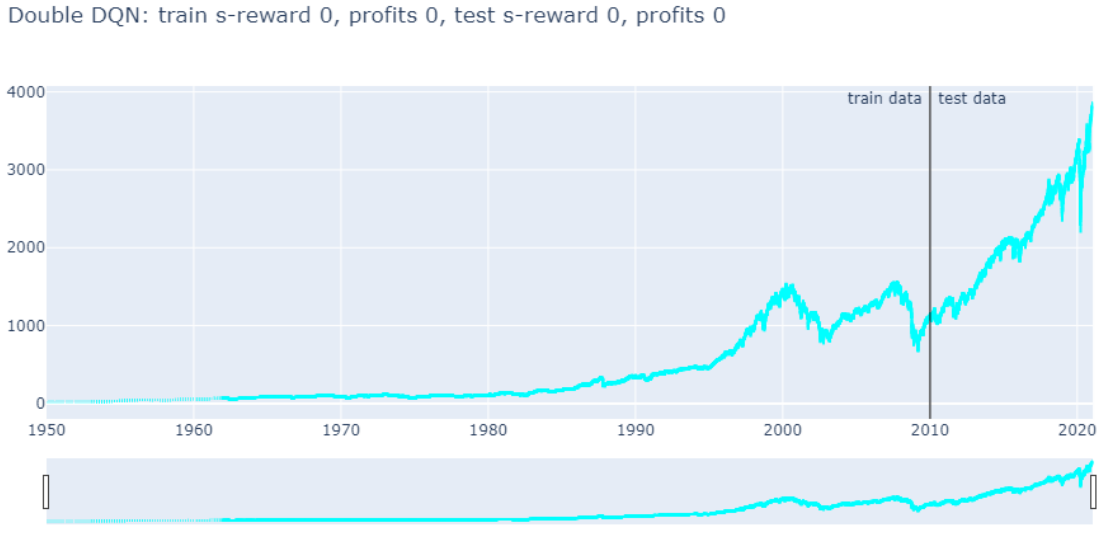


Figure 22

2-Step Double DQN - Losses (Left) and Rewards (Right) throughout Epochs

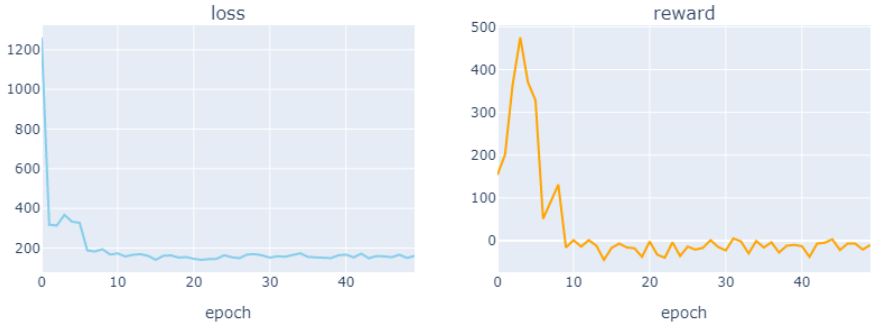


Table 11

2-Step Dueling DQN Results

Epoch+1	Epsilon	Total Step	Log (Reward)	LogLoss	Elapsed Time (s)
5	0.09999999999999920	75485	381.8	565.809531	327.9053783
10	0.09999999999999920	150970	536.8	457.533558	329.2875743
15	0.09999999999999920	226455	904.6	440.2342634	331.8600135
20	0.09999999999999920	301940	580.2	406.1678416	331.2742934
25	0.09999999999999920	377425	586.2	366.4386781	328.6193447
30	0.09999999999999920	452910	554	455.9574716	336.2375886
35	0.09999999999999920	528395	518.4	427.573368	339.2278426
40	0.09999999999999920	603880	954.4	524.1384289	339.2786179
45	0.09999999999999920	679365	852.2	397.559438	338.0871801
50	0.09999999999999920	754850	961	394.1469872	346.9240909

Figure 23
2-step Dueling DQN Choice of Actions across Time-Horizon

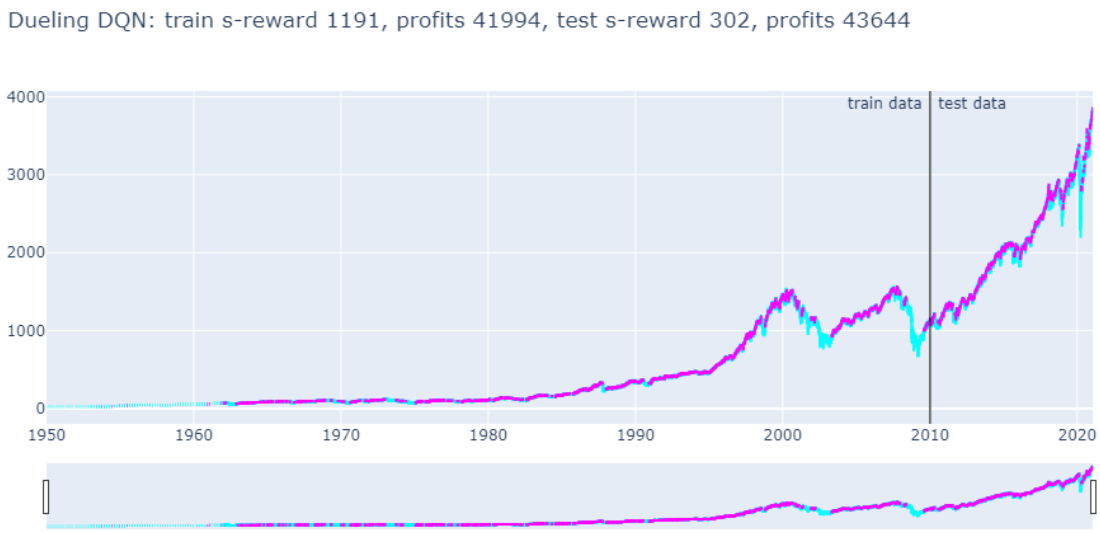
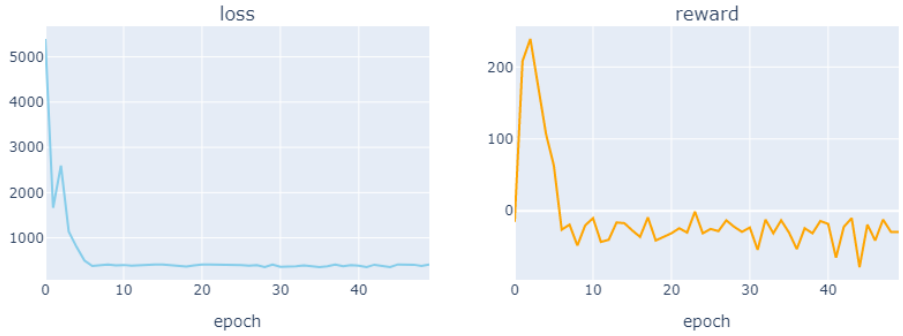


Figure 24
2-Step Dueling DQN - Losses (Left) and Rewards (Right) throughout Epochs



Discussion

The addition of the 2-step rollout for the DQN models does impact performance when Dueling DQN's are used. Further improvements to this study could include experimentation with 3-step or 4-step Dueling DQN models in order to see if this could improve profits.

Conclusion

Overall, a reinforcement learning application to the financial markets is challenging to implement using simply price data to develop a successful trading strategy. The sensitivity of training a reinforcement learning agent is clear with advanced methods unable to significantly outperform the baseline DQN model. We analyzed many techniques throughout this research including Double DQN, Dueling DQN, N-Step DQNs, priority buffers, multivariate analysis, and complex reward functions. Each model adds a layer of complexity but they still struggle to develop a policy that converges to an optimum. This was not very surprising given the noise and autocorrelation that exists in the financial markets. Continued exploration will be needed to develop a strategy that is successful in trading the S&P 500.

Directions for Future Research

When it comes to directions for future research, there are many parts to be done to further improve performance and substantiate additional key considerations. We will lay out some of those both for the signal processing aspect and for the RL trading agent aspect.

Recommended Directions for Signal Processing

Throughout this paper, we took raw inputs from the real markets and RL trading agents took actions based on action-values iteratively calculated by Q-learning. In practice, having a signal processing method in place before feeding into the RL trading agents deems to be appropriate, particularly considering the very nature of the financial market – high noise to signal ratio (i.e., non-linear dimensionality reduction, kernel, and transformation related techniques). In addition, in dealing with signal processing, particular attention needs to be paid to the very imbalanced nature of financial data. Although we introduced the concept of prioritized replay buffer and its potential utility in alleviating imbalanced nature among samples – by providing

uneven weights, we can potentially also explore other loss function constructions instead of the standard mean square error but rather some others like Q-like loss function.

Recommended Directions for RL Trading Agent

Among many, the first thing that can be explored is natively dealing with Partially Observable Markov Decision Process (POMDP) by using recurrent neural nets (RNN) architecture in estimating Q-values. In particular, as we deal with more high frequency data, the POMDP nature of financial market data would be considerably more important in terms of better capturing temporally dependent signals – whereas with the less frequent data the classic MDP assumption may hold reasonably well as in the classic random walk theory in stochastic simulations. In addition, particular attention is necessary in handling the time-series data in order to avoid potential data leakage issues. In particular, any form of stochastic gradient descent (SGD) optimizations assumes the aforementioned i.i.d. property for each batch learning process. To meet that assumption, a random shuffling across different batches tends to be carried out in most SGD-involved learning processes. However, it is critical to ensure the agent is not looking ahead in the time dimension and to make sure dependencies across different batches are still minimized if any random shuffling is implemented.

References

Chakole, J, Kurhekar, M. (2020). Trend following deep Q‐Learning strategy for stock trading. Expert Systems. 37:e12514.

Dixon, M. F., Halperin, I., & Bilokon, P. (2020). Machine learning in finance: From theory to practice. Cham: Springer.

Lapan, M. (2020). Deep Reinforcement Learning Hands-On - Second Edition. Packt Publishing.

Li, Y., Ni, P. & Chang, V. (2020). Application of deep reinforcement learning in stock trading strategies and stock forecasting. Computing 102, 1305–1322

Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602

Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. arXiv preprint arXiv:1511.05952

Sutton, R.S., 1988, Learning to Predict by the Methods of Temporal Differences, Machine Learning 3(1):9-44

Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: AAAI, vol 2, p 5. Phoenix, AZ

Wang Z, Schaul T, Hessel M, Van Hasselt H, Lanctot M, De Freitas N (2015) Dueling network architectures for deep reinforcement learning. arXiv preprint