Coursework Report for Module INM707 Deep Reinforcement Learning

Lina Katabi and Rachael Olatunji

Purpose and Motivation for this project

The definition of trading can be described as the purchasing, exchanging or selling of assets, regularly seen in the financial market [1]. Financial report and news increasingly show stocks that are heavily automated with over half of US equity trading driven by Quants and high frequency traders [2] which employs both mathematical and artificial intelligent models to analyse the financial data without necessarily establishing a link to financial theory. Along with the training and testing that would take place during the process, deep reinforcement learning would increase the model's performance, hence making it beneficial for this problem. Due to the financial market being highly stochastic, complex and dynamic, the models that are not controlled by rules have proven to be more reliable, therefore demonstrating that deep learning is a promising future to the strategy of trading. One of the categories of machine learning is deep reinforcement learning, which is a method that joins together reinforcement learning to deep learning [3].

Description of problem

Reinforcement learning involves an agent exploring its environment, gaining or losing rewards based on wanted or unwanted actions which is mastered through trial and error [4]. The purpose of this report would be to investigate whether an agent can learn from the trades carried out in the stock market environment. The objective of the agent is to optimises its learning strategy (by taking position on the market) and maximise its profit at the end of a period.

The period would characterize the episode length for this scenario in the training part. In each episode, the agent will run through it and decide to position itself as short, long or neutral and will then be rewarded if only the daily positioning is matched correctly. The agent task is to maximise its long-term return chosen over a period by returning on daily basis the optimal trade, through the process of stating the reward for each action beforehand over time.

The environment to which the agent interacts in, is considered arduous to dissect due to the many factors connected that affect the stock market such as current events, interest rates etc. and how difficult the stock market is to predict. Figure [1] shows how deep reinforcement learning works. To begin with, the environment is created, this is where the agent operates. Afterwards the rewards are then specified beforehand for when the agent is created and then trained and validated for the policy to finally then be deployed [5].

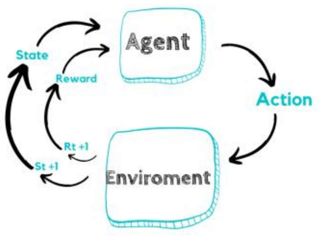


Figure [1]. Deep reinforcement learning

This problem we would be addressing would be modelled as a Markov Decision Process (MDP). In reinforcement learning, a mathematical framework is used to explain the environment, this is what is called MDP. The combination of actions (enabling choice) and rewards distinguishes Markov decision processes from Markov chains. A Markov decision process reduces to a Markov chain if each state has only one action (e.g., "wait") and all rewards are the same (e.g., "zero") [7].

The purpose of this report would be to investigate whether an agent can learn from the trades carried out in the environment of a stock market, to optimises its learning strategy and maximise its profit at the end of the day. This essay would display the effort of imploring reinforcement learning (RL) agent in the stock market. In this project, we evaluated two unique agents, one is an example of a model-free agent which is Q-learning agent and the second trains a critic to estimate the return or future rewards, in turn making it a value-based agent. To obtain background information in the deep reinforcement learning, the paper *DEEP REINFORCEMENT LEARNING WITH DOUBLE Q-LEARNING* et al. (2015) was read and the similarities and differences between Q-learning and DQN was learned which would later be presented in this report [6].

Dataset

As the US market is one of the most traded indexes and amongst the most efficient markets, the full simulation was run on the SPY exchange traded fund which follows the S&P500, throughout this project. Kaggle.com [8] was used to gather the data. The model will still have a partial perspective of the market because it is unable to account for most market data. However, utilising some technical indications per paper, we can then simplify and attempt to represent the current index position.

Defining the states

The market characteristics that were used to represent the index and designate states are mentioned below:

Daily Close Returns: percentage changes at time t between close price at t and t-1.

Open: the price of the index when the stock exchange open for the day.

High: the maximum price of the index has traded for the day.

Low: the minimum price of the index has traded for the day.

Close: the price of the index when the stock exchange close for the day.

WR (6): Williams Overbought/Oversold Index is a momentum indicator that compare the most recent closing price to the highest high of the last 6 trading days.

WR(12): Williams Overbought/Oversold Index is a momentum indicator that compare the most recent closing price to the highest high of the last 12 trading days.

RSI(14): The relative strength index is a technical indicator used in the analysis of financial markets. It is intended to chart the current and historical strength or weakness of a stock or market based on the closing prices of the last 14 days period at time, t.

Defining the action space

Given three alternative positions that the agent will choose at each time step, one simple discrete action space was developed which will be represented by [0,1,2]

State transition probability function

In MDP, the state transition probability function determines the likelihood of moving from one state to another whilst still accounting for the agents' actions. The state at which time step t+1 is considered the current state function, where action has been taken at a specific time t, as shown below in figure [2].

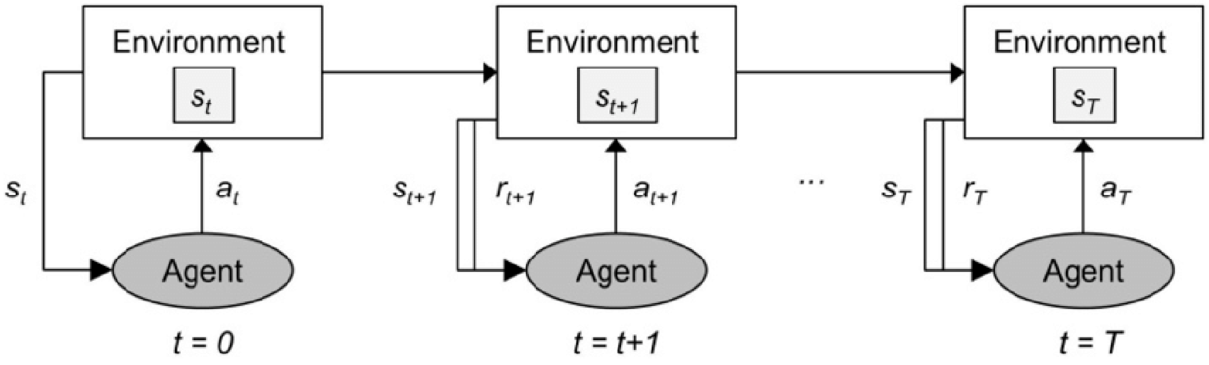


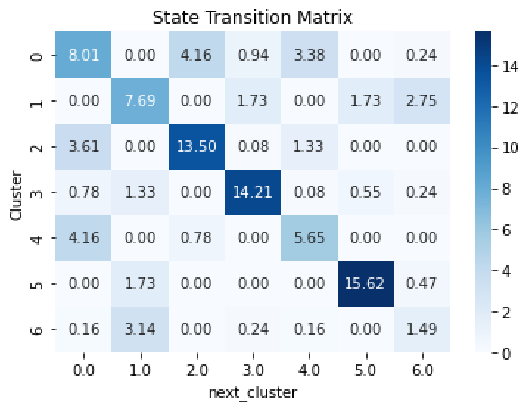
Figure [2]. In an MDP, the agent-environment interaction.

However, for this scenario, the agent's actions will not decide which state it stops in. It just monitors the market environment's next state (The internal status of the agents, on the other hand, is defined in the advance section e.g., cumulative rewards, current positions, and the last sequence of actions would be a portion of the state description).

The notion that states are not ever explicitly duplicated, leading to environments with unlimited alternative states, is among the obstacles in representing a financial trade system as a Markov Decision Process. To a certain degree, this can be overcome by establishing a great though simplified environment that compensates (explicitly or implicitly) for most significant rational market characteristics. An even more challenging problem to solve without jeopardising the model's realism is that a basic state transition function is rarely applicable: acting in state, s may lead to different an unrelated new state at varying time scales with the same s, a, s' sequence, leading to the agent seeing an entirely different reward.

To describe our stock trading system as an MDP, the states would be discretised through the means of clustering using the k-means algorithm as the cluster centroids generate a set of ongoing market features taken out of the training data, resulting into a limited number of states. The clustering method is utilised, wherein the n state of a market is specified, and the training sample is used to run the clustering algorithm to produce the states, which will then be used to represent the centroids of the developed clusters.

The states are grouped into 7 categories/clusters and is visualised in figure [4]. The centroids clusters are presented using two features: Daily close changes and the WRI(12), despite the fact that we employed 8 market features in our fundamental setup in figure [3].



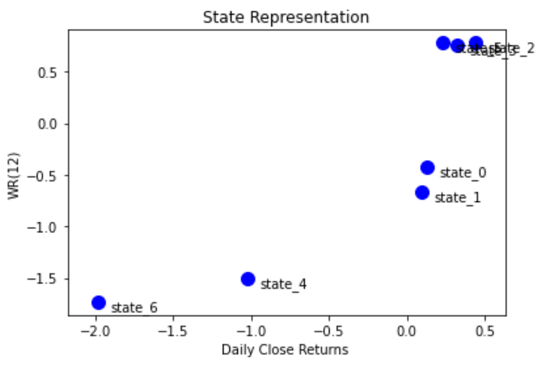


Figure [3]. State transition function probability

Figure [4]. State clustering

Figure 3 depicts the odds of transitioning from one state to another after clustering and running the historically the data. The market environment tends to stay in the same state as it was in the previous iteration, which is comparable with real financial climates where market features do not vary drastically from one day to the next.

Maximising the cumulative reward

The agent's objective is to optimize the total amount of money earned. The issue we have in our scenario is that market rewards are very stochastic, meaning that for the identical s,a,s', the agent could acquire entirely different rewards. As a result, we chose to employ an anticipated reward function, which we dynamically created in the pre-processing phase based on the historical timeseries. The reward function derived, would be characterized for this scenario as the period chosen for training, where each day an agent would be allowed to position itself as short, long or neutral and will then be rewarded if only the daily positioning is matched correctly.

Depending on the action, the reward function would attribute different reward stated as below:

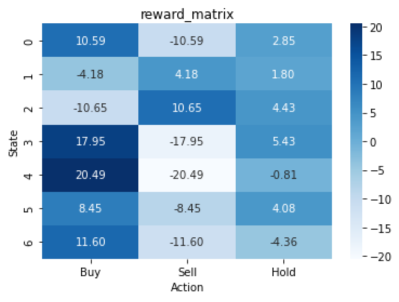
Buy: gain the next day return which would be positive if the price increase.

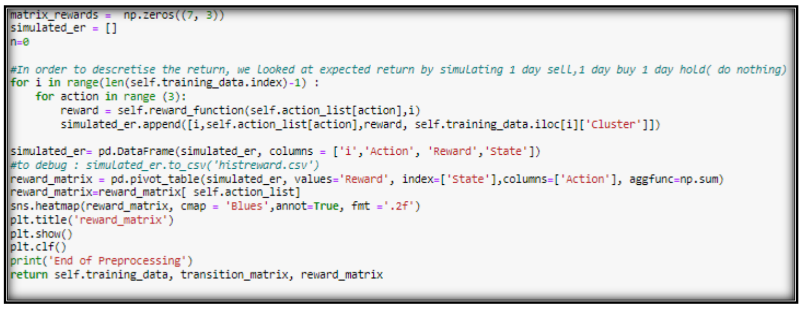
Sold: gain the negative next day return which would be negative if the price increase.

Hold: will get rewarded if the next day return is lowered than the trading cost. It will also get penalised if the next day move is greater than 4% which is considered a highly daily change.

The reward matrix which will feed our Q-learning model can be seen in Figure 5.

Following the training is the reward matrix presented in figure [5], which are influenced by the actions and how the market evolves after each iteration. The reward matrix is motivated by the stochastic nature of financial market value reward as the agent moves from one state to the next.




```
matrix_rewards =  np.zeros((7, 3))
simulated_er = []
n=0

#In order to descretise the return, we looked at expected return by simulating 1 day sell,1 day buy 1 day hold( do nothing)
for i in range(len(self.training_data.index)-1) :
    for action in range (3):
        reward = self.reward_function(self.action_list[action],i)
        simulated_er.append([i,self.action_list[action],reward, self.training_data.iloc[i]['Cluster']])

simulated_er= pd.DataFrame(simulated_er, columns = ['i','Action', 'Reward','State'])
#to debug : simulated_er.to_csv('histreward.csv')
reward_matrix = pd.pivot_table(simulated_er, values='Reward', index=['State'],columns=['Action'], aggfunc=np.sum)
reward_matrix=reward_matrix[ self.action_list]
sns.heatmap(reward_matrix, cmap = 'Blues',annot=True, fmt ='.2f')
plt.title('reward_matrix')
plt.show()
plt.clf()
print('End of Preprocessing')
return self.training_data, transition_matrix, reward_matrix
```

Figure [5]. Reward- matrix and corresponding code

Policy and Parameters for Q learning

The Bellman Optimality Equation is utilized by the Q-learning algorithm to continuously adjust the Q-values for every state-action pairing up until the Q-function converges to the maximum Q-function. The Bellman Equation is used to calculate the worth of a state and assess how good it is to use that state. The maximum will be obtained in the optimal state.

The equation can be found below inf figure [2]. It finds our agent's next state by combining the present state and the reward associated with it, as well as the optimum reward and a discount rate that indicates its value to the current state. The model's learning rate influences how quickly or slowly it will learn.

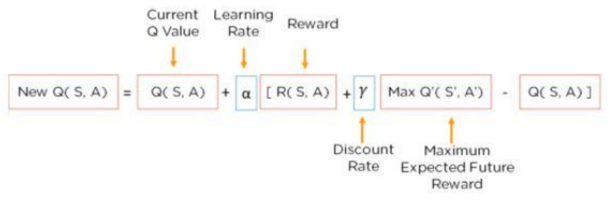

Current Q Value
Learning Rate
Reward
New Q(S, A) = Q(S, A) + α [R(S, A) + γ Max Q'(S', A') - Q(S, A)]
Discount Rate
Maximum Expected Future Reward

Figure [2]. Bellman equation

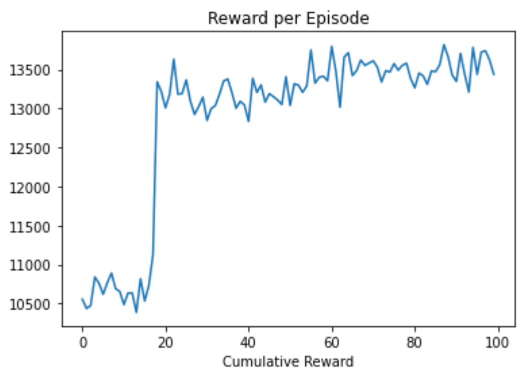
Due to the high stochasticity of the market environment, we employed a low learning rate of 0.1(range between 0 -1) to encourage a slower learning from the agent. And as we are focus on long term maximum reward, we went for a relatively high future reward discount of 0.8.

Keeping this in mind, we went for an epsilon greedy policy which will explore 10% (epsilon 0.1) of the time randomly due to the constant dynamic nature of the environment.

The epsilon-greedy approach was employed to choose the action that frequently gives the estimate of the optimal reward, with the goal of achieving an equilibrium between exploration and exploitation.

Q-learning Results

Through our experiment based on in-sample data we can observe that the maximum reward per episode is trending specially after 20 episodes to which the average reward per episode seems to go sideways. However it's worth noting the low learning rate and episode in this example. Using the last episode and plotting the buy and sell signal we can see an abundance of buy during positive momentum and more sparse selling signal during downward event.



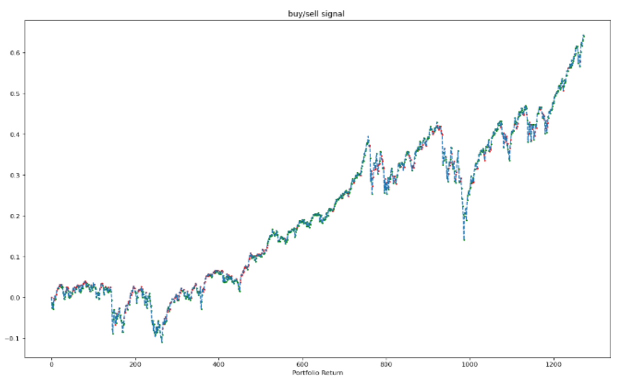
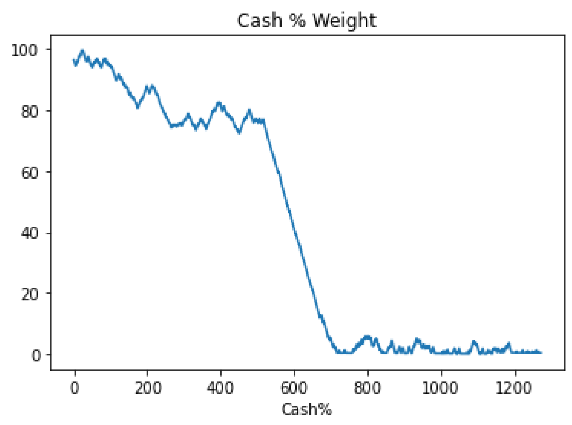


Figure [3]. Average Max Rewards per episode Figure [4]. agent Buy/Sell Position and market.

More realistic life situation where portfolio is constraint by itself on how much cash it can invest, the results are not very favourable with the strategy buy and selling only 1 stock at each signal. Therefore, do not try this strategy at home.



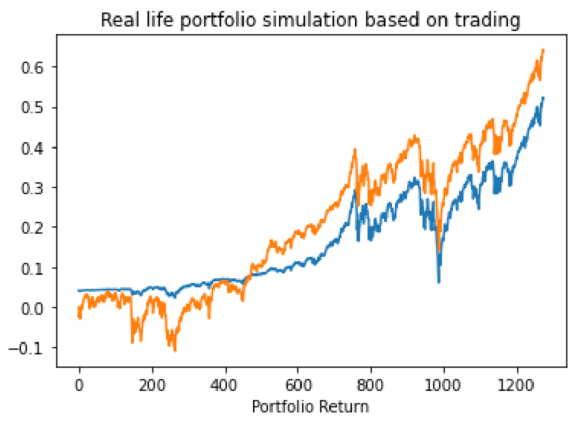
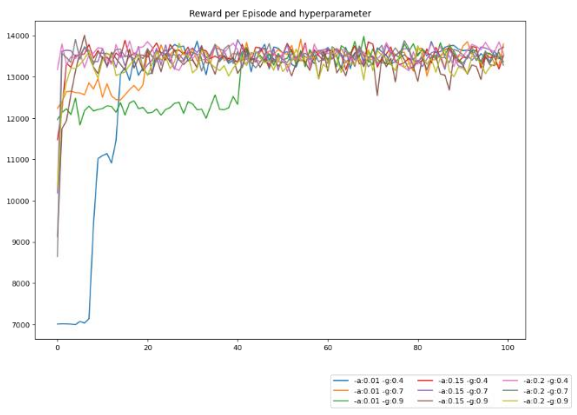


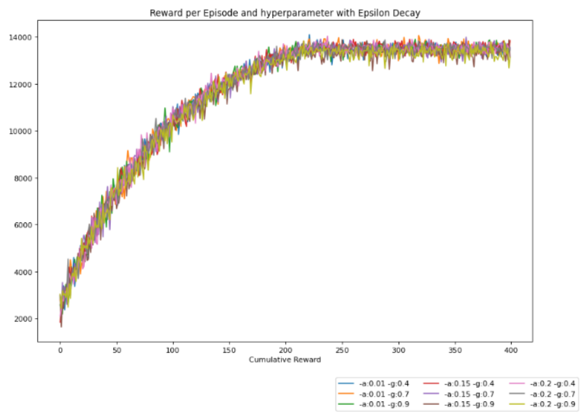
Figure [5]. Percentage of cash weights

Figure [6]. Graph of portfolio simulation based on trading. Market in Orange, vs Portfolio in Blue

Results for different Q-learning hyperparameter's and Policy.

We execute the Q-learning algorithm for 100 episodes per run as presented like a grid search. To determine the optimal hyperparameters that gives the best performance, gamma, ϒ would be adjusted to be between 0.5 and 0.99 and alpha α would range between 0.01 and 0.2. The quantity of ϒ does not appear to have a significant influence on the agents' performance. Learning rate, on the other hand, appears to have a higher influence.





In addition, we have run a further policy where epsilon decay as we go through each episode bringing a gradual move from exploration to exploitation. A further parameter that should be specified is the agent's greediness (ϵ) decay rate. We set ϵ decay rate of 0.99 and a starting epsilon of 0.9 As the agent's training progresses and it learns more about the environment in which it operates, these parameters are updated in accordance with the equation. For an agent in a stochastic environment, a unique reward may be given for the same action carried out, in the same state, during its transition in the next state.

Advanced reinforcement learning algorithm

For this next section, we chose to implement an advance reinforcement learning algorithm based on DQN. We will investigate Double DQN and duelling DQN and compare which gives the optimum results.

Advance reward function and the environment

For this section, we used a similar problem as above, but was we move from a grid view to a neural network we amended the environment with the following:

- The agent state uses the same set describe in the above, however for better representation in the network, we standardise the High, Low, and Open by the Close price and excluded the latter from the state description. The reason for the exclusion is due to the natural increase in price during the period and therefore we could end up in similar state at the start and end of the period but with high different close prices. The state observation is also enhanced with the portfolio number of shares and cash amount.
- The reward policy is not defined anymore by a Q-matrix but is calculated each time we perform a step. If the agent sells a position than the reward will be Buy price – Sell price. We have added a small negative reward for when the agent tries to sell and do not have enough shares.

DDQN and Duelling network

The state remains a snapshot as we believe that the momentum ratios will give an indication of the past trend without having to pass through a time series. In this retrospect we chose to implement a RNN network instead of the general LSTM network describe in financials Deep learning algorithm.

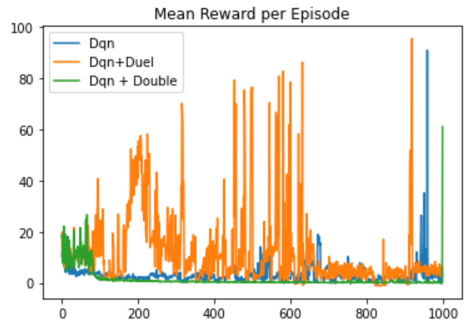
The off-policy reinforcement learning technique employs double Q-learning (DQN), a procedure that doubles its estimation in order to prevent overestimation issues from arising, which is common in the average Q-learning algorithm [9]. One of the main reasons to apply this would be to reduce the estimation and reward if the agent took a benefiting action by chance. With the intention of it is being easier for the network to discriminate between actions and the improvements in learning, duelling DQN is employed [10].

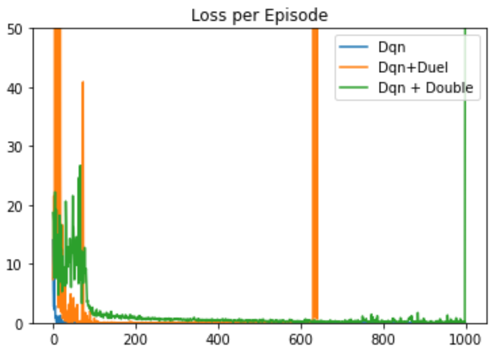
Atari

This written script utilizes the breakout environment, implementing deep Q-learning. Whilst the agent performs each action and travels through the environment, it understands and maps the environments observable state to a specific action. In each condition, an agent will choose an action which is a weighted reward based on the highest projected long-term reward. A Q-Learning Agent learns to fulfil its task in such a way that the proposed action optimises future rewards. This method is classified as an "Off-Policy" method, which means that its Q values are updated as if the best action was taken, even if it wasn't. For the breakout environment, a board moves across the floor at the bottom of the screen, bouncing the ball up into the air as it meets it. The purpose of this game would be to destroy the block at the top of the screen. The agent must be trained to control the board as to moves from left to right, returning the ball into the air and removing the blocks, after each interaction of the ball and board.

Results

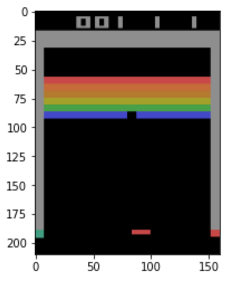
The experiment results do not tend to be very comprehensive when it come to the application of the DQN with improvement per the below. The loss functions seem to be more stable with a “drop” via the end for all the agent while the rewards do not show positive learning.





A few reasons may explain the following:

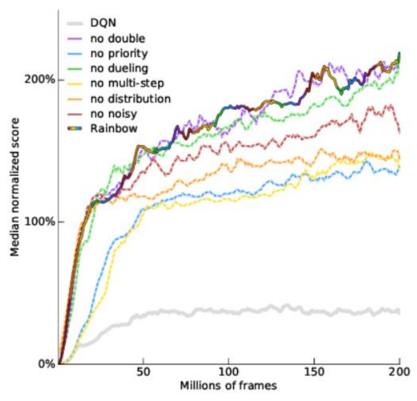
- The Reward have not been clipped which could type the scale when occurring outliers’ event and even shift the learning.
- Despite having a lost trending lower mostly, it does not mean that policy is the best. This can be since the neural network is not finding improvement to the policy. This is confirmed by the jump in loss at the end which could be an exploding gradient problem.
- **Atari**

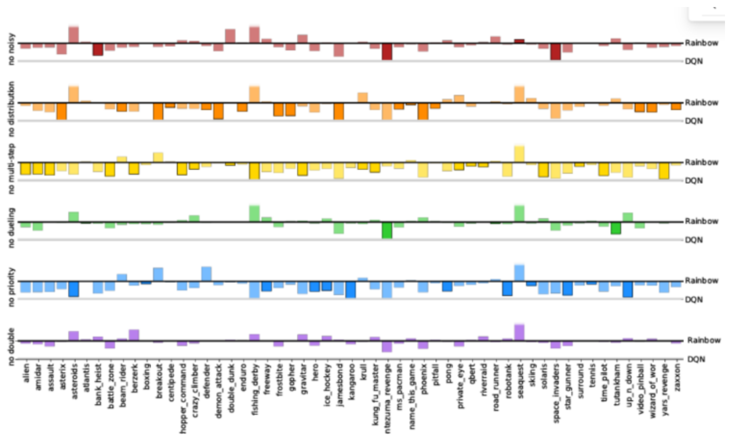


For this section of the project, I have opted to use the Breakout-V0 environment. Alongside Pong, this is highly popular as a first game to start working on the Atari arcade. The arcade game consists of a layer of bricks at the top of the screen, which needs to be hit by the ball without the agent dropping it on the way back or the agent will lose a turn.

To solve the problem, I based our decision on the research paper Rainbow: Combining Improvements in Deep Reinforcement Learning. The paper focuses on the use of Rainbow DQN, which combines several improvements into a single learner (Double Q-Learning to tackle overestimation bias, Prioritized Experience Replay to prioritize important transitions, duelling networks, multi-step learning, distributional reinforcement learning instead of the expected return, noisy linear layers for exploration). The conclusion of the experiment is that a rainbow model will outperform based on simulation of multiple Atari games as demonstrated below. However, making those improvements would significantly increase sensitivity to hyperparameters combinations. This motivated us to reproduce this analysis with only Prioritized Experience replay, Duelling and Double learning.

Result from research paper

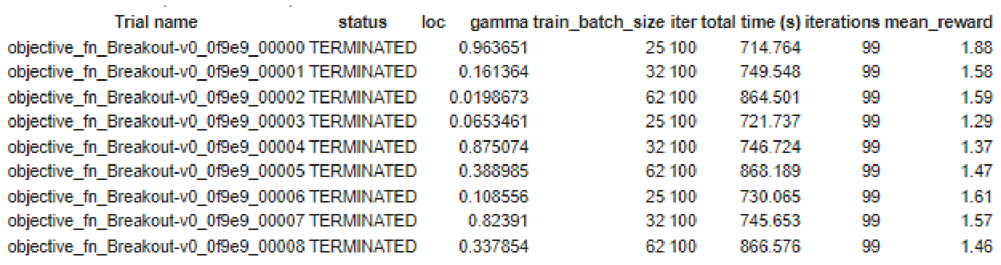




If we go by the gross assumption that more improvement will yield better the result than we optimise the learning batch size and the gamma hyperparameter on the model with the three improvements. In general, smaller or larger batch size doesn't guarantee better convergence. Batch

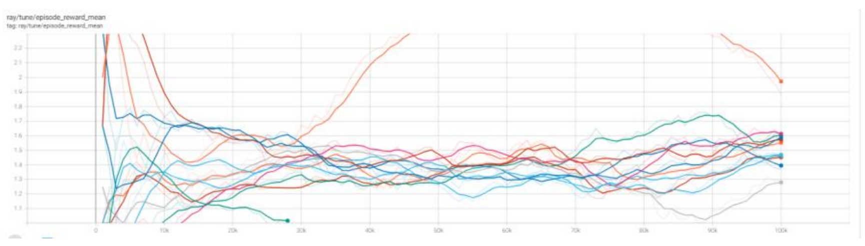
There is a trade-off for bigger and smaller batch size which have their own disadvantage, making it a hyperparameter to tune. We have also added Gamma as a parameter to tune while adding a learning rate of 0.001. In hindsight, tuning the three parameters may have been better as I cautiously put a lower learning rate to avoid the risk of gradient descent to overshoot. However, a too low learning rate may have a too slow gradient descent.

The tuning returned the best results for a gamma of 0.9636 and a batch_size of 25. Oddly we can't observe a pattern for gamma that would necessarily yield better results and its worth noting that the mean reward is very low. One of the reasons is the 100k steps with 100 iteration is too low for the agent and therefore any results below could be off the convergence.



Trial name	status	loc	gamma	train_batch_size	iter	total time (s)	iterations	mean_reward
objective_fn_Breakout-v0_0f9e9_00000	TERMINATED		0.963651	25	100	714.764	99	1.88
objective_fn_Breakout-v0_0f9e9_00001	TERMINATED		0.161364	32	100	749.548	99	1.58
objective_fn_Breakout-v0_0f9e9_00002	TERMINATED		0.0198673	62	100	864.501	99	1.59
objective_fn_Breakout-v0_0f9e9_00003	TERMINATED		0.0653461	25	100	721.737	99	1.29
objective_fn_Breakout-v0_0f9e9_00004	TERMINATED		0.875074	32	100	746.724	99	1.37
objective_fn_Breakout-v0_0f9e9_00005	TERMINATED		0.388985	62	100	868.189	99	1.47
objective_fn_Breakout-v0_0f9e9_00006	TERMINATED		0.108556	25	100	730.065	99	1.61
objective_fn_Breakout-v0_0f9e9_00007	TERMINATED		0.82391	32	100	745.653	99	1.57
objective_fn_Breakout-v0_0f9e9_00008	TERMINATED		0.337854	62	100	866.576	99	1.46

During tuning, we allowed TensorBoard to visualise the results for all combinations.



References

[1] Capital.com. 2022. *What is Trade: Meaning and Definition*. [online] Available at: https://capital.com/trade-definition.

[2]. Ft.com. 2022. Volatility: how ‘algos’ changed the rhythm of the market. [online] Available at: https://www.ft.com/content/fdc1c064-1142-11e9-a581-4ff78404524e.

[3]. Techopedia.com. 2022. What is Deep Reinforcement Learning (Deep RL)? - Definition from Techopedia. [online] Available at: https://www.techopedia.com/definition/34029/deep-reinforcement-learning-deep-rl.

[4]. SearchEnterpriseAI.com 2022. What is Reinforcement Learning? A Comprehensive Overview. [online] Available at: https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning#:~:text=Reinforcement%20learning%20is%20a%20machine,learn%20through%20trial%20and%20err.

[5]. KDnuggets. 2022. Three Things to Know About Reinforcement Learning - KDnuggets. [online] Available at: <https://www.kdnuggets.com/2019/10/mathworks-reinforcement-learning.html> [Accessed 24 April 2022].

[6]. Arxiv.org. 2022. [online] Available at: <https://arxiv.org/pdf/1509.06461.pdf.

[7]. En.wikipedia.org. 2022. Markov decision process - Wikipedia. [online] Available at: https://en.wikipedia.org/wiki/Markov_decision_process

[8]. Kaggle dataset

[9]. Medium. 2022. Deep Reinforcement learning: DQN, Double DQN, Dueling DQN, Noisy DQN and DQN with Prioritized…. [online] Available at: https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823

[10]. Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. and de Freitas, N., 2022. Dueling Network Architectures for Deep Reinforcement Learning. [online] arXiv.org. Available at: https://arxiv.org/abs/1511.06581

Figure [1]. Google.com. 2022. Redirect Notice. [online] Available at: https://www.google.com/url?sa=i&url=https%3A%2F%2Fkitrum.com%2Fblog%2Freinforcement-learning-for-business-real-life-examples%2F&psig=AOvVaw2QNd9Q9aX1aXdUgcFLXmVP&ust=1650707543967000&source=images&cd=vfe&ved=0CAwQjRxqFwoTCNiN2_6yp_cCFQAAAAAdAAAAABAD

Figure [2]. 2022. [online] Available at: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-q-learning

Github linked : https://github.com/Linakatabicity/DLModule