Tabular Q-Learning vs DQN

Athanasakis Evangelos 2019030118
Yiorgos Fragkogiannis 2019030039

July 20, 2024

1 Introduction

The purpose of this project is to implement Tabular Q-Learning and Deep Q-Learning in a stock portfolio environment and analyze the scalability of the these two algorithms as the number of stocks increases. The main goal is to (i) assume an unknown environment, and solve small version of the problem and learn while optimizing, and (ii) to solve larger versions of the problem using Deep Neural Networks as approximators.

2 Task 1

For this environment, the parameters of the Markov chain driving the price of each stock are unknown. While the agent is aware that each stock has two states, H(High), L(Low), it is not aware of the stock rewards nor the transition probabilities.

In this task, two separate "toy" scenarios, whose optimal policies are already known from the previous phase of the project (policy iteration algorithm) are used. These environments are used as ground truth to make sure that the implementation of the Tubular Q-learning algorithm is correct.

2.1 Environments

Environment 1: This environment consists of N = 2 stocks, transition probabilities for stock 1: $p^1_{HH} = 0.5$, $p^1_{HL} = 0.5$, $p^1_{LH} = 0.5$, $p^1_{LL} = 0.5$ and for stock 2: $p^2_{HH} = \frac{9}{10}$, $p^2_{HL} = \frac{1}{10}$, $p^2_{LH} = \frac{1}{10}$, $p^2_{LL} = \frac{9}{10}$, rewards for stock 1: $r^1_H = 10\%$, $r^1_L = -2\%$ and for stock 2: $r^2_H = 5\%$, $r^2_L = 1\%$, $\gamma = 0$ and transaction fee = -0.9

Environment 2: This environment consists again of N = 2 stocks. The environment's transition probabilities, rewards and γ are similar as for the environment above. The transaction fee for this scenario is much more reasonable with a value of -0.01.

2.2 Implementing the Tabular Q-Learning algorithm

In order to find the optimal policy that provides the best rewards for the created environments we use the Tabular Q-Learning algorithm. The Hyper-parameters of this algorithm are:

- Learning Rate (alpha)
- Discount Factor (gamma)
- Alpha Decay (exponentially decreased)
- Epsilon Decay

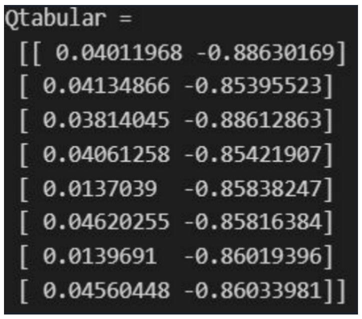
It should also be noted that during training the learning rate decreases as time passes to make learning more stable and minimise oscillations. Moreover, many episodes of medium size (100 steps each) where used for the training.

To optimise the algorithm's performance, an Optuna study was implemented. This study finds the optimal hyper-parameters for our algorithm by minimising the output of an objective function that was defined. The output of the objective function is a sum of the number of episodes needed for the convergence of the algorithm (**optimize convergence speed**), the Mean Square Error of the found Q table elements compared to the elements of the optimal Q table found with Policy iteration (**optimise accuracy**) and the number of mistaken policy actions in the outputted policy compare to the optimal policy found (**make sure to always find correct policy**). The above elements were weighted appropriately before they were summed since they are not all equally important, i.e. convergence speed is much less important than finding a correct policy and accurate Q table values, hence it has a much lower weight.

After many tests, the best **learning rate** was found equal to $\alpha \approx 0.7$, the best learning rate decay rate (**alpha decay**) equal to ≈ 0.004 and the best exploration decay rate (**epsilon decay**) equal to ≈ 0.95 with epsilon starting from 1.

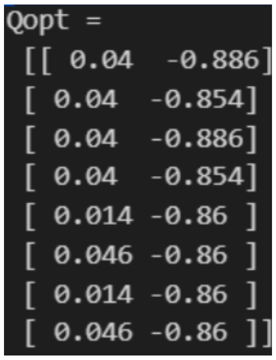
2.3 Verifying Q-Learning

In order to verify that the found policies are correct, they are compared to the optimal policies found in Phase 1 of this project using Policy Iteration (these are considered ground truth). The algorithm in both cases seems to converge to the optimal policy for a number of episodes (epochs) of around ≈ 3000. The policies are indeed optimal:



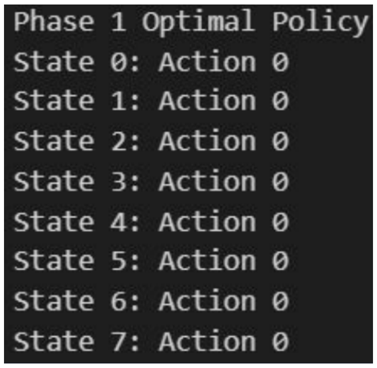
```
Qtabular =
 [[ 0.04011968 -0.88630169]
 [ 0.04134866 -0.85395523]
 [ 0.03814045 -0.88612863]
 [ 0.04061258 -0.85421907]
 [ 0.0137039  -0.85838247]
 [ 0.04620255 -0.85816384]
 [ 0.0139691  -0.86019396]
 [ 0.04560448 -0.86033981]]
```

(a) Q table of Q-Learning implementation for environment 1



```
Qopt =
 [[ 0.04  -0.886]
 [ 0.04  -0.854]
 [ 0.04  -0.886]
 [ 0.04  -0.854]
 [ 0.014 -0.86 ]
 [ 0.046 -0.86 ]
 [ 0.014 -0.86 ]
 [ 0.046 -0.86 ]]
```

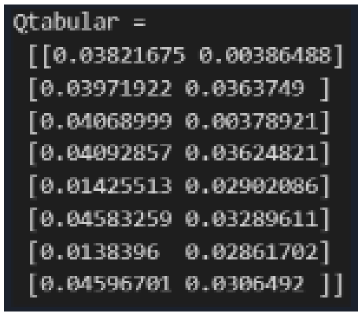
(b) Optimal Q table



```
Phase 1 Optimal Policy
State 0: Action 0
State 1: Action 0
State 2: Action 0
State 3: Action 0
State 4: Action 0
State 5: Action 0
State 6: Action 0
State 7: Action 0
```

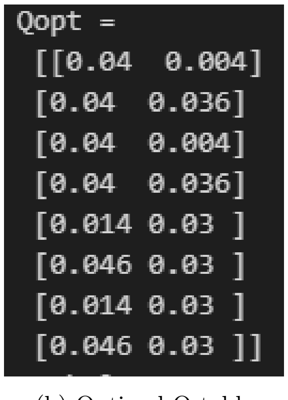
(c) Optimal Policy

Figure 1: Outputs for environment 1



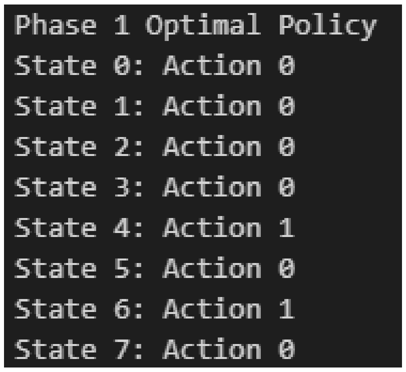
```
Qtabular =
 [[0.03821675 0.00386488]
 [0.03971922 0.0363749 ]
 [0.04068999 0.00378921]
 [0.04092857 0.03624821]
 [0.01425513 0.02902086]
 [0.04583259 0.03289611]
 [0.0138396  0.02861702]
 [0.04596701 0.0306492 ]]
```

(a) Q table of Q-Learning implementation for environment 2



```
Qopt =
 [[0.04  0.004]
 [0.04  0.036]
 [0.04  0.004]
 [0.04  0.036]
 [0.014 0.03 ]
 [0.046 0.03 ]
 [0.014 0.03 ]
 [0.046 0.03 ]]
```

(b) Optimal Q table



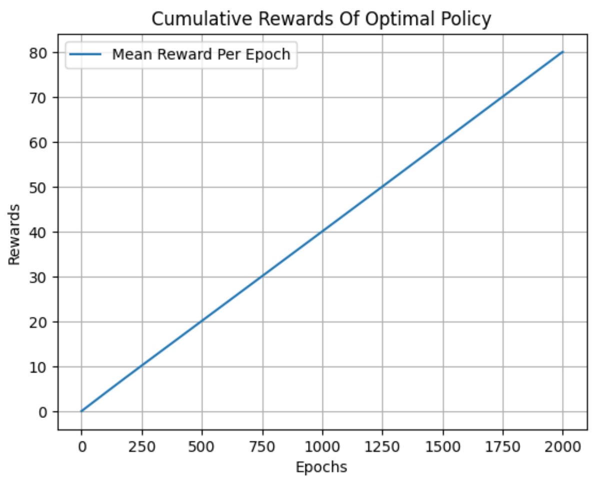
```
Phase 1 Optimal Policy
State 0: Action 0
State 1: Action 0
State 2: Action 0
State 3: Action 0
State 4: Action 1
State 5: Action 0
State 6: Action 1
State 7: Action 0
```

(c) Optimal Policy

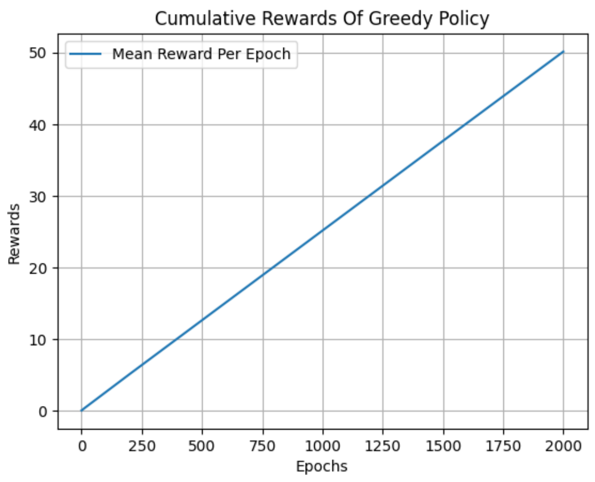
Figure 2: Outputs for environment 2

2.4 Comparison of Optimal Policy Found With Greedy Policy

In the next step, a comparison of the optimal policy found by the Tabular Q-Learning Algorithm was made with a greedy policy of always performing the action *Switch*. The cumulative rewards of the experiment are shown below and it is clear that the policy found by the Q-Learning algorithm is performing much better.



(a) Cumulative Rewards Of Optimal Policy



(b) Cumulative Rewards Of Greedy Policy

Figure 3: Comparison Of Optimal Policy With Greedy Policy

3 Task 2

In this section of the project the same logic is followed as for task 1 but for a larger environment.

A function was created for the generation of the environment were:

- Rewards are chosen uniformly in [-0.02, +0.1].
- Transition probabilities are equal to 0.1 for half the stocks and 0.5 for the other half.

Tabular Q-learning was applied to the medium scenario in order to show the algorithm learns to perform optimally here as well. For this case an Optuna study was used once again to find the optimal hyper-parameters for the algorithm. The outputs that are generated for an environment of 5 stocks are:

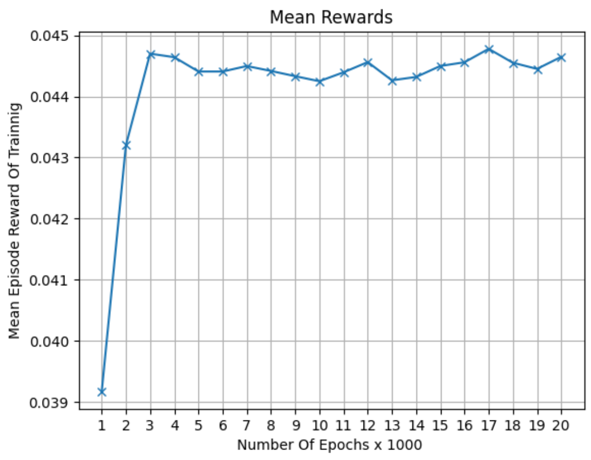
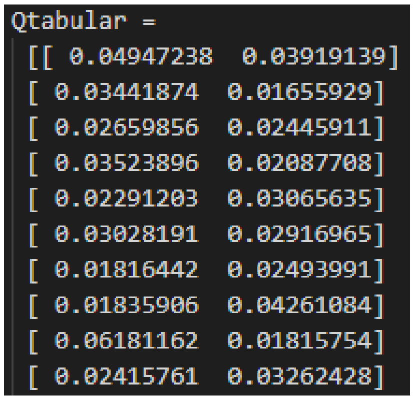


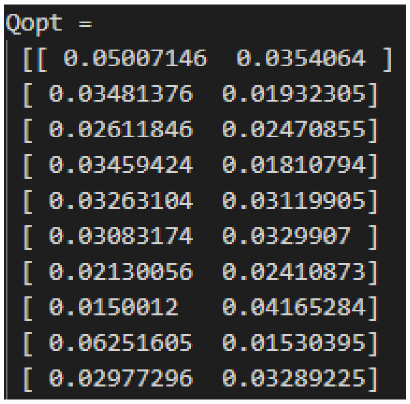
Figure 4: Mean Reward per round

By comparing the generated Q values good accuracy can be observed. Since there are too many states only the 10 first Q values are provided:



```
Qtabular =
 [[ 0.04947238  0.03919139]
 [ 0.03441874  0.01655929]
 [ 0.02659856  0.02445911]
 [ 0.03523896  0.02087708]
 [ 0.02291203  0.03065635]
 [ 0.03028191  0.02916965]
 [ 0.01816442  0.02493991]
 [ 0.01835906  0.04261084]
 [ 0.06181162  0.01815754]
 [ 0.02415761  0.03262428]
```

(a) Q table of Q-Learning implementation for medium environment



```
Qopt =
 [[ 0.05007146  0.0354064 ]
 [ 0.03481376  0.01932305]
 [ 0.02611846  0.02470855]
 [ 0.03459424  0.01810794]
 [ 0.03263104  0.03119905]
 [ 0.03083174  0.0329907 ]
 [ 0.02130056  0.02410873]
 [ 0.0150012   0.04165284]
 [ 0.06251605  0.01530395]
 [ 0.02977296  0.03289225]
```

(b) Optimal Q table

Figure 5: Outputs for medium environment

4 Task 3

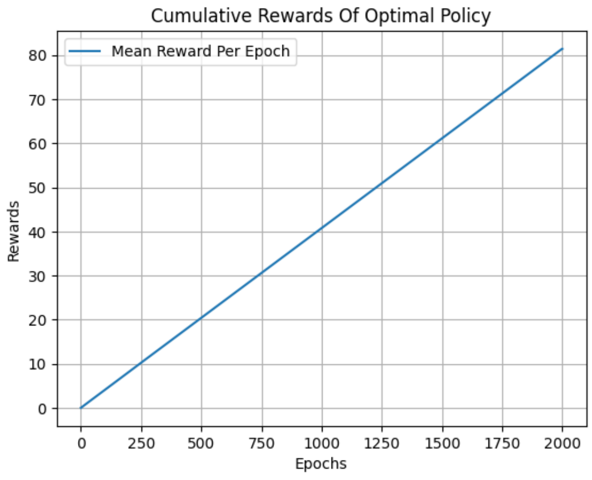
In this section of the project a **Deep Q-learning Neural Network** is implemented to be able to tackle large environments. In this scenario, the states of the stocks are unknown.

4.1 Network Architecture

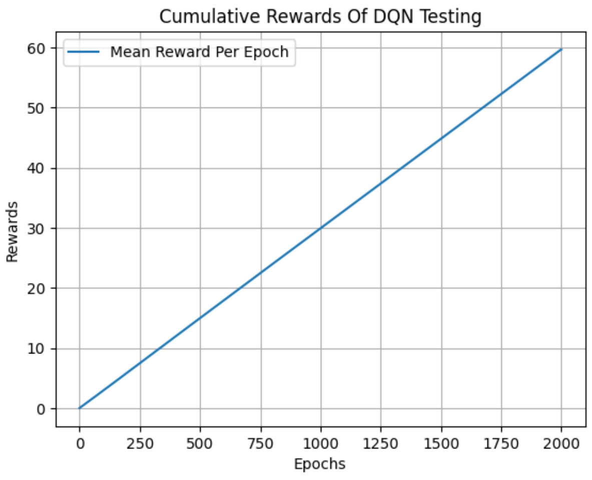
Input layer: For the input layer, the number of nodes is the same as the number of stocks plus one (num of nodes = num of stocks + 1). For each stock the mean reward it gets up to this point is provided to one of the nodes and the final node gets as input the stock that is currently being used.
Hidden layers: One hidden layer is being used, which has the same number of nodes as the input layer.
Output layer: In the output layer 2 nodes are being used, one for each action (keep//switch).
Hyper-parameters:

- Optimizer: Adam Optimizer
- Loss Function: MSE
- Activation Function: ReLU
- Learning rate = 0.01
- Replay Memory Size = 10000
- Batch Size = 64

To figure out whether or not our network runs optimally we compare the rewards its gets to the rewards the optimal policy provided from Tabular Q-Learning gets. We can see that the DQN does not converge to the optimal policy since the rewards that the trained model gets do not match the ones the optimal policy can find. This is something that can be reduced by using even more epochs(with the expense of computational time) and further fine-tuning the networks hyper-parameters. Using a second hidden layer can also have an impact on the output. Despite all these, the DQN can find a pretty good policy that can approach the optimal one. This can be seen in the following plots of the rewards of the Tabular Q-Learning (Optimal) and the DQN ones.

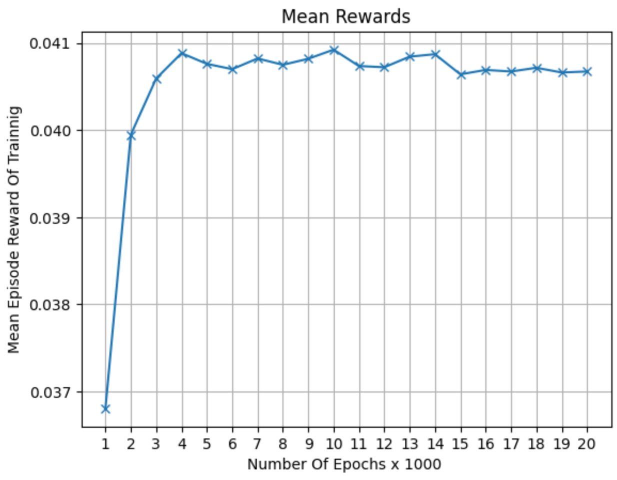


(a) Cumulative Rewards of Optimal Policy Found By the Tabular Q-Learning During Testing for 2000 episodes

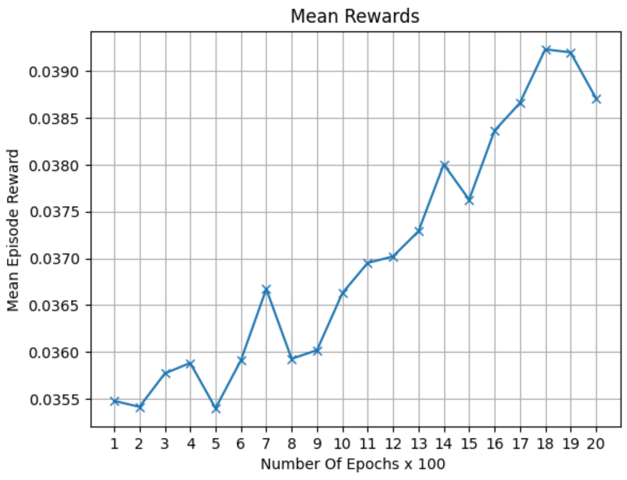


(b) Cumulative Rewards of DQNs Policy During Testing for 2000 episodes

Figure 6: Comparison of the cumulative profit of the testing of Tabular Q-Learning with DQN



(a) Mean Episode Reward Of Training Of Tabular Q-Learning



(b) Mean Episode Reward Of Training Of DQN

Figure 7: Comparison of the mean episode reward of the training of Tabular Q-Learning with DQN

5 Conclusions

The goal of this project was to compare the capabilities of two famous algorithms, Tabular Q-Learning and DQN, within a stock portfolio management environment to analyze their scalability and performance as the number of stocks (environment) increases.

For **small environments** (task 1) with known optimal policies, Tabular Q-Learning proved effective. By using an Optuna study, the hyper-parameters of the algorithm were fine-tuned to achieve convergence within a reasonable number of episodes, ensuring the accuracy of the Q-values and the optimality of the derived policies. The results validated the implementations, as the policies closely matched the known optimal policies, and the cumulative rewards indicated superior performance compared to a simple greedy policy. For **medium environments**, Tabular Q-Learning still demonstrated good performance, with the Hyper-parameters again optimized via Optuna. Despite the increase in state space complexity, the algorithm maintained high accuracy in estimating Q-values, which was evident from the comparison of Q-tables and the cumulative rewards, despite being a little slower. In **large environments**, where the state space becomes intractable for Tabular Q-Learning, a Deep

Q-Learning Neural Network was implemented. Although the DQN did not always converge to the optimal policy within the tested epochs, it still performed reasonably well, suggesting that with further fine-tuning and increased training epochs, it could approximate the optimal policy more closely. The cumulative rewards and mean episode rewards from the DQN indicated its capability to learn a robust policy, even if it was not always optimal.

Overall, the above experiments highlight the trade-offs between Tabular Q-Learning and DQN. Tabular Q-Learning is effective for small to medium environments where the state space is manageable, providing accurate and optimal policies. However, as the environment scales and the states of the stocks are unknown, DQN becomes necessary due to its ability to approximate Q-values in large state spaces, despite requiring more computational resources and careful tuning of network parameters. Future work could explore further improvements in DQN performance through deeper networks, advanced exploration strategies, and more sophisticated training regimes.