Technical University of Crete
School of Electrical and Computer Engineering
Course: **Reinforcement Learning and Dynamic Optimization**

Assignment 3 - Phase 2

Angelopoulos Dimitris - 2020030038
Michalochristas Konstantinos - 2020030111

For the second phase of the exercise we assume we do NOT have full environment knowledge, unlike phase 1. We will solve small and larger versions of the problem using Q-Learning and Deep Q-Learning.

Task 1 Initially we have to develop the (tabular) Q-Learning algorithm, assuming that we do not know the stock reward r_i^H, r_i^L nor the Markov probabilities $P_{xx'}^i$ where $x \in \{H, L\}$ for any stock $i = 0, \ldots, N-1$.

The algorithm goes as follows. First of all, we begin from a random state $s \in \mathcal{S}$. Then we choose an action a based on the ϵ-greedy algorithm. Having knowledge of the state and the action, we let the environment evolve to observe the next state s' and receive a reward r. Getting the tuple of information (s, a, r, s'), we perform an update step as follows:

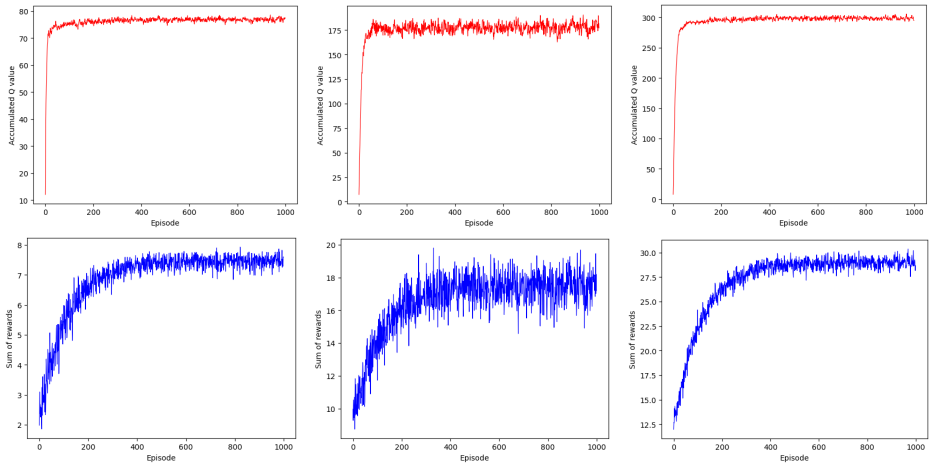
$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s,a))$$

After performing the step, we update the current state and repeat for a certain time horizon T. This means that we have to find a sufficient time horizon to guarantee that our agent finds the optimal policy. We set the time horizon T to be the following: $T = kT_{eff}$. Where T_{eff} is a function $T_{eff} = \lceil 1/(1-\gamma) \rceil$. This function increases for larger γ meaning that the agent has to explore more in each episode since the rewards do not get discounted as much. The parameter k is for tuning and it is chosen to be $k = 10N$.

Question 1 Assuming that that $N = 2$ and $r_0^H = 2r_1^H = 0.06$. We set the stock rewards for the Markov L states to be $r_i^L = -r_i^H$ and $P_{xx'}^i = 0.5$, for $i = 0, 1$. This parametrization yields an optimal policy (computed via PI) to always keep the stock we are holding.

Assuming now that our agent does not have sufficient information of the environment. We run the Q-Learning algorithm following the logic explained above. The policy of this algorithm is also optimal and is indeed to keep the stock you hold.

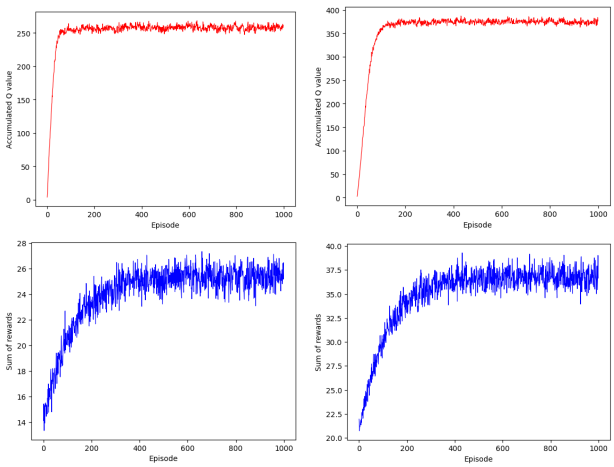
Question 2 Now we will set $\gamma = 0.9$ and experiment for $N = 2, 3$ stocks to make sure that our model-free agent accurately estimates the optimal action-value function $Q^*(s, a) \approx \mathbb{E}[\hat{Q}^*(s, a)]$ (and thus the optimal policy). We consider 3 separate cases, (i) one case where the optimal policy is in some states to switch and in some to stay for, $N = 2$ stocks and two cases for $N = 3$ stocks; (ii) one where the stocks are competitive and (iii) one where one of the stocks is much better than the rest.



First of all we can verify that our agent finds a policy that yields a consistent amount of accumulated reward over a horizon of training episodes (blue graph). The agent yields a constant amount action-value functions (red graphs). The accuracy of Q-Learning for every case 100% except the non trivial one where we get 91.6% accuracy. Thus, when the accumulated Q-Value and reward have a constant mean that means that the agent chooses the action that yields a close to maximum reward.

More specifically for case (i) the Q-Learning policy is the same as the PI policy. For case (iii) the Q-Learning policy is also optimal. But in case (ii) the policy (even though it yields a Q-Value estimate close to the V-value of PI) is almost the same except for 2 states meaning that the Q-Learning agent learns to play almost as good as the PI agent in a non-trivial case where the decision of switching or staying is hard. This suboptimality might result from the fact that Q-Learning suffers from overestimation issues [1].

Tasks 2 After running some more experiments we can see that the Q-Learning agent performs (almost) optimally. More specifically for $N = 4, 5, 6$ we get approximately 90-95% policy accuracy.



Tasks 3 Now we will use the Deep Q-Learning algorithm that uses a Multi-Layer Perceptron(MLP) as a function approximator for the action-value function $Q(s, a; \boldsymbol{\theta})$. This algorithm consists of 2 Deep Q-Networks, one that is updated constantly and one other, called Target Network, that is updated periodically. For this to work we have to save every new sample we get from the environment to a replay memory $\mathcal{D}$. When we collect enough data, we start sampling a random batch $\mathcal{B}$ of them. With that batch we train our MLP while we interact with the environment. Using that $\mathcal{B}$ we perform a gradient descent step on the the Q-Network. The Target Q-Network has its parameters updated every 50 episodes. The hyperparameters for the MLPs used are: $\alpha = 0.001$, $|\mathcal{B}| = 64$, $episodes = 1000$.

Neural Network Architecture The NN is a fully connected MLP with input layer of size $N+1$, 2 hidden layers of size $|H| = 64$, an output layer of size N and ReLU activation functions. Where N is the amount of stocks we have.

1. Input: In the case where we know the state s we are currently at; we feed the MLP the state s which contains the information of the stock we are currently holding and the Markov H, L states for every stock. In the case where we do not know the state s and we just have continuous values (of stocks daily profits for example) then we give them as input plus the stock we hold at the current time step.

2. Output: The MLP will give us N Q-Values. Every Q-Value corresponds to an action.

3. Optimizer: Stochastic Gradient Descent. This is a variation of Gradient Descent that computes a gradient estimate using batch $\mathcal{B}$.

4. Loss Function: Mean Squared Error (MSE) of the target and the network output.

Now we will simulate the results of the Deep Q-Learning agent and compare the to the results of (tabular) Q-Learning. Initially we plot the accumulated reward versus the episode and compare.

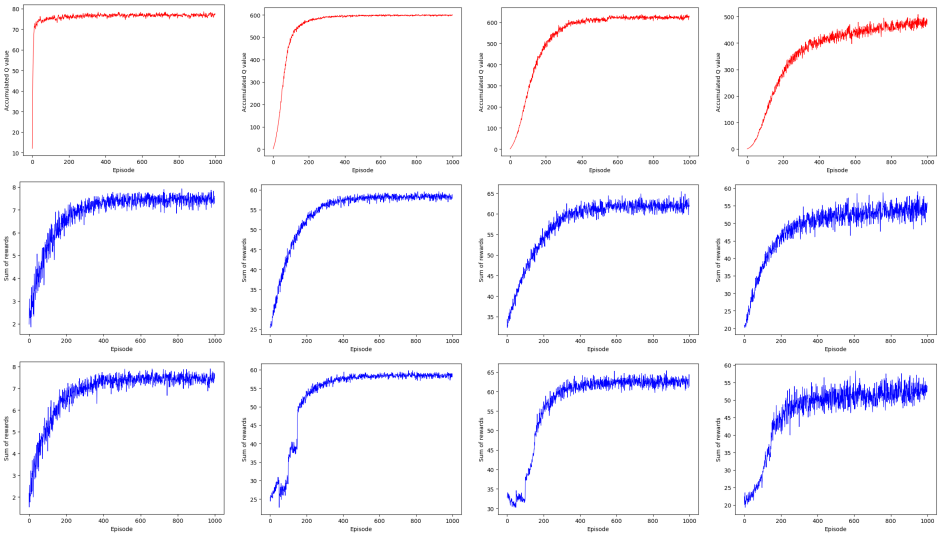


Figure 1: First and second rows are Q-Learning graphs of accumulated Q/reward. Third row contains the plots of the accumulated reward of Deep Q-Learning versus the episodes. First column is for $N = 2$. Next three are for $N = 6, 7, 8$.

We can verify that our Deep Q-Network correctly approximates the action-value function and thus it yields a policy very close to the Q-Learning. Also for the cases of $N = 6, 7, 8$ we can notice that the DQN algorithm converges in less episodes Note: as N increases we need to increase the amount of episodes and let the agent to continue learning longer.

Also we recorded the time taken for each agent to complete 1000 episodes versus the amount of stocks N.

	T (Q-Learning)	T (DQ-Network)
$N = 2$	$15s$	$\approx 8m$
$N = 6$	$\approx 5m$	$\approx 40m$
$N = 7$	$\approx 13m$	$\approx 43m$
$N = 8$	$\approx 31m$	$\approx 60m$

We can observe that the rate with which the time increases is much faster for the Q-Learning algorithm even though The DQN method seems slower for small N. As we increase the number of stocks Q-Learning will become even slower and the Neural Network approach will be consistent. These said, we would use tabular Q-Learning for small state spaces and DQN methods for larger more complex or continuous state spaces.

The Colab code for the assignment can be found in the following link.

References

[1] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum, 1993.