Reinforcement Learning and Dynamic Optimization

Project 1: VNF Placement --- Phase 2

George Konofaos - 2018030175

Task 1

Assume you know still that each VNF traffic demand has two states (H,L), but you **do not** know the parameters of the Markov chain traffic patterns for each VNF. Namely, you do not know: λH,λL nor p_{xx} , where xε{H,L}. Implement (tabular) Q-learning for the small (N=2, M = 2) scenarios you chose at Phase 1, and ensure that Q-learning eventually learns to perform optimally.

The first task we are assigned is to implement is tabular Q-learning for the base case N=2,M=2.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max a' Q(s',a') - Q(s,a)]$$

Implementation Details:

- **State and Action Space**

  The state space consists of all combinations of traffic demand states for the two VNFs, resulting in four possible states: (HH, HL, LH, LL). The action space includes the possible placements of VNFs in physical nodes, namely M^N

- **Reward Function**

  The reward function is designed to minimize costs associated with power consumption, reconfiguration, and SLA (Service Level Agreement) violations. It combines these factors into a total cost
  Important note: In my implementation i refer to reward as the cost , hence i want to minimize the reward.

$$Total\ Cost = wON \times Power\ Cost + wR \times Reconfiguration\ Cost + wSLA \times SLA\ Violation\ Cost$$

- **Implementation**

  The Q-learning algorithm implementation involves initializing the Q-table, selecting actions using an epsilon-greedy policy, updating the Q-values based on observed rewards and state transitions, and ensuring convergence over multiple episodes.

Simulation Setup:

- **Transition probabilities:** trans_probs=0.6 for staying in the same state, and 0.5 for transitioning to a different state, this is a sample example for the sake of the environment and trying to create a realistic field to train on.

- **Weights and costs:**Weights will be adjusted to the scenarios i will use to showcase that Q-learning works and will be discussed later. Costs follow the same logic as assignment 3, cost_on=1,cost_r=1,cost_SLA=1.

- **Hyperparameters** :

  epsilon=1.0 since we do not know the markov chain traffic patterns we need to do more exploration early on.

  epsilon_decay=0.995 is the value that the epsilon will be mult by after each update forcing it to explore less and less.

  epsilon_min=0.1 this will ensure that the algorithm will still have a very low chance of exploring even at the very end of the process.

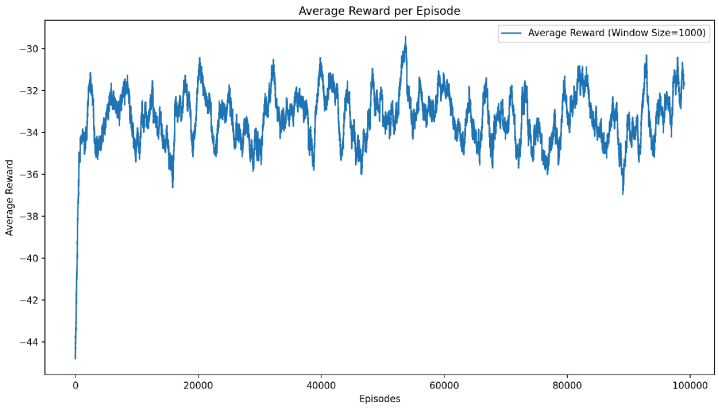
  Episodes: I have 100000 episodes for each experiment and each episode ends when np.random.rand() < 0.1 since we have non terminating episodes.Change after the Presentation: i changed the terminating case to **100 steps** because as i was corrected, the mean episode lasted 10 steps which could be not enough for Q-learning to converge.

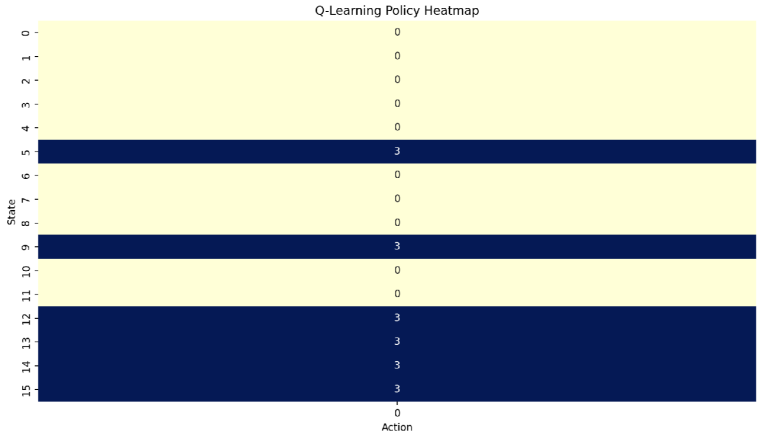
Results and Analysis

To showcase the rewards i decided to look at the avg. reward(loss) per episode since cumulative reward wasn't providing info regarding convergence.

Scenarios used: [W_on, W_r,W_sla]

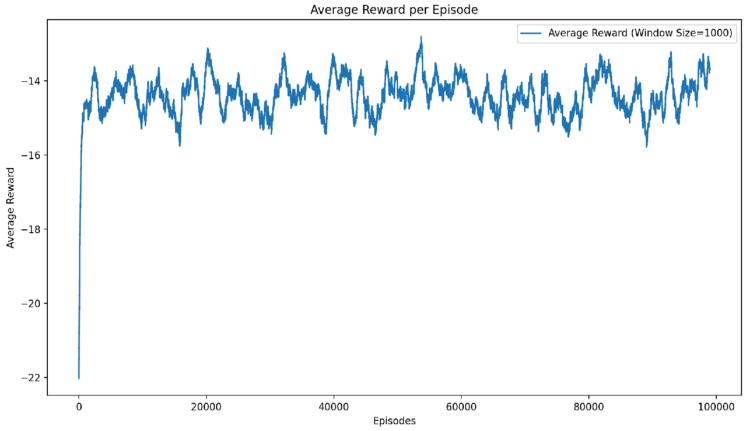
1.[3,1,1] {expected policy: Group all}

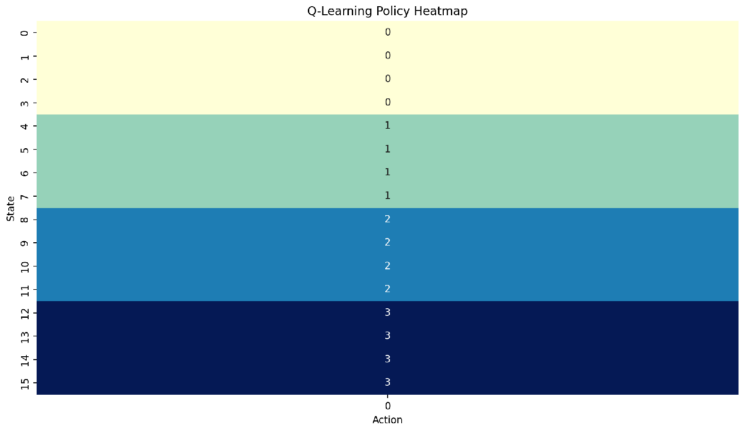




Observations:As we can see in this scenario , Q-learning converges to the Optimal Policy. The values seem to converge to a range of values that i would assume is varied due to some statistical randomness. We can confirm it converges from the heatmap since we see that it has found the optimal policy choosing the actions 0:{0,0} and 3{1,1} to group them in one physical node.

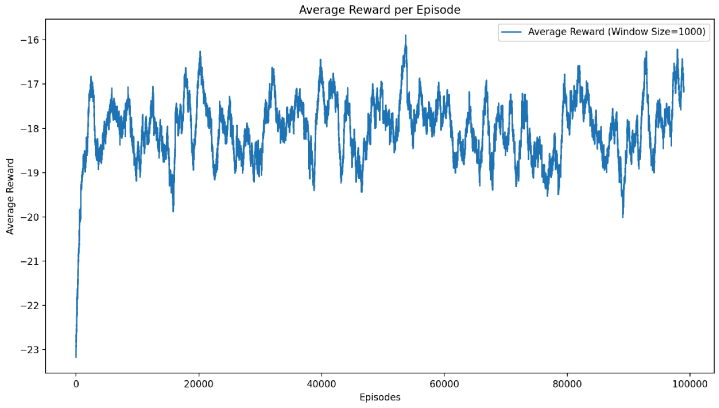
2.[1,3,1]{expected policy: Maintain the same locations}

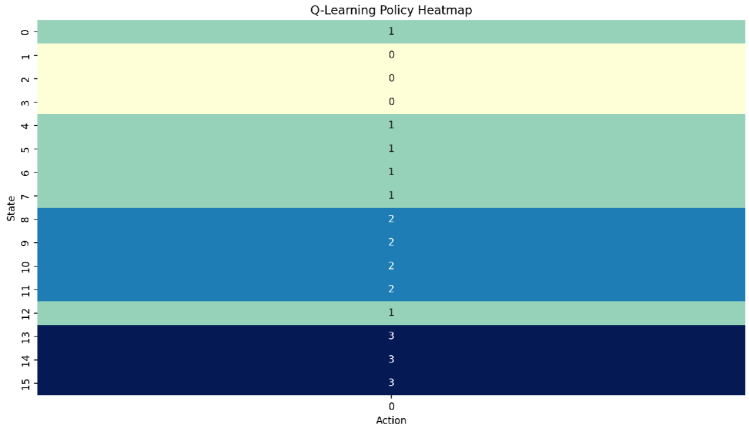




Observations: Same as the above example we can see the avg. reward (loss) converging to a range of values while starting from lower.Here the optimal policy is to not reconfigure the nodes and thus Q-learning finds the optimal policy leaving the VNFs at their location.

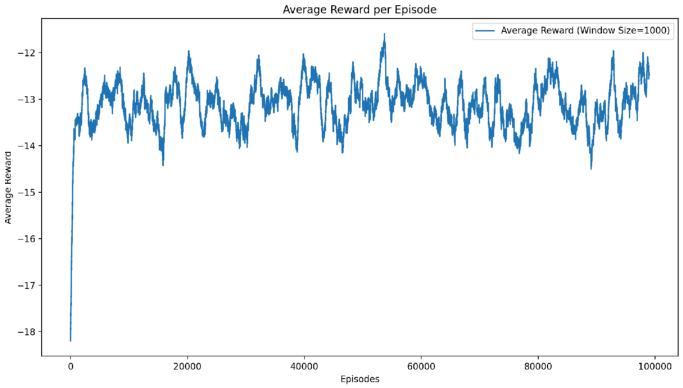
3.[1,1,3]{expected policy: Split VNFs when violation would occur}

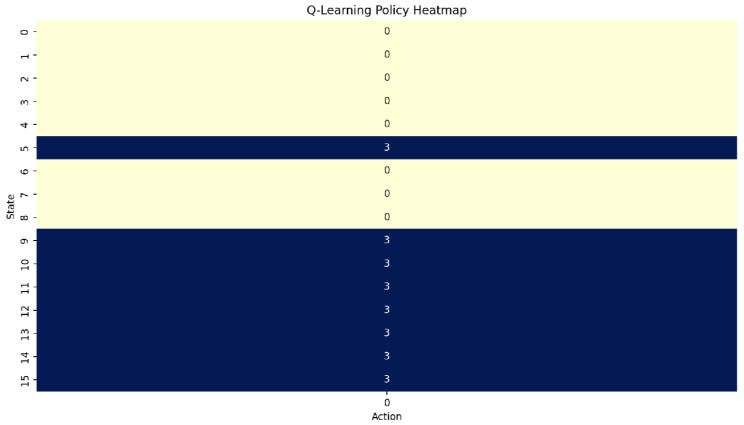




Observations: The same holds for this example, but we can see that the range seems to be bigger. Im not confident that this is the optimal policy and it is probably because it prioritizes not violating the SLA but seems to be unsure about the rest, it decided to go with example 2 for the rest of the cases and leaving the Vnfs where they are, but it is important that in state 0{0,0,H,H} and in state 12{1,1,H,H} it decides to change VNFs to separate nodes.Here i would like to have explored more with the hyperparameters to see if there is a better outcome than this.

4.[1,1,1]





Observations: In this case we can see that convergence and we also see that Q-learning has chosen scenario 1. {group all} as optimal policy, this confirms that it has converged to the optimal policy as it is the same exact policy that policy iteration had decided in assignment 3.It is important to note that even though it seems as not corner case scenario , because of the way costs are calculated in this scenario it seems that it outweighs the other costs and thus it does not represent a more normal scenario where it tries to balance all costs.

Key takeaways: From this graph we can see that the policy converges to a set of values, normally we would need to see if it is close the to values of optimal policy but this isn't needed now since in this experiment is easy to observe that it has the correct choice of actions , and thus the optimal policy

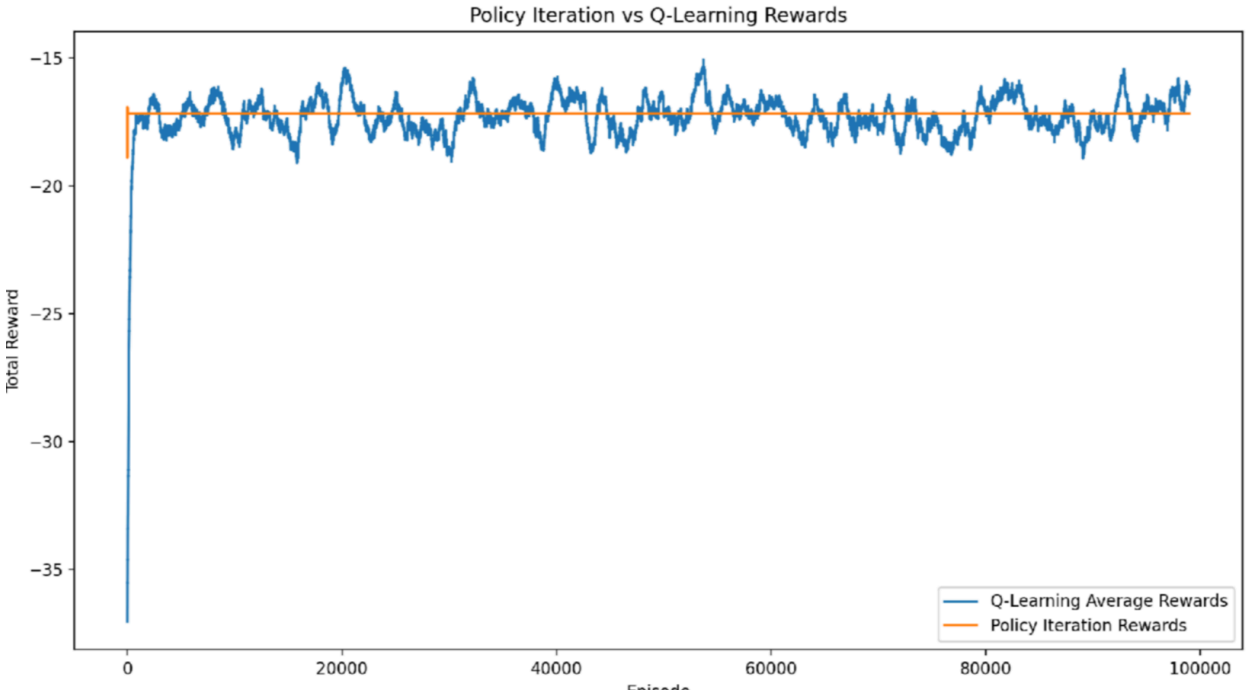
Task 2

Apply now your tabular Q-learning algorithm to the large(r) scenario you tried for Phase 1, and show that it again learns to perform optimally. (assume the same environment as Task 1).

In this task we increased N=M=3 to see if Q-Learning can reach results in a timely manner.Note that my implementation of policy iteration reached results in 1 hour in my pc.

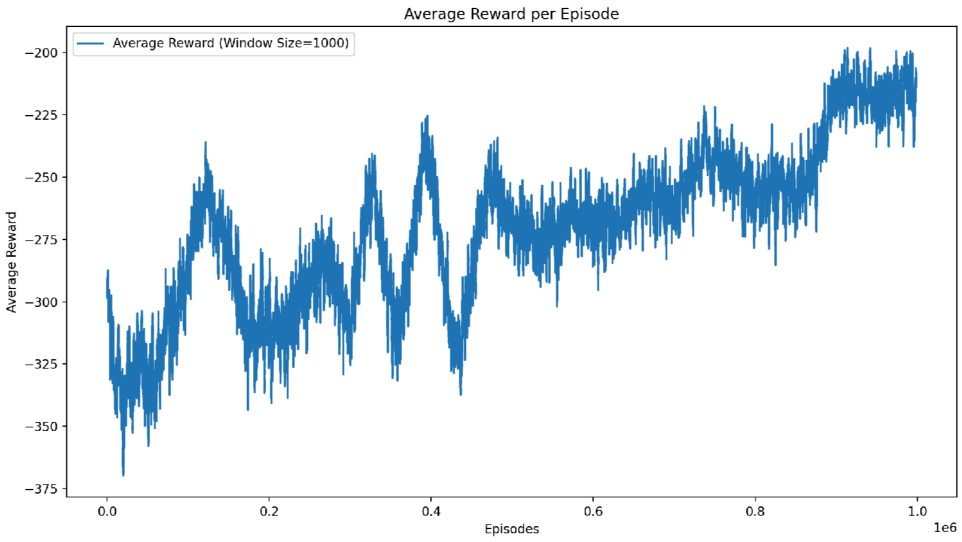
For all the cases we see similar outcomes to task 1 and the results came in under 10 mins avg.

I will provide an indicative case [3,1,1]



Observations: In the case N=M=3 it is very hard to decide if Q-learning has reached optimal policy by looking and comparing actions in the corner case scenarios so i compared it to the rewards that policy iteration provides.

Next i tried to see if Q-learning could handle M=N=4 and it took considerable amount of time to finish executing.



Observations: As we can see here Q-Learning hasnt converged and it is still improving even though i increased the episodes to 1000000.This is the point where transitioning to a Deep Q Learning method to see how it can handle bigger complexity scenarios.DQN Uses a neural network to approximate the Q-values, enabling it to handle large and continuous state spaces efficiently. The neural network generalizes well across similar states, reducing the need to explicitly store every possible state-action pair.

Task 3

Consider now a scenario that is large enough for tabular Q-learning to converge slowly, but still within reasonable time (e.g. 1-2 hours). You can just use the scenario of Task 2, or enlarge it slightly (if tabular Q-learning runs in just a few minutes). Implement now a Deep Q Learning method to solve the problem of Task 2. You can implement DQN, Actor-Critic (or other Deep RL variants). (**Note:** Traffic should still be Markov chain based as before, again with unknown parameters, **but now your agent does not know that there are two states!** hence, you cannot use H,L as part of your state...you just observe a continuous traffic demand, without knowing what guides it initially). Compare your Deep Q learning method with the tabular method.

- What is the input you will use for your DNN(s)? What is the output?
- Your goal is to show that you can still achieve similar performance as with the tabular method (which we know will converge to optimality), but that you learn this policy faster than tabular (due to the approximation and generalization performed by the DNN(s)).

For Task 3 i decided to implement a normal DQN and configure it to solve the problem of VNFs slicing.

Environment Setup:

- The state of the environment consists of the locations of the VNFs and their respective traffic demands. Unlike previous scenarios, the agent does not know the discrete states (H, L) guiding the traffic demands but instead observes continuous traffic demands

Deep Q-Learning Method

- Deep Q-Learning (DQN) is employed to solve the VNF placement problem by approximating the Q-value function using a neural network. This approach allows the agent to handle the continuous state space effectively, providing a more scalable solution compared to tabular Q-learning.

Neural Network Architecture

The neural network used in the DQN agent has the following structure:

- **Input Layer**: Takes the state vector, which includes the locations of the VNFs and their continuous demands.Input has the format of (M+N) locations concatenated with the demands,(Locations as noted in the

presentation is needed in order for the neural network needs to know about reconfiguration costs)

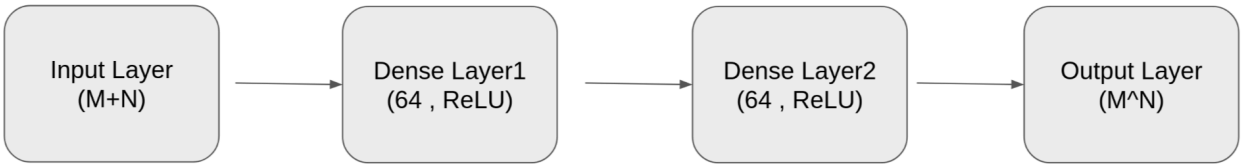
Hidden Layers:

- **Layer 1:**
  - **Type:** Fully Connected (Dense)
  - **Neurons:** 64
  - **Activation Function:** ReLU
- **Layer 2:**
  - **Type:** Fully Connected (Dense)
  - **Neurons:** 64
  - **Activation Function:** ReLU

Output Layer:

- **Dimensions:**M^N (All possible actions)
- **Purpose:** Outputs Q-values for each possible action.

Visualisation of the Neural Network Architecture



Hyperparameters

These values are close to the default ones. I would like to have explored more values but i didint have enough time.

Learning Rate (α): 0.001

Discount Factor (γ): 0.99

Exploration Rate (ε):
Initial ε: 1.0
Decay Rate: 0.995
Minimum ε: 0.01
Gradually shifts from exploration to exploitation.

Batch Size: 64
Standard size for stable and efficient learning.

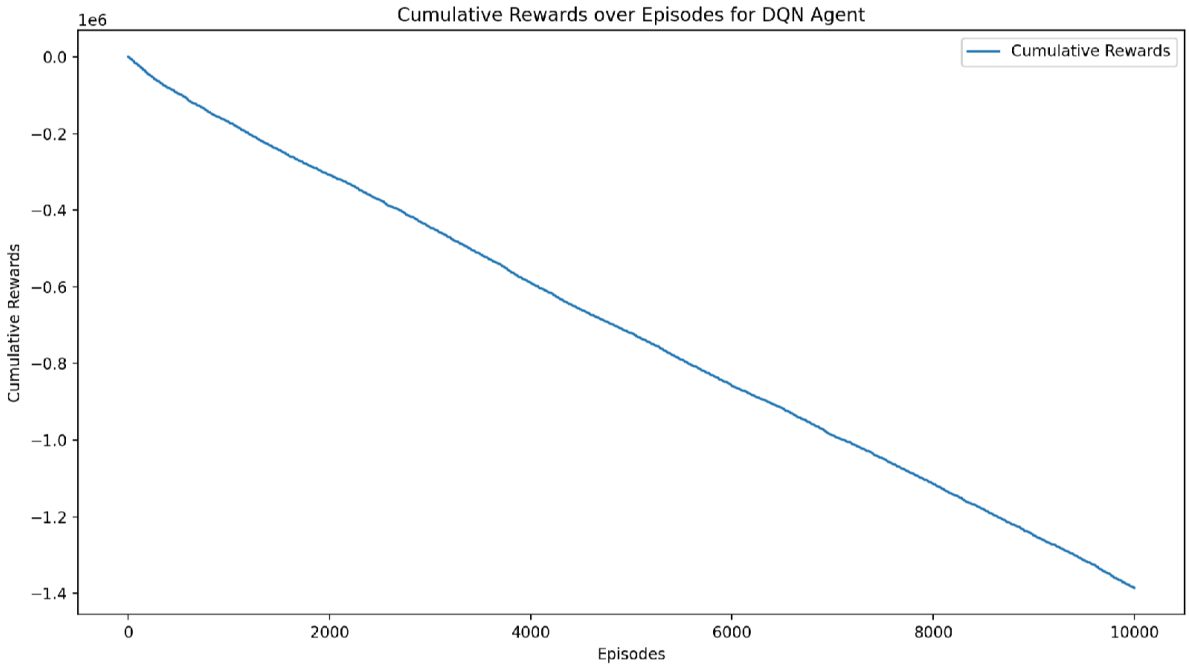
Memory Size: 10000

Episodes: 100000

Target Network Update Frequency: Every 10 episodes

- **Reason:** Helps stabilize training by reducing correlations between the Q-network updates.

Notes on implementation
DQN network for this task, the results are not as expected. The network does not appear to be converging. In the code I provided, I have configured both the DQN and the environment for the problem to be solved. However, the implementation seems to be flawed.If i had more time available i would like to see what is not working maybe it is something in my representation of the environment of there is tweaking needed in the hyperparameters.



Observation: This is the result i got from trying scenario [3,1,1] with N=M=4 , it run only for 10000 episodes and 10000 took longer than i would expect, maybe more episodes are needed or more layers.

Next steps to improve my DQN implementations.

Experiment with hidden layers and units for the network architecture.

Ensure an adequate number of training episodes.

Confirm the environment's dynamics, including the reward function, state representation, and action space, before training.