Partners: Dhruv Karthik, Rahul Maganti

HW 4: Actor-Critic & Q-Learning

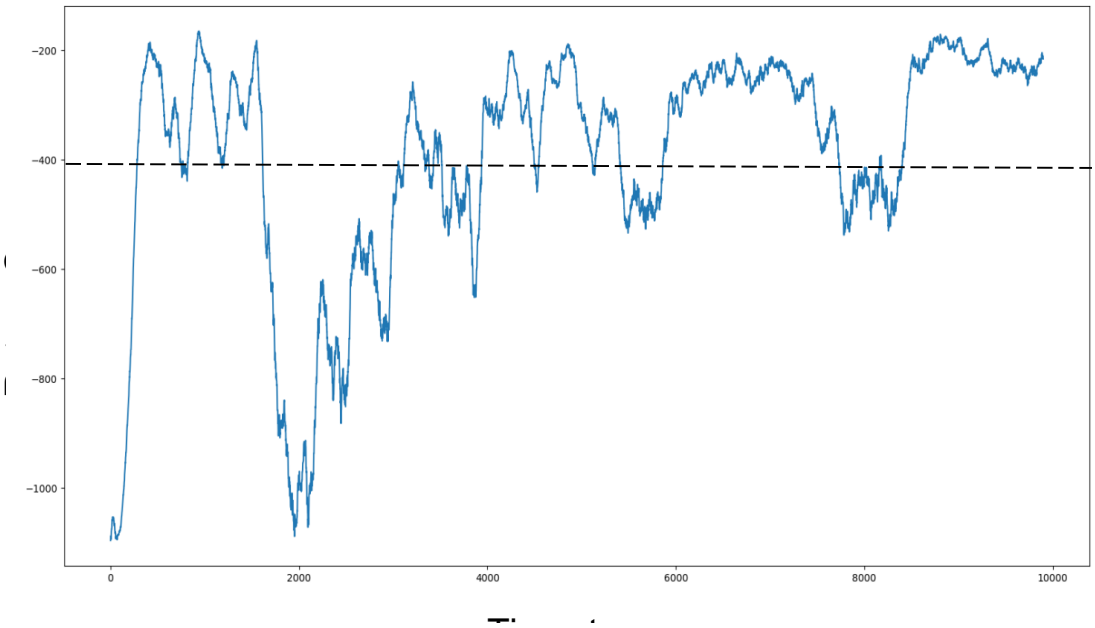
Problem 1: Actor Critic Algorithms

1

The *actorcritic_skeleton.py* file contains the algorithm, along with *pkl* files illustrating the weights on a learnt model

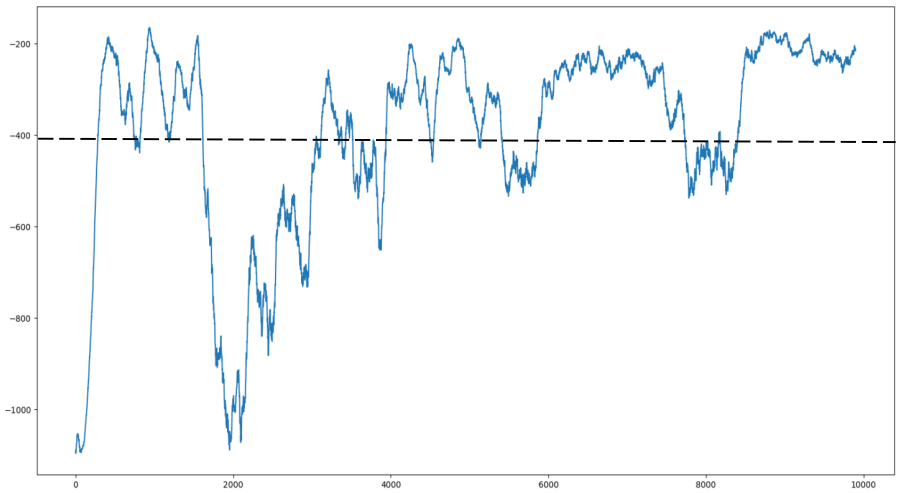
2

Hyperparamters: alpha_policy = 1e-2, alpha_value = 5e-2



3

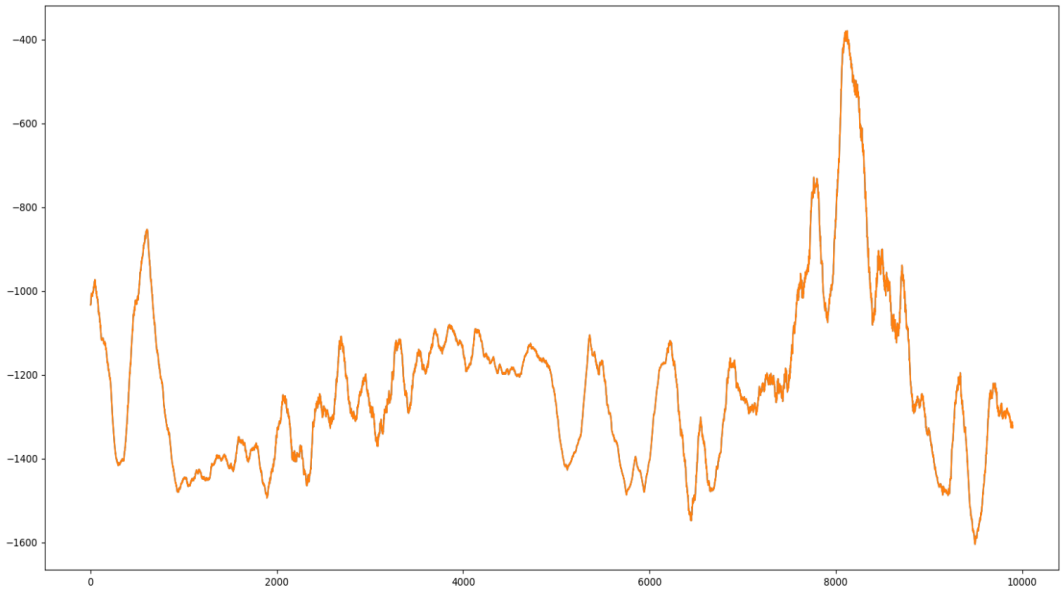
Hyperparamters: alpha_policy = 1e-2, alpha_value = 5e-2 (only one that worked well)



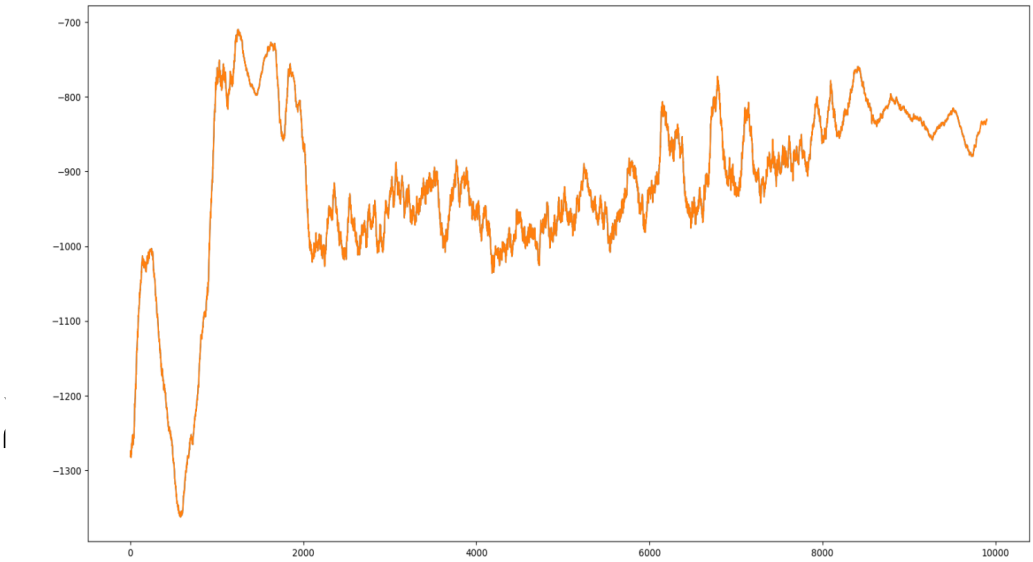
Partners: Dhruv Karthik, Rahul Maganti

HW 4: Actor-Critic & Q-Learning

Hyperparamters: alpha_policy = 5e-2, alpha_value = 1e-2



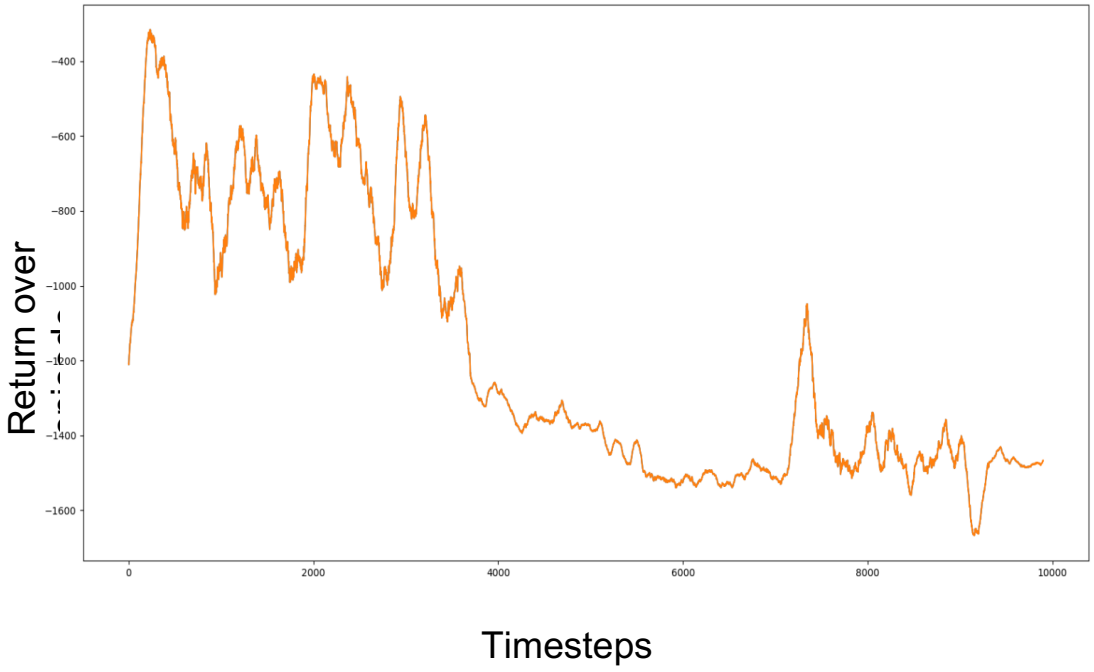
Hyperparamters: alpha_policy = 1e-2, alpha_value = 1e-2



Partners: Dhruv Karthik, Rahul Maganti

HW 4: Actor-Critic & Q-Learning

Hyperparamters: alpha_policy = 5e-2, alpha_value = 5e-2



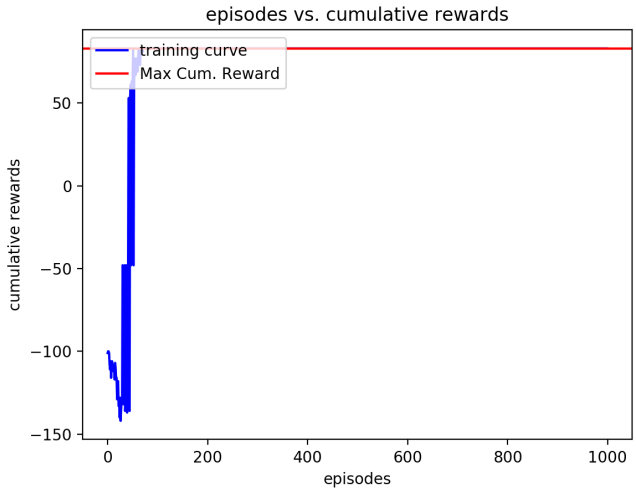
When the actors learning rate (alpha_policy) was larger than the critics learning rate (alpha_value), the critic was not effectively able to evaluate the value at a certain state. As a result, the model did not converge. In fact, the critic likely directed the actor in a suboptimal direction. The only case where the model converged was where the critics learning rate was higher, as this meant the critic was better at estimating the value function and guiding the actor toward convergence.

Partners: Dhruv Karthik, Rahul Maganti

HW 4: Actor-Critic & Q-Learning

Problem 2: Q Learning

1. (-Greedy Policy) Run Qlearning on MAP3. Create a plot where the x axis is the episode number and y axis is the total reward for the episode. Compare this with the output of the evaluate greedy policy() function. Is there a difference, and if so, why?



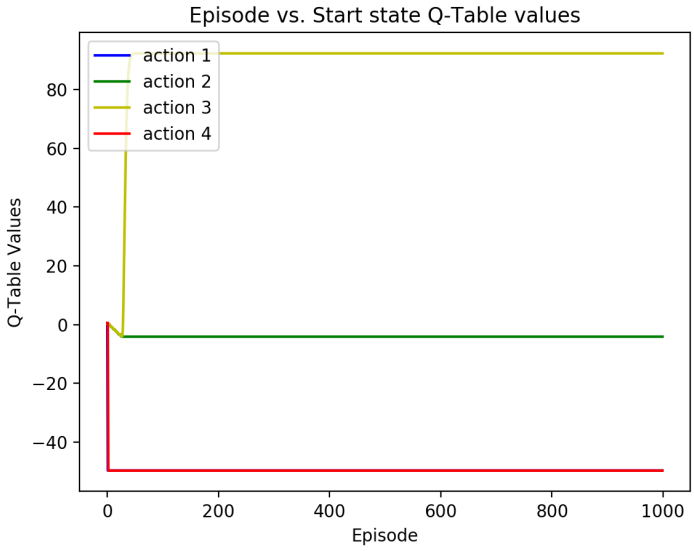
The output of the evaluate_greedy_policy function and the q learning method are not different for MAP 3. Both yield a maximum reward of 83 for the map. The greedy policy converges relatively more quickly, given it does not incorporate any randomness in choosing actions.

2. (Optimal Q values) Run Q learning on MAP2. Compute the Qvalue for the start state. (Hint, you know the optimal policy already, just need to add a few numbers to get the discounted return).

The Q-value for the start state is the expected discounted returns over the optimal policy.
Q_Value for start state:

Partners: Dhruv Karthik, Rahul Maganti

HW 4: Actor-Critic & Q-Learning



After Convergence(ep=1000)	0	1	2	3
Q_Value	-49.72	-4.14	92.32	-49.74

Before Convergence(ep=10)	0	1	2	3
Q_Value	-49.72	-4.14543238e+00	92.32	-49.74

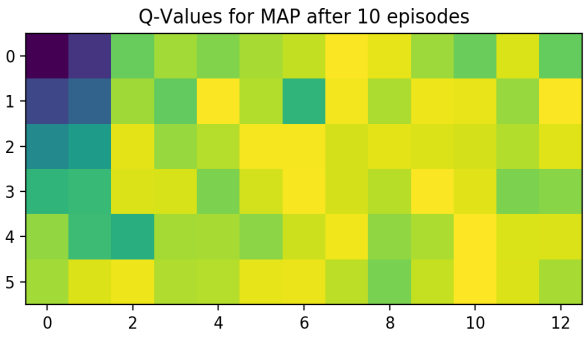
a) **It takes approx. 23 iterations for the q learning algorithm to converge to its optimal value.**

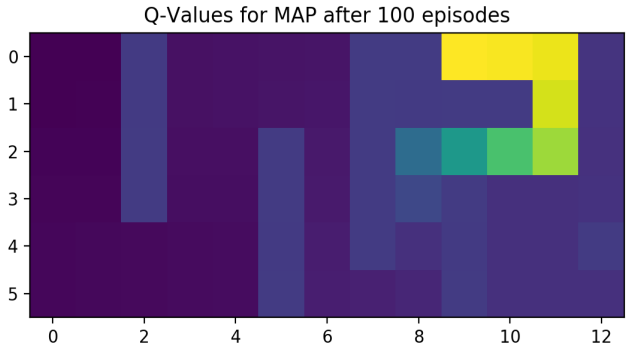
b) **Although the Q values may not be at the converged values, using a greedy policy can still be relatively effective. The reason is that q learning is based on the idea of selecting the action that yields the maximum q-value (expected discounted rewards) for every step of a given trajectory. While randomness is NOT incorporated into (which adds the ability for the algorithm to perform exploratory actions) in a greedy policy, if we are close to convergence, these explorative actions are not particularly useful.**

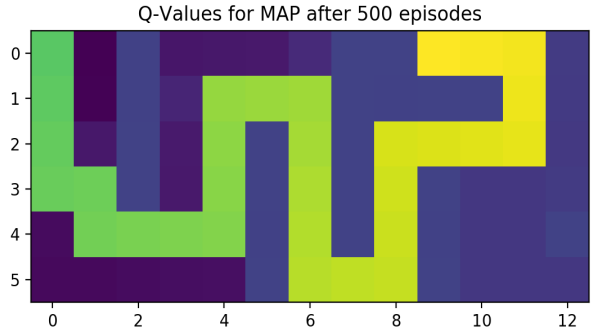
Partners: Dhruv Karthik, Rahul Maganti

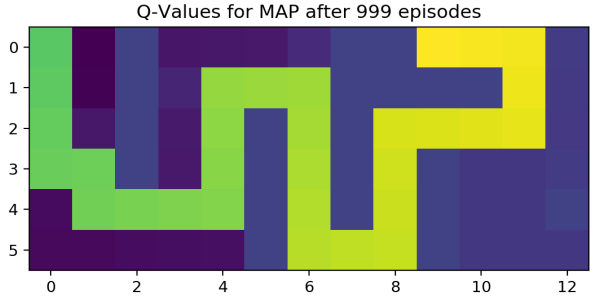
HW 4: Actor-Critic & Q-Learning

3. (Visualization) Run Q learning on MAP4. Create a matrix that is the same size as the map where each entry is the maximum Q value over actions for that state. Create the matrix after training for 0, 10, 100, 500, 1000 episodes. use matplotlib.pyplot.imshow() to visualize theses matrices.









4. (Reading check) Q learning seems very similar in structure/equations to Value iterations. Why use Q-learning at all? (The answer to this is why value iteration is not considered a Reinforcement Learning algorithm.

Q learning is a model-free learning algorithm, meaning it does not have access *a priori,* knowledge about the state- transition probabilities. Given (s, a, s', r), Q learning also learns the transition probabilities to generate an optimal policy that maximizes rewards over actions given for each state. In value iteration, the state-transition probabilities are known and the agent must optimize this policy over some parametrization. In effect, the agent has all the information about the environment and can therefore simulate it to find the right decisions. This does not fall under the class of RL algorithms because there is no inherent necessity to balance the tradeoff between exploration vs. exploitation.