313326985 Shahar Shcheranski
206172686 Sarit Hollander

Report

Section 1 – Tabular Q learning

1. Value iteration can be used when we know the probabilities of moving from one state to another. Due to the fact that we are not always able to access all states and we do not always know the transition probabilities from state to state, we were unable to implement this method in a complex dynamic environment.
2. In model-free methods, each state-action pair is approximated, so a transition matrix is not required.
3. Q-learning is an off-policy algorithm that chooses the maximum Q over all possible actions, which looks like it follows a greedy policy without exploration. SARSA is an on-policy algorithm that learns action values relative to its policy.
4. Using decaying ε − greedy, there is a chance of ε that we perform exploration, which may eventually improve the value of the action, in contrast to the greedy algorithm, which will always select the action with the maximum value and not perform exploration.

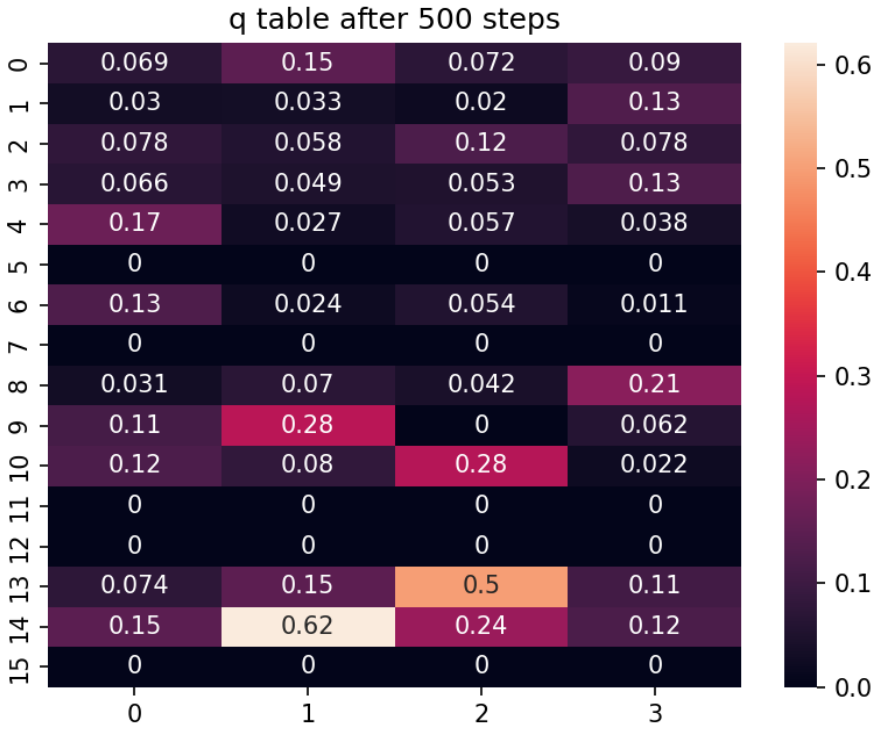
frozen-lake (v1)

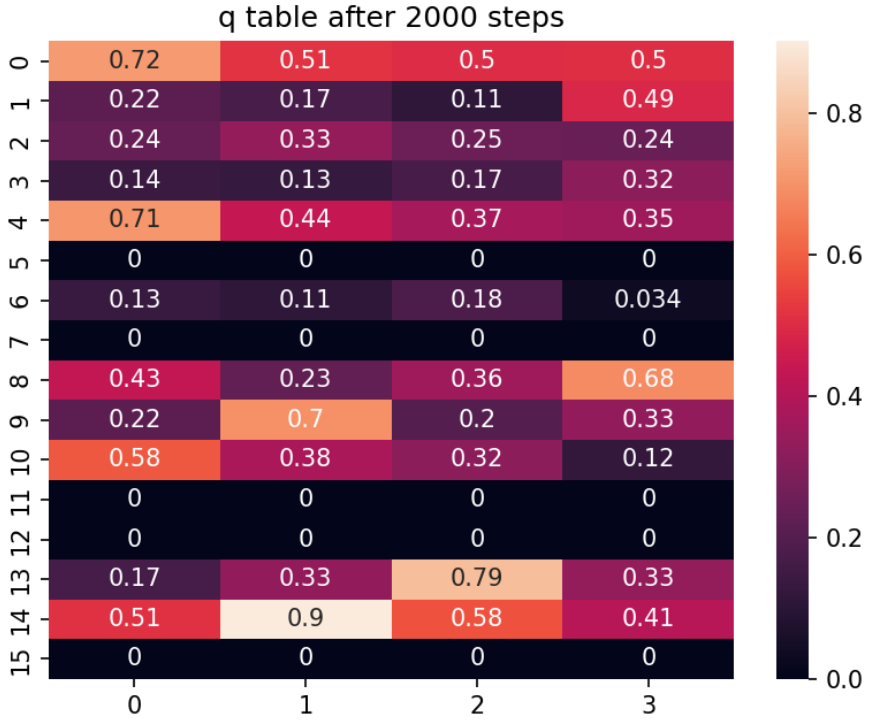
Hyper-parameters:

- gamma = 0.9995
- Learning rate = 0.1
- epsilon decay = 0.995
- epsilon = 1
- min epsilon = 0.01

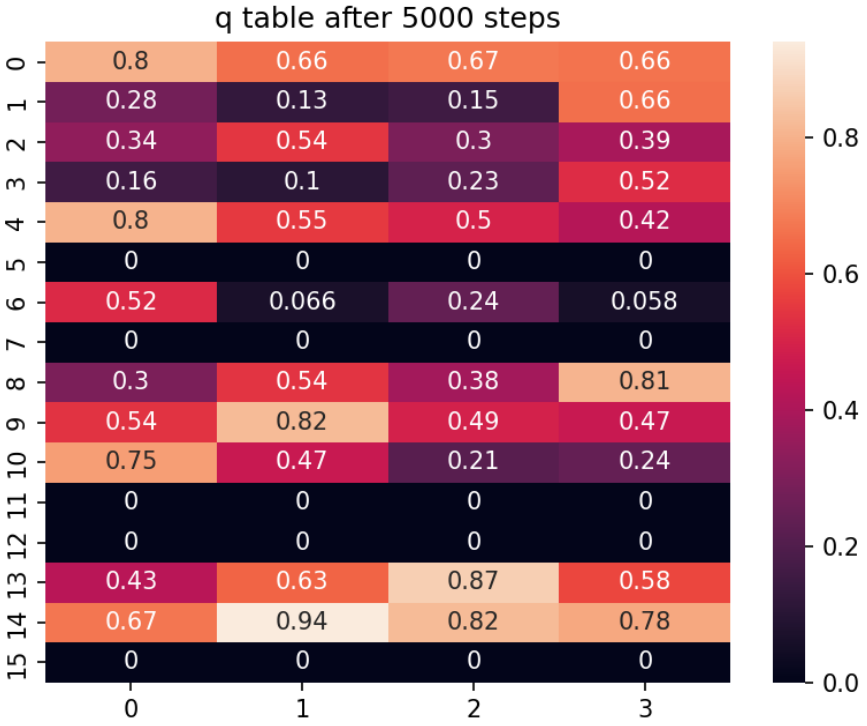
313326985 Shahar Shcheranski
206172686 Sarit Hollander

results:



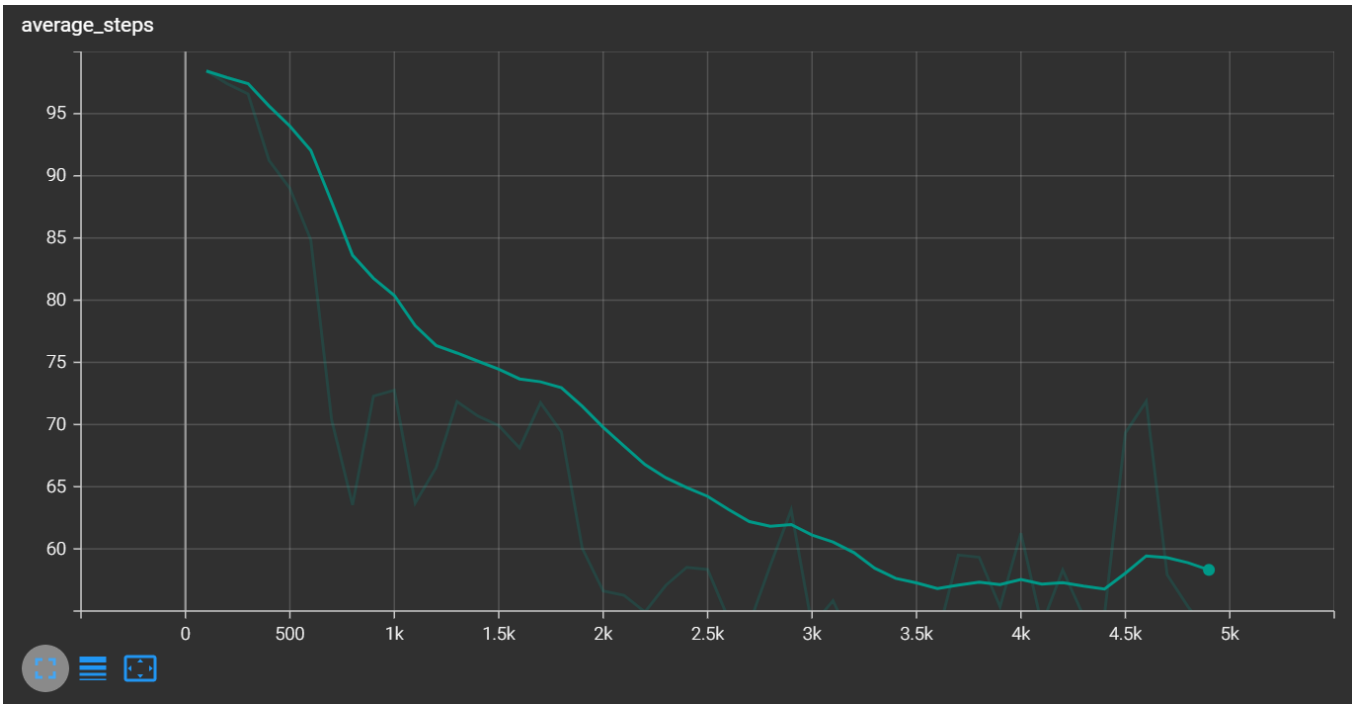


313326985 Shahar Shcheranski
206172686 Sarit Hollander



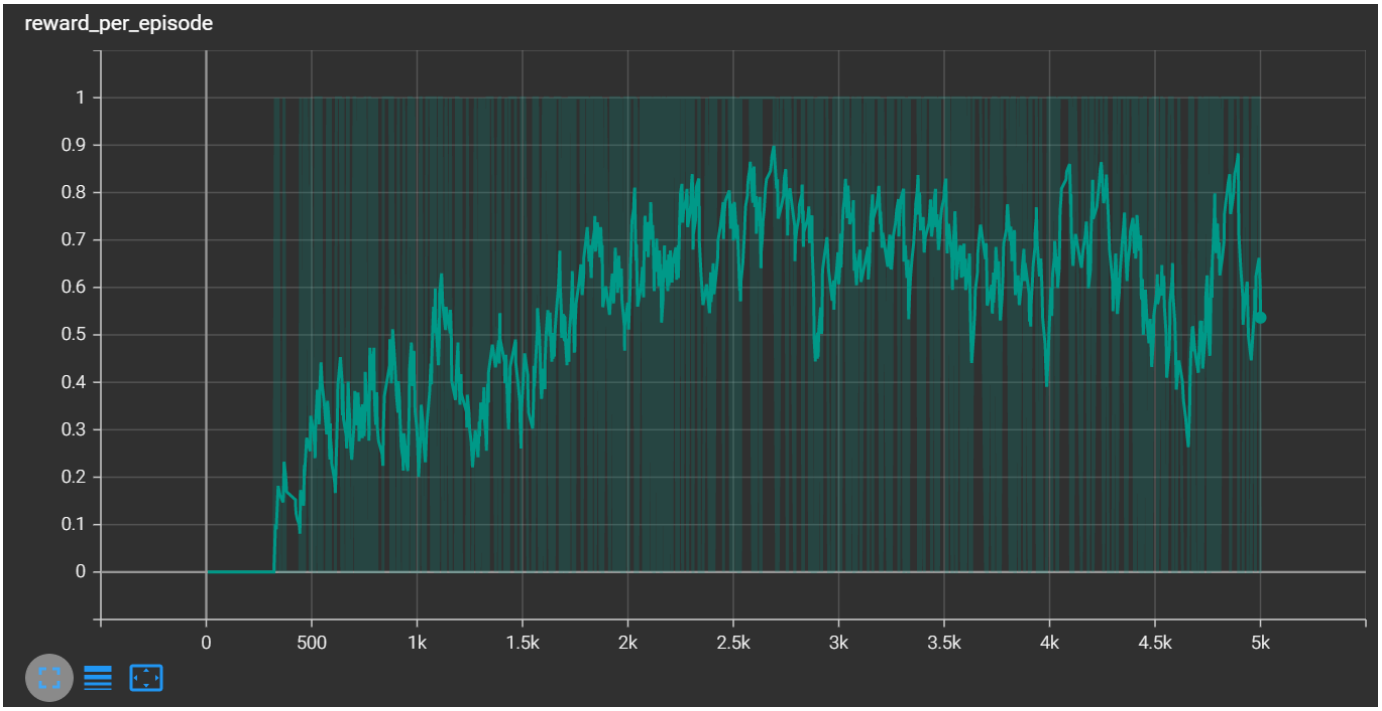
Plots:

Average number of steps to the goal over last 100 episodes



313326985 Shahar Shcheranski
206172686 Sarit Hollander

Reward per episode:



Section 2 – Deep Q-learning

1. By randomizing the samples, we are preventing them from being correlated and related within the same batch, so the model is more generalizable.
2. As the target is not stationary (chasing a moving target), the algorithm has difficulty convergenting. By "freezing" the target parameters for a given number of iterations and updating every C steps, the model becomes more stable.

313326985 Shahar Shcheranski
206172686 Sarit Hollander

Hyperparameters

- 3 hidden layers:
  - number of hidden units: [64, 32, 16]
  - optimizer: Adam
  - activation: relu in hidden, linear in last layer
  - batch size: 64
  - gamma: 0.95
  - epsilon decay: 0.995
  - min epsilon: 0.01
  - learning rate: 0.005
  - min learning rate: 1e-10
  - replay memory size: 20000
  - C: 2

- 5 hidden layers:
  - number of hidden units: [32, 32, 32, 32,32]
  - optimizer: Adam
  - activation: relu in hidden, linear in last layer
  - batch size: 64
  - gamma: 0.995
  - epsilon decay: 0.995
  - min epsilon: 0.01
  - learning rate: 0.005
  - min learning rate: 1e-10
  - replay memory size: 20000
  - C: 2

We tried out values around the ones we found to be common after adjusting our parameters.
Once we got the value of epsilon decay to be 0.995, our solution improved. This is probably due to a lack of exploration at first. As a result, we have a much smaller and less varied model because our agent has less experience of new states.
Our results were also affected by the parameter C to update the target model weights, and as C gets larger, the model has a hard time converging.

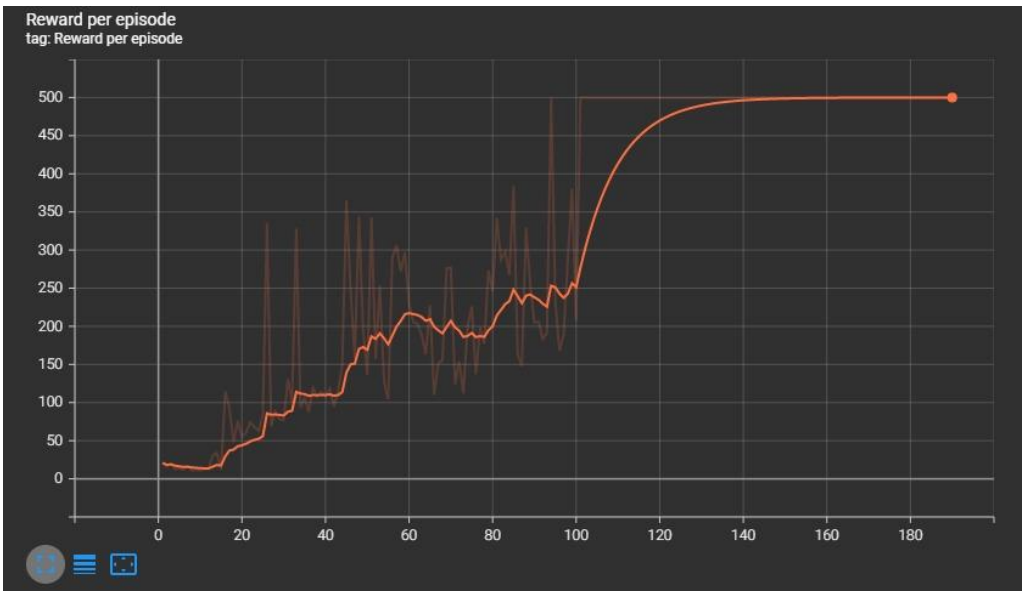
313326985 Shahar Shcheranski
206172686 Sarit Hollander

Number of episodes until the agent obtains an average reward of at least 475.0 over 100 consecutive episodes:

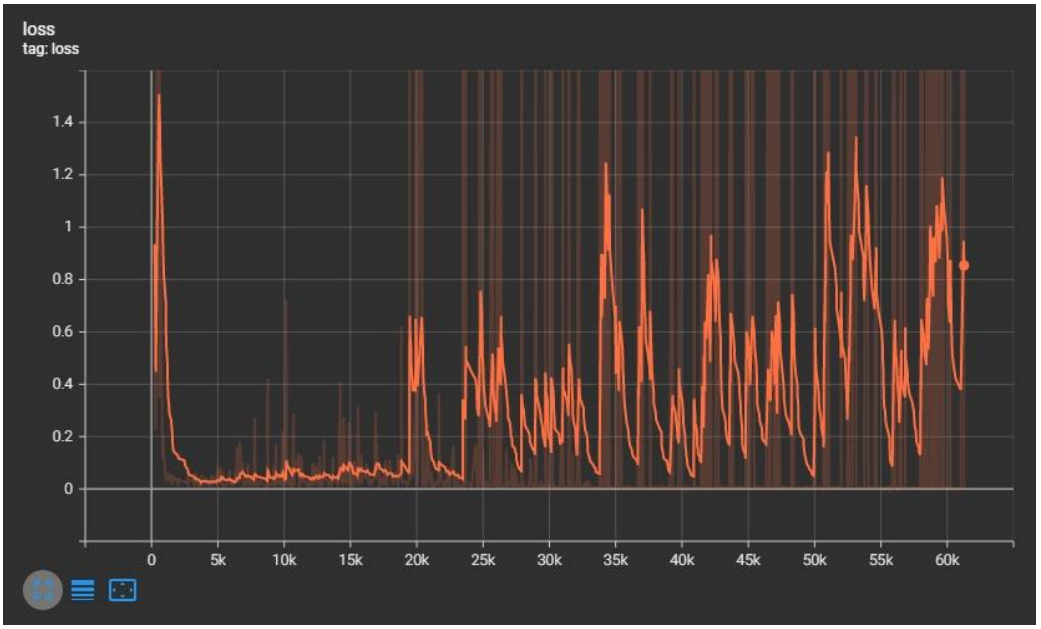
- 3 layers: **190**
- 5 layers: **181**

3 layers plots:

Reward per episode



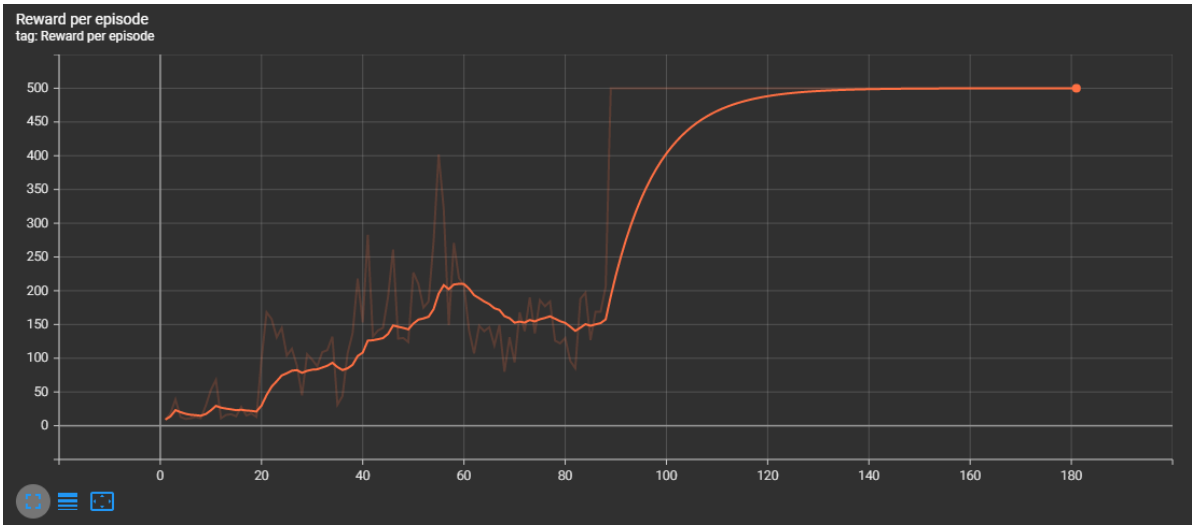
Loss



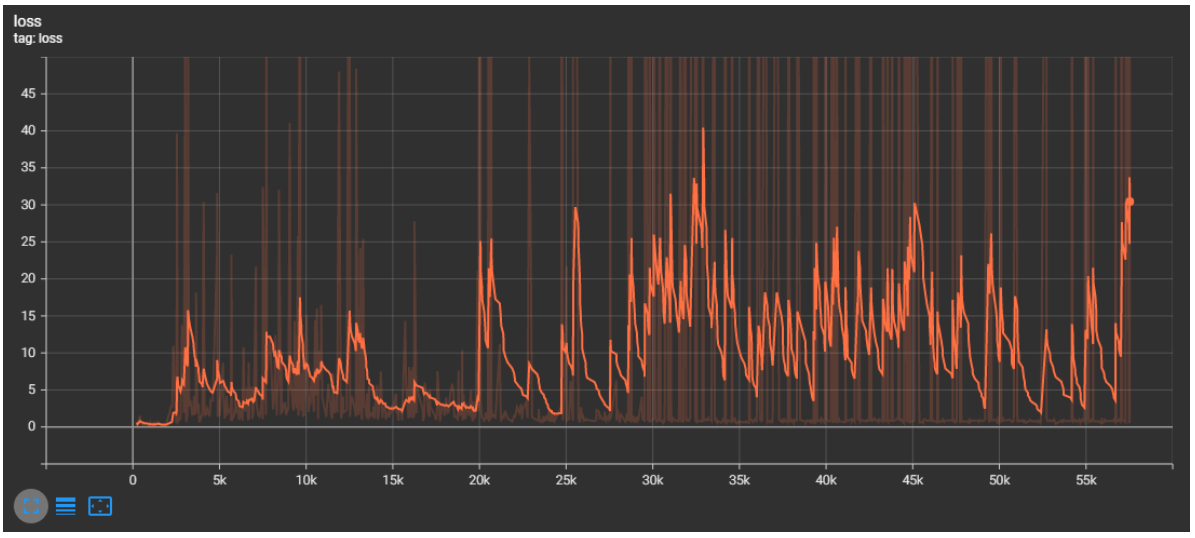
313326985 Shahar Shcheranski
206172686 Sarit Hollander

5 layers plots:

Reward per episode



Loss



313326985 Shahar Shcheranski
206172686 Sarit Hollander

Section 3 – Improved DQN

Short explanation

Our suggestion for improvement is the implementation of Dueling DQN.

The intuition behind Dueling DQN is that the importance of choosing the right action is not equal across all states. Dueling DQNs achieve this goal by decoupling the action from the state, one network evaluates the state value $V(s)$ and the other network evaluates the action value $A(s,a)$ (the "advantage" function). The output of the two networks is then combined to a single Q-function.

Our intuition to implement the Dueling DQN is that in the CartPole environment there are states where choosing the right action is more important than other states. For example, choosing the right action when the pole is about to fall is more important than choosing the right action when the pole is in the middle. When the pole is about to fall to a certain side, choosing the right action (i.e. balance to the opposite side) is the most important.

Hyperparameters

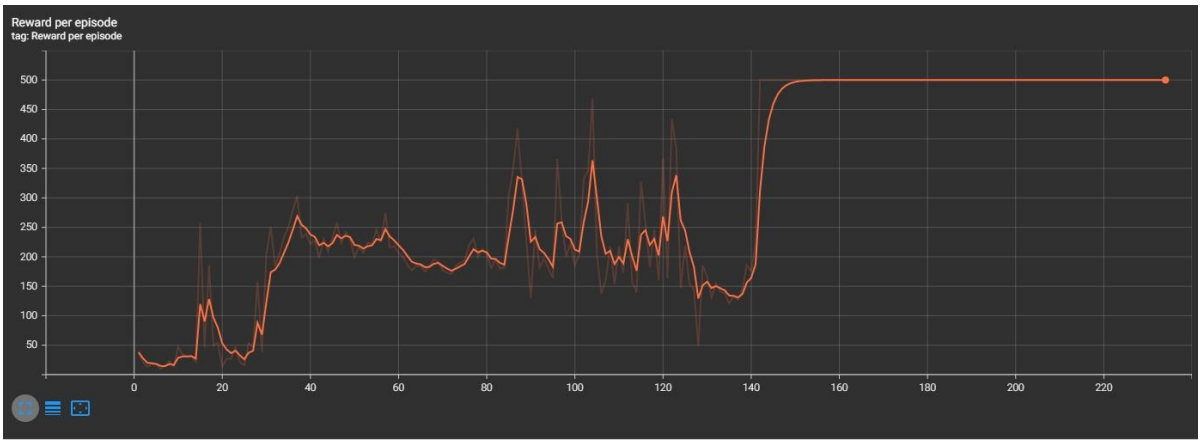
- number of hidden layers: 3
- number of hidden units: [64, 32, 16, 1 (state value), 2 (action advantage)]
- optimizer: Adam
- activation: relu in hidden, linear in the two last layers (state value and action advantage)
- batch size: 64
- gamma: 0.995
- epsilon decay: 0.995
- min epsilon: 0.01
- learning rate: 0.005
- min learning rate: 1e-10
- replay memory size: 20000
- C: 2

313326985 Shahar Shcheranski
206172686 Sarit Hollander

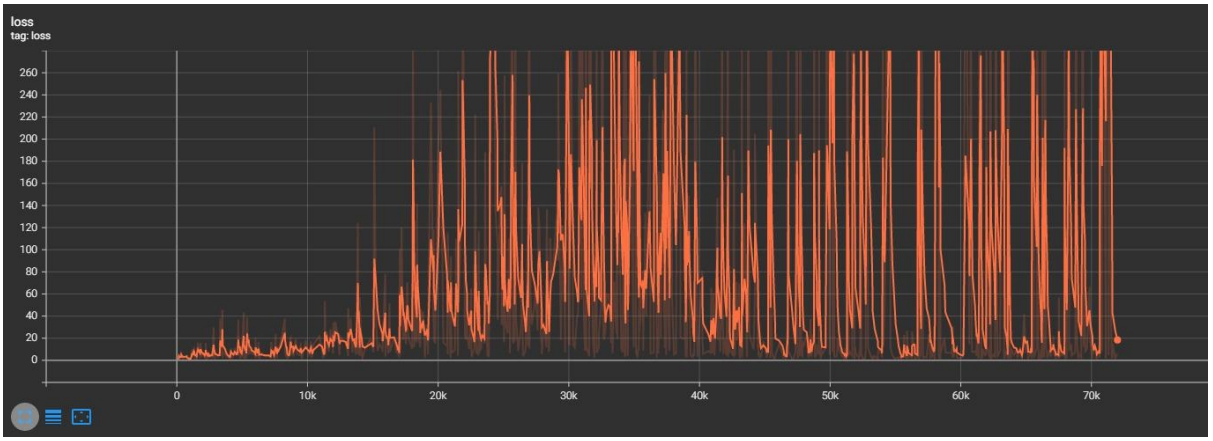
Number of episodes until the agent obtains an average reward of at least 475.0 over 100 consecutive episodes: **234**

Plots:

Reward per episode



Loss



Comparison with the results of section 2

As we can see from the plots above, in section 2 we get an average of 475 reward over the last 100 episodes in episode 190 compared to section 3 where we get this average in episode 234. That is, in section 3 we get this average after a larger number of episodes compared to section 2.

Also, the loss we receive at each step in section 3 is greater than the loss we receive in section 2.