Deep Reinforcement Learning

Assignment 1 – From Q-learning to Deep Q-learning (DQN)

# Section 1 – Tabular Q learning

1. Why methods such as Value-Iteration cannot be implemented in such

environments? Write down the main problem.

Value-iteration is a model-based method and as such it uses the transition matrix for calculation of the value function:

In scenario like the one described, the transitions (), are hard to model so model-based method will not fit well to this problem.

1. How do model-free methods resolve the problem you wrote in previous

question? Explain shortly.

Using the Q-function to pick action and directly getting the next state by interacting with the environment help us to avoid modeling the unknown transition matrix of this environment.

1. What is the main difference between SARSA and Q-learning algorithms?

Explain shortly the meaning of this difference.

The main difference between these two algorithms is in the Q function update rule. Q-learning updates the q-function under the assumption that the next action will be the one that maximize the Q function ()

And SARSA updates the q-function using action picked by the policy (e.g., .

For this reason, Q-learning try to learn under more optimistic assumption in contrast to SARSA that try to learn under more realistic assumption.

1. Why is it better than acting greedily (choosing an action with 𝑎𝑟𝑔𝑚𝑎𝑥"𝑄(𝑆′, 𝑎))?

When using a greedy policy our agent will always take actions to exploit known rewards and avoid exploring new unknown states and rewards (will remain in subspace of the problem, local maxima). a non-greedy policy such as *decaying - greedy* can “help” the agent to explore new states and rewards.

### Hyper-parameters

Epsilon = 0.2

Epslon decay=0.1

Epsilon decay steps = 1000

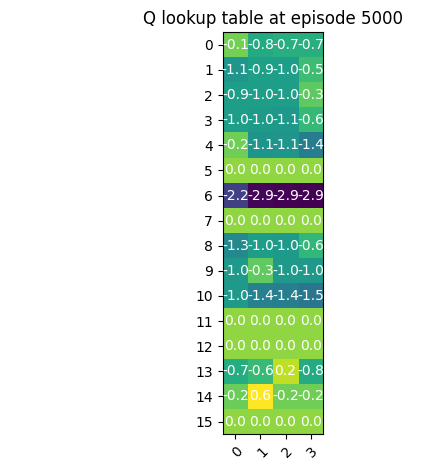
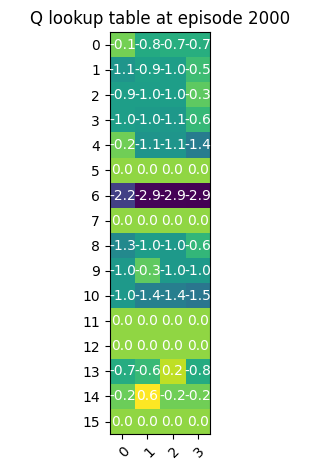
Learning rate = 0.1

Gamma = 0.99

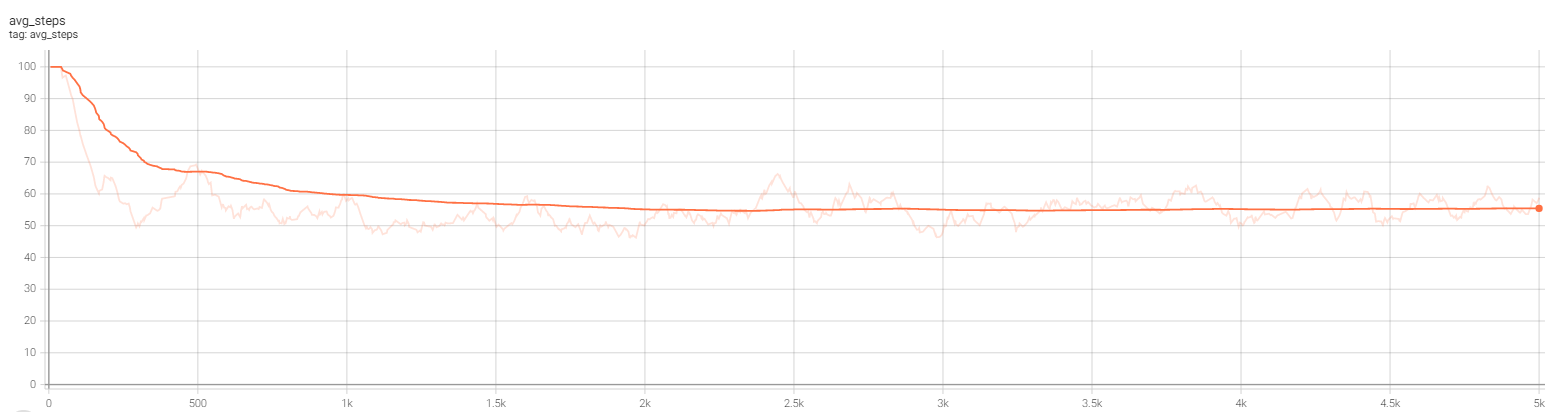
**Proggram assignment:**

*python q\_learning.py -e frozen\_lake -a lookup --epsilon 0.2 -v section1 --epsilon\_decay\_factor 0.1 --epsilon\_decay\_steps 1000*

תמונה שמכילה שולחן

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט, שוג'י, בניין

התיאור נוצר באופן אוטומטי

תמונה שמכילה שוג'י, כלוב

התיאור נוצר באופן אוטומטי

# Section 2 – Deep Q-learning

1. Why do we sample in random order?

We use random order when picking the sample to break any correlation in the samples and make sure not to overfit specific sample.

1. How does this improve the model?

The method helps the Loss function to converge by improving the stability and achieving better results faster.

## 3 layers

### Hyper-parameters

Hidden layers: 32, 32, 32

Epsilon: 0.5

Epsilon decay factor: 0.9995

Epsilon decay steps: 1

Min epsilon: 0.002

Discount factor: 0.97

Learning rate: 0.004

Batch size: 512

Target update episodes: 2

Experience replay capacity: 50000

Learning rate decay factor: 0.98

**Proggram assignment:**

*Python q\_learning.py -e cart\_pole -a dqn\_cart --experiment section1 --epsilon 0.5 -- epsilon\_decay\_factor 0.9995 --epsilon\_decay\_steps 1 --experiment section2 --steps 1000 --episodes 5000 --learning\_rate 0.004 --discount\_factor 0.97 --lr\_decay\_factor 0.98*

**Average of 475 after 164 episodes.**

Chart, line chart

Description automatically generatedRewards

### LossesChart, line chart, scatter chart Description automatically generated

## 5 layers

### Hyper-parameters

Layers: 16 32 32 16 16

Epsilon: 0.5

Epsilon decay factor: 0.9995

Epsilon decay steps: 1

Min epsilon: 0.002

Discount factor: 0.97

Learning rate: 0.005

Batch size: 512

Target update episodes: 2

Experience replay capacity: 50,000

Learning rate decay factor: 0.995

**Proggram assignment:**

*Python q\_learning.py -e cart\_pole -a double\_td1 --epsilon 0.7 --epsilon\_decay\_factor 0.9991 --epsilon\_decay\_steps 1 --experiment section2 --steps 1000 --episodes 5000 --learning\_rate 0.004 --discount\_factor 0.97 --lr\_decay\_factor 0.99 --layers 16 32 32 64 --target\_update\_episodes 3*

**Average of 475 after 141 episodes.**

### Chart, line chart Description automatically generatedRewards

### Losses

Chart, line chart

Description automatically generated

# Section 3

We used Double DQN as an improvement. This method is proposed in a paper:

Deep Reinforcement Learning with Double Q-learning. 2016 Hasselt et al. (A of Google DeepMind)

The method suggests using two identical networks, one learns through the experience replay exactly like a regular DQN network, the other is using copies of the first model. The Q-value is approximates using the second network.

### Hyper-parameters

Layers: 16 32 32 64

Epsilon: 0.7

Epsilon decay factor: 0.9991

Epsilon decay steps: 1

Min epsilon: 0.002

Discount factor: 0.97

Learning rate: 0.004

Batch size: 512

Target update episodes: 3

Experience replay capacity: 50,000

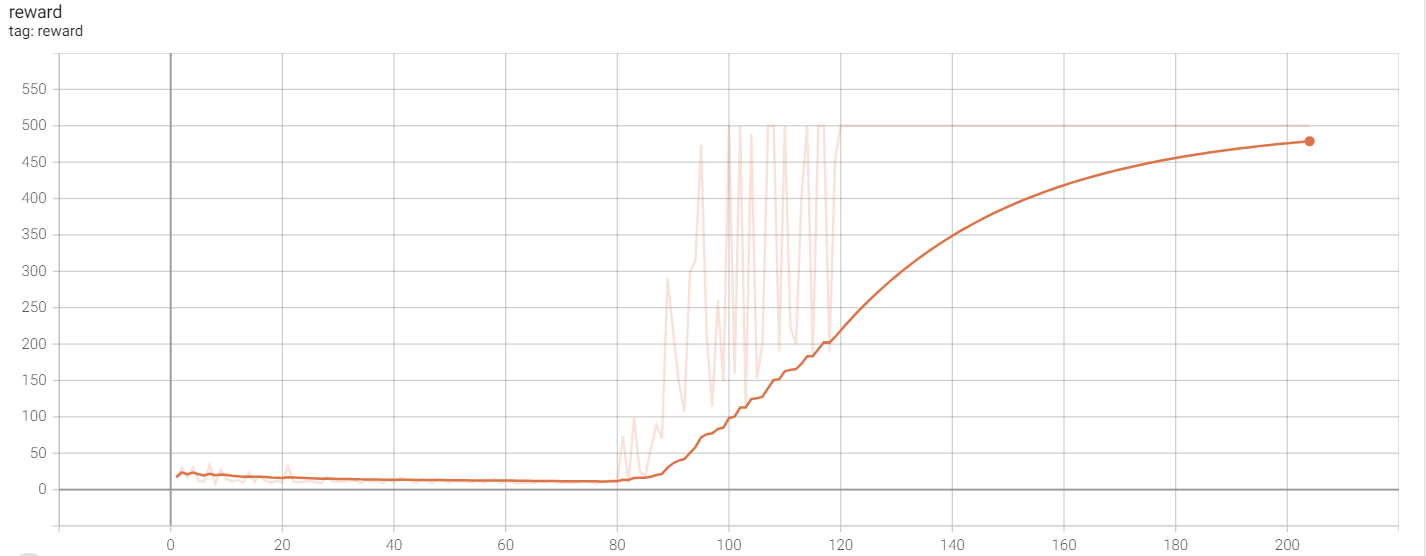
Learning rate decay factor: 0.99

**Proggram assignment:**

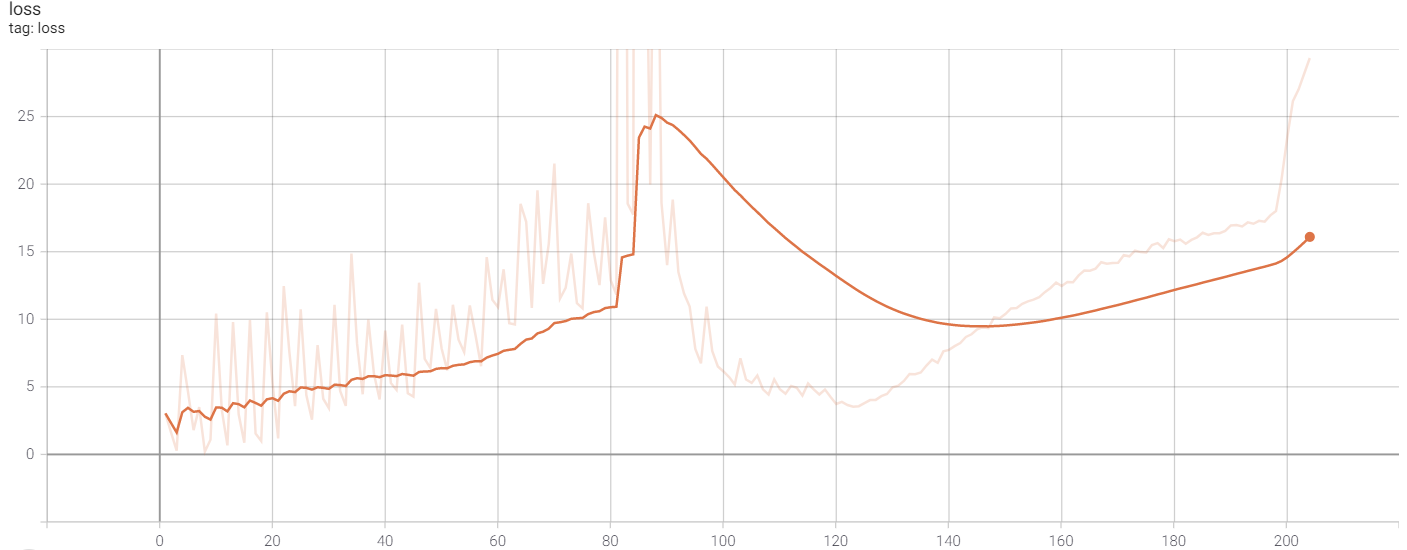
*python q\_learning.py -e cart\_pole -a double\_dqn\_cart --epsilon 0.7 --epsilon\_decay\_factor 0.999 --epsilon\_decay\_steps 1 --experiment section2 --steps 1000 --episodes 5000 --learning\_rate 0.02 --discount\_factor 0.96 --lr\_decay\_factor 0.9995 –layers 8 16 32 32 --target\_update\_episodes 5*

**Average of 475 after 186 episodes.**

### Rewards



### Losses



**Comparison with the results of section 2**

The results shows that the Double DQN take more time to converge, this may be related to the two networks that need to be trained instead of one. due to this fact the Q values are approximated greater number of episodes. The method of double DQN suppose to be more robust and learn better and not necessarily faster.