DRL – assignment 1

Tomer shaked - 315822221

Elad sofer - 312124662

Guy Hadad - 316508126

Section 1

1. The main problem is that if that environment is unknown, we don’t know the dynamics of the transition matrix that needed for the value iteration algorithm. another limitation is when the environments are too complex, and it is not feasibily to use tabular method to restore all the q values or the computional cost of search and optimizations.
2. Model-free methods solve the problem by directly learning from interactions with the environment, without relying on a transition matrix. These methods estimate the value or policy directly based on the observed experiences.
3. The main difference between SARSA and Q-learning algorithms is how they update their Q-values. SARSA updates Q-values using the action our policy taken in the current state, while Q-learning updates Q-values based on the action with the maximum expected reward in the next state. Its means that SARSA is an on-policy algorithm (update Q while following the current policy) while Q-learning is an off-policy algorithm (update Q while following the other policy-behavior policy).
4. It is better because it explores new states and actions so our model can improve. If our agent always be greedy, it might lead our policy to stuck on sub-optimal result. Decaying epsilon method balance better the exploration-exploitations trade-off. So, in the first timestamps our agent more explores the environments with different random actions and over the time when the epsilon decreases our policy more exploit the knowledge that it explored and exploit it to more optimal solution.
5. Hyper-parameters:

alpha = 0.2

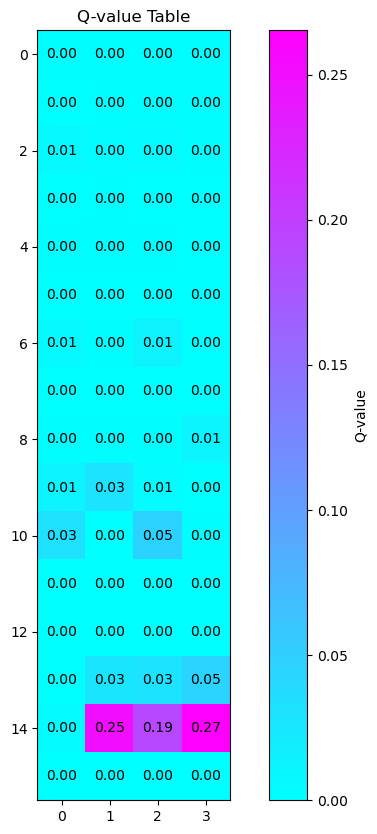
gamma = 0.95

init epsilon=1.0

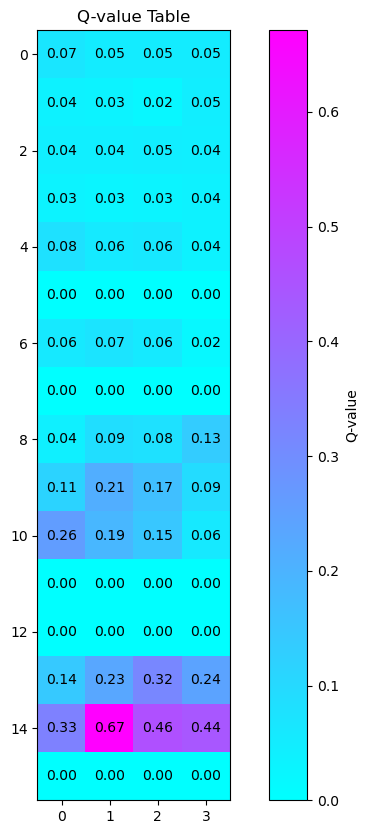
min epsilon=0.05

decay ratio=0.9

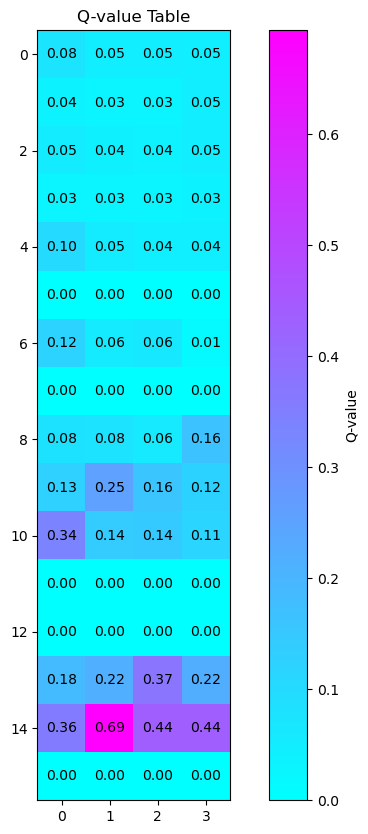
Q Table after 500 episodes:



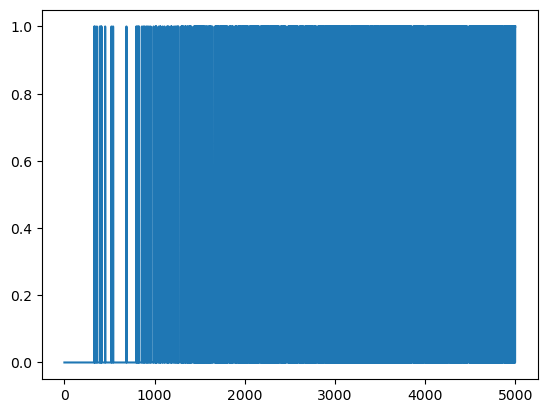
Q table after 2000 episodes:

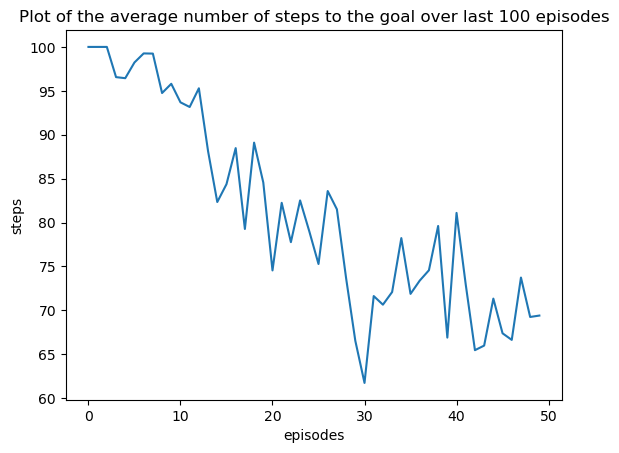


The final Q table:



Plot of the reward per episode:





Section 2:

1. We sample in random order to:

* We train sequentially so we may forget previous experiences.
* Random sampling helps break the temporal correlation in the sequence of experiences. Consecutive experiences are often highly. This correlation can lead to instability during training, as the learning algorithm might focus too much on the most recent experiences. Random sampling helps decorrelate the samples.

1. The use of a target network, where the parameters are updated less frequently (every C steps), stabilizes the training of the model in the learning. This approach reduces the variance in target values, providing a more consistent and slowly changing target for the learning algorithm. It prevents the model from chasing a moving target, ensuring smoother convergence during training. The strategy mitigates the risk of oscillations or divergence, resulting in more robust and reliable updates to Q-value predictions.
2. Hyper-parameters:

Learning rate = 0.0001

Epsilon max = 0.9

Epsilon min = 0.025

Gamma = 0.99

FC network with 3/5 hidden layers and dropout and ReLU between each layer

Hidden dimensions = [128,128,128] [64,64,64,64,64]

Dropout = 0.05

Activation function = ReLU

Epsilon decay = 0.998

Batch size = 64

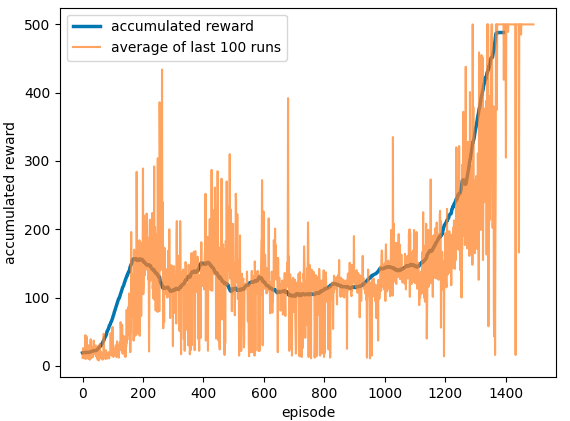
C = 5

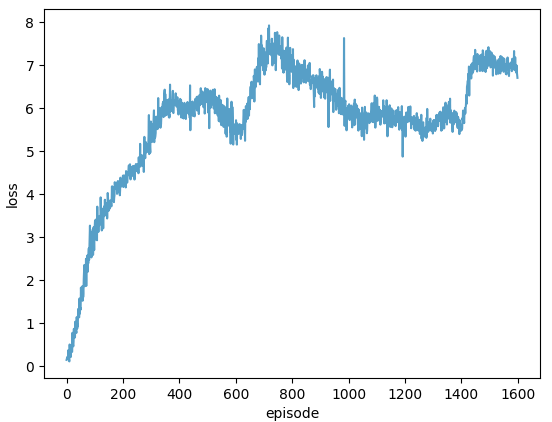
Replay size = 10000

Optimizer = Adam

Loss function = Smooth L1 Loss

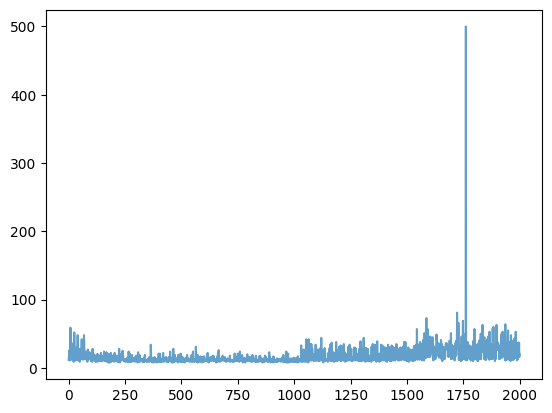
* The loss and the optimizer were very important for the robustness of the learning process.
* The epsilon greedy parameters were critic to get high rewards.
* The learning rate and the C were important to the convergence to satisfied results.
* We didn’t see significant difference between shallow and deep NN.





Our model obtains an average reward of at least 475 over 100 consecutive episodes after 1364 episodes.

To get a nice and "normal" decreasing loss plot we had to set c =512 but the model don’t get sufficient result.



To get our best results we define c = 5:

We read that the increase in loss in DQN, despite better rewards, can be attributed to the non-stationarity of the Q-function, fluctuations in experience replay, exploration strategies, and issues with learning rate or optimization, causing the neural network to experience temporary instability and adjust its estimates.

Section 3:

The hyperparameter was like the previous section because we did a lot of tuning until we get a satisfied result.

In the paper " [Continuous control with deep reinforcement learning"](https://arxiv.org/abs/1509.02971)

The researchers use soft update, it means θ′←τθ+(1−τ)θ′.

We noticed that this algorithm is very sensitive to changes and causes the model to become unstable. It would reach a much higher accumulated reward fast but it would have trouble staying at a high accumulated reward so we changed the updating function

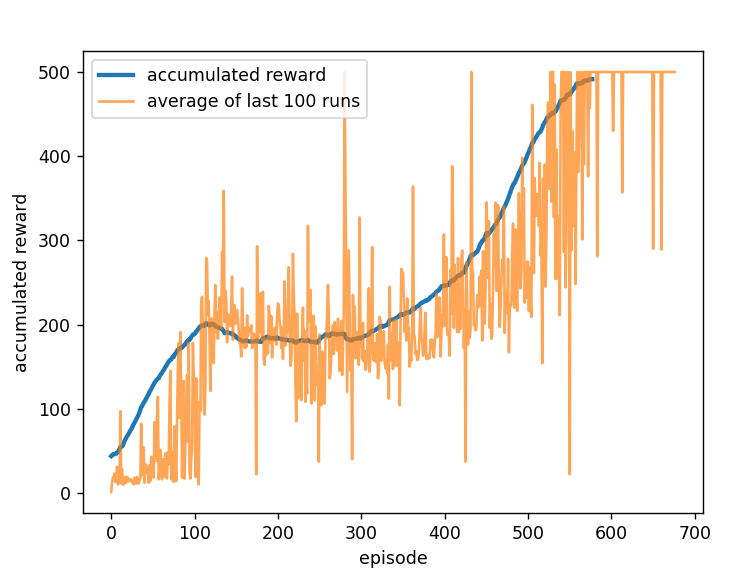
θ′←τ\*(t/last\_acc\_reward)\*θ+(1−τ\*(t/last\_acc\_reward))θ′.

This modification makes the last iteration of every episode have a larger affect on the update of the models and also it also lowers the tau value for episodes which did not survive for very long.

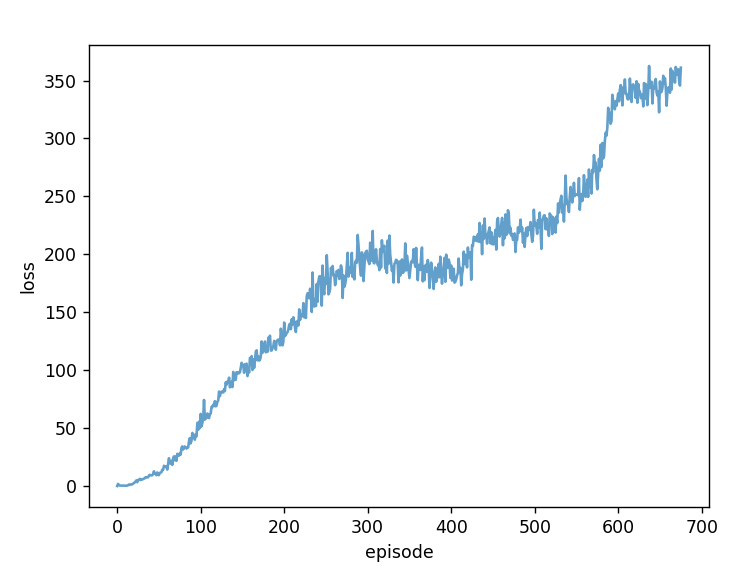
The Hyperparameters used:

T = 100000  
g = 0.99  
epsilon = 0.9  
lr = 0.0001  
episodes = 1450  
c = 5  
batch\_size = 64  
replay\_size = 10000  
Tau = 0.4 # this is only the basic tau which is multiplied by ‘t’ the iteration number and divided by ‘acc\_reward\_list[-1]’ the accumulated reward of the last epoch.

The model reached an average accumulated reward above 475 for 100 consecutive episodes at episode 675.

 We get this accumulated reward graph:

The loss function:



We can see that section 3 converges much faster than section 2, this is due to the fact that we do not wait C episodes to update the target.