Navigation Project Report

Krithika Govindaraj

1. Introduction

Q-tables are mainly used for RL tasks that have a very small set of actions and states. Maintaining a Q-table for a large number of actions and states as well as in continuous RL tasks is impossible. Hence we use Q-networks. The main issue with these networks however is correlation between consecutive steps that inhibits accurate learning. Randomness is introduced in the system using experience replay. There is also instability in the Q-target which can cause convergence issues. This is addressed using fixed Q-targets.

2. Implementation

The Q-network is implemented using deep learning to map the states to actions. We create a simple neural network, in this case it does not consist of CNNs as we are not using pixel data. The neural network consists of fully connected layers and relu as activation functions.

The Q-Local network and the Q-Target networks are used as substitutes for the Sarsa(0) formula used for the traditional method that uses Q-tables. The Q-local network uses each step to improve the weights and produce the accurate action to take for each state. The Q-target after each episode ends, and acts as a check on convergence of the Q-local network.

The Adam optimizer is the most commonly used optimizer for deep learning tasks, and we use the same in our implementation of the network. We use mean-square error in order to define the loss, which makes sense in this case we are dealing with raw numerical values. The loss is the MSE between the Q-local and Q-target values.

We also use the epsilon-greedy policy which chooses the most likely action with the probability (1-eps), and chooses an action based on a uniform distribution with probability (eps).

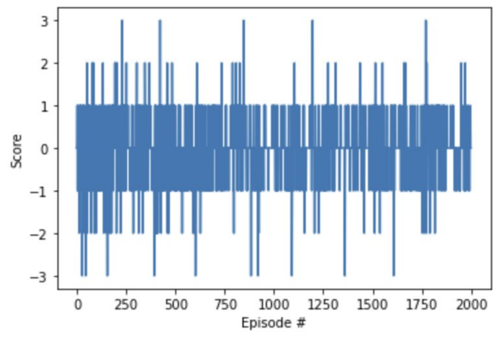
The algorithm is run for 2000 episodes with a maximum of 1000 steps within each episode. Once the average score remains constantly above 13 for a couple of hundred runs, the model checkpoint is saved, and the RL agent is said to be trained.

3. Experiments

3.1 Implementing the Q-network without experience relay

In order to understand the significance of the experience replay, the navigation project was implemented without experience replay. The results are as follows:

```
Episode 100     Average Score: -0.16
Episode 200     Average Score: -0.09
Episode 300     Average Score: -0.08
Episode 400     Average Score: 0.021
Episode 500     Average Score: -0.12
Episode 600     Average Score: -0.10
Episode 700     Average Score: -0.09
Episode 800     Average Score: 0.021
Episode 900     Average Score: 0.13
Episode 1000    Average Score: -0.04
Episode 1100    Average Score: -0.10
Episode 1200    Average Score: 0.032
Episode 1300    Average Score: -0.05
Episode 1400    Average Score: 0.051
Episode 1500    Average Score: 0.04
Episode 1600    Average Score: 0.021
Episode 1700    Average Score: 0.031
Episode 1800    Average Score: -0.07
Episode 1900    Average Score: 0.061
Episode 2000    Average Score: 0.031
```

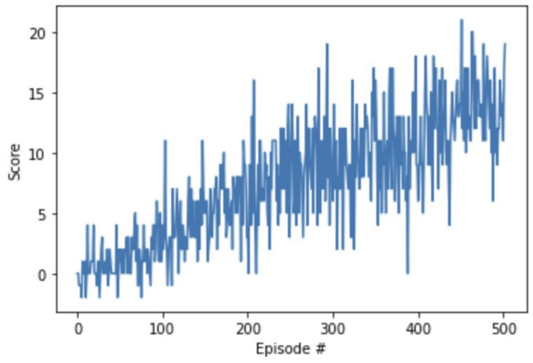


From the diagram, we can see that the average score does not show improvement at all. The value remains constantly low, and this can be attributed to high correlation, that is throwing off the learning.

3.2 Implementing the Q-network with experience replay

From the diagram, we can see that the average scores improve immediately. The agent learns about the environment faster as the correlation is broken which enables it explore the environment without being biased by previous states.

```
Episode 100     Average Score: 1.06
Episode 200     Average Score: 4.92
Episode 300     Average Score: 8.44
Episode 400     Average Score: 9.82
Episode 500     Average Score: 12.83
Episode 503     Average Score: 13.07
Environment solved in 403 episodes!     Average Score: 13.07
```



Hence the agent fully learns the optimal policy for the current problem after 403 episodes.

4. Future work

Different types of Q networks can be used to improve the working of the current implementation. Double Q networks and dueling Q networks can be used as alternate solutions. In the Double Q-learning algorithm, we have a model Q and a target model Q' instead of two independent models. We use the Q' for action selection and Q for action evaluation. In the dueling Q networks we split the state-dependent action advantages and the state-values into two separate streams.