Report

The problem was solved using a simple Deep Q-Network algorithm. Firstly, the neural network architecture consists of 3 layers, each with the ReLU activation function. The input size is 37, corresponding to the number of states of the agent, while the output size is equal to the possible actions i.e. four. There are two such networks, one local and one target. While the former one is used to expected Q-values, the later computes the maximum predicted Q-values. The difference between the two calculates the loss as a mean squared error which is back-propagated through the network to update the weights. The spikes in the bellow figure are due to the target network being updated.

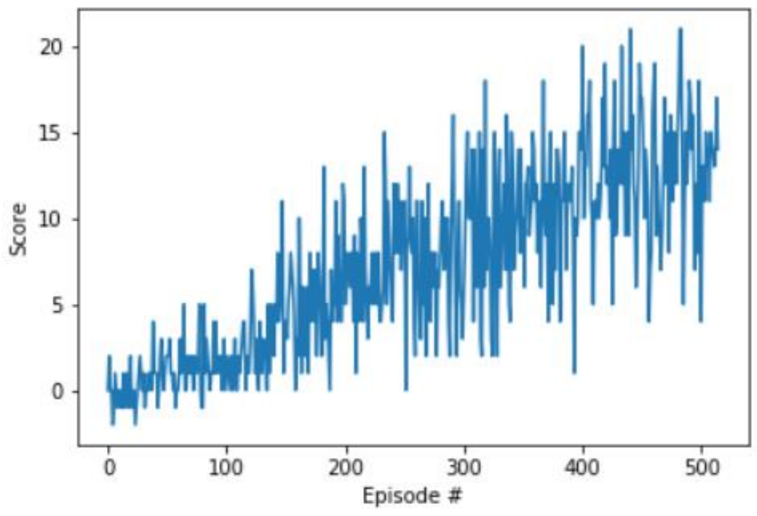


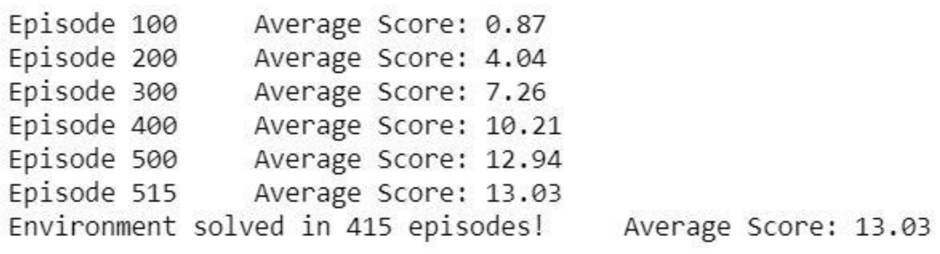
Figure 1. The scores over the episodes

One of the hyperparameters chosen is the discount factor, set to 0.99. This allows for future rewards to be highly considered. Another hyperparameter is the batch size. Using 64 allows the model to better generalize. The hyperparameter tau, chosen as 0.001, is used to determine by how much to update the target weights. Finally, but not least important, the hyperparameter corresponding to the how often to update the target network weights was set to four i.e. every four steps.

The action was selected according to a greedy-epsilon policy: the action corresponding to the highest Q-value is chosen with a 1-epislon possibility, otherwise the action is selected randomly. The agents starts with a epsilon value of 1 (thus encouraging exploration), and it linearly decreases until 0.01, this exploiting its accumulated knowledge.

An important feature of the training algorithm is Replay Experience Buffer. This is used to store the most recent states, rewards and actions that the agent has experienced. Then, a batch from these is sampled and it is used to train the network. This helps to reuse past experiences of the agent. The size of the buffer is 100,000, keeping the most recent events and overriding the old ones.

The result can be observed in the figure bellow. Over time, the agent starts to learn the environment, thus it accumulates more positive rewards and the scores increases. It took 415 episodes in order to solve the environment.



```
Episode 100     Average Score: 0.87
Episode 200     Average Score: 4.04
Episode 300     Average Score: 7.26
Episode 400     Average Score: 10.21
Episode 500     Average Score: 12.94
Episode 515     Average Score: 13.03
Environment solved in 415 episodes!     Average Score: 13.03
```

Figure 2. The average score accumulated over the episodes

One improvement to the network is represented by upgrading the replay buffer to a prioritized version. Instead of randomly sampling the events with equal chance, the experiences from which the agent had more to learn could have a higher priority for sampling. Another improvement consists of the method selecting the highest Q-value. A network should select the action with the highest Q-value, but the Q-value used should be obtained from a different network.