1 Learning Algorithm

For this project I used a Deep Q-Network (DQN) model. The aim is to train a policy that tries to maximize the discounted, cumulative reward $R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$, where R_{t0} is also known as the return.

I used PyTorch for the implementation of the DNN. The experience gained by the agent while acting in the environment is saved in a memory buffer, and a small batch of observations from this list is randomly selected and used as the input to train the weights of the DNN (i.e., Experience Replay). At each step, the agent follows an ϵ-greedy approach to select an action. The value of ϵ is continuously decayed after every episode to gradually focus on exploitation rather than exploration.

1.1 Model architecture

I used a DQN with two hidden layers, with ReLU as the activation function in both layers. The input layer (of size 37) is connected to the first hidden layer with 32 neurons, which is then connected to the second hidden layer with 64 neurons. Finally, the second hidden layer is connected to the output layer of 4 neurons, with a linear activation function. The network uses an Adam optimizer. Note that I did consider different architectures with more layers and neurons, however, the aforementioned one gives a good compromise between performance and computation time.

As shown in Figure 1, the agent's performance when using a batch size of 32 was marginally higher towards the later episodes. Therefore, the batch size was set to 32.

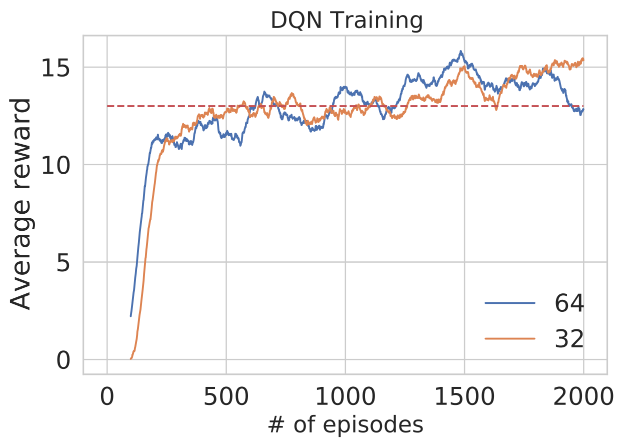

Figure 1: Average reward for each batch size.

1.2 Hyper-parameter tuning

The implemented agent requires three hyper-parameters as input. They are, (i) the learning rate (α) of the optimizer, (ii) the discount factor (γ) of future rewards, and (iii) the decay rate of ϵ (ϵ-decay).

First I tuned the values for α and γ using a grid search, while keeping the value of ϵ-decay constant at 0.999. The value for α was selected from $\{0.01, 0.001, 0.0001\}$ and the value for γ was selected from $\{0.9, 0.99, 0.999\}$.

Figure 2 depicts the rolling average reward over the last 100 episodes of the agent while trying the nine possible combinations of learning rates and discount factors over 2,000 episodes. As the figure shows, the best results were achieved with $\alpha = 0.0001$ and $\gamma = 0.99$ (the $\alpha = 0.0001$ and

$\gamma = 0.9$ combination has more fluctuations in average reward). With α and γ selected, I selected a suitable value for ϵ-decay from $\{0.9999, 0.999, 0.99, 0.9\}$. Figure 4 depicts the rolling average reward for the different ϵ-decay values. Since ϵ-decay $= 0.999$ seems to provide better than the others, it was selected.

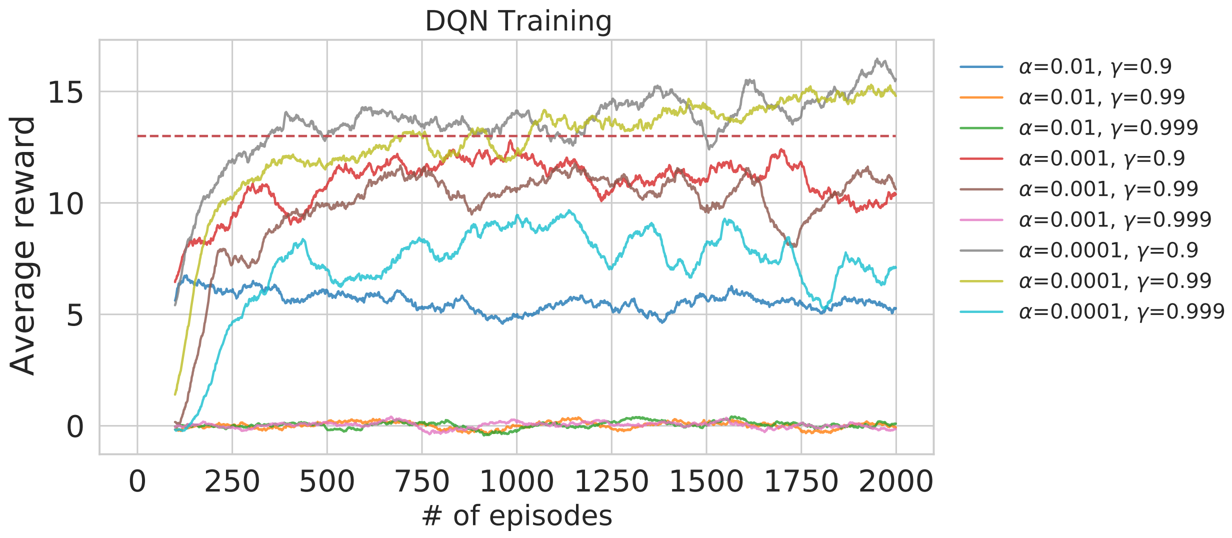

Figure 2: Average reward for each batch size.

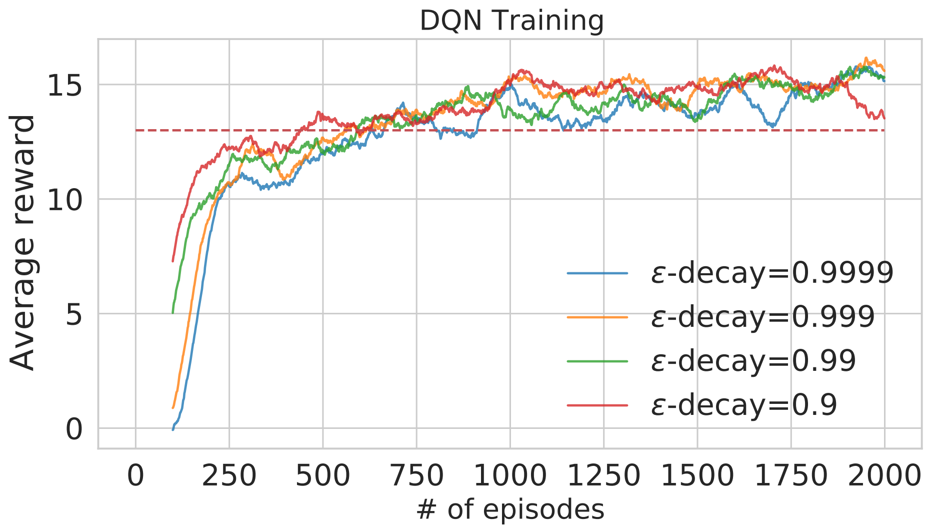

Figure 3: Average reward for each batch size.

There are a few other parameters that need to be set for the agent to solve the environment. Those values, listed below, were selected using a trial and error approach.

- Size of the replay buffer - 10,000
- Total episodes - 2,000
- ϵ minimum - 0.01

- Maximum number of steps - 1,000

2 Results

Figure 4 shows the reward per episode, as well as the rolling average reward for over the last 100 steps for the best DQN model. The agent can solve the environment (i.e., get an average score of +13 over 100 consecutive episodes) in 558 episodes.

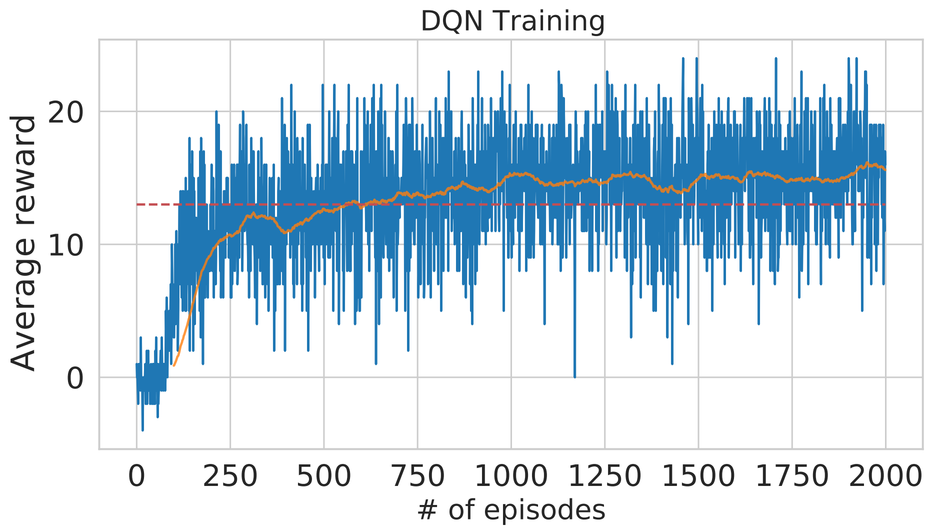

Figure 4: Average reward for each batch size for the selected model.

3 Future Work

A more exhaustive search for hyper-parameters could result in faster convergence of the DQN model. It would also be interesting to see how different network architectures affect the agent's behavior. The use of a Double Q-learning Network (DDQN) instead of a simpler DQN may increase the accuracy of the agent. Although there is room for improvement, the current implementation solved the environment well above the required threshold.