Learning Algorithm

In the lesson on Deep Q-Networks (DQNs), I experimented with a DQN implementation in the context of an OpenAI Gym task. I adapted the provided solution to that task to this navigation project, making changes to account for the fact that I was solving a different environment.

At the heart of this learning algorithm, deep Q-learning, is a multilayer perceptron. The first layer of neurons takes a state (37 inputs), performs a linear transformation, passes the results through the ReLU activation function, and returns 64 outputs, which go into the second layer of neurons. The second layer transforms them into 64 outputs again in the same manner. Finally, there's a third layer which takes the second layer's outputs and performs a linear transformation to produce four final outputs without passing them through an activation function. The final outputs are the four action values associated with taking the four possible actions in the 37-dimensional input state.

In this implementation of deep Q-learning, experience replay and a fixed target Q-network are employed to boost training stability and efficiency. The former decorrelates the sequential observations made by the agent in an episode of training. The latter decorrelates the target and predicted Q-values in a Q-network update step. Here are the major steps of this implementation.

1. Initialisation. Two instances of the multilayer perceptron are created and initialised with the same dimensions and seed. One is the target Q-network and the other is the local Q-network.
2. Training. The agent is trained in an episodic task. During each episode, each transition (time step) is stored as a tuple, which is also called an observation in this report, in a replay buffer. It comprises the agent's current state, the action it takes, the reward for this state-action pair, the next state, and whether the transition ends the episode. The action is selected based on the local Q-network's outputs and by following the epsilon-greedy policy (behaviour policy). This allows the agent to strike a balance between exploiting its experience and exploring the state-action space.
3. Updating both Q-networks once every *UPDATE_EVERY* transitions. This should be considered a part of training.
    - *BATCH_SIZE* tuples are sampled randomly from the replay buffer.
    - For each sampled tuple, the target Q-network is used to compute the target Q-value by following the greed policy (target policy). It means that the learning algorithm is an off-policy algorithm because the behaviour and target policies are different.
    - For each target Q-value, the difference between it and the Q-value predicted by the local Q-network is computed. The sum of these differences over the

sampled tuples (loss function) is used to update the local Q-network's weights by backpropagation.
    - After updating the local Q-network, the target Q-network is updated by blending it with the updated local Q-network. A small fraction (*TAU*) of the updated local Q-network's weights is added to a large fraction (1-*TAU*) of the target Q-network's weights to update the latter.
4. Termination. When an episode ends, the cumulative reward for the episode is calculated. When the average cumulative reward over 100 consecutive episodes is at least 13, the algorithm terminates.

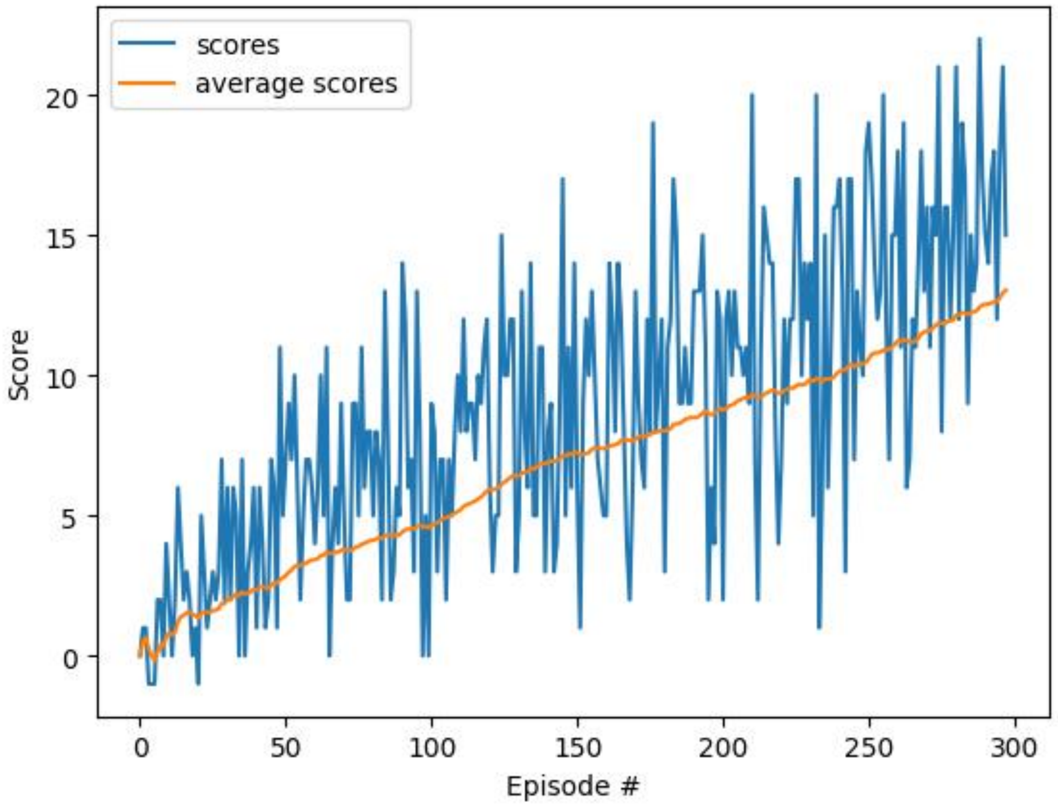
At first, I used the default hyperparameters in the provided solution to the OpenAI task.

- Maximum number of episodes (*n_episodes*): 2000.
- Maximum number of transitions in an episode (*max_t*): 1000.
- Epsilon-greedy policy parameters. Initial probability of exploring, minimum probability of exploring, and probability decay rate (*eps_start*, *eps_end*, and *eps_decay*): 1.0, 0.01, 0.995.
- Number of transitions before updating the Q-networks (*UPDATE_EVERY*): 4.
- Number of tuples sampled from the replay buffer (*BATCH_SIZE*): 64.
- Discount rate in Q-learning (*GAMMA*): 0.99.
- Learning rate of the local Q-network (*LR*): 5e^{-4}.
- Update rate of the target Q-network (*TAU*): 1e^{-3}.

After experimenting with the environment, I observed that the high value of *max_t* was encouraging the agent to procrastinate, leading to slow learning. Therefore, I decreased it to 300. This allowed me to expose the agent to a broader range of initial conditions (more episodes) in a unit of computational time. I reasoned that by diversifying the initial conditions, I was forcing the agent to explore the state-action space already, so I decided to tone down the epsilon-greedy policy by reducing *eps_start* and *eps_decay* to 0.1 and 0.987 respectively. The following hyperparameters were used in the final learning algorithm.

- Maximum number of episodes (*n_episodes*): 2000.
- Maximum number of transitions in an episode (*max_t*): 300.
- Epsilon-greedy policy parameters. Initial probability of exploring, minimum probability of exploring, and probability decay rate (*eps_start*, *eps_end*, and *eps_decay*): 0.1, 0.01, 0.987.
- Number of transitions before updating the Q-networks (*UPDATE_EVERY*): 4.
- Number of tuples sampled from the replay buffer (*BATCH_SIZE*): 64.
- Discount rate in Q-learning (*GAMMA*): 0.99.
- Learning rate of the local Q-network (*LR*): 5e^{-4}.
- Update rate of the target Q-network (*TAU*): 1e^{-3}.

Plot of Rewards



Using the learning algorithm, I solved the environment in 198 episodes. In the 100 episodes after that, the trained agent achieved an average episodic cumulative reward (average score) of 13.04.

Ideas for Future Work

If I want to improve the learning algorithm further, I will start by tuning the remaining hyperparameters, especially *LR*. From personal experience, the performance of deep learning is heavily influenced by the learning rate.

Then, I will refine the deep Q-learning algorithm by adding combinations of the six modifications constituting the Rainbow algorithm. Prioritised experience replay is particularly important. My results prove that the deep Q-learning algorithm is already capable of teaching the agent a reasonably good policy. Prioritised experience replay could incentivise the agent to focus on the states where small differences in its policy result in huge differences in Q-values.

After that, I will go beyond value-based methods and experiment with policy-based methods.