Eric Chen

CID : 01936805

Reinforcement Learning (Second Half) Coursework

Question 1 (i) By observing both graph, we can note that the variance of the loss for the online learning is higher compared to the one for mini-batch learning because we can see more oscillations in the curve. As a result, the method which gives the most stable results is mini-batch learning because instead of using each transition in the order they were collected as in online learning, mini-batch learning relies on randomly selecting several samples from the data. This maintains a more uniform data distribution across the environment.

Question 1 (ii) The mini-batch learning method is more efficient at improving the Q-network's predictive accuracy because it has a lower overall loss than online learning. We know that in online learning parts of a Q-network which have not been trained on for a while will be overwritten. For mini-batch learning, the entire state-action space is updated at once, allowing the updates to be spread across the network. As a result, this explains why mini-batch learning can lead to a better predictive accuracy per episode of interaction.

Question 2 (i) We know that the initial state is located at the bottom of the environment. Because the agent begins its training at the bottom, it is more likely for it to explore the bottom right of the environment than the top right because the distance between them is smaller. The wall which separates the top and the bottom does not help the agent to explore the top part of the environment. As a result, Q-value predictions will be more accurate for the bottom-right compared to the top-right because the agent would explore more this part of the environment.

Question 2 (ii) By looking at the greedy policy, the agent will go to the top and bump into the wall. As a consequence, the agent would not reach the goal. This is because the loss is computed with the predicted reward. As a result, the agent cannot really explore the environment and it will favor the shortest path to go the goal state. This is why the agent bumps into the wall.

Question 3 (i) Now that we have introduced the Bellman equation, the agent has a better estimation of the true Q-values by using the maximum Q-value of the next state. The agent can now care about rewards in the future and is more far-sighted because it can have a better understanding of what action to use for the next state.

Question 3 (ii) When plotting the loss curves, Figure 3 (a) shows a graph where the loss quickly decreases and then begins to oscillate. When we use the target network, we observe in Figure 3 (b) that there are periodic peaks. This is because we update the target network with the weights of the Q-network every 10 episodes. We use the target network in the Bellman equation to compute the loss as a way to counter the effects of excessive generalisation. Because the target network is periodically updated with the parameters of the Q-network, the loss drastically changes at this moment which explains the shape of the curve for Figure 3 (b).

Question 4 (i) If ϵ is always 0.0, the agent will never reach the goal. Indeed, we know that ϵ represents a trade-off between exploration and exploitation. If we have a high value of ϵ, the agent will explore more the environment, looking for more options and paths, while having a low ϵ means allowing less exploration and exploits the exiting path. As a result, by having ϵ equal to 0.0, the agent will never explore the environment and will always execute the greedy policy. Thus, the agent will never know that there is an existing path which is better that allows to bypass the wall and to reach the goal.

Question 4 (ii) Before the Q-network has converged, if the agent continues on in a straight line and hits the wall on the left of the environment after reaching the goal, it can be due to the fact that the agent has not learnt enough information from the states near the goal state. For instance, the agent may have never explored these states. Due to this lack of information regarding the Q-values, when executing the greedy policy, the agent can go to the left instead of going back to the goal state even it is not the optimal action because the action which leads to the highest Q-value is the left one.

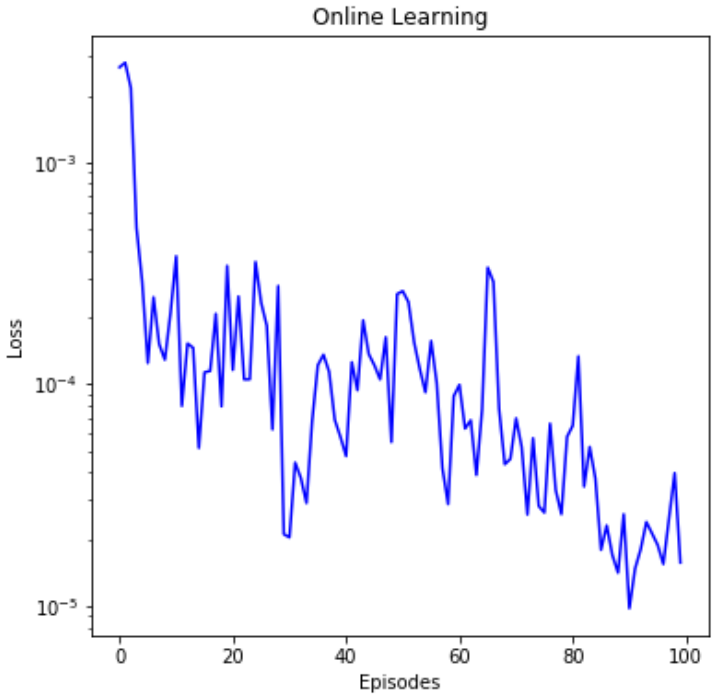

Figure 1 (a) Loss vs Episodes (Online Learning)

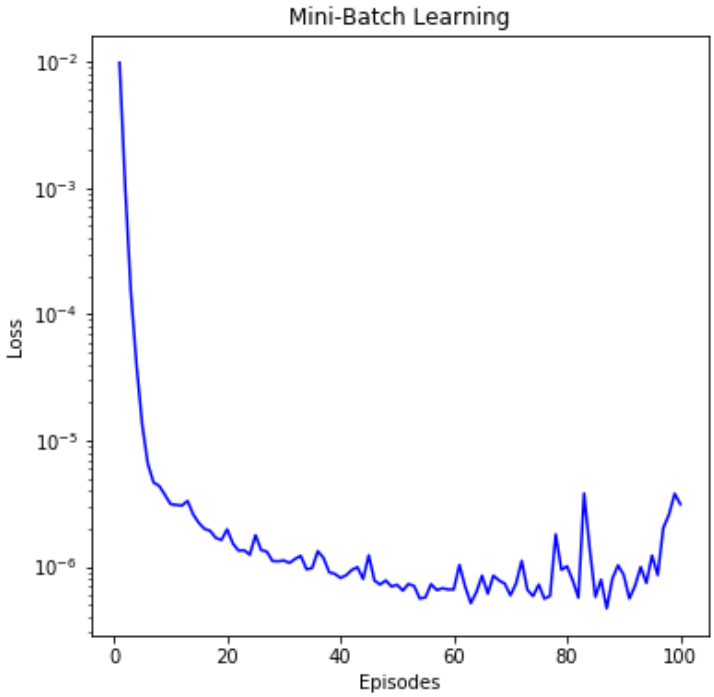

Figure 1 (b) Loss vs Episodes (Mini-Batch Learning)

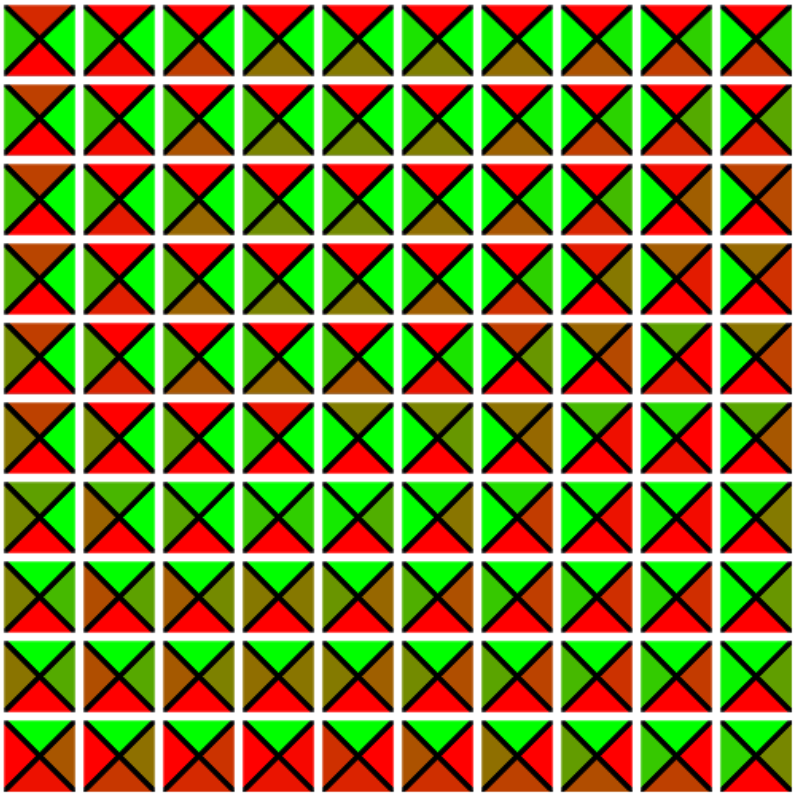

Figure 2 (a) Visualisation of Q-values

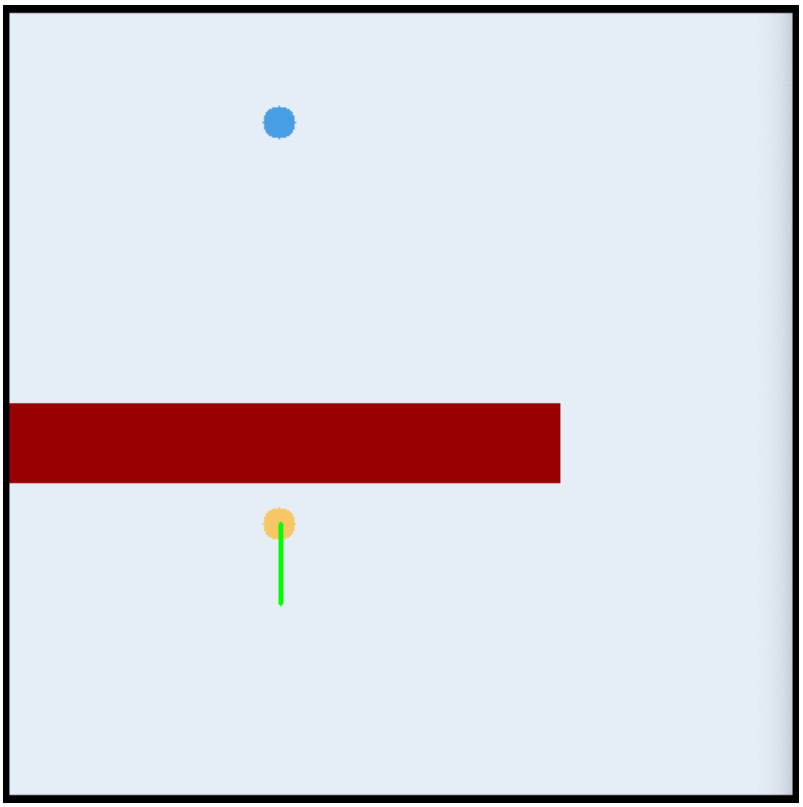

Figure 2 (b) Visualisation of the greedy policy

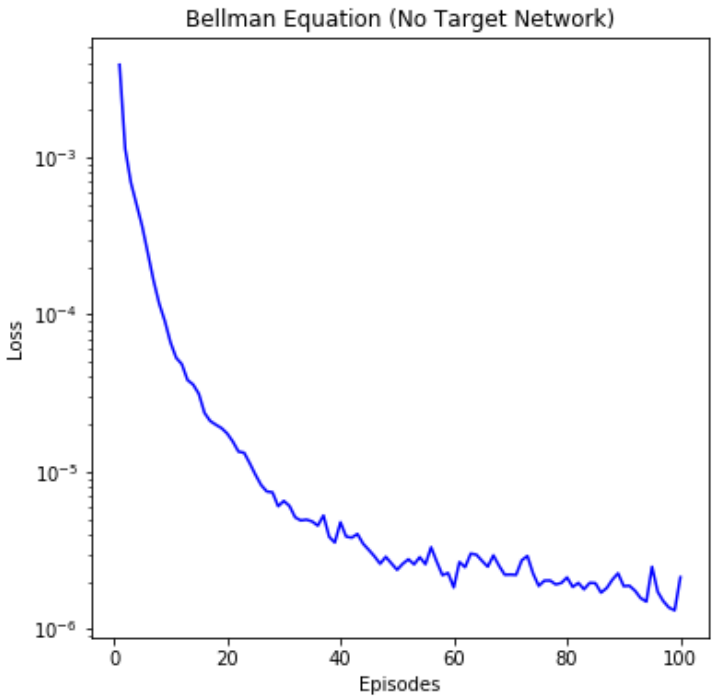

Figure 3 (a) Loss Vs Episodes (No target network)

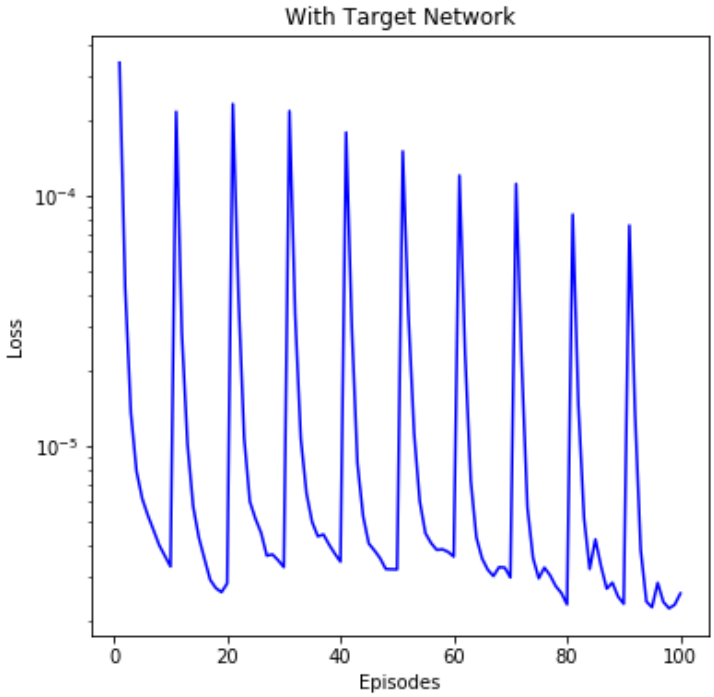

Figure 3 (b) Loss vs Episodes (With target network)

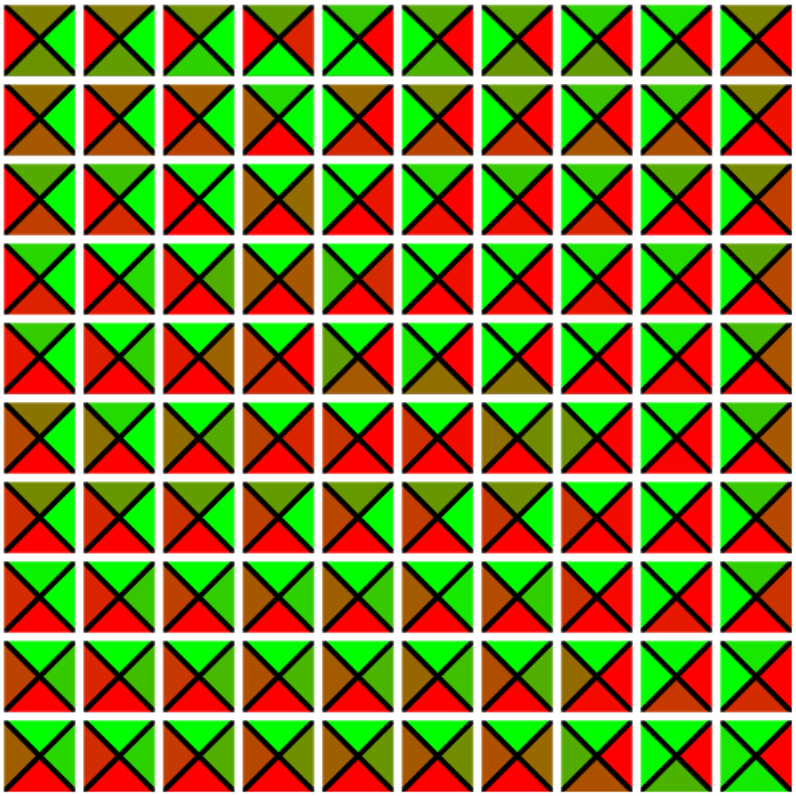

Figure 4 (a) Q-values (Exploration vs Exploitation)

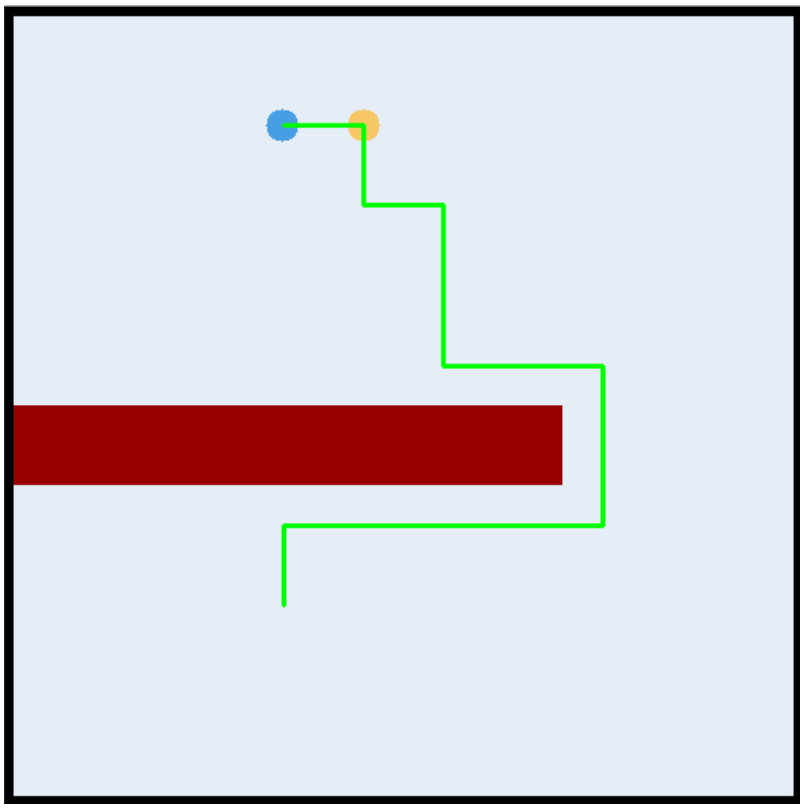

Figure 4 (b) Greedy policy for an episode length of 20 (The blue node is the goal, reached in 19 steps, and corresponds to the 2nd row-4th column in Figure 4 (a))

Description of Implementation for Part 2

For the implementation of Part 2, we use a similar structure as for Part 1. In the agent.py file, we will introduce 4 classes: $ReplayBuffer$, $Network$, DQN and $Agent$. We choose to implement Prioritized Experience Replay and Double Q-Learning with target network. We also have the option to perform early stopping. Every 3 episodes, we will let the agent execute the greedy policy for the first 100 steps. If the agent reaches the goal in less than 100 steps, then it will always execute the greedy policy for the following episodes and stop early the entire training. Else, we will continue the training.

First at all, in the class $Agent$, we will initialise some of our parameters (line 26 to 60). For instance, we will take an episode length of 200 and create a dictionary of continuous actions (line 37 to 39). We choose to only use 3 actions : right, top and bottom. Then, we can start by looking at the function get_next_action (line 85 to 105). This function will return the next action depending on the situation. As stated earlier, the first case (line 91 to 93) represents the option of executing a greedy action (line 140 to 147) every 3 episodes and for the first 100 steps of the episode. The second case (line 94 to 97) represents a normal training when we choose the action according to a ϵ-greedy policy thanks to the function _choose_next_action (line 72 to 82). The last case (line 98 to 100) is used after performing early stopping and it consists in always taking the greedy policy until the end of the training.

We can now look at the function set_next_state_and_distance (line 108 to 138) in the class $Agent$. This function will be used to train the agent if we have not performed yet early stopping (line 109) and there are 2 cases. For the first case (line 110 to 114), we look at the remaining distance to reach the goal (line 111) while we execute the greedy policy. If the distance is inferior to 0.03, this means that when executing the greedy policy, the agent has reached the goal in less than 100 steps so we can stop early the training (line 112). This means that for the next episodes, nothing will be updated. For the second case (line 115 to 138), we first convert the distance to a reward (line 117). Then, we create a transition that we will add to the replay_buffer and we will assign a maximum weight to it (line 119 to 129). Adding a transition and its corresponding weight is done by using the functions add_transition and add_weights from the class $ReplayBuffer$ (line 243 to 247). Then, we wait for the replay_buffer to have the required length to sample mini-batch (line 131) and we compute the loss using Double Q-Learning (line 133). The weight for each transition is also computed and we update the weight for the prioritized experience replay (line 133) thanks to the function set_weight (line 249 to 252). For the Double Q-Learning, the implementation can be found in the class DQN (line 171 to 232) and depends on the class $Network$ (line 151 to 167). After that, the value of ϵ (for exploration) will vary depending on the number of steps taken and the number of episodes, but it will decrease overall (line 136). Then for each 100 steps, we will update the target network (line 137 to 138).