Cesare Magnetti

CID: 01413438

Reinforcement Learning (Second Half) Coursework

Question 1

For the first two parts of this question the agent was let to randomly explore the environment and only immediate rewards were considered.

Question 1 (i) It is clear from the diagrams reported in Fig. 1 that using an experience replay buffer (1b) significantly lowers the variance of the Q-Network loss and allows for a more stable training. This is due to the subsequent training samples being highly correlated in online learning (they come in temporal order). Furthermore, since the function is continuous, the updates will also affect close by regions (which have just been updated in the previous step), this will cause rapid changes in the Q-values function leading to the higher variance portrayed in diagram (1a). On the other hand, mini-batch learning de-correlates the training data by uniformly sampling over past transitions causing the whole Q-space to be updated at once.

Question 1 (ii) The diagrams in 1 are also showing a lower loss for mini-batch learning (1b), which leads to a more efficient estimation of the actual Q-values. This is again expected since Neural Networks are non-linear functions and we are trying to find the optimal parameters to minimise a Mean Squared Error (MSE) loss via gradient descent: one transition is not sufficient to tune the parameters to find the optimal value for that state. Keep in mind that we are additionally gathering the output for the chosen action, hence only part of network is trained at each backward pass, meaning that even more transitions are needed for the minimisation process.

Question 2 (i) The Q-values visualisations showing what the agent has learnt are reported in Fig. 2a. We can see that the top right corner state-action values are significantly wrong, mostly indicating the complete opposite expected value, whereas the bottom right corner correctly suggest to move up and left. This is due to the fact that random movement is more likely to transition over the bottom right corner, as it does not have to go through the obstacle, which reduces the chance of arriving to the top right corner. More transition over such states means a more accurate estimate of their state-action values.

Question 2 (ii) Looking at the greedy policy shown in Fig. 2b it is trivial to see that the agent will not reach the goal as it will be blocked by the obstacle. This is due to the agent focusing on immediate rewards ($1 - ||state_{xy} - goal_{xy}||^2$), urging

it to follow a straight path towards the goal to maximise its return. Furthermore, 100 episodes of 20 steps are definitely not enough for the agent to explore the whole environment under random movement.

From this part on wards we introduce the Bellman's equation and we stop only considering immediate rewards.

Question 3 (i) The agent is now able to bootstrap over future state-action values to infer the cumulative discounted rewards of his actions. This step is necessary for the agent to understand that moving away from the shortest path towards the goal, in order to overcome the obstacle, will be highly beneficial.

Question 3 (ii) The Q-network losses with and without the use of a target network are reported in Fig. 3. We can see that approximately every 10 episodes the Q-network loss jumps when using the target network to infer the current values. This is due to the sudden change of weights when the target network is set equal to the Q-Network, which leads to a different estimation from what was expected. Choosing an excessively large delay will lead to have biased estimates of the true current Q-value, as the two networks will differ too much from each other. On the other hand, reducing the delay too much will result in a possibly unbounded estimation of the Q values due to recursive updates caused by the smoothness of the Q-function itself (reason why the target network was introduced in first place).

From this part on wards we introduce an $\epsilon - greedy$ exploration of the environment rather than taking random actions. The agent was trained for 400 iterations of length 20 steps, Q-Learning algorithm was used (with a target network to estimate current state values) with a learning rate of 0.0002 and an ϵ decay of 5% at each iteration. An experience replay buffer with a maximum capacity of 5000 transitions was used and a batch size of 100 was chosen to uniformly sample from it at each step. Ressults are shown in Fig. 4.

Question 4 (i) With $\epsilon = 0$ the agent would not reach the terminal state as it would never explore the environment and always exploit on the random initialisation of the Q-Network (which is for sure not the optimal value function).

Question 4 (ii) The mentioned statement could happen since, due to the smoothness of the Q-function, the value of neighbourhood states to the goal could be slightly boosted by the agent reaching the goal, leading to the agent possibly exploiting on biased values close to the goal. This can lead the agent to enter a region where the Q-function (since it has not converged yet) is not correctly estimating the actual state values and the agent will move along a wrong trajectory (and hence hitting the wall perhaps). Ideally, after the network has converged to an optimal Q-function this will not happen.

(a) Q-Network loss using online learning.

(b) Q-Network loss using replay buffer.

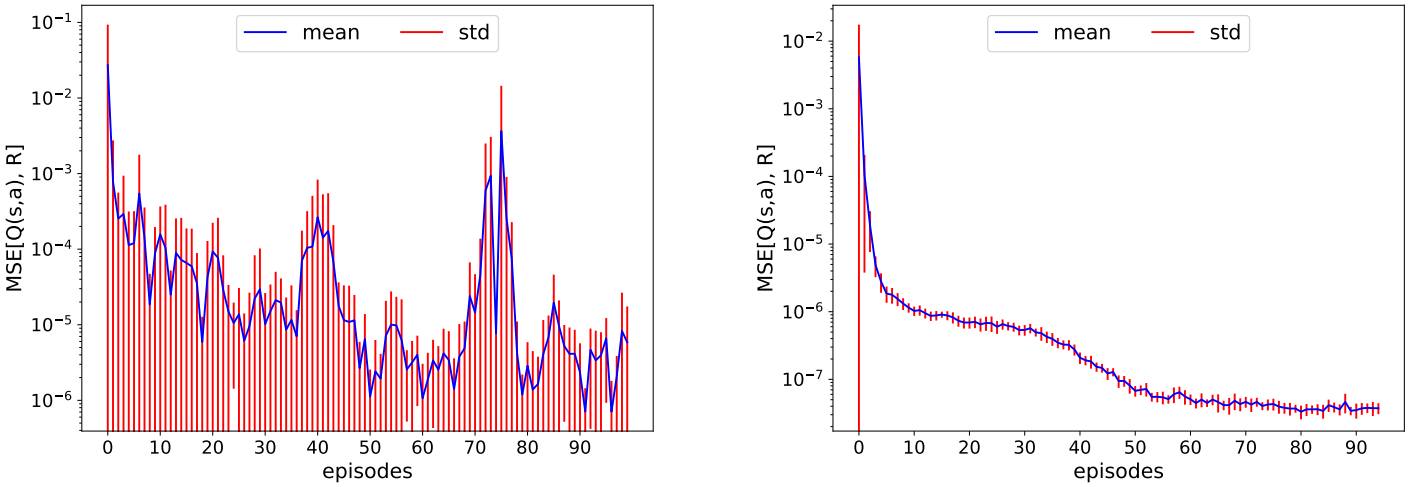
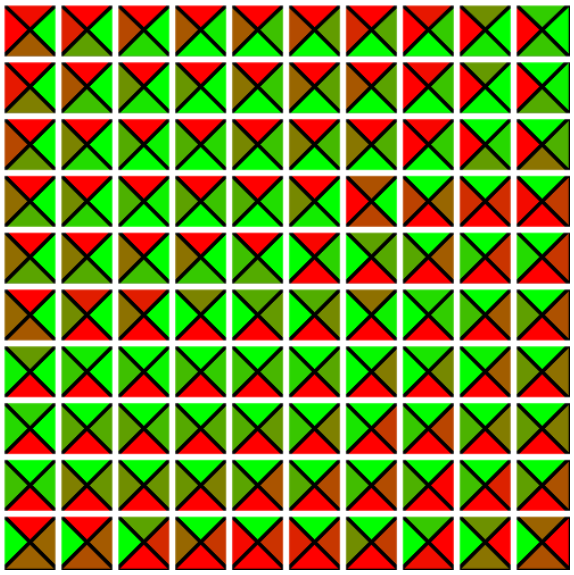


Figure 1: diagrams showing results for question 1.1, where the agent randomly took 20 Steps for 100 episodes. It is clear that mini-batch learning (with $batchsize = 100$) shown in b) drastically reduces the variance compared to a). It is also important to notice that b) achieves a lower loss by an order of magnitude. Both observations are explained in Question 1.1.

(a) Visualisation of the Q-value estimates.



(b) Visualisation of the greedy policy.

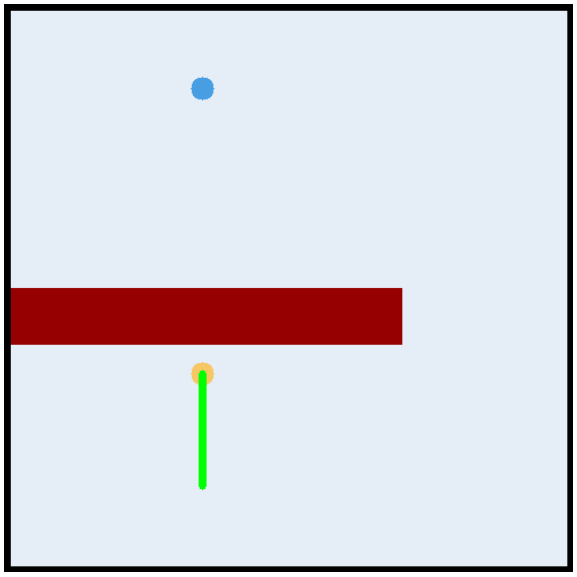


Figure 2: diagrams showing results for question 1.2, where the agent randomly took 20 Steps for 100 episodes using an experience replay buffer (with $batchsize = 100$). a) color-coded values of taking each action in each state (the greener the better), we can see that there is no clear sequence of actions that the agent should take to successfully reach the goal. It is clear from b) that the agent will not arrive to the goal. Both observations are explained in Question 1.2.

(a) Q-Network loss.

(b) Q-Network loss with a target network.

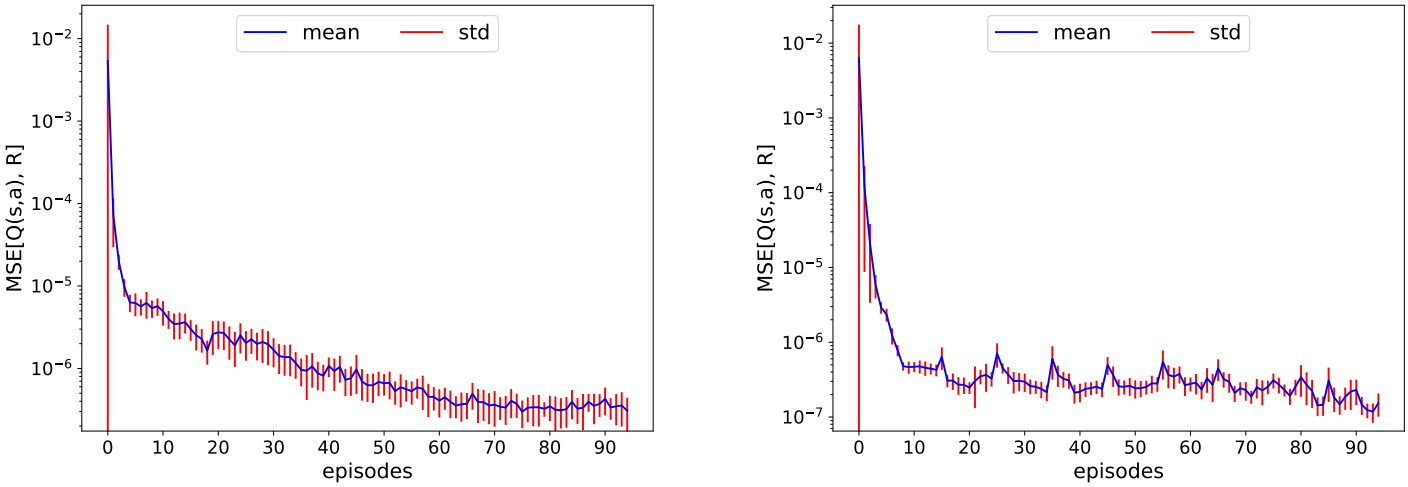
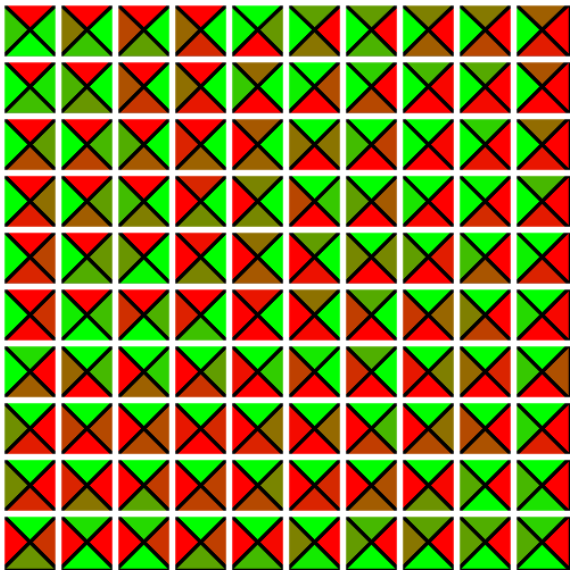


Figure 3: diagrams showing results for question 1.3, where the agent randomly took 20 Steps for 100 episodes using an experience replay buffer (with $batchsize = 100$). We can see that b) shows slightly lower variance throughout training and that around every 10 episodes (exactly when we update the target network), the loss makes a small jump. Both observations are explained in Question 1.3.

(a) Visualisation of the Q-value estimates.



(b) Visualisation of the greedy policy.

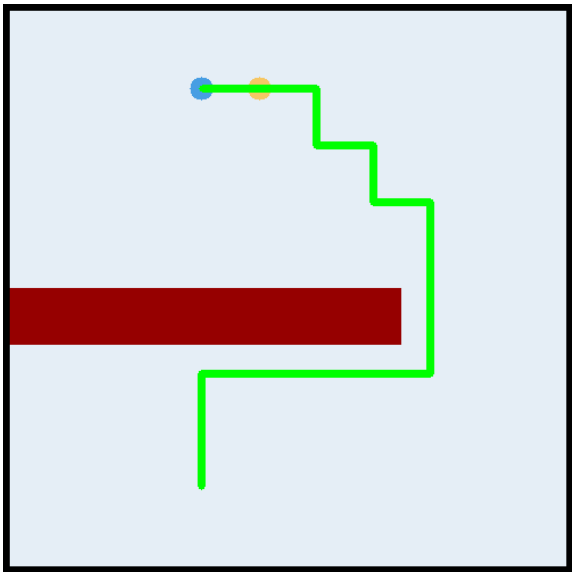


Figure 4: diagrams showing results for question 1.4, where the agent was trained under an $\epsilon - greedy$ policy for 400 iterations of length 20 steps, using Q-learning. a) color-coded values of taking each action in each state (the greener the better), we can see that now, the favorable actions depicted in a) resemble the trajectory shown in b). It is clear from b) that the agent arrived to the goal. Both observations are discussed in more detail in Question 1.4.

Description of Implementation for Part 2

The following paragraphs will describe the main decisions that were taken throughout the implementation of question 2 and the reasons behind them. Note that as the code is well commented, there will not be any explanation on it in this report. Firstly, a Q-Learning implementation with the following hyper-parameters was found to perform well under a number of different environments.

- *Q-Network architecture*: many experiments were conducted to fine-tune network architecture, the best model found consisted of an input layer taking the xy coordinates of the state, 2 hidden layers with 100 features coupled with ReLU and an output layer with 4 nodes, one for each possible action.
- ϵ *decay rate*: ϵ must be decayed throughout the episodes in order to achieve an exploitable optimal policy, many decay strategies were tried (exponential, linear, inverse, etc.), in the end it was chosen to settle with a simple 5% decrease in ϵ at the end of every episode as it gave more consistent results.
- *variable episode length*: We want the agent to learn how to get to the goal quickly, therefore we should not allow it to take a large number of steps. However, in the early epochs, we want the agent to explore as much of the environment as possible. Hence, a variable episode length was chosen, starting from 1500 steps and decreasing by 2.5% at the and of each episode, without going under 150 steps per episode. Again this hyper-parameter was found through experiments.
- *target network updates*: after some trial and error, consistency was found updating the target network every 1000 steps.
- *loss function*: after some literature review, it was concluded that state of the art DQN suggest the use of the Huber loss rather than MSE loss due to its higher resistance to outliers. Significant improvement was seen upon this change.
- *buffer hyper-parameters*: an initial maximum capacity of 5000 was chosen, but was quickly updated to 10000 as the agent could not converge to the goal state otherwise. As the network architecture was increased from a single hidden layer to 2 hidden layers, it was decided to double the batch size from an initial guess of 128 to 256, as bigger network requires more training samples to converge.

Secondly, a simple modification to the DQN class (which handled the training of the Q-network) sufficed to implement Double Q-Learning, great improvement was observed upon this simple change. Finally, a prioritised replay buffer was implemented, and an α prioritisation factor of 1 was found to give more consistent results. A strong boost in performance was found by developing an early-stop condition, which tested the greedy policy when the agent managed to get to the goal state at low values of ϵ and reduced the learning rate until the agent consistently hit the target.