EL2805 Reinforcement Learning
Computer Lab 2

Núria Casals
casals@kth.se
950801-T740
Robin de Groot
robindg@kth.se
981116-T091

Problem 1: Deep Q-Networks

In this report we only answer the questions that require an explicit textual answer. The questions that just required programming were left out here though the required files can be found in our .zip submission.

(b) Why do we use a replay buffer and target network in DQN?

Without an Experience Replay Buffer, using function approximation, the subsequent updates of the main network Q_θ will fall nearby the previous ones and therefore will be highly correlated. This fact could affect the convergence of the Stochastic Gradient Descent algorithm since it requires independent sampling. That is why it makes sense to incorporate a replay buffer. All experiences (s, a, r, s_{t+1}) are added to a buffer of length L from which N experiences are uniformly sampled at each timestep, and all of those N experiences are used to update the network Q_θ at that timestep. This results in a smoother convergence due to a better estimate of the true gradient.

The DQN algorithm updates many state/action values at the same time, and the network Q_θ keeps changing at each timestep. The target network $Q_{\theta'}$ acts as a stable network that is periodically synchronized with the network Q_θ after C timesteps and therefore it reduces the variability of the Q-values used to update the network Q_θ, finding a more robust model.

(d) Explain the layout of the network that you used; the choice of the optimizer; the parameters that you used (γ, L, TE , C, N, ϵ) and which modification did you implement (if any). Motivate why you made those choices.

We have played around with the network size a lot, staying within the recommendations of a maximum of two layers and maximum 128 neurons per layer. We ended up going for a one hidden layer network with 64 neurons. This network still trained fast due to having only one hidden layer, and increasing the number of layers and neurons per layer did not improve performance for us.

The optimizer we used is the Adam optimizer. This is sort of a default for many neural network training tasks and since it worked well also for us, we decided not to deviate from using this optimizer.

The parameters that we used for the network and training are (0.99, 16384, 900, 256, 64, linearly decreasing ϵ) for (γ, L, TE , C, N, ϵ) respectively.

For the discount factor we have tested multiple values, but we found that a rather high value of 0.99 made the network converge fastest. This can be explained because the algorithm is more able to 'look into the future' by assigning a higher value to Q-values of future states, and since landing correctly and receiving the big reward can be quite long in the future, this has a big impact on this problem. A

lower value might be able to let the lander land more quickly, but it would likely require many more episodes to become successful.

The buffer length we arrived at is somewhat in the middle of the suggested range in the assignment document. This is the buffer length we started at and it turned out to work well. We also tested smaller buffer sizes but that worsened performance. We also tested a larger buffer, which performed less well than our chosen buffer length. This is likely due to the fact that in a larger buffer there are more experiences from the random agent or from longer ago, both making the experiences sampled less relevant.

Determining the number of episodes was even more of a trial-and-error process. We started with 100 episodes and saw that the model did not converge yet, after which we decided to let it run for more episodes. To make sure we let the model have enough time to converge, if the parameters allowed for it of course, we then set the episode number at 900. This gave the model enough time to converge and we saw that the model needed less than 800 episodes to converge.

The values for the update frequency and the batch size were determined in tandem as the recommended formula to follow is $C \approx L/N$. The batch size L was determined already, and for the batch size N a value that is a power of two is most efficient for parallelisation and thus computation performance. A value of 64 was chosen for the batch size, which is in the advised range, leading the update frequency C to be set at $\frac{16384}{64} = 256$. These values worked well for us in practice.

A linearly decreasing epsilon from 0.99 to 0.05 was chosen. This is a simpler implementation of a decreasing epsilon. In the beginning, it is important that the model explores many different (state, action) pairs which is done through random actions, meaning that we should start with a high epsilon value. Epsilon should decrease so the model can get closer to the optimal policy as fewer random actions will be taken. Another option is to let epsilon decrease exponentially. We found that a linearly decreasing epsilon let the model achieve good enough results, so we decided not to diverge from that.

(e)

(1) Plot the total episodic reward and the total number of steps taken per episode during training. What can you say regarding the training process?

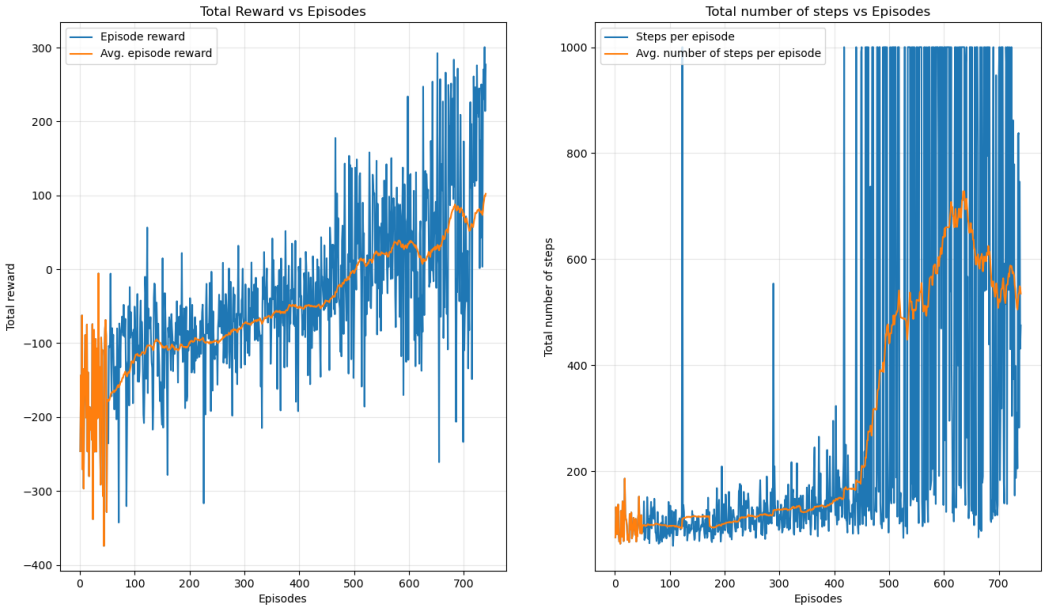

Figure 1: Total episodic reward and the total number of steps taken.

As we can observe in figure 1, the total episodic reward generally increases when the number of episodes increases. Using the computed MSE loss function of the previous experiences, the Stochastic Gradient Descent updates the parameters θ and improves the main network Q_θ. The

function approximation approach implies that, until a certain threshold where the model starts forgetting, the more experience (episodes), the better the final model.

We can also observe that the number of steps increases drastically after 400 episodes of training. This behaviour is because, at that point, the average reward starts being positive, which means that the proportion of successfully landed trials is bigger and, to land successfully, a larger number of steps is needed.

The last episodes used for the training process use a fewer number of steps per episode and they achieve higher total reward. This fact can be seen as a consequence of the training process, which achieved a refined model that can land correctly with fewer steps.

(2) Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor $\gamma_1 = 1$, and a discount factor $\gamma_2 << \gamma_0$. Redo the plots for γ_1 and γ_2 (don't change the other parameters): what can you say regarding the choice of the discount factor? How does it impact the training process?

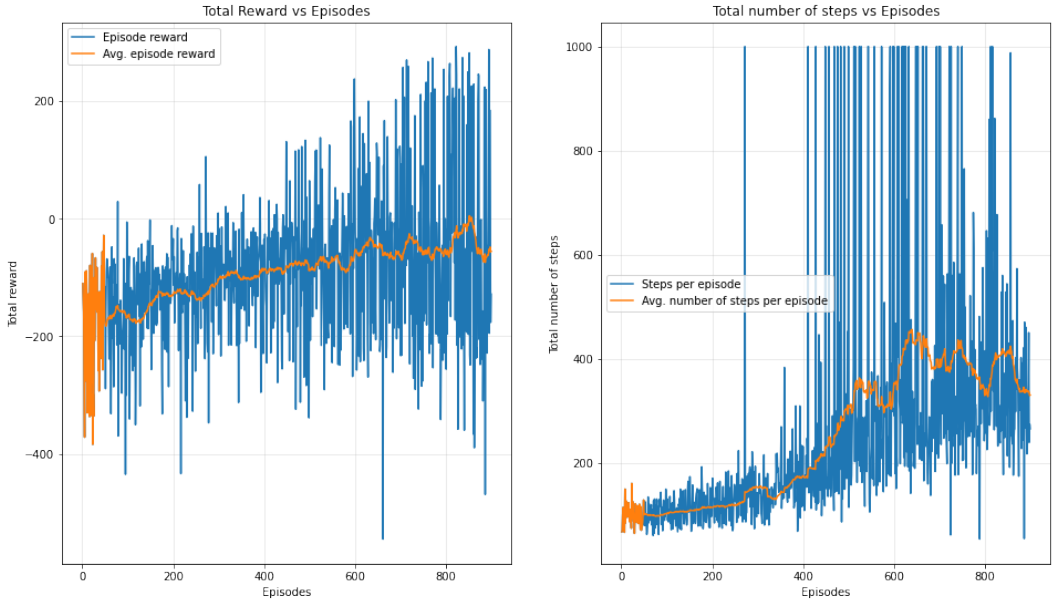

Figure 2: Total episodic reward and the total number of steps taken with $\gamma = 0.5$.

The discount factor determines how much the agent cares about rewards in the distant future relative to those in the immediate future.

Setting γ small impacts the training process because the agent gives a lot of importance to the few next future steps and rewards but does not take into account a larger future horizon, or at least not to the same extent. This translates to a slower learning process as shown in figure 2, where total episodic reward and the total number of steps taken with a discount factor $\gamma = 0.5$ are shown. We can observe an increasing trend of the average episodic reward that means that the model is learning but at a much slower pace.

Figure 3 illustrates the same problem with a discount factor $\gamma = 1$. We can observe that there is no learning from the experience as the number of episodes increases. With $\gamma = 1$, the agent evaluates each of its actions based on the sum of all of its future rewards and the future has the same importance as the present. Setting $\gamma = 1$ means that the problem formulation is not discounted anymore and, for all policies that obtain an average a positive reward at each timestep, the (non)-discounted reward would sum up to infinity and the algorithm will not converge.

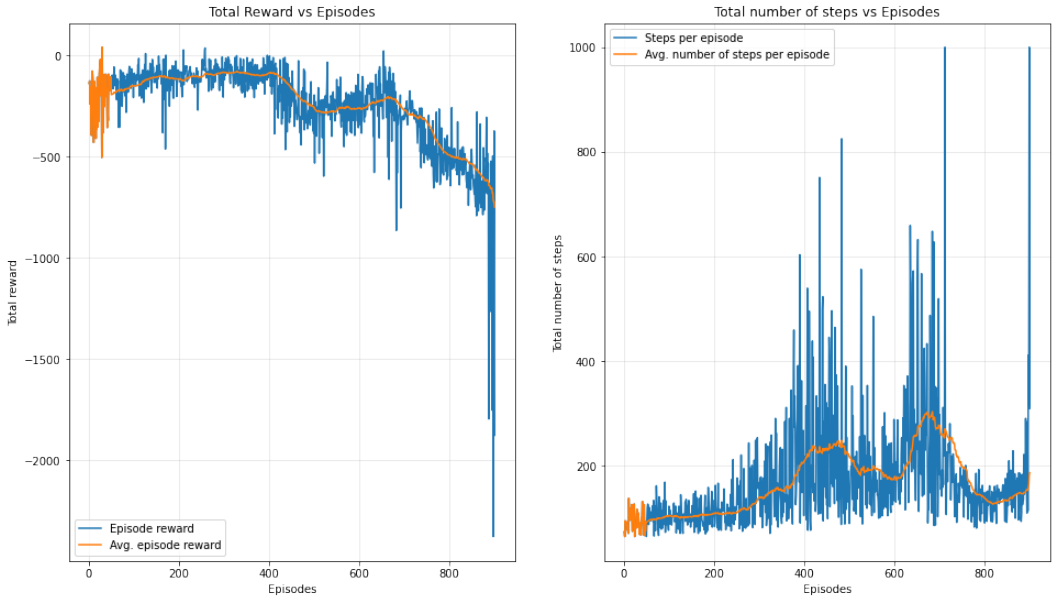

Figure 3: Total episodic reward and the total number of steps taken with $\gamma = 1$.

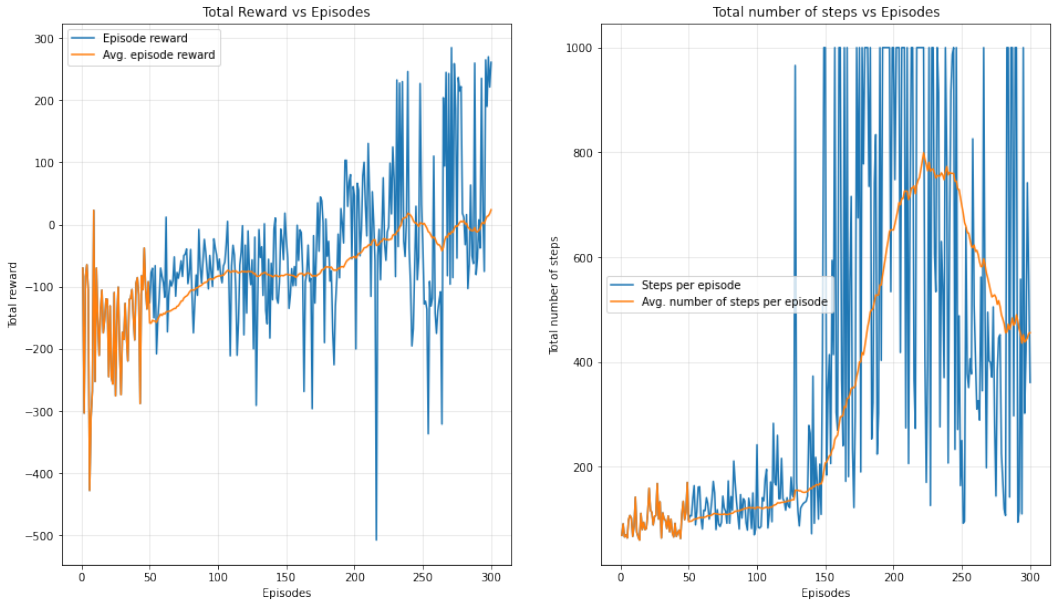

Figure 4: Total episodic reward and the total number of steps taken with 300 episodes

(3) For your initial choice of the discount factor γ_0 investigate the effect of decreasing (or increasing) the number of episodes. Also investigate the effect of reducing (or increasing) the memory size. Document your findings with representative plots.

Since we were already almost at the maximum of the advised number of episodes (and our model had already achieved good enough performance in less than our 900 episodes), we decided to only decrease the number of episodes. We tested both episode numbers 300 and 600, visible in figures 4 and 5 respectively. The 300 episodes did not converge to our threshold value of 100 rolling average reward over 50 episodes. The 600 episode run did reach that threshold before time ran out. Since a lower number of episodes does not fundamentally change anything about the model and its convergence, this is more likely to be luck than a significant change to the algorithm. Still, in general,

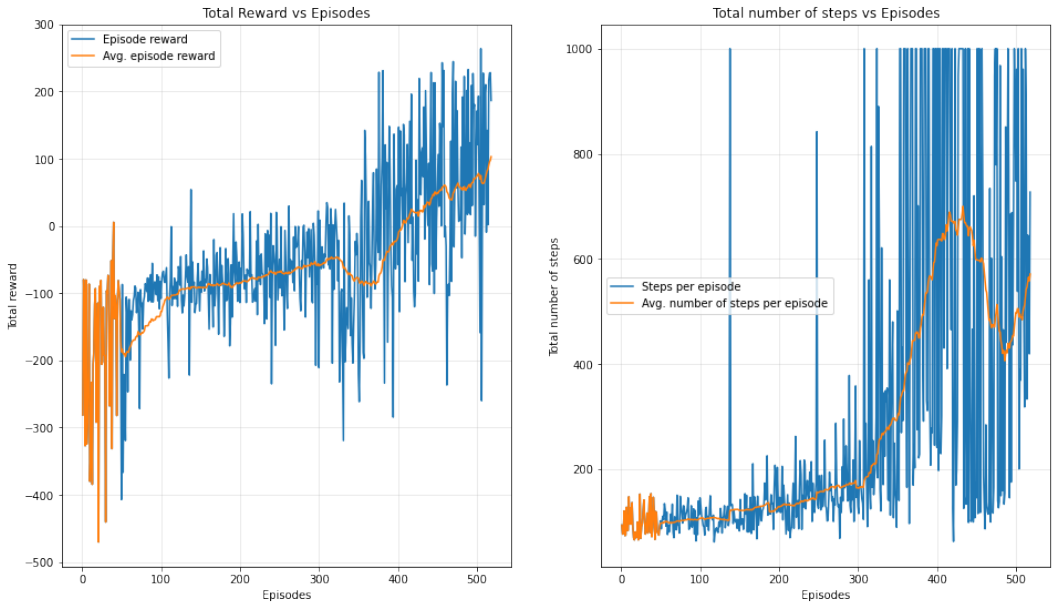

Figure 5: Total episodic reward and the total number of steps taken with 600 episodes

there is an optimum in the number of episodes. It shouldn't be too low as the model doesn't get the chance to learn enough about the environment and thus underperforms, but it also shouldn't be too high, as the model can then start forgetting imporant information and underperform due to that.

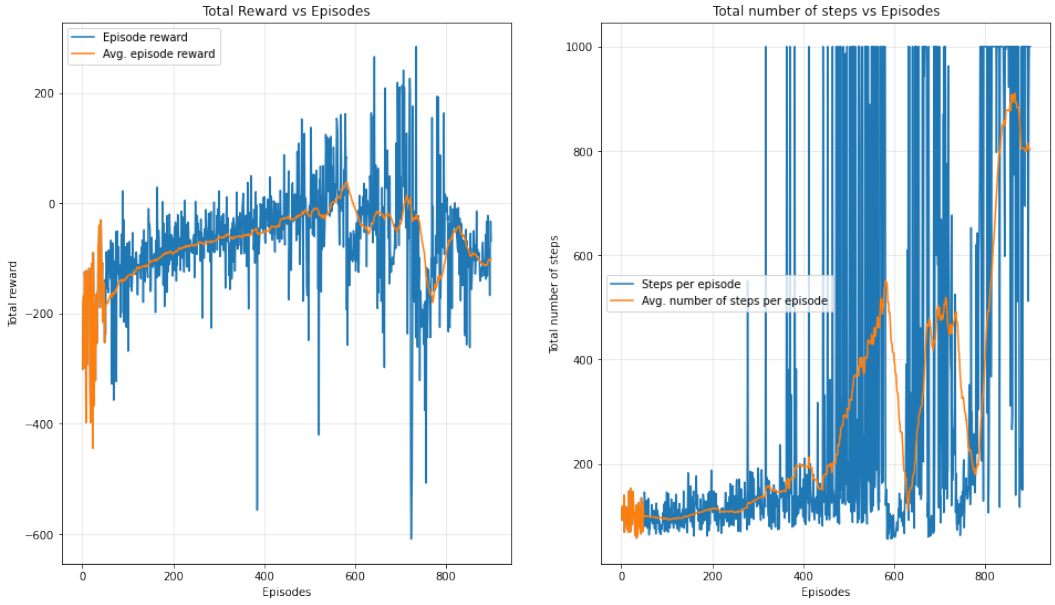

Figure 6: Total episodic reward and the total number of steps taken with buffer length 8192.

Changing the buffer size by decreasing and increasing it by 8192 (one power of 2 down from 16384) showed results we were already expecting, and can be observed in figures 6 and 7 for buffer lengths of 8192 and 24576 respectively. A smaller buffer does not allow the model to converge within the time, and is therefore very undesirable. With the larger buffer, the reward at the end of the 900 episodes was at 92.7, which is almost the threshold of 100 but not quite. This could partly be due to the unchanged update frequency and batch size, resulting in the parameters not following the $C \approx L/N$ formula

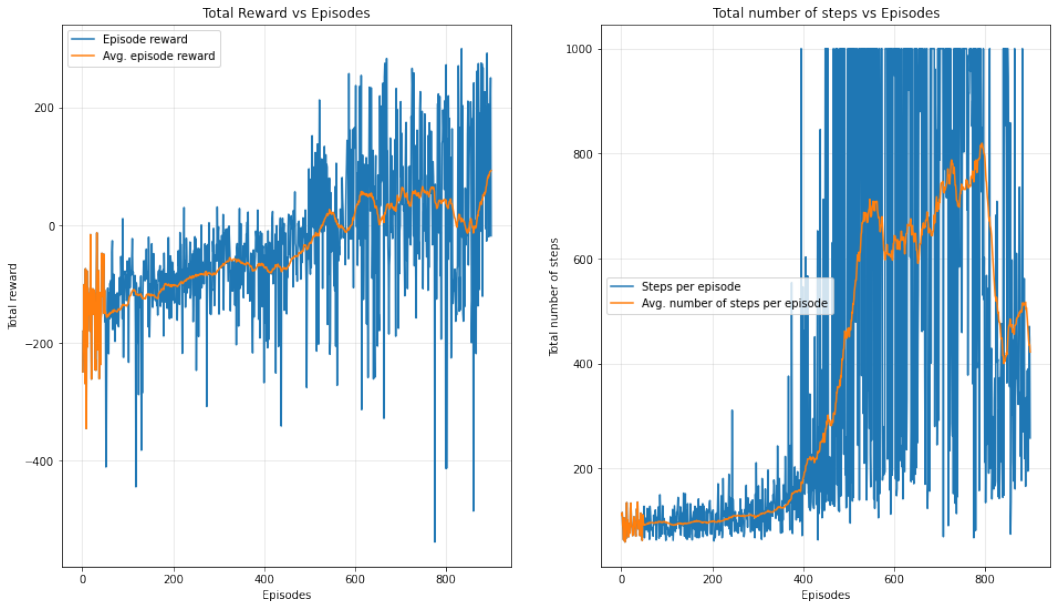

Figure 7: Total episodic reward and the total number of steps taken with buffer length 24576.

anymore, though this is not the only possible explanation. In conclusion, we are happy with the value of 16384 we chose for the buffer length.

(f)

(1) Consider the following restriction of the state $s(y, \omega) = (0, y, 0, 0, \omega, 0, 0, 0)$, where y is the height of the lander and ω is the angle of the lander. Plot $max_a Q_\theta(s(y, \omega), a)$ for varying $y \in [0, 1.5]$ and $\omega \in [-\pi, \pi]$. You should obtain a 3D plot. Does the value of the optimal policy you found make sense? Explain it.

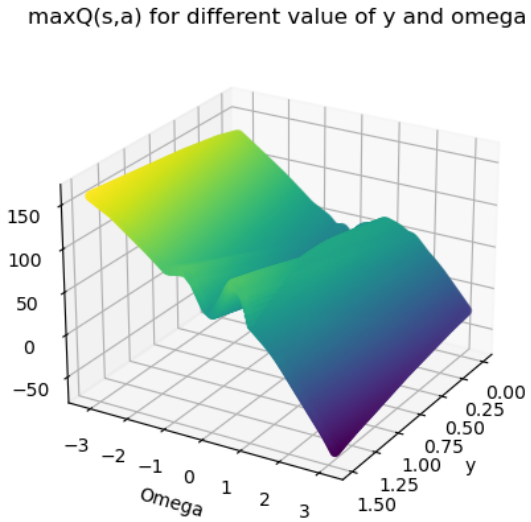

Figure 8: $max_a Q(s(y, \omega), a)$ for different y and ω.

The result of running the model with this restriction is visible in figure 8. We observe that the value for y barely seems to matter for how much reward the model anticipates in the (near) future. This can probably be attributed to the velocity being 0 in the states that we input into the model, as the lander is just hanging still in the air, though this is likely no complete explanation.

We also observe that the angle the lander is at has a big influence on the maximum Q-value, but that the impact of a negative and positive angle are approximately opposite. We have tried to come up with a sensible explanation for this behaviour, but unfortunately we cannot figure out a way in which this makes sense.

(2) Let s be the same as the previous question, and plot $argmax_a Q(s(y, \omega), a)$ for varying y and ω (as in the previous question). You should obtain a 3D plot. Does the behaviour of the optimal policy make sense? Explain it.

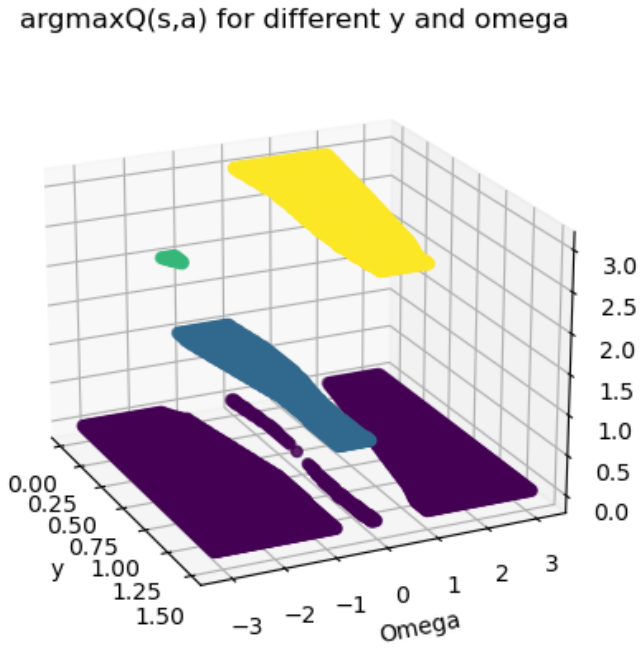

Figure 9: $argmax_a Q(s(y, \omega), a)$ for different y and ω.

In figure 9 we observe that the height y does not influence the value of $argmax_a Q(s(y, \omega), a)$ for actions 0, 1 and 3. However, the height is relevant when choosing action 2 (fire main engine). This action is mainly chosen when y is close to 0 (i.e. when the agent wants to land). This policy behaviour makes sense because in this setting, the velocity is always set to 0, and it makes sense that the main engine is only fired when the lander is going toward the ground.

We also observe that action 3 (fire right orientation engine) is taken when $\omega \in [0, \pi/2]$ and action 1 (fire left orientation engine) is taken when $\omega \in [-\pi/2, 0]$. This behaviour makes sense because the lander fires the right (or left) engine when it is slightly inclined to go back to $\omega = 0$.

Last, the action 0 (do nothing) is chosen when ω is too large or too small, meaning that the lander is considerably inclined and the best policy is to do nothing.

(g) Compare the Q-network you found with the random agent in (a). Show the total episodic reward over 50 episodes of both agents.

In figure 10 we can see the total episodic reward over 50 episodes of both agents. It is visible that the trained model obviously and significantly outperforms the random agent.

Figure 10: Q-value for DQN model vs. the random agent.