Lab 2

EL2805 *December 23, 2022*

Kaj Munhoz Arfvidsson
19980213-4032

Erik Anderberg
19960806-7857

1 Problem 1

b) An experience replay buffer is used to decrease the correlation between the experiences used when updating θ, as too strong correlation would negatively impact convergence.

We use the target as a means to measure future outcome, i.e. in some sense we want the main network to approach target. However, this is problematic since the the target is dependent on time. Therefore it's easier to slow down the update rate of the target such that the target seems fixed.

d) We found that the network should have two hidden layers with 64 neurons in size. The suggested Adam optimizer worked well. Here we set the learning rate to $5 \cdot 10^{-4}$, increasing it made the performance significantly worse. The discount factor was set to 0.99 and we had a exploration rate that linearly decreased from 0.99 to 0.05 over 50 % of the episodes. This was a result of some experimentation where we started with a linear decay over 90 % of the episodes, but noticed a lower percentage gave quicker improvements. We ran 400 episodes in total with batches of 100 experiences each. When sampling batches we used the combined experience replay modification. The replay buffer was initialized with 5 000 random walks and made to contain 20 000 experiences in total. As suggested, the target was updated every $L/N = 200$ steps. With this setup we got results shown in Fig. 1.

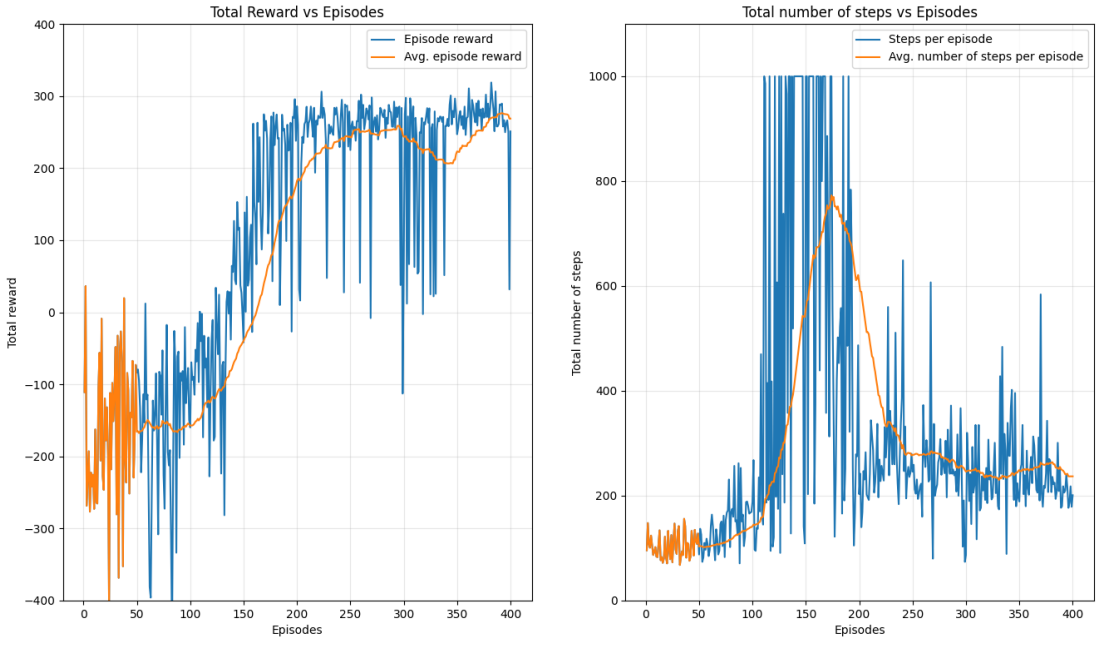
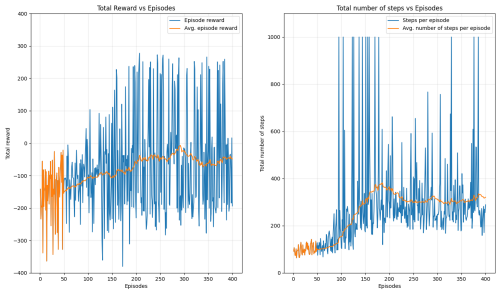


Figure 1: DQN performance

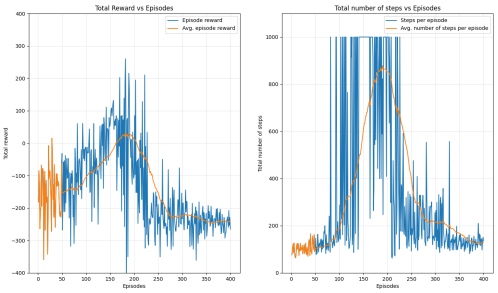
e) In the beginning the episodic rewards are mostly negative with lots of fluctuations. Over time the average episodic rewards go up (although some episodes still have negative total rewards), and the curve flattens out on a positive level after a certain number of episodes.

The number of steps per episode starts out relatively low, but increases a lot over time around when the average total episodic rewards go up (the algorithm is learning how to successfully land). Towards the end the average number of steps starts decreasing again while maintaining a high average total episodic reward, meaning it's further optimizing the landing process (landing faster).

The agent will become more greedy as we decrease the discount factor since it will consider future rewards less. This can be seen in the training where it will take longer to reach good total rewards. Since the agent is penalized for using fuel it lacks the incentive to act early and will crash more often. If, instead, the discount factor is 1 then we cannot guarantee that the neural network will converge to the target. The results of training with different discount rates are shown in Fig. 2.



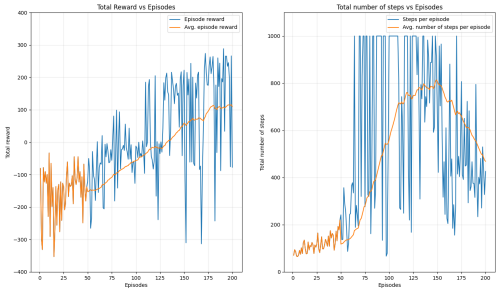
(a) Discount of 0.1



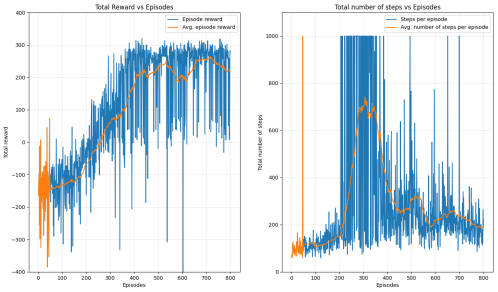
(b) Discount of 1

Figure 2: Effects of different discount rates.

When decreasing the number of episodes there is less time for the agent to learn. Likewise if we increase the number of episodes the agent has more opportunity to learn, however, at some point the learning rate will saturate. This can be seen in Fig. 3.



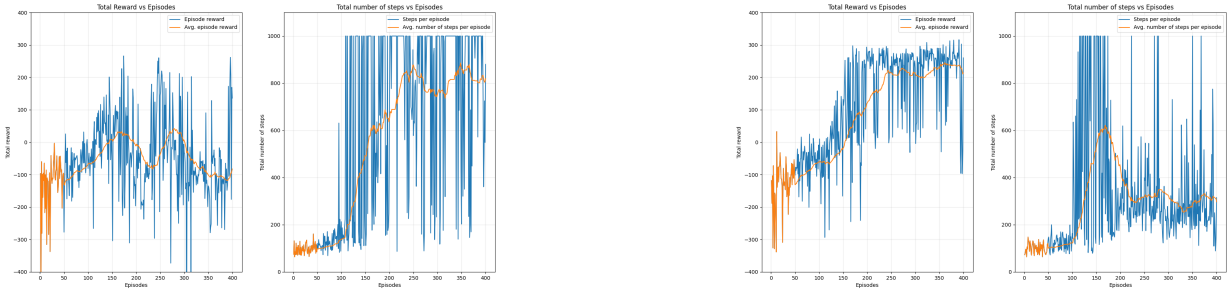
(a) Low number of episodes



(b) High number of episodes

Figure 3: Effects of different number of episodes.

When reducing the replay buffer size we have a smaller pool of experiences to learn from. The probability will be higher that the agent learns from recent experiences but there will be less variations. The same goes if the buffer is large but filled with many randomly generated experiences. In Fig. 4 we only change the buffer size, keeping the number of initial random experiences proportionally the same. We can see that when the buffer is smaller there is less experience to learn from.



(a) Buffer with 5 000 experiences. *(b) Buffer with 10 000 experiences.*

Figure 4: Effects of different buffer sizes.

f) The plots of maximum Q values and argmax for those values are shown in Fig. 5.

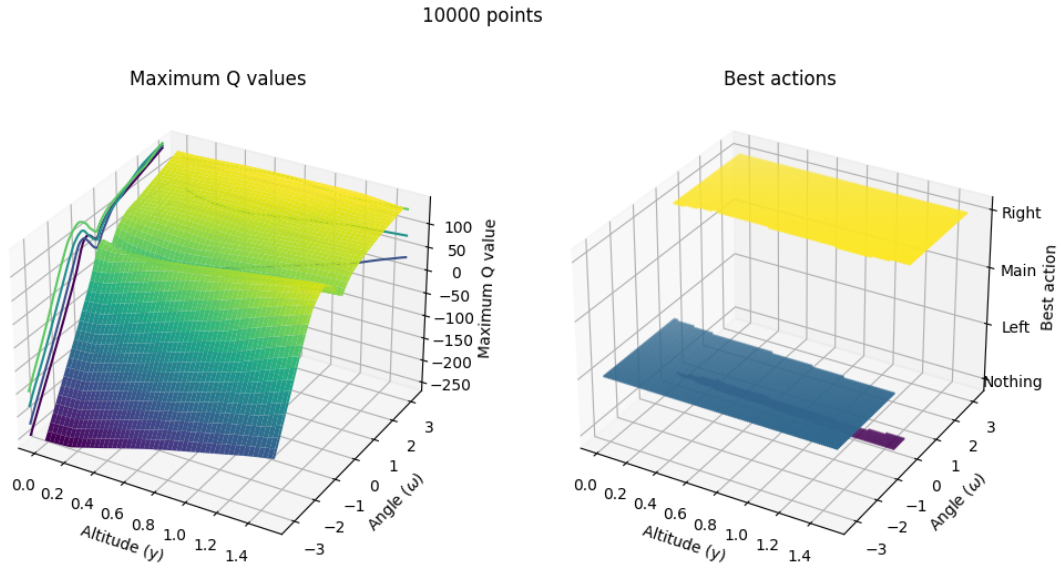


Figure 5: Max and argmax Q values

The maximum Q_θ value plot has a clear valley along $\omega = 0$, but is not as symmetric around that line as expected (intuitively, leaning to the left should be as problematic as leaning to the right when trying to land, which was the case with many other models). For y the Q_θ values decrease with y, which is also a bit unexpected since we'd expected higher values closer to ground.

The $\underset{a}{argmax}\ Q_\theta$ plot shows the best actions to take given the state (y, ω). If the lander is tilted to the left (positive ω) it makes sense to go the right to keep a steady angle. The best action is never to use the main engine, which can probably be explained by the fact the the velocity is zero in all states. Having zero velocity at a high altitude could mean that it's more efficient to fall for a while before braking, and having zero velocity close to the ground probably means there's no need to break.

g) The network we trained (as seen in Fig. 1) starts out like the random agent (focusing on exploration but learns and improves over time, eventually giving it a positive average total episodic reward (shifting

focus to exploitation). The random agent on the other hand, doesn't improve over time as it only chooses random actions (exploration without exploitation), as seen in Fig. 6.

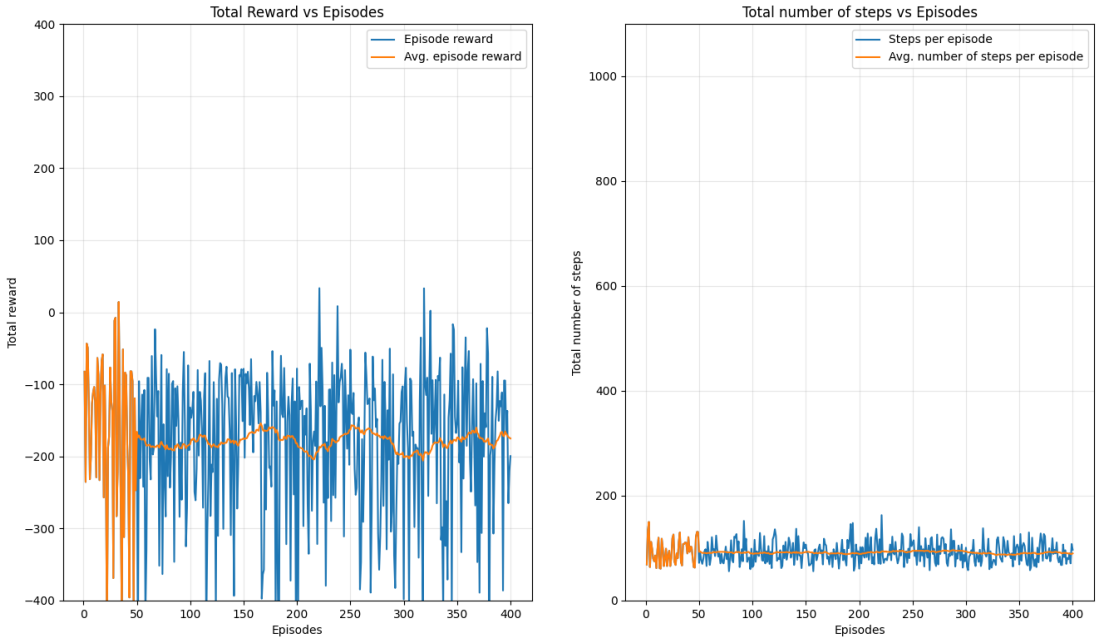


Figure 6: Random agent

h) The validity was verified using `DQN_check_solution.py`.

```
Policy achieves an average total reward of 271.6 +/- 10.1 with confidence 95%.
Your policy passed the test!
```

Sometimes when running the solution checker we noticed it can get stuck. After some investigation we found that the agent lands on a corner and tries to place itself upright, see Fig. 7. Even after waiting for a very long time the environment will still not terminate.

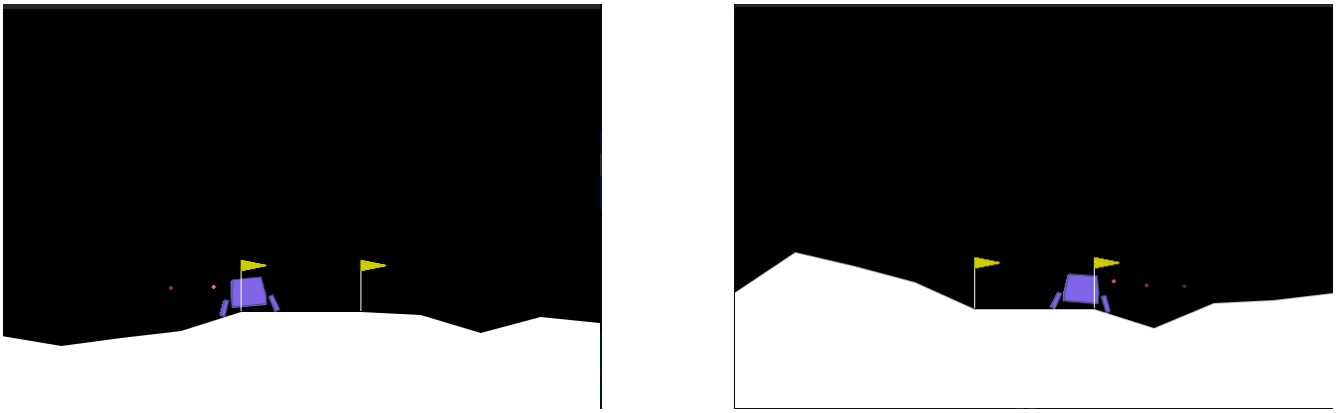


Figure 7: Agent stuck on corner.