Computer Lab 2
— Report —

Mohammed El Mendili
Personnummer: 19961029-3533
mohammed.el-mendili@polytechnique.edu

Adrian S. Wiltz
Personnummer: 19950910-7430
wiltz@kth.se

Abstract

This report answers the questions of Computer Lab 2, presents the relevant figures and is supplementary to our submitted code.

1 Problem 1 – Deep Q-Networks (DQN)

1.1 a) Familiarization with the Code

—

1.2 b) Replay Buffer and Target Network

Experiences from one single episode are strongly correlated, since they follow a trajectory, which affect the convergence of the parameters of the neural network negatively. To avoid the correlation among the experiences, a replay buffer is used, and the data used for the training of the neural network is randomly chosen from it. This results in an improved convergence.

The target network is the neural network that shall be tracked and is used as a reference for the neural network that is being trained. If however the target is computed directly from the neural network that is being trained, the target changes in every iteration and cannot be tracked so well anymore. This hampers the algorithms convergence as well. Therefore the target network is additionally used as a fixed target to resolve this problem, and is only updated after `C` steps.

1.3 c) Implementation

see submitted code (no modifications implemented)

Overview on the submitted files for Problem 1:

- `DQN_agent.py`: completed code file for the agents
- `DQN_check_solution.py`: for checking the agents (code line `from DQN_agent import Agent` needed to be added)
- `problem_1.py`: Code for the computation of the neural network that solves the task
- `neural-network-1.pth`: neural network that solves the task
- `some_plot_p1f1.py`: code for creating the plots for problem 1, f)
- `some_plot_p1g.py`: code for creating the plots for problem 1, g)

Preprint. Under review.

1.4 d) Setup for Computation of Neural Network

Our computed network that solves the task has one hidden layer. One layer showed to be sufficient when we used a sufficiently large amount of neurons (124 neurons) to ensure a high learning capacity of the network. The optimizer is the stochastic gradient descent algorithm `torch.optim.Adam` that was recommended with learning rate 0.00006 in order to avoid a too fast adaption to the provided experience and overfitting, and the clipping value 1.846938775510204, which was determined by a random search on the interval $[0.5, 2]$. The other selected parameters are:

- Discount factor γ: 1.0
- Buffer size L: 29025
- Number of episodes T_E: 620
- Target network frequency update C: `int(L//N)`
- Training batch size N: 38
- exploration rate ε: linear with $\varepsilon_{\text{max}} = 0.9$ and $\varepsilon_{\text{min}} = 0.05$, `Z = int(0.93 * N_episodes)`

Since every episode was limited to 1000 time steps and thereby the overall reward bounded, we chose the discount factor $\gamma = 1.0$ such that all rewards are weighted in the same way and the entire reward is being optimized. In order to avoid that the experiences used for training are strongly correlated, we chose a large replay buffer ($L = 29025$), and a smaller training batch size. Due to the small learning rate of the optimizer and the smaller batch size, we used many episodes for training ($T_E = 620$). The update frequency was chosen according to the recommendation (for the motivation of using an update frequency $C > 1$, see b)). With regard to the exploration rate, ε_{max} was chosen as 0.9 since the replay buffer was already initialized with random data, and $\varepsilon_{\text{min}} = 0.05 > 0$ to avoid overfitting in the end of the training and to ensure an ongoing learning. After having identified suitable intervals for each parameter, the particular parameters have been determined by using a random search.

1.5 e) Analysis

1.5.1 e) (1) Training process

Figure 1 depicts the total episodic reward and the total number of steps taken per episode of the training of out neural network that solves the task. It can be seen, that the number of steps per episodes increases significantly until it reaches the maximum of 1000 steps. For the respective episodes, the average total episodic reward is around zero, which indicates that the lunar lander mostly neither crashes nor manages to land. This is due to $\gamma = 1$. The agent seems to learn to collect rewards in by hovering above the surface instead of risking to crash (from about episode 350 till 530). It takes some more training, until it learns to successfully land (from episode 530 on), and the steps per episode are again significantly decreasing. It seems, that for large discount factors close to one, many episodes are required for training. However, the average reward was nearly monotonously increasing. Besides the no "forgetting" could be observed at the end of the training.

1.5.2 e) (2) Impact of the discount factor on the training process

By comparing Figure 1 with Figures 2-5, it can be seen that for a decreasing discount factor, the number of steps per episode decreases. For $\gamma = 0.01$, the maximal number of episodes is only seldomly reached compared to Figure 1. On the other, for a smaller discount factor (at least in combination with the other chosen parameters), the neural network does not anymore succeed in learning a good policy, although in the case regarded in Figure 3, the agent obtains several high rewards (> 200) in a row (roughly between episode 450 and 550).

1.5.3 e) (3) Impact of number of episodes and memory size on the training process

Consider at first the number of episodes. From Figures 6 (400 episodes) and 7 (100 episodes), it can be seen that for too few episodes, the amount of training data is too small and the network is not able to learn a good policy. In Figure 6, it can be still seen how the agent learns not to crash and to hover above the ground.

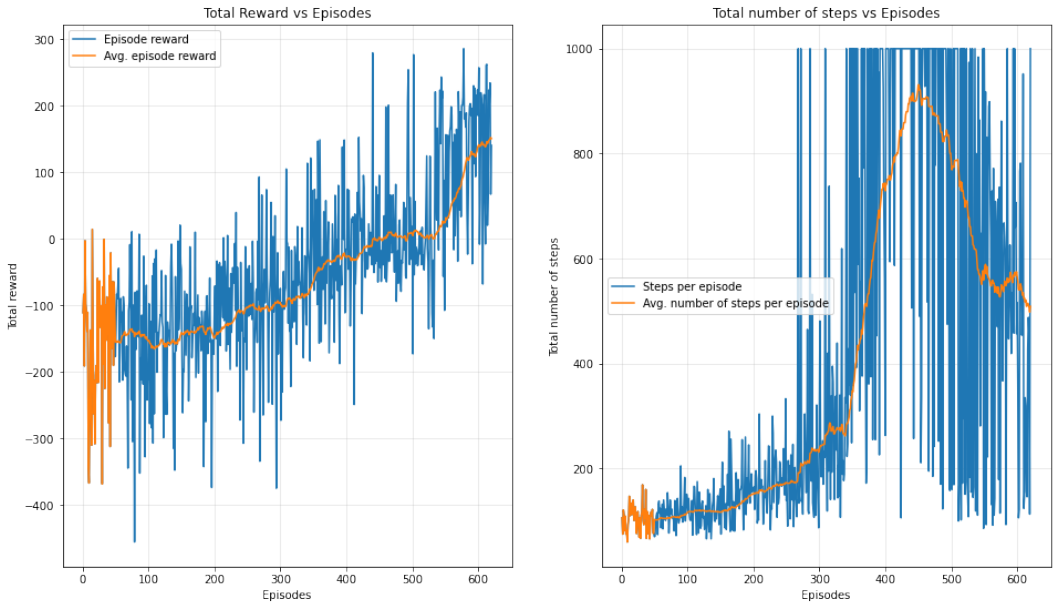

Figure 1: Total episodic reward and total number of steps taken per episode during training. The maximum average reward (average over past 50 episodes) that was reached is about 161.

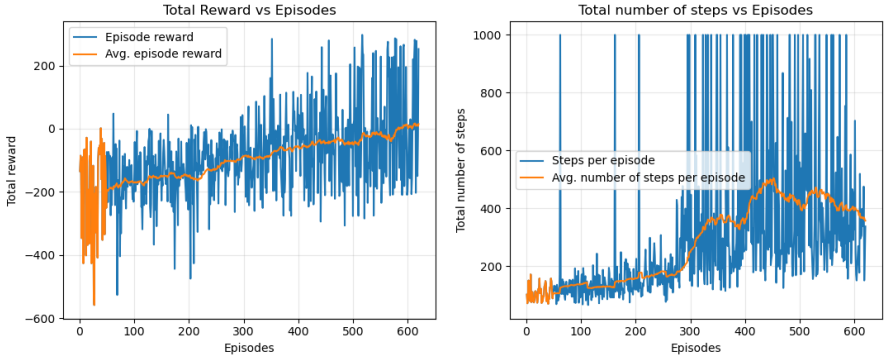

Figure 2: Total episodic reward and total number of steps taken per episode during training. Discount factor $\gamma = 0.8$, other parameters are the same as in Figure 1.

For $T_E = 1000$, the interpretation is not so clear since for an increased number of episodes the agent explores for more episodes due to the choice of Z, which is dependent on the number of episodes T_E. In general, the plots look similar to Figure 1 and the agent learns to successfully land in the end of the training (or at least starts to do so). The number of the steps per episode stays throughout the second half of the training (from episode 500) mostly at the maximal number of steps (1000), and only decreases in the end, but does not reach such low values as in Figure 1.

We proceed now to the impact of the memory size on the training, i.e., the number of neurons in the hidden layer. From Figure 9 it can be seen that the agent is hardly able to achieve a better average reward in the end compared to the random policy from the beginning. This is due to the small memory size and the complexity of the policy due to the many possible trajectories.

The findings for large memory sizes is not so clear, see Figures 10 (300 neurons) and 11 (1000 neurons). However, it seems that a large memory sizes hampers the learning and the convergence to a good trajectory. None of the networks has learnt a good policy, although the network with 1000 neurons achieved around episode 500 a positive total episodic average reward. However, in both cases the number of steps per episode remained close to the maximum of 1000 steps.

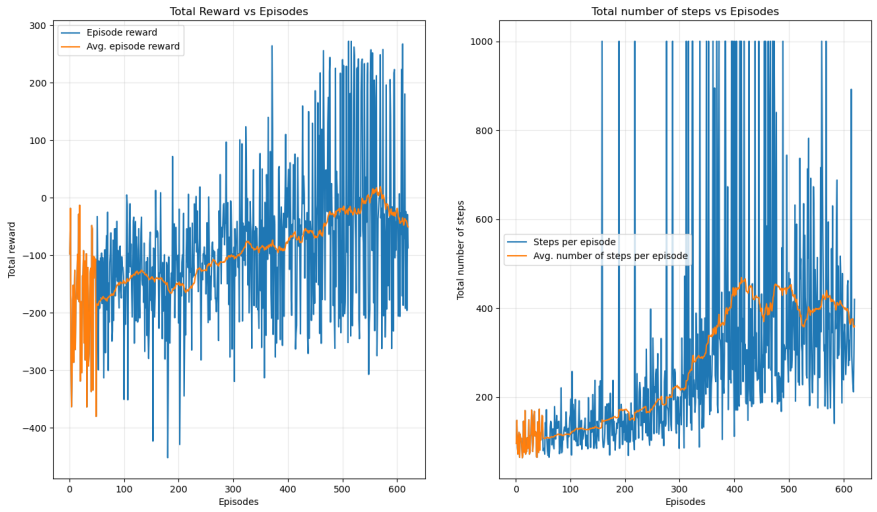

Figure 3: Total episodic reward and total number of steps taken per episode during training. Discount factor $\gamma = 0.5$, other parameters are the same as in Figure 1.

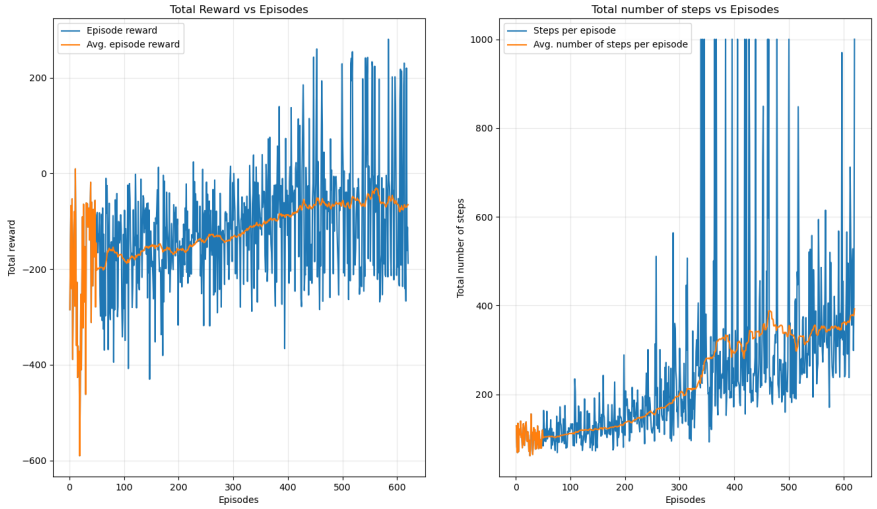

Figure 4: Total episodic reward and total number of steps taken per episode during training. Discount factor $\gamma = 0.2$, other parameters are the same as in Figure 1.

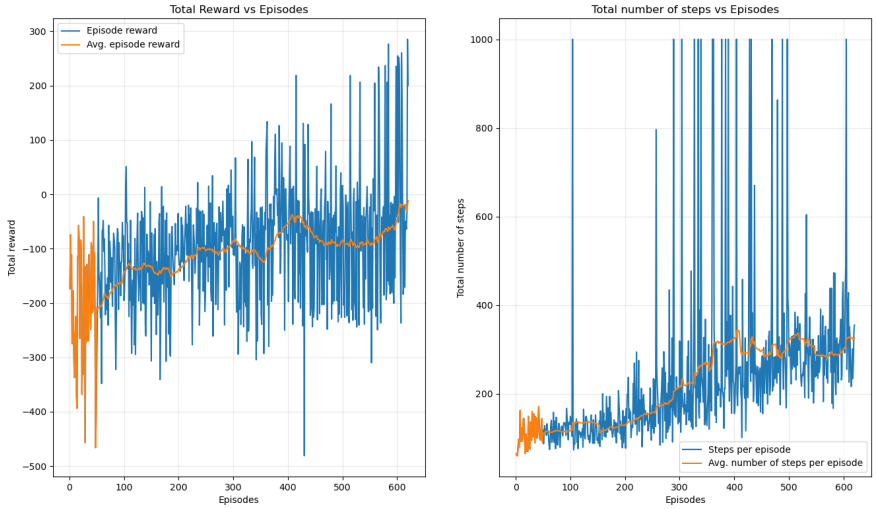

Figure 5: Total episodic reward and total number of steps taken per episode during training. Discount factor $\gamma = 0.01$, other parameters are the same as in Figure 1.

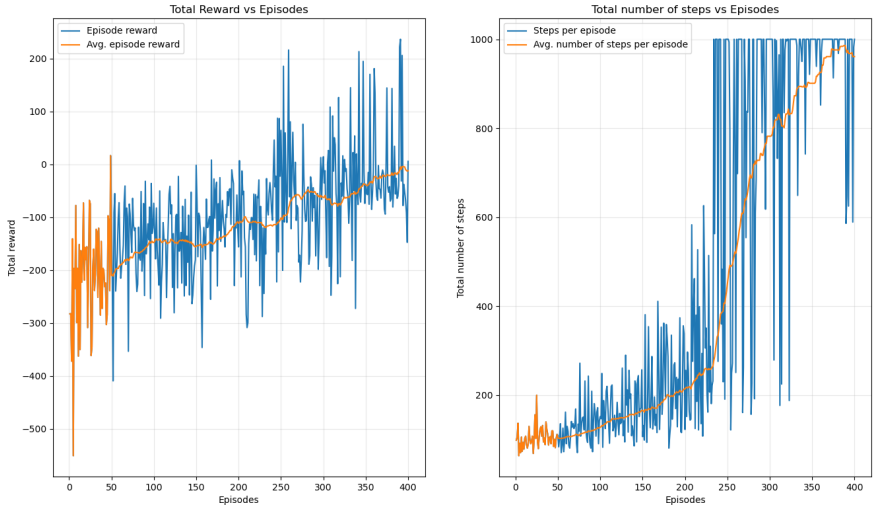

Figure 6: Total episodic reward and total number of steps taken per episode during training. Number of episodes $T_E = 400$, other parameters are the same as in Figure 1.

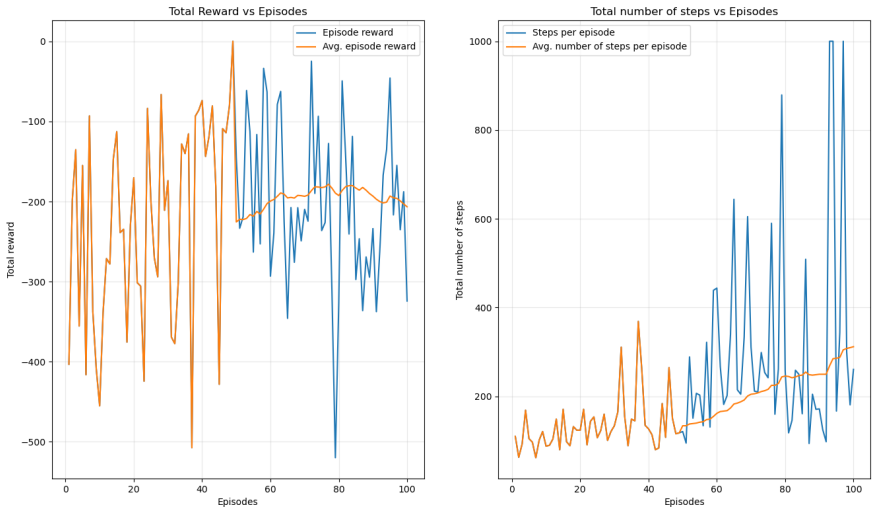

Figure 7: Total episodic reward and total number of steps taken per episode during training. Number of episodes $T_E = 100$, other parameters are the same as in Figure 1.

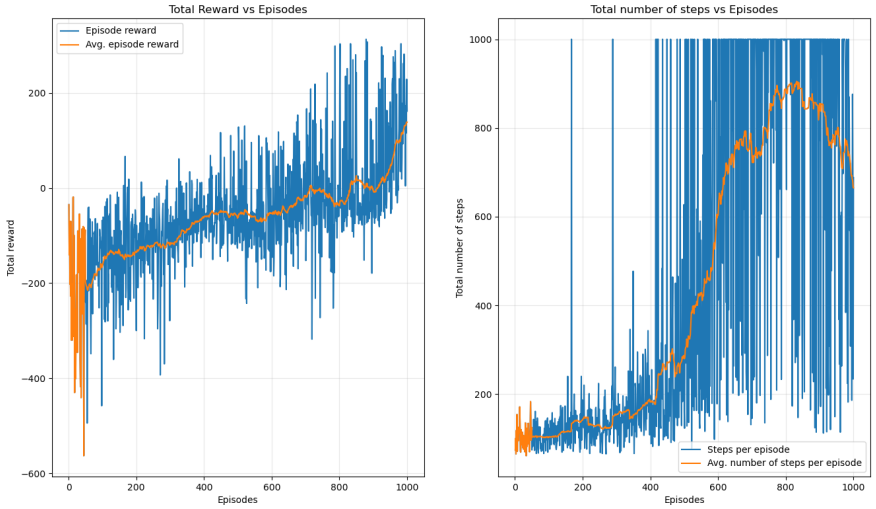

Figure 8: Total episodic reward and total number of steps taken per episode during training. Number of episodes $T_E = 1000$, other parameters are the same as in Figure 1.

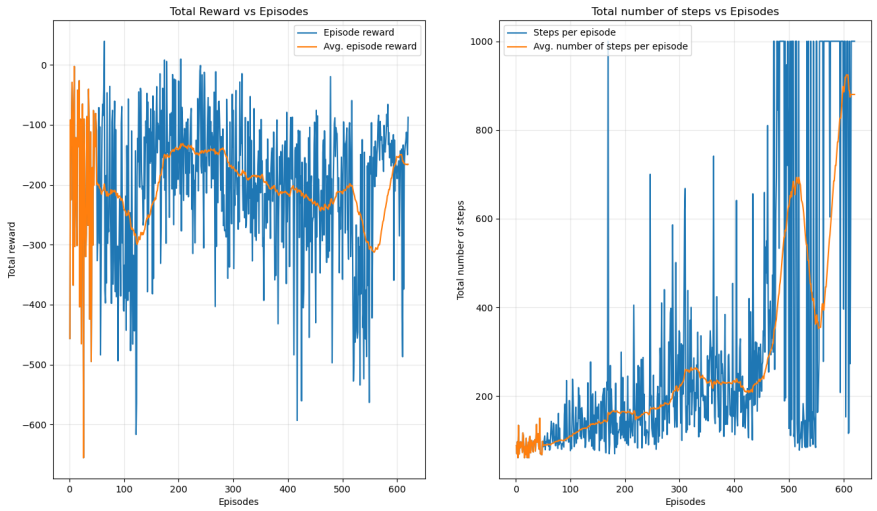

Figure 9: Total episodic reward and total number of steps taken per episode during training. 8 neurons in the hidden layer, other parameters are the same as in Figure 1.

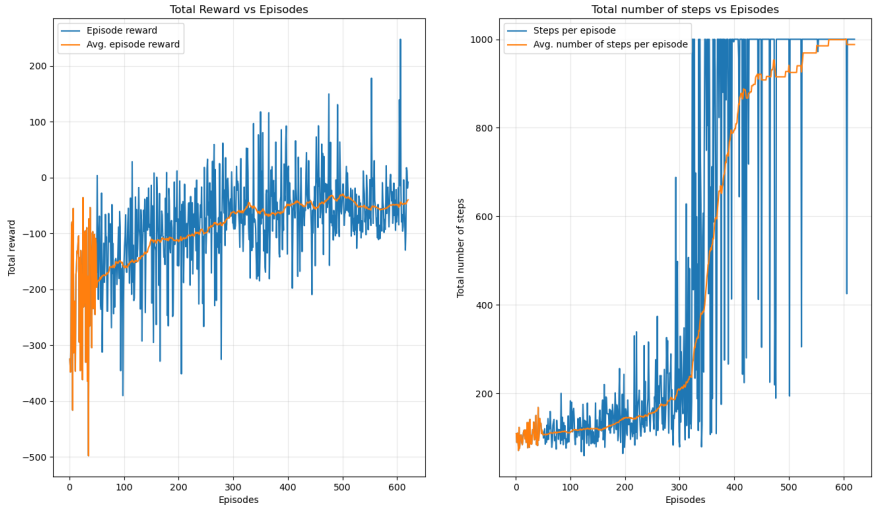

Figure 10: Total episodic reward and total number of steps taken per episode during training. 300 neurons in the hidden layer, other parameters are the same as in Figure 1.

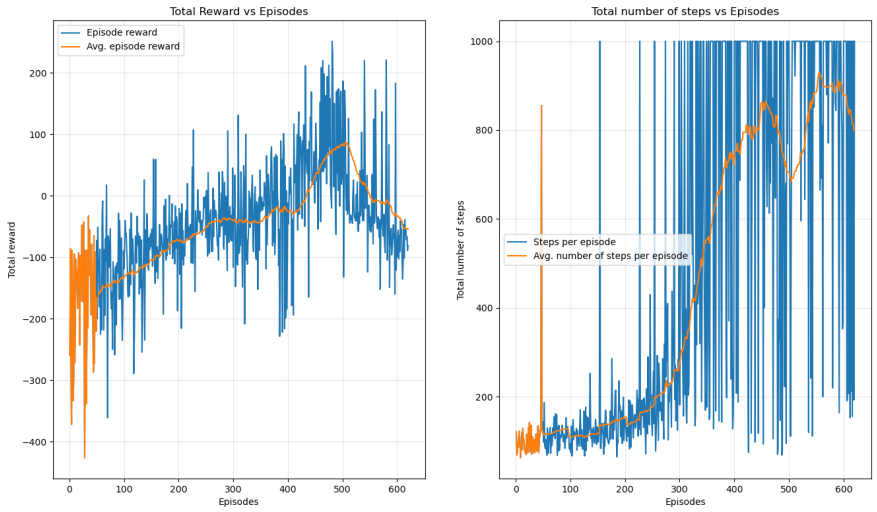

Figure 11: Total episodic reward and total number of steps taken per episode during training. 1000 neurons in the hidden layer, other parameters are the same as in Figure 1.

1.6 f) Investigation of the obtained Q-network

1.6.1 f) (1) Analysis of optimal Q-values

Figure 12 depicts the Q-values for the optimal action, i.e., $\max_a Q_\theta(s(y,\omega),a)$ over the states $s(y,\omega) = (0,y,0,0,\omega,0,0,0)$. Remember, that $V^{\pi^*}(s) = \max_a Q_\theta(s = s(y,\omega),a) = \mathbb{E}\left[\sum_{t\geq 0}^{1000} \gamma^t r(s_t,a_t) \middle| \pi^*, s_0 = s\right]$. It makes sense that the plane is descending along the y axis towards zero, since the higher the agent is above the ground the more time it has to collect rewards on its way until it lands. Besides, it is clear, that if $\omega = \pm\pi$, i.e., it is oriented upside down, the Q-value is the smallest compared to all other values for states with the same y-value, since it is difficult for the agent to turn upright again and to successfully land. Additionally, it must use its side engines which induces negative reward. Regarding the Q-values for a fixed y-value, it is clear, that the highest values can be found around $\omega = 0$ since then the agent has the correct attitude for landing. Only in the surroundings of $\omega = 0$ even higher values can be found. This is because in these cases, the lander touches the ground at first with only one leg without immediately completing the task by touching the ground with both.

1.6.2 f) (2) Analysis of the optimal policy

Figure 13 depicts the Q-values for the optimal action, i.e., $\text{argmax}_a Q_\theta(s(y,\omega),a)$ over the states $s(y,\omega) = (0,y,0,0,\omega,0,0,0)$. It can be seen that for $\omega \neq 0$, the respective engine is activated to bring the lander to an attitude close to $\omega = 0$, which coincides with the considerations regarding Figure 12. Close above the ground, the action 0, i.e., no action is chosen. This is the case, because for the regraded states the vertical and horizontal velocities are zero ($\dot{x} = \dot{y} = 0$) as well as $x = 0$. Hence, the lander does not need to activate any engines in order to accelerate towards the ground. Besides, a slight deviation from $\omega = 0$ is appreciated as pointed out in the previous section.

1.7 g) Comparison of computed policy with random policy

Figure 14 depicts the total episodic reward and its average over 50 episodes for the agent using our computed policy. The same is depicted in Figure 15 for the agent using a random policy. It can be clearly seen, that the training of our neural network resulted in a network that (mostly) solves the task to land the lunar lander and therefore yields a significantly higher positive reward. It can be even stated, that the best episode with respect to the totally achieved episodic reward of the random agent

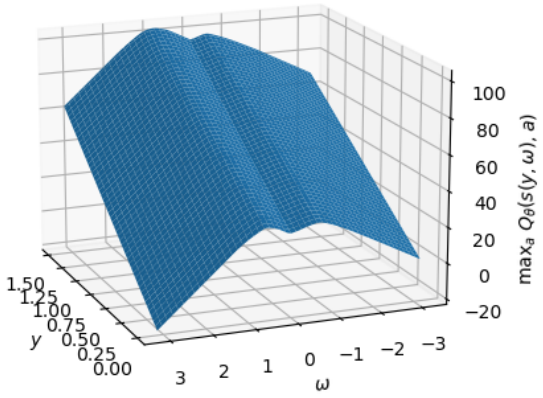

Figure 12: Q-values $\max_a Q_\theta(s(y,\omega),a)$ over the states $s(y,\omega) = (0,y,0,0,\omega,0,0,0)$

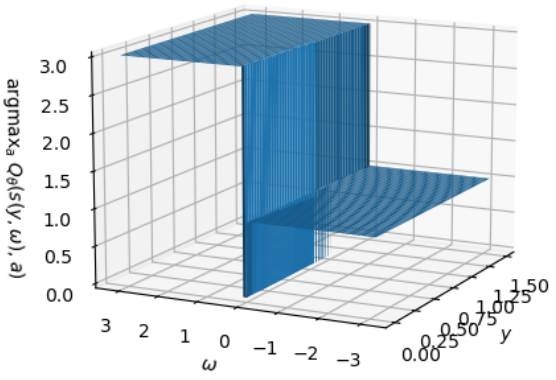

Figure 13: Policy $\text{argmax}_a Q_\theta(s(y,\omega),a)$ over the states $s(y,\omega) = (0,y,0,0,\omega,0,0,0)$

Figure 14: Total episodic reward of agent using our computed policy over 50 episodes.

Figure 15: Total episodic reward of random agent over 50 episodes.

is worse than the worst episode of the agent using the learnt policy. However, the episodes of the agent using the learnt policy require more steps than those of the random agent (cf. Figures 1 and 16).

1.8 h) Storing final Q-network

Our final network is `neural-network-1.pth` and can be found among the submitted files in folder `Problem 1`.

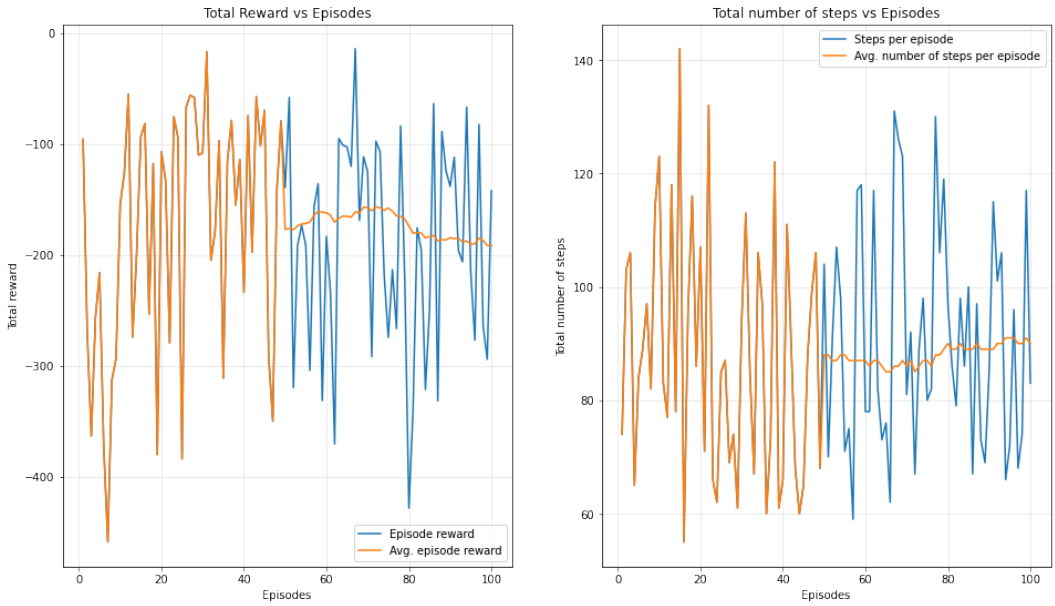

Figure 16: Total episodic reward and total number of steps taken per episode of the random agent.

2 Problem 2 – Deep Deterministic Policy Gradient (DDPG)

- **(b-1)** Using critic's target network instead of critic's network to update the actor network could result in a severe slowing of the training because the target is updated in a **soft** way.
- **(b-2)** DDPG is off-policy, like DQN. That's because we use experience replay to train the networks, which is as if we used a different policy than the one we have currently to get the training samples.
  Yes, sample complexity could be an issue for off-policy methods. In fact, off-policy methods converge to the true optimal Q-fct when they visit all states-actions almost surely and sample complexity could have an impact on this.
- **(d) layout** We simply used the suggested layout in the lab which results in better performances. We note however that this problem recquires indeed a big value for L as decreasing it resulted a big decrease in the performance.
- **(d) Learning rate** It seems better to have big learning rate for the critic as the actor is most important in this problem (our strategy is encoded there). Small learning rates results in general in a more calibrated learning, which is indeed more crucial for actor than for critic.
- **(e-1)** In figure 18 we plot the evolution of training for our selected parameters: $T_E = 300, \gamma = 0.99, \text{lr}_{\textbf{actor}} = 10^{-5}, \text{lr}_{\textbf{critic}} = 10^{-4}, L = 30000, \tau = 10^{-3}, N = 64, \mu = 0.15, \sigma = 0.2, d = 2$ and Figure 17 illustrates the architectures that we used for the Actor-critic networks. We achieve a **total reward of 255.5 +/- 7.2 with confidence 95% over 50 episodes**, which solves the problem.
- **(e-2)** Figure 20 illustrates the training evolution when we only set $\lambda = 0.2$ (other parameters are as before) and Figure 20 illustrates the training process when $\lambda = 1$. Although our selected $\lambda = 0.99$ is close to 1, setting $\lambda = 1$ worsen and destabilizes the training especially in the last episodes. Moreover, setting $\lambda = 0.2$ is completely not appropriate as the algorithms seems to not learning anything (its performance is worse than the random agent).
- **(e-3)** We also investigated the effect of changing L. Figure 21 illustrates the training for $L = 15000$, the performance was worsened and the training became more unstable (more fluctuations). This is due to the fact that this environment is quite rich and a value of 15000 is not enough to learn it correctly. On the other hand, a very big value of L doesn't work neither if the number of episodes if fixed. In fact, Figure 22 illustrates the training for $L = 50000$. The perfrmance start to increase a bit at the very last episodes which suggests that it probably

```
Network model: actor_net(
  (input_layer): Linear(in_features=8, out_features=400, bias=True)
  (input_layer_activation): ReLU()
  (input_layer1): Linear(in_features=400, out_features=200, bias=True)
  (output_layer): Linear(in_features=200, out_features=2, bias=True)
  (output_layer_activation): Tanh()
)

Network model: critic_net(
  (input_layer): Linear(in_features=8, out_features=400, bias=True)
  (input_layer_activation): ReLU()
  (input_layer1): Linear(in_features=402, out_features=200, bias=True)
  (output_layer): Linear(in_features=200, out_features=1, bias=True)
)
```

Figure 17: Used Actor-Critic Neural Networks architectures

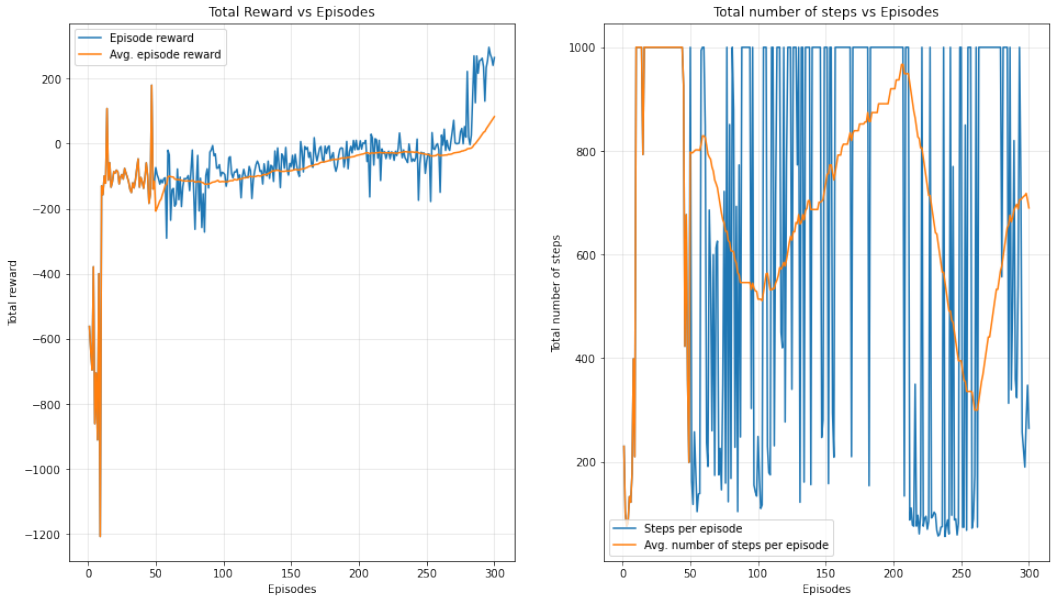

Figure 18: Training plots for our selected parameters. $T_E = 300, \gamma = 0.99, \text{lr}_{\textbf{actor}} = 10^{-5}, \text{lr}_{\textbf{critic}} = 10^{-4}, L = 30000, \tau = 10^{-3}, N = 64, \mu = 0.15, \sigma = 0.2, d = 2$

needs more episodes to converge (to fill the Buffer with new relevant experiences other than the random one that we used before beginning the training).

- **3D-plots** As in problem 1, we plot for the same restriction on the state space, in Figure 23, we plot $Q_\theta(s(y,\omega), \pi_\theta(s(y,\omega)))$ function of y, ω. In figure 24, we plot $\pi_\theta(s(y,\omega))$ function of y, ω.

  As in problem 1, the Q-values (Figure 23) are decreasing along the y-axis for decreasing y-values. Surprisingly, the Q values are not again symmetric around $\omega = 0$. For $\omega < 0$, it is however plausible that the surface has a "kink". The maximum values for a fixed y can be found for $\omega \leq 0$ always for ω-values slightly smaller than zero, whose absolute values are increasing for increasing y-values. As already explained in problem 1, this is due to the fact that then the agent lands first with one leg, which means an extra reward without immediately completing the task. However, for too large deviations from $\omega = 0$, it is expensive to turn the lander upright again, such that the Q-values are decreasing for

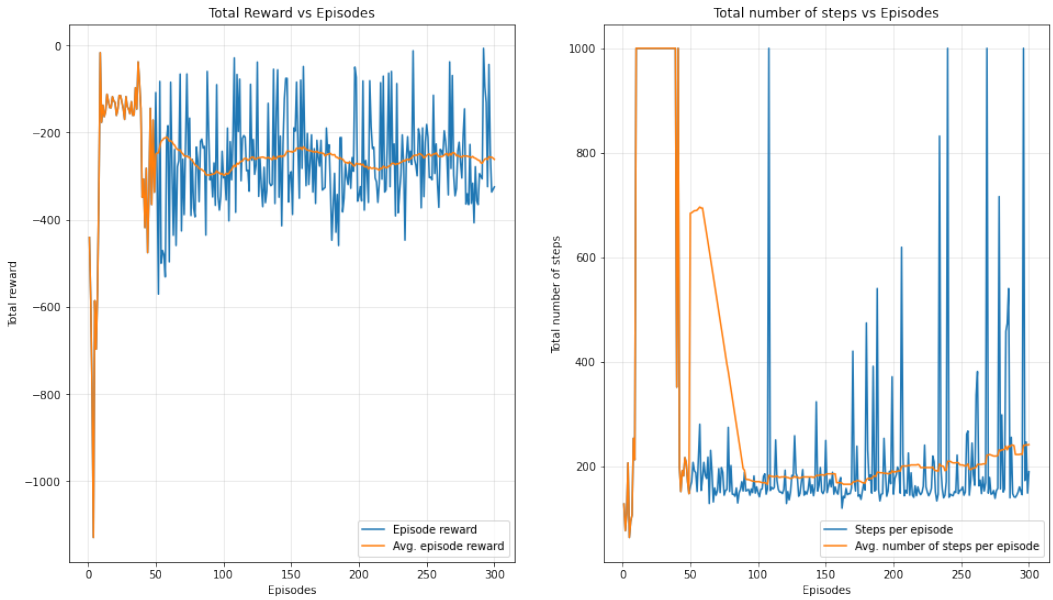

Figure 19: Training plots for $\lambda = 0.2$, all other parameters are the same as before

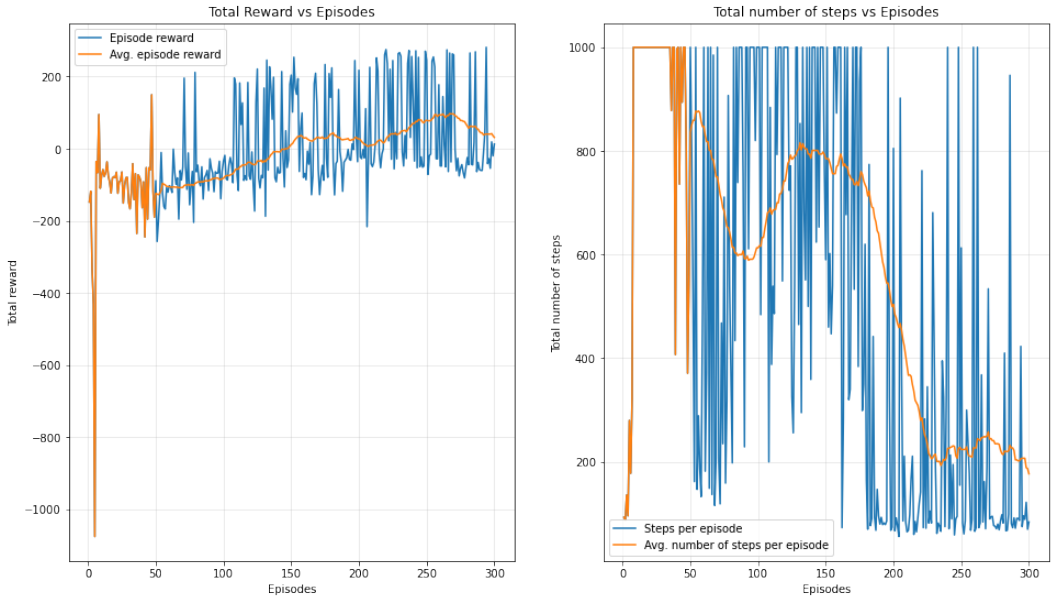

Figure 20: Training plots for $\lambda = 1$, all other parameters are the same as before

$\omega \ll 0$. Yet, we have no explanation why this characteristic of the Q-values cannot be observed for $\omega > 0$. Still, also for $\omega > 0$, the "kink" can be seen.

Regarding now Figure 24, it can be seen that for $\omega < 0$ the optimal policy is $\pi_\theta = -1$, and for $\omega > 0$ the optimal policy is $\pi_\theta = 1$ which means that the left and right engines are used in such a way that the lander assumes an attitude close to $\omega = 0$. Since the vertical velocity is zero, i.e., $\dot{y} = 0$, it also makes sense that the main engine is not being used for the regarded states.

- **(g) Comparison with Random Agent**: Our policy achieves more than 250 reward at the end of the training while the random agent is stuck at -250 rewards as illustrated in Figure 25

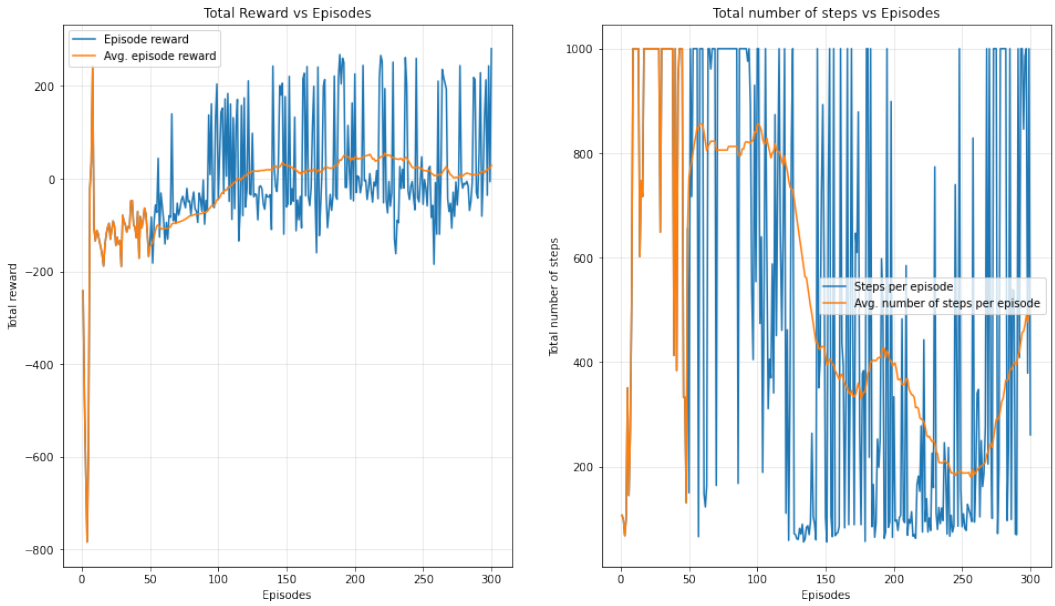

Figure 21: Training plots for $L = 15000$, all other parameters are the same as before

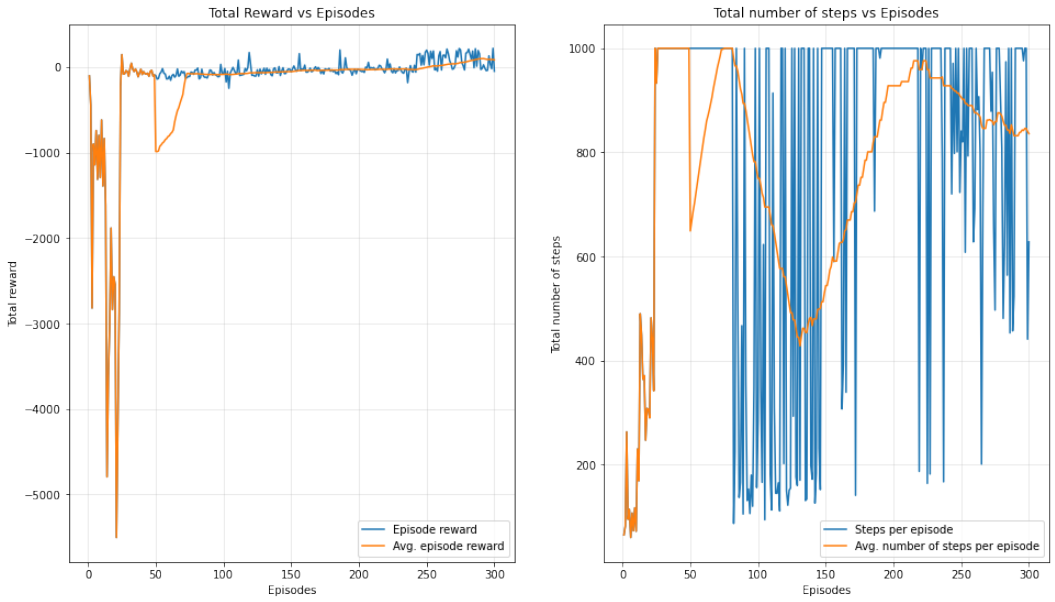

Figure 22: Training plots for $L = 50000$, all other parameters are the same as before

3 Problem 3 – Proximal Policy Optimization (PPO)

- **(b-1)** The idea of PPO is to stabilize the training by clipping the ratio of new policy and old policy. In PPO, we use Monte-Carl estimates of the reward-to-go as target values, and hence we no longer need a target network.
- **(b-2)** 3. PPO is on-policy since it trains while collecting samples from the same policy. Does not uses an experience replay samples.
  Yes, sample complexity could be an issue for on-policy methods. In fact, on-policy methods converge to the true optimal Q-fct when they visit all states-actions almost surely and sample complexity could have an impact on this.

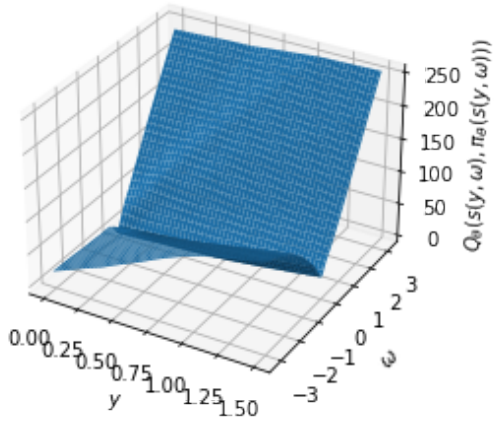

Figure 23: Plot of $Q_\theta(s(y,\omega),\pi_\theta(s(y,\omega)))$ when $s(y,\omega) = (0,y,0,0,\omega,0,0,0)$

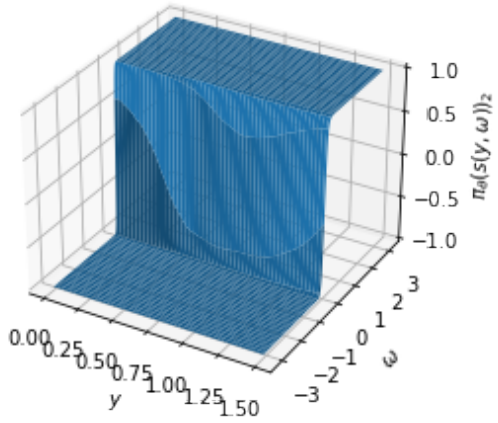

Figure 24: Plot of $\pi_\theta(s(y,\omega))$ when $s(y,\omega) = (0,y,0,0,\omega,0,0,0)$

- **(d) layout** We simply used the suggested layout in the lab which results in better performances. We note that, even though not suggested, we applied norm clipping of the networks' gradients before performing back propagation (2-norm limited at 1), as we found that it considerably accelerates the training (score 200 with VS score of 100 without for the same number of episodes).
- **(d) Updates of the actor** We don't think updating the actor less frequently could benefit the training as the clipping used in PPO (i.e. actor updated but not so much) replaces this idea.
- **(e-1)** In figure 27 we plot the evolution of training for our selected parameters: $T_E = 1600, \gamma = 0.99, \text{lr}_{\textbf{actor}} = 10^{-5}, \text{lr}_{\textbf{critic}} = 10^{-3}, M = 10, \epsilon = 0.2$ and Figure 26 illustrates the architectures that we used for the Actor-critic networks. We achieve a **total reward of 200 +/- 35.2 with confidence 95% over 50 episodes**, which solves the problem.
- **(e-2)** Figure 28 illustrates the training evolution when we only set $\lambda = 0.2$ (other parameters are as before) and Figure 29 illustrates the training process when $\lambda = 1$. Although our selected $\lambda = 0.99$ is close to 1, setting $\lambda = 1$ worsened and destabilized the training. Moreover, setting $\lambda = 0.2$ is completely not appropriate as the algorithms seems to not learning anything (its performance is similar to the random agent).
- **(e-3)** We also investigated the effect of changing ϵ. Figure 30 illustrates the training for $\epsilon = 0.001$, where the training became clearly very slow (though not unstable). In fact,

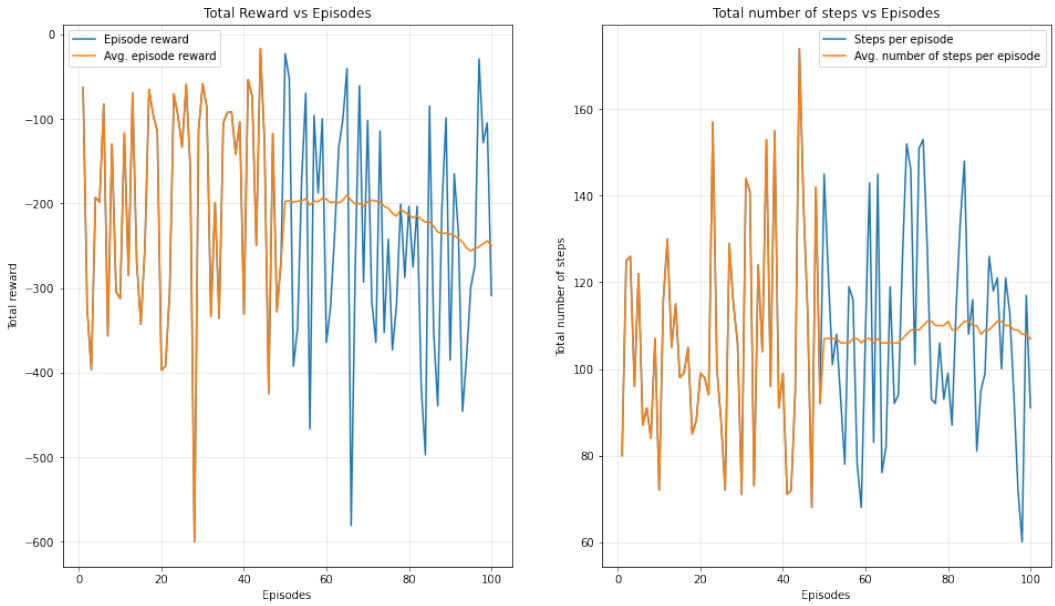

Figure 25: Random Agent performance

```
critic_net(
  (input_layer): Linear(in_features=8, out_features=400, bias=True)
  (input_layer_activation): ReLU()
  (input_layer1): Linear(in_features=400, out_features=200, bias=True)
  (output_layer): Linear(in_features=200, out_features=1, bias=True)
)
actor_net(
  (input_layer): Linear(in_features=8, out_features=400, bias=True)
  (input_layer_activation): ReLU()
  (input_mean): Linear(in_features=400, out_features=200, bias=True)
  (output_mean): Linear(in_features=200, out_features=2, bias=True)
  (act_mean): Tanh()
  (input_sigma): Linear(in_features=400, out_features=200, bias=True)
  (output_sigma): Linear(in_features=200, out_features=2, bias=True)
  (act_sigma): Sigmoid()
)
```

Figure 26: Used Actor-Critic Neural Networks architectures

reducing ϵ means that the clipping used in PPO becomes more severe and hence the actor policy updates becomes very small which slows down the training process. On the other hand, a very big value of ϵ doesn't work neither if the number of episodes if fixed. In fact, Figure 31 illustrates the training for $\epsilon = 0.99$. The performance is still very good for this choice of $epsilon$. We achieved an average total reward of 161.5 +/- 32.5 with confidence 95%, which solves the problem. We noticed, however, that the performance decreased a bit dureing last episodes, which may siggest that for biggest values of $epsilon$ the number of episodes should decrease to have better performance (i.e. this speeds up the training). This is compatible with our observation for low values for ϵ.

- **3D-plots** As in problem 1, we plot for the same restriction on the state space, in Figure 32, we plot $V_\omega(s(y,\omega))$ function of y, ω. In figure 33, we plot $\pi_\theta(s(y,\omega))$ function of y, ω. The same comments as for problem 2 could be made. However, we can notice that ft he policy

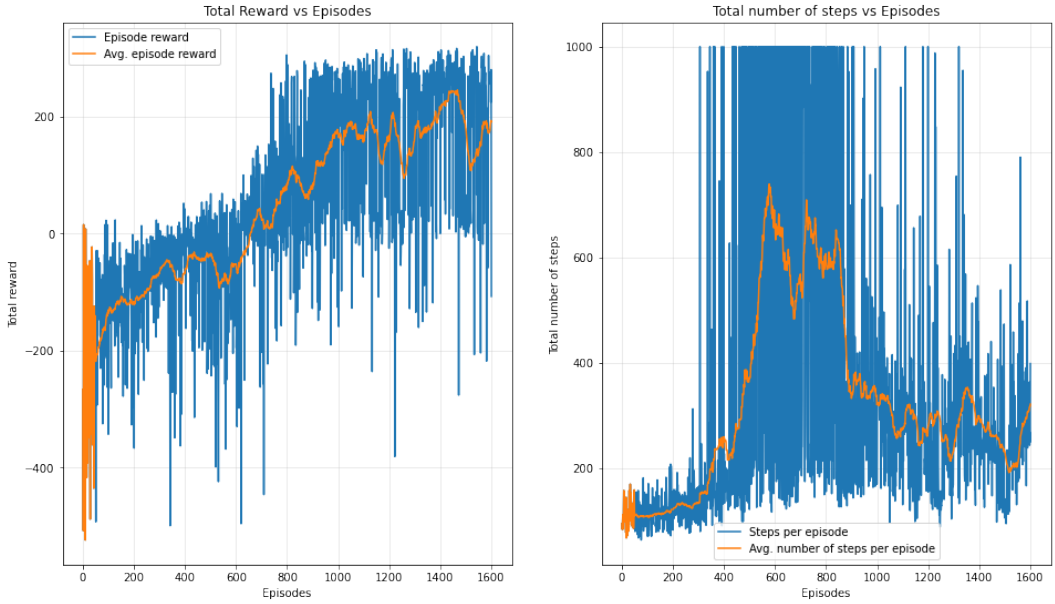

Figure 27: Training plots for our selected parameters. $T_E = 1600, \gamma = 0.99, \text{lr}_{\textbf{actor}} = 10^{-5}, \text{lr}_{\textbf{critic}} = 10^{-3}, M = 10, \epsilon = 0.2$

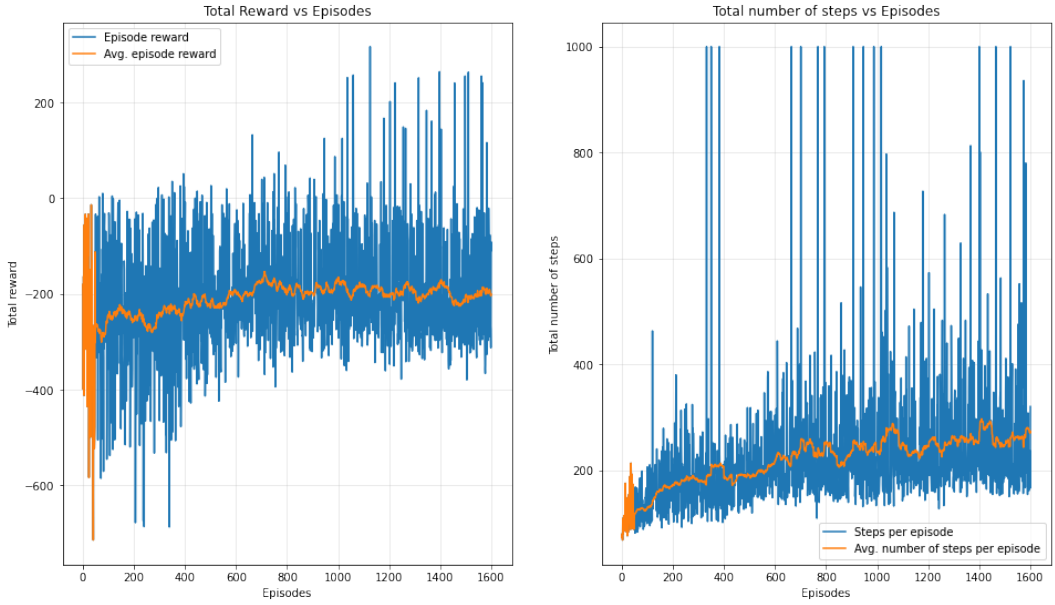

Figure 28: PPO Training plots for $\lambda = 0.2$, all other parameters are the same as before

the plot is not very smooth. This is mainly due to the fact that we are using a stochastic policy with parameters (means and standard deviation) given by our actor network.

- **(g) Comparison with Random Agent**: Our policy achieves more than 200 reward at the end of the training while the random agent is stuck at -200 rewards as illustrated in Figure 34

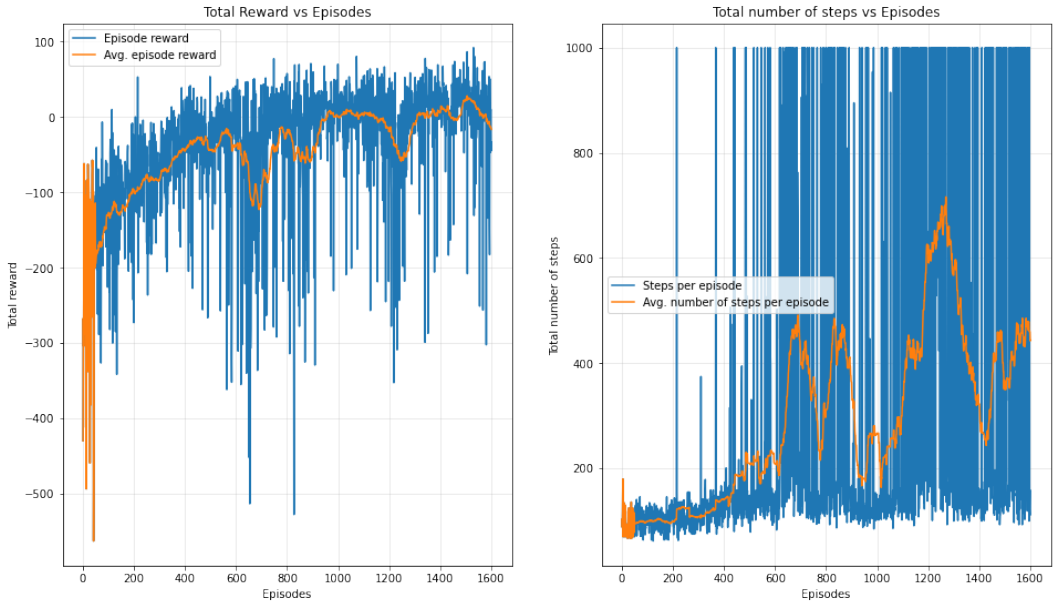

Figure 29: PPO Training plots for $\lambda = 1$, all other parameters are the same as before

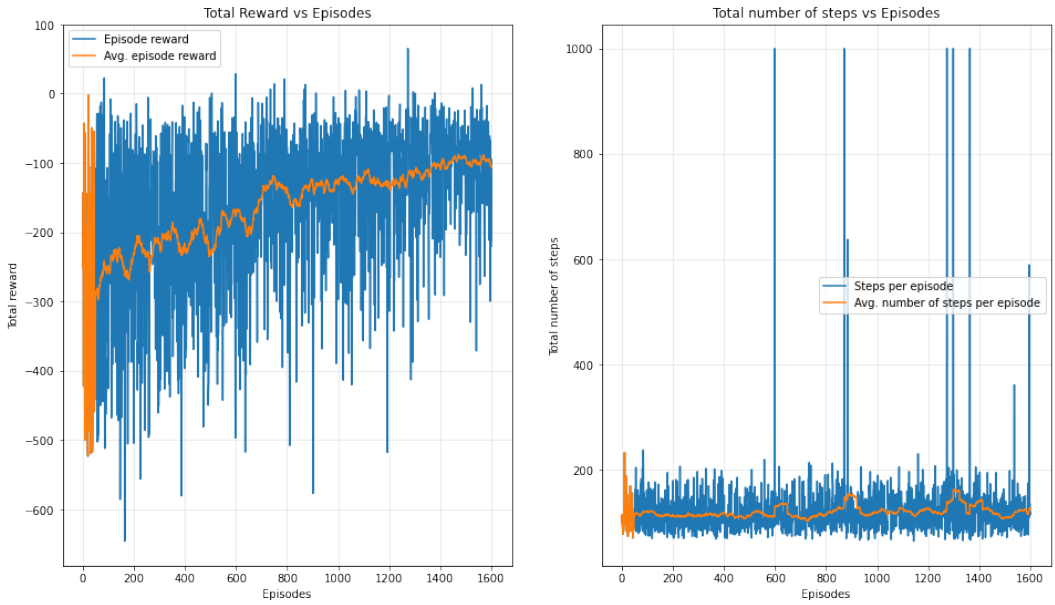

Figure 30: PPO Training plots for $\epsilon = 0.001$, all other parameters are the same as before

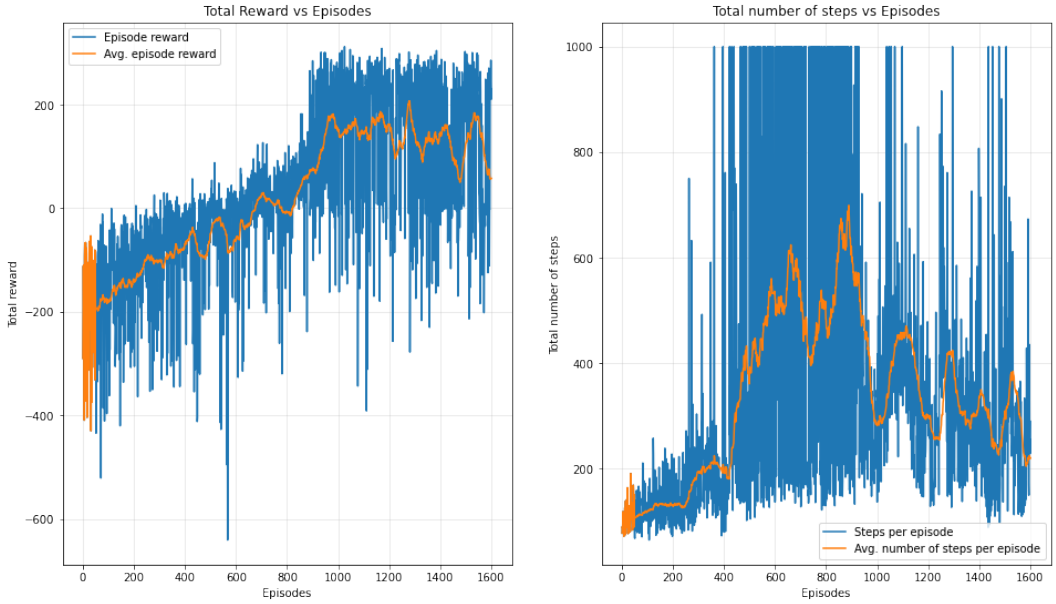

Figure 31: PPO Training plots for $\epsilon = 0.99$, all other parameters are the same as before

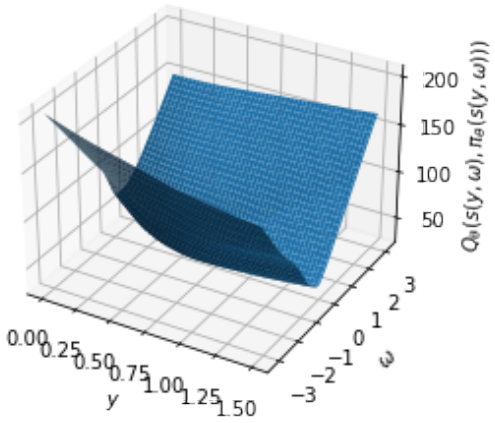

Figure 32: PPO Plot of $V_\omega(s(y,\omega)))$ when $s(y,\omega) = (0, y, 0, 0, \omega, 0, 0, 0)$

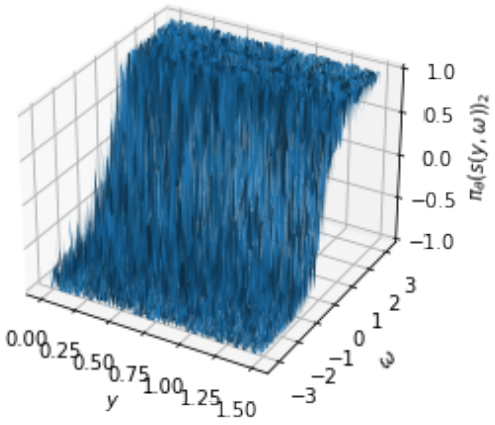

Figure 33: PPO Plot of $\pi_\theta(s(y,\omega))$ when $s(y,\omega) = (0, y, 0, 0, \omega, 0, 0, 0)$

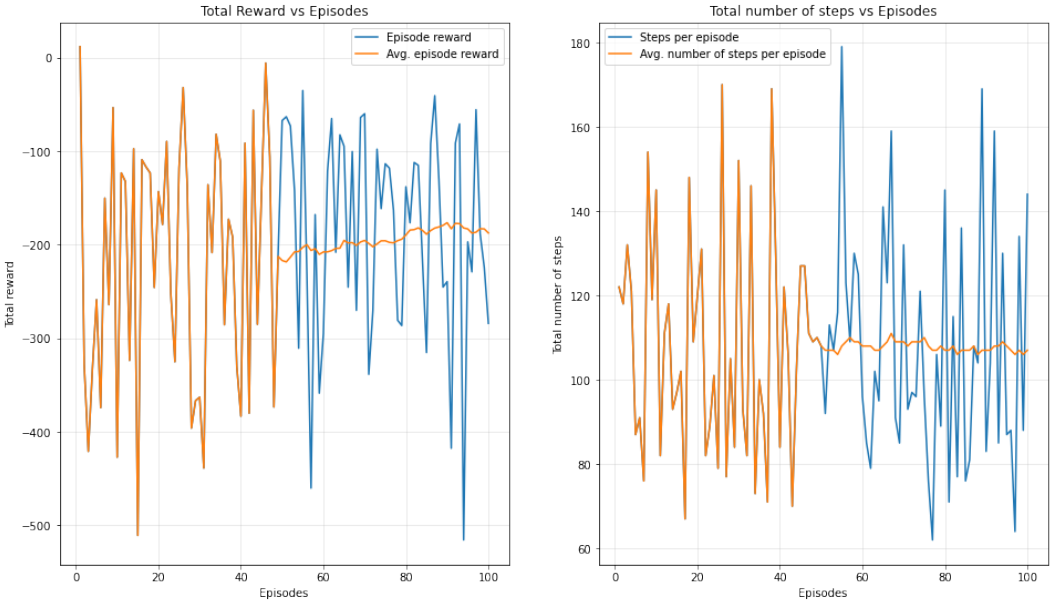

Figure 34: Random Agent performance