Reinforcement Learning - Assignment 1

Gustavo Teodoro D. Beck
gtdb@kth.se
940218-0195

Fernando García Sanz
fegs@kth.se
970718-0312

The lunar lander

Figure 1: Lunar lander in OpenGym.

In this lab we solved a classical problem in optimal control theory: the lunar lander. The environment is implemented in the OpenGym library [1] and the goal is to train different networks to make a successful landing on the moon (the landing pad is always at coordinates $(0, 0)$, figure 1). In each problem, a different approach will be implemented and discussed. Worth noticing that the episode finishes if the lander crashes or comes to rest, receiving an additional -100 or +100 points. If the lander moves away from the landing pad it loses reward and firing the main engine is -0.3 points each frame.

The state of the problem s is an 8-dimensional variable: s_1 and s_2 are respectively position in the x axis and y axis; s_3 and s_4 are the x, y axis velocity terms; s_5, s_6 are the lander angle and angular velocity; s_7 and s_8 are the left and right contact points (boolean values that indicate if the spaceship touched land). The action space, instead, can be discrete or continuous, and for each one of the scenarios we have to apply different methods:

1. Discrete action space: Four discrete actions are available: do nothing (0), fire left orientation engine (1), fire main engine (2), fire right orientation engine (3).
2. Continuous action space: In this case the action a is a 2-dimensional variable, whose values are between -1 and 1. a_1 controls the main engine: from -1 to 0 is off, from 0 to 1 throttles from 50% to 100% of the power (engine can't work with less than 50% power). a_2 instead is used to control direction: from -1 to -0.5 it fires the left engine, while from -0.5 to 0,5 is disabled, and from 0,5 to 1 it fires the right engine.

1 Deep Q-Networks (DQN)

1.1 Problem 1 - Tasks

1.1.1 Why do we use a replay buffer and target network in DQN?

One requirement of stochastic gradient descent training is that the data have to be independent and identically distributed. In order to prevent data correlation, instead of using the latest experiences, a

[1] `https://gym.openai.com/envs/LunarLander-v2/`

large buffer of past experiences can be used in order to sample the training data from it. This allows updating the network, and consequently minimizing the effects of data correlation. It is convenient that before starting the training process, the buffer is filled with random experiences, in order to count on enough different data for the first stages of the procedure. Every new episode of the learning process, the buffer is updated based on the new experience generated by the updated network, and the oldest experiences are dropped from the buffer.

With regards to the target network, it is a copy of the training network and it is periodically synchronized with the values of the training network that is updated constantly in every step of an episode. Therefore, the target network is never updated through back-propagation. The idea behind this technique is based on the following: as $Q(s, a)$ is computed via $Q(s', a')$, two states with only a step of difference in between, which makes the network difficult to distinguish between them. The update of the parameters that calculated $Q(s, a)$ could indirectly affect to $Q(s', a')$, making the training process quite unstable. Keeping a copy of the original network and using it to calculate $Q(s', a')$ improves the stability of the process. Since now these two different Q functions are computed by two different networks, the results will be more stable, because training the main network doesn't affect the results produced by the target network, until this one is synchronized.

1.1.2 Explain the layout of the network

(a) The choice of the optimizer.

In order to perform this task, Adam (Adaptive Moment Estimation) has been the chosen optimizer. Adam optimizer is an extension of the well-know stochastic gradient descent. It has several beneficial features, such as:

- It is computationally efficient.
- Appropriate for large problems in terms of amount of data or number of parameters.
- Well suited when gradients are either noisy or sparse.

A crucial difference between the classical SGD and Adam is in terms of the learning rate. While SGD keeps a single learning rate for all weight updates, Adam keeps a learning rate for each network weight.

When compared to other optimizers, Adam is capable of reaching better results way faster.

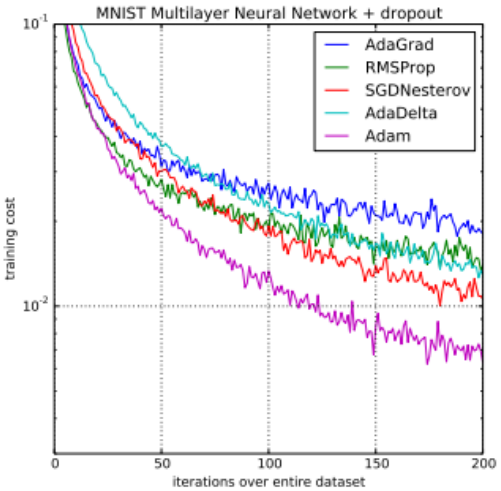

Figure 2: Comparison of different algorithms training a multilayer perceptron.

As proved by the previously mentioned facts, we consider that Adam is the best optimizer to use in this case, due to the amount of data recorded in the problem, and most importantly, due to its performance when compared to other well-known optimizer algorithms.

(b) The parameters that you used (γ, L, T_E, C, N, ϵ) and which modification did you implement (if any). Motivate why you made those choices.

- γ: The value chosen for gamma has been 0.99. As the episodes will have an elevated number of states, we have considered that using a big discount factor is beneficial in order to take into account future states as much as possible.
- L: The size of the employed replay buffer has been 15,000. We have considered that this size is more than enough to store a huge number of different experiences, ensuring minimal correlation when sampling.
- T_E: The total number of episodes employed for training has been equal to 500. After performing a grid search on this parameter, we found out that the network converges around its peak performance when an approximate number of 500 episodes has elapsed.
- C: The value of C has been approximated by $\lfloor \frac{L}{N} \rfloor$, being this equal to 234.
- N: The batch size employed has been equal to 64. As the replay buffer size was set to 15,000, we considered that using also an intermediate value withing the interval proposed would be the best choice.
- ϵ: The value of epsilon has been computed every episode, being bounded in between 0.99 and 0.05. An exponential decay has been employed via the following formula:

$$\epsilon_k = \max\left(\epsilon_{min}, \epsilon_{max} \times \left(\frac{\epsilon_{min}}{\epsilon_{max}}\right)^{\frac{k-1}{T_E \times Z - 1}}\right) \tag{1}$$

  Where $Z = 0.925$ and k is the episode index {1, 2, 3, ...}.

In addition to these parameters, the learning rate employed α has been set to 10^{-4}. In our initial experiments, $\alpha = 10{-}3$ was used, but it proved to be rather unstable. Therefore, by reducing its value by one magnitude order, we experienced a more consistent learning process, without sudden performance drops.

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

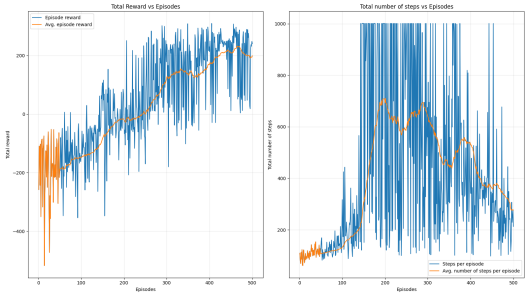

Figure 3: DQN episodic reward and total number of steps.

The above figure shows the reward at each episode and the total number of steps each episode took. It can be seen that at the beginning, the results were rather bad, taking just a few steps in each episode and returning quite low rewards. Nevertheless, once the number of steps taken began to increase, the rewards did it too. Then, the network trained during many episodes around the maximum number of steps available (1000). This maximizes the exploration process during training, producing better learning. Finally, there is a point in which the network is able to perform more optimally, needing fewer steps in order to maintain or even keep increasing the rewards. At the final stage of the training, the network has reduced a lot the number of steps per episode while obtaining the maximum reward values of the whole training.

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

Figure 4: DQN episodic reward over 50 episodes.

Once simulating both policies, it became clear that our model outperforms the random policy, which achieved an average of total reward of -172.3 ± 27.5 with confidence 95%. Observing figure 3 it is possible to see that in the beginning of the training, DQN did poor choices that led to very bad results. But once it has explored enough experiences, it's decisions start to become more accurate an less dependent on randomness, therefore, performing better.

(e) Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor γ_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?

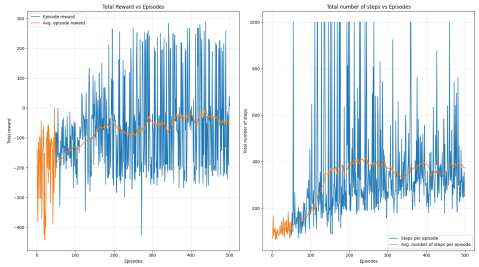

Figure 5: DQN discount factor $\gamma = 0.01$.

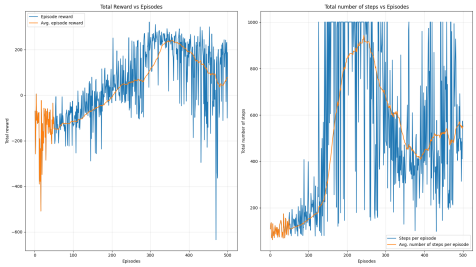

Figure 6: DQN discount factor $\gamma = 1.0$.

For the first case, $\gamma = 0.01$, it can be seen that the results are quite bad. The obtained averaged rewards are in between -100 and 0, and the network does not seem to improve anymore. As the discount factor is so small, future states are barely taken into account, forgetting about the long-term rewards. This is why the network does not perform correctly; it does not care about reaching the goal that much, just about which is the immediate best action to perform.

For the second case, $\gamma = 1$, the results, if not as good as the ones shown in section *(c)*, are acceptable. In this case, all rewards are weighted the same, giving equal importance to future profits as the one given to present ones. This can force the network to perform poorly in some cases, taken not proper present actions with the aim of reaching a better future reward. Generally, immediate rewards should be weighted more than future ones, and this is why the network does not perform as well as the original one.

(f) 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.

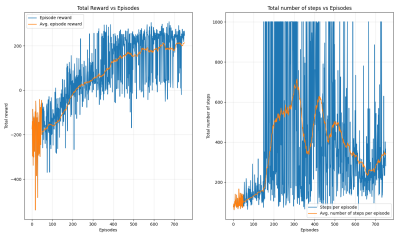

Figure 7: DQN increase in number of $episodes = 750$.

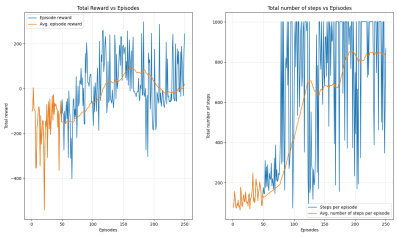

Figure 8: DQN decrease in number of $episodes = 250$.

The two different experiments have been done increasing and decreasing the total number of episodes T_E employed in the original network (500) by a 50%. When the total number of episodes is 750, it can be seen that the averaged episodic reward seems to slowly improve for a bit, although it also seems to be converging around values slightly bigger than 200. When training over less episodes, the results never reach values as good as the ones obtained in networks that trained for more episodes. It can be seen, regarding to the averaged episodic reward, that the results are still unstable, fluctuating a lot over time. Therefore, it can be concluded that this network needs to train for longer in order to achieve competent results.

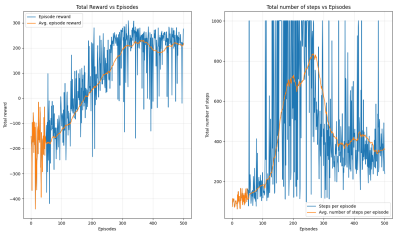

Figure 9: DQN increase in $memory = 30,000$.

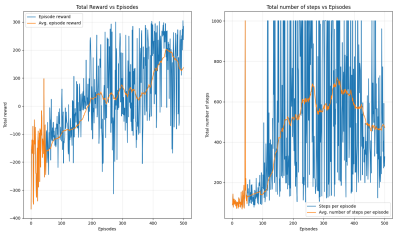

Figure 10: DQN decrease in $memory = 5,000$.

The importance of the replay buffer size has also been analyzed. In order to do so, we have employed the minimum and maximum recommended sizes, 5,000, and 30,000 experiences capacity. When the size is the biggest one, the results are not different from those obtained when the size was equal to 15,000. This proves that the memory capacity employed was good enough in order to sample different types of experiences. Nevertheless, when the size is equal to 5,000, the results are slightly worse than those obtained before. This could mean that the buffer size is not as big as necessary to store enough different experiences. Hence, when the sampling is performed, is less likely to retrieve a good amount of uncorrelated experiences, causing the learning process slightly worse than when a bigger buffer is used.

(g) For the Q-network Q_θ that solves the problem, generate the following two plots:

- *Consider the following restriction of the state $s(y, \omega)$ = (0, y, 0, 0, ω, 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.*

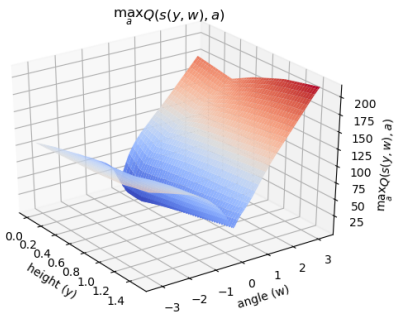

Figure 11: DQN maximum value given angle (ω) and height (y).

The values of the optimal policy make sense, considering that rare are the cases when the model is initialized with a combination of coordinate $x = 0$ and angle $\omega = 0$. Therefore, the model did not train with this kind of situation and does not know how to behave in this

scenario, which leads to a low value. It is worth mentioning that the influence of the height y has a low impact on the value function, probably. Additionally, training for more episodes would lead to a sharper plot since it would know better what to do in each scenario.

In terms of the angle (ω), with more episodes, the less relevant positive or negative angles are, as the function is symmetric and should not give priority for positive angles over negative ones.

- *Let s be the same as the previous question, and plot $argmax_a Q_\theta(s(y, \omega), a)$ 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.*

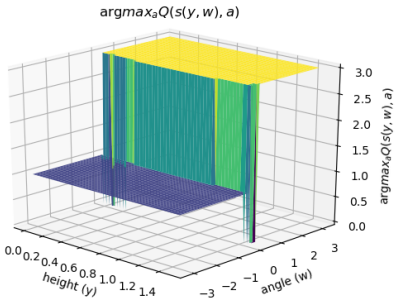

Figure 12: DQN action given angle (ω) and height (y).

The behaviour of the optimal policy makes sense. Depending on the angle (ω) the lander has to adjust its position to land with both legs on the ground with angle zero, that's why it prioritizes action 1 (moving left) and 3 (moving right).

2 Deep Deterministic Policy Gradient (DDPG)

2.1 Problem 2 - Tasks

2.1.1 Why don't we use the critic's target network when updating the actor network? Would it make a difference?

In order to update the actor network, the model backpropagates the mean of all of the values generated by the critic's network. We don't use the critic's target network in this case since we desire to use the most recent version of the critic's network in order to update the actor network. Otherwise, we would be updating the actor network based on a soft-updated (synchronized) critic network, which could make the learning process slower. In addition, if the actor network was updated based on the critic's target network, this soft-synchronization would also propagate to the actor's target network, which once more would slow down the learning process.

2.1.2 Is DDPG off-policy? Is sample complexity an issue for off-policy methods (compared to on-policy ones)?

It is an off-policy algorithm, as it uses off-policy data generated by the actor network and appended to the replay buffer. Then the Bellman's equation learns the Q-function, and uses the Q-function to learn the policy. Bellman's equation does not care which states, actions, or next states from the replay buffer are used, because the optimal Q-function should satisfy the Bellman equation for all possible transitions. Therefore, randomly sampling from the replay buffer guarantees the exploration of the experiences and the updates of the networks will promote the exploitation, since the decisions made in each step should be more target oriented to goal (landing the lander).

In terms of sample complexity[2], one advantage of off-policy methods is the fact that they can explore the state space, as in some occasions the off-policy (behaviour-policy) can initially be random.

[2]Time required to find an approximated optimal policy

This allows the agent to explore. However, this might take some time which would lead to a bad performance with regards to the sample complexity.

On the other hand, on-policy methods can find quicker the approximated optimal policy, but since they do not explore as much as the off-policy methods, they can reach to a local minimum and never find a satisfactory optimal policy. Therefore, adding randomness to the on-policy method is an important factor to avoid this issue and guarantee exploration.

2.1.3 Explain the layout of the network

(a) The parameters that you used and which modification did you implement (if any). Motivate why you made those choices.

- γ: The value chosen for gamma has been 0.99. As the episodes will have an elevated number of states, we have considered that using a big discount factor is beneficial in order to take into account future states as much as possible.
- L: The size of the employed replay buffer has been 30,000. We have considered that now that the actions are continuous, having a bigger buffer will be more beneficial since now the possible actions to be performed increase.
- T_E: The total number of episodes employed for training has been equal to 300. This value was suggested in the problem statement and after testing with others, we found out that training during 300 episodes was enough to get good and consistent results.
- N: The batch size employed has been equal to 64. We kept the previous batch size as we considered that it is big enough to train with different experiences.
- α_{actor}: We set the actor learning rate equal to 5×10^{-5}, which provided stable learning.
- α_{critic}: We set the critic learning rate equal to 5×10^{-4}, which provided stable learning.
- d: The policy update frequency, which denotes when the policy should be updated, as well as the soft updates of the target networks, was set to two. This implies that only in half of the episode steps this will be produced.
- μ: Factor that multiplies the previous noise value for noise generation was set to 0.15.
- σ: Standard deviation of the normal distribution employed to sample noise was set to 0.2.
- τ: Parameter employed for the network soft update was set to 10^{-3}, as suggested, providing good results.

(b) In general, do you think it is better to have a larger learning rate for the critic or the actor? Explain why.

In general is preferable to have a larger learning rate for the critic, this way we delay the convergence of the action's decision making process, which leads to a better exploration since the actions in the beginning are random due to the initialization of the actor network. This directly affects the convergence rate of the critic networks, leading to a better estimation of actor network and therefore, a more stable learning process.

(c) Compare the policy you found with the random agent. Show the total episodic reward over 50 episodes of both agents. Plot the total episodic reward and the total number of steps taken per episode during training. What can you say regarding the training process?

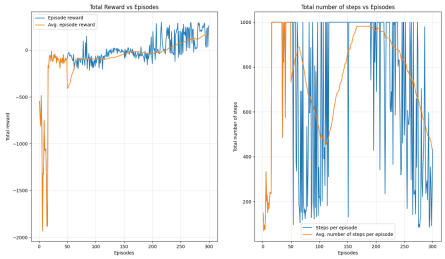

Figure 13: DDPG episodic reward and total number of steps.

In the above image it can be seen how the model behaved during the whole training process. During the first episodes, quite low rewards were obtained (close to -2,000). Nevertheless, once the number of steps per episode began to increase, the rewards improved, getting close to -100, although with some fluctuations. It is around episode 100 when the model starts to be more stable and the rewards begin to improve, slowly but consistently learning. Around episode 150, the averaged episodic reward is close to 0. It is worth mentioning that during most of these episodes, the number of steps taken was 1,000, the maximum number of steps per episode. It is from this moment that the model starts to improve faster, getting higher rewards, and also being able to take less steps per episode while getting better rewards. When 300 episodes have elapsed, the model is getting the best averaged episodic reward values of the while training process.

```
Checking solution...
Episode 49: 100%|                                                  | 50/50 [00:35<00:00,  1.42it/s]
Policy achieves an average total reward of 241.5 +/- 13.3 with confidence 95%.
Your policy passed the test!
```

Figure 14: DDPG episodic reward over 50 episodes.

Random policy achieves an average total reward of -225.1 ± 38.2 with confidence 95%.

(d) Once you have solved the problem, do the following analysis:

- *Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor γ_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?*

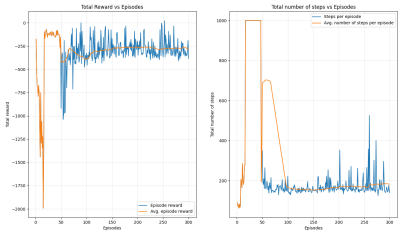

Figure 15: DDPG discount factor $\gamma = 0.01$.

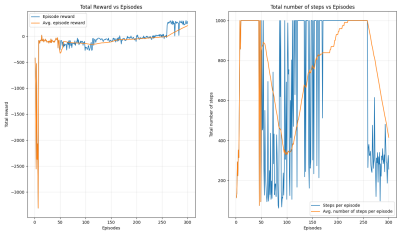

Figure 16: DDPG discount factor $\gamma = 1.0$.

For the first image, representing the training process for a discount factor of 0.01, it can be seen that the behavior during the initial episodes is quite similar to the one described in the previous analysis. Nevertheless, in this case, the averaged reward converges around -250, while in the previous case we could not see any clear convergence. The number of steps taken per episode falls around 200 from the 50^{th} episode, not changing during the rest of the training. Since the discount factor is so low, the model mainly cares about short-term decisions, not long-term, and probably it makes the decision of choosing the action whose

profit is more immediate but likely not the best one towards the global goal of the task. This is why the model trains for way less episodes than the previously analyzed one.

When talking about the second case, the one with discount factor equals to one, several differences can be spotted. Its behaviour is quite more similar to the model described in *(c)* than the model described in the previous paragraph. In this case, using a unitary discount factor, all rewards, short and long-term, are given the same importance. For some cases, this can be irrelevant, but for some others, a short-term decision should be more important than a long term one (for instance, when a huge reward is way ahead in the future). For these cases, a more immediate but safer choice could prime more than a risky one that takes in consideration a probable future reward. This is when the importance of the discount factor becomes patent, in order to weight the decision making process. In this case, although the final reward is also good, the learning process does not seem that consistent when looking at the number of steps per episode.

- *For your initial choice of the discount factor γ_0 investigate the effect of reducing (or increasing) the memory size. Does training become more stable? Document you findings with relevant plots.*

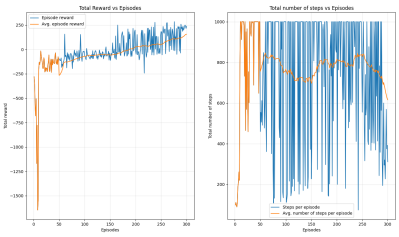

Figure 17: DDPG increase in $memory = 45,000$.

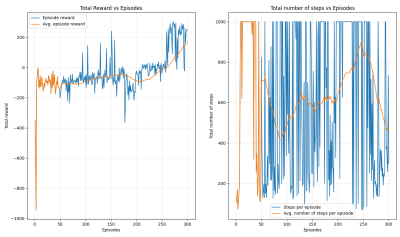

Figure 18: DDPG decrease in $memory = 15,000$.

Analyzing these two figures, it can be seen that there are not huge differences in model behavior. It seems that the one which uses the bigger buffer achieves slightly better results, also training during more time with a higher averaged number of steps per episode. This, of course, could be a consequence of utilizing a bigger buffer, which allows storing more experiences and, therefore, sample uncorrelated items more times than if the buffer was smaller. While it is true that the network with the bigger buffer reaches a positive averaged reward around 100 episodes before, the final results in terms of averaged episodic reward are not that different. This proves that counting on a greater buffer makes the training more constant and stable, but that in terms of final rewards if trained during enough time, it does not really make a difference.

(e) For the networks π_θ and Q_ω that solves the problem, generate the following plots:

- *Consider the following restriction of the state $s(y, \omega)$ = (0, y, 0, 0, ω, 0, 0, 0), where y is the height of the lander and ω is the angle of the lander. Plot $Q_\omega(s(y, \omega), \pi_\theta(s(y, \omega)))$ 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.*

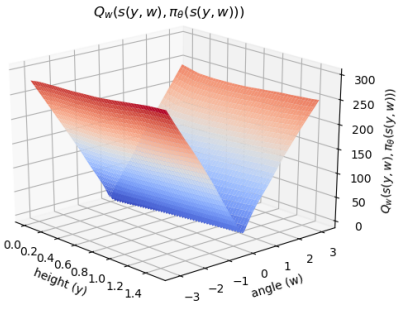

Figure 19: DDPG value function given angle (ω) and height (y).

When compared to figure 11, we can see that exploring more leads to sharper and more symmetric graphs. In addition, due to the continuous action space the policy had more flexibility to learn how to react in each scenario which lead to better plots.

- *Let s be the same as the previous question. Remember that the action is a bi-dimensional vector, where the second element denotes the engine direction. Plot the engine direction $\pi_\theta(s(y,\omega)))_2$ 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.*

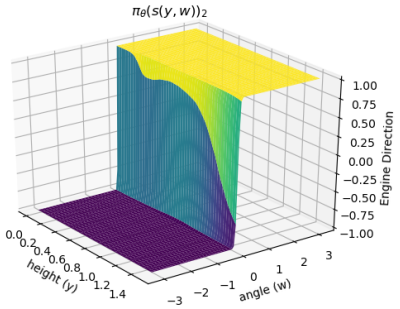

Figure 20: DDPG engine direction given angle (ω) and height (y).

In this case it is even more clear the preference of the policy to propel left or right when the lander is rotated (in comparison to figure 12).

3 Proximal Policy Optimization (PPO)

3.1 Problem 3 - Tasks

3.1.1 Why don't we use target networks in PPO?

Because in PPO we are not trying to solve the algorithm based on Bellman's equation, and therefore, we do not have the overestimation problem that has been discussed in the previous sections. As we are trying to solve a supervised learning problem, there is no need for using a target network.

3.1.2 Is PPO an on-policy method? If yes, explain why. Furthermore, is sample complexity an issue for on-policy methods?

It is an on-policy method because of the following: it involves collecting a small batch of experiences interacting with the environment. This batch is employed to update its decision-making policy.

The process is then repeated, collecting a new batch of experiences, but using now the updated decision-making policy.

Regarding to sample complexity, it depends. The stochasticity plays an important role in order for the model to not to get stuck in a local minimum when sampling an action. For instance, PPO draws its actions based on a normal distribution, computed from the mean and the standard deviation retrieved by the actor network. Although this might delay the convergence to an optimal policy, it assess the model's exploration. If the on-policy does not rely on any randomness, this might lead to a fast convergence to an optimal policy or to a local minimum.

3.1.3 Explain the layout of the network

(a) The parameters that you used and which modification did you implement (if any). Motivate why you made those choices.

- γ: The value chosen for gamma has been 0.99. As the episodes will have an elevated number of states, we have considered that using a big discount factor is beneficial in order to take into account future states as much as possible.
- L: The size of the employed replay buffer has been set to 30,000. Nevertheless, in this case, this is not important, as the buffer is reseted every episode and an episode can only have a maximum of 1,000 steps. Therefore, there will not be any buffer with more than 1,000 experiences stored.
- T_E: The total number of episodes employed for training has been equal to 1,600. This value was suggested in the problem statement and after testing with others, we found out that good results were obtained in the later stages of the training process, so this value actually makes sense.
- N: The batch size employed has been equal to the number of elements stored inside the buffer per episode, so it can actually vary every training episode.
- M: The value of this parameter was set to 10, as suggested in the assignment statement. This value indicates how many backward passes per episode to do.
- α_{actor}: We set the actor learning rate equal to 10^{-5}, which provided stable learning.
- α_{critic}: We set the critic learning rate equal to 10^{-3}, which provided stable learning.

(b) Do you think that updating the actor less frequently (compared to the critic) would result in a better training process (i.e., more stable)?

Similarly to the actor network's learning rate, proposed in problem 2, by updating the actor network less frequently this will lead to a better exploration of the state space, which would directly affect the convergence of the critic network, leading to a better estimation of the actor network and therefore, a more stable procedure.

(c) Compare the policy you found with the random agent. Show the total episodic reward over 50 episodes of both agents. Plot the total episodic reward and the total number of steps taken per episode during training. What can you say regarding the training process?

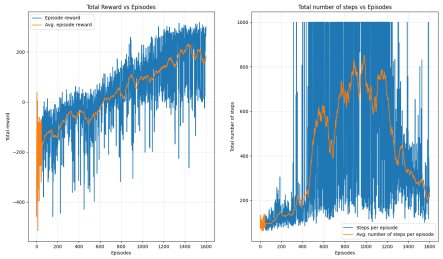

Figure 21: PPO episodic reward and total number of steps.

PPO algorithm, in comparison to the two previous methods perform as many updates. However, instead of updating every single experience is added to the buffer, it fills the buffer and then updates based on M size. In the end, assuming that we use the same parameters that used so far, PPO will roughly update the same amount of times that DDPG did. Observing the average reward through the training it can be seen that the learning process is stable and the reward increases continuously. Around episode 800 (in the middle of the training) the model consistently returns positives rewards and in the end achieved rewards above 200 points.

In terms of the number of the steps, the process is very intuitive. In the beginning the networks are a bit loss until it updates enough and starts to go deeper in terms of steps in order to explore as much as possible, up to a point that it explored enough and it is time to explore its knowledge and start to earn more point. At this stage the average number of steps decreases so the lander can reach the landing pad as fast as possible and achieve better results.

```
Checking solution...
Episode 49: 100%|                                                                | 50/50 [00:18<00:00,  2.64it/s]
Policy achieves an average total reward of 227.9 +/- 25.7 with confidence 95%.
Your policy passed the test!
```

Figure 22: PPO episodic reward over 50 episodes.

Random policy achieves an average total reward of -200.2 ± 38.7 with confidence 95%.

(d) Once you have solved the problem, do the following analysis:

- *Let γ_0 be the discount factor you chose that solves the problem. Now choose a discount factor γ_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?*

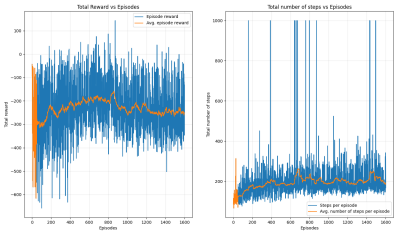

Figure 23: PPO discount factor $\gamma = 0.01$.

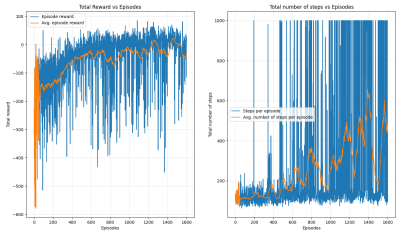

Figure 24: PPO discount factor $\gamma = 1$.

Similarly to problem 1 and 2, the smallest the γ less complex is the problem, but lowest is the variance, which leads to a smaller consideration of future values. Therefore, for $\gamma = 0.01$ we don't explore much. For $\gamma = 1.0$ we take to much into account the future rewards and makes the learning process to unstable.

- *For your initial choice of the discount factor γ_0 investigate the effect of reducing (or increasing) ϵ. Does training become more stable? Document you findings with relevant plots.*

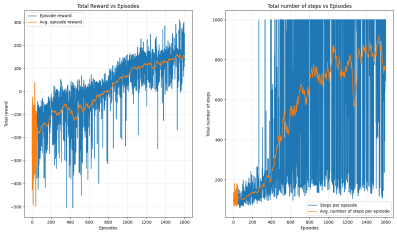

Figure 25: PPO $\epsilon = 0.05$.

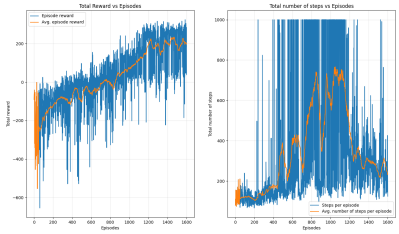

Figure 26: PPO $\epsilon = 0.8$.

The effect of the ϵ is with regards to the learning step of the actor network. If the epsilon is small we restrict the backpropagation to a smaller range, which leads to a more stable learning process, but slow.Therefore, if we increased the number of episode we could achieve good rewards in the end. On the other hand, a bigger epsilon might lead to a faster learning process because it increases the range, but it might be unstable.

(e) For the networks π_θ and V_ω that solves the problem, generate the following plots:

- *Consider the following restriction of the state $s(y, \omega)$ = (0, y, 0, 0, ω, 0, 0, 0), where y is the height of the lander and ω is the angle of the lander. Plot $V_\omega(s(y, \omega), \pi_\theta(s(y, \omega)))$ 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.*

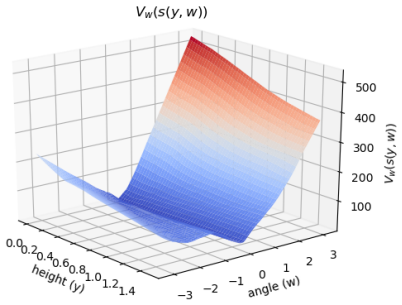

Figure 27: PPO value function given angle (ω) and height (y).

The conclusions for the value function for this plot is very similar to the ones made in problem 1 and 2. Although, one can say that the shape here is different (smoother) due to the last activation function (`tanh`) that computes the actions.

- *Let s be the same as the previous question. Remember that the action is a bi-dimensional vector, where the second element denotes the engine direction. Let the mean value of $\pi_\theta(s)$ be $\mu_\theta(s)$: plot the engine direction $\mu_\theta(s(y, \omega)))_2$ 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.*

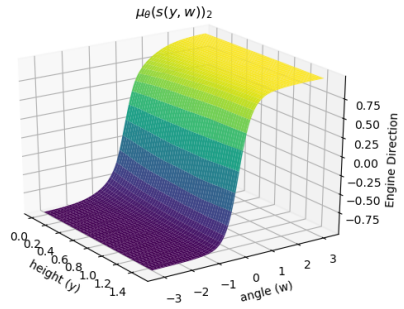

Figure 28: PPO engine direction given angle (ω) and height (y).

For this problem, we compute the mean (μ) that helps sampling the actions based on a Gaussian distribution. In this case, this μ is computed from a `tanh` activation function is provides a more smoother transition between the negative and positive angles.

4 How to run

The models saved and provided within the code have been generated via the *Google Colab* platform, by means of their GPUs. This is relevant because the original code provided in the `<problem>_check_solution.py` files will not work with GPU generated models, only with CPU ones. In order to run the models in the checker files, it is necessary to add the following when loading the model:

```
model = torch.load('model.pth', map_location=torch.device('cpu'))
```

References

[1] J. TORRES.AI, "Deep q-network (dqn)-i." https://towardsdatascience.com/deep-q-network-dqn-i-bce08bdf2af, Aug 2020.

[2] "Deep deterministic policy gradient." https://spinningup.openai.com/en/latest/algorithms/ddpg.html#:~:text=DDPG%20is%20an%20off%2Dpolicy,DDPG%20does%20not%20support%20parallelization., 2018.

[3] C. Trivedi, "Proximal policy optimization tutorial." https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6, Aug 2019.

[4] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning." https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/#:~:text=Adam%20combines%20the%20best%20properties,do%20well%20on%20most%20problems., July 2017.