RLD report

Christopher Heim, Thomas Floquet

Contents

List of Figures

List of Tables

Chapter 1

TME 4

1.1 Introduction

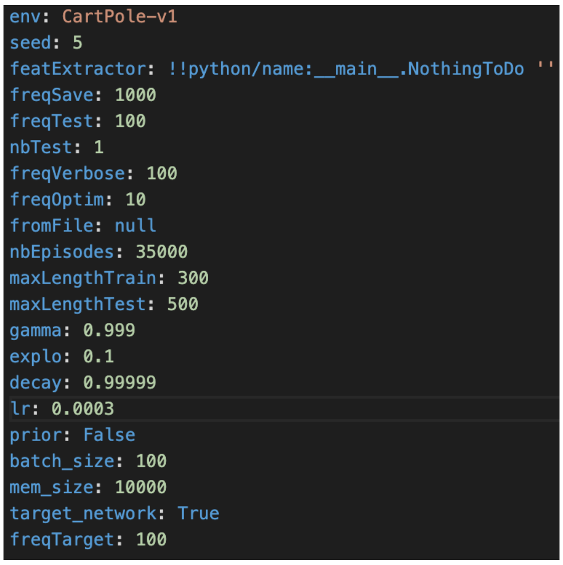
In this chapter, we will study the DQN algorithm with Target Network and Prioritized experience replay.

1.2 Cartpole

We will study the following hyperparameters on Cartpole:

- Loss function: MSELoss vs SmoothL1Loss
- Size of the memory buffer
- Exploration parameter ϵ
- Network architecture
- Update frequency of the target network

Unless specified, we will use the following hyperparameters:



```
env: CartPole-v1
seed: 5
featExtractor: !!python/name:__main__.NothingToDo ''
freqSave: 1000
freqTest: 100
nbTest: 1
freqVerbose: 100
freqOptim: 10
fromFile: null
nbEpisodes: 35000
maxLengthTrain: 300
maxLengthTest: 500
gamma: 0.999
explo: 0.1
decay: 0.99999
lr: 0.0003
prior: False
batch_size: 100
mem_size: 10000
target_network: True
freqTarget: 100
```

FIGURE 1.1: Hyperparameters by default for Cartpole

1.2.1 Loss function

Interest of the SmoothL1Loss: gives less importance to extreme values and in some cases can prevent exploding gradient. We obtained the following curves:

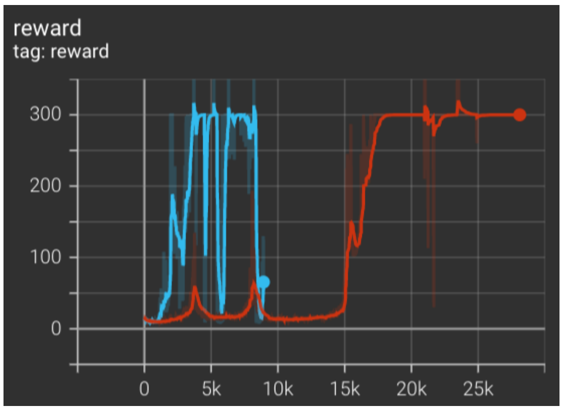


FIGURE 1.2: Comparison between the SmoothL1Loss (red) and the MSELoss (blue)

Conclusion: The MSELoss converges earlier but seems to be less stable.

1.2.2 Size of the memory buffer

In this section, we will study the influence of the size of the memory buffer.

1. Memory size of 100
2. Memory size of 1000
3. Memory size of 10000

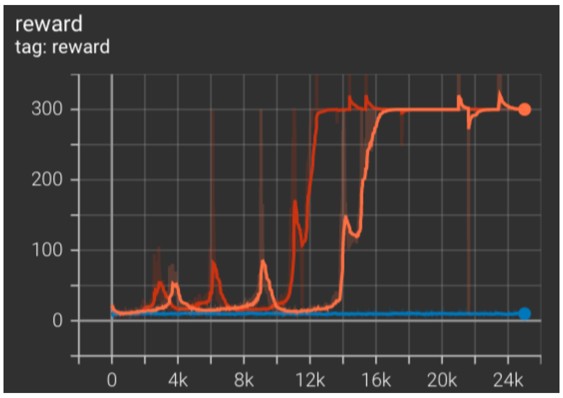


FIGURE 1.3: Comparison of the memory sizes
10 blue, 1000 red and 10000 orange

With a memory size of 100, the agent does not learn an optimal strategy and does not get a reward higher than 15. With a memory size of 1000, the agent converges faster than with a memory size of 10000 and achieves similar results.

Conclusion: The memory size needs to be big enough for the model to learn a good strategy and it has great influence on the results.

1.2.3 Exploration parameter ϵ

In this section, we compare the influence of the exploration parameter ϵ

1. Exploration of 0.01
2. Exploration of 0.1
3. Exploration of 0.2

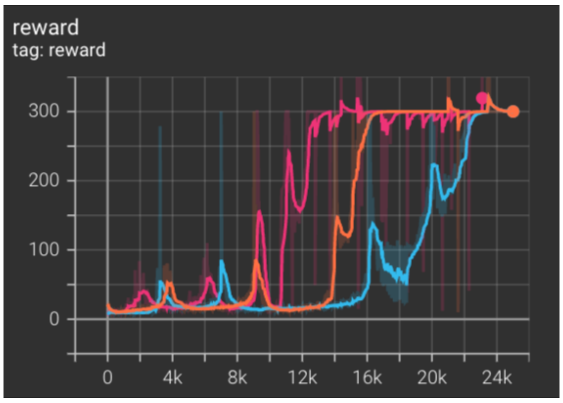


FIGURE 1.4: Comparison of the exploration parameter
0.01 blue, 0.1 orange, 0.2 rose

With *epsilon* $= 0.01$, the convergence is much slower because the agent might be temporarily stuck on an non-optimal trajectory. With *epsilon* $= 0.2$, the convergence is a little bit faster than with $\epsilon = 0.1$, but it seems less stable because once the convergence is achieved the exploration is still too important.

Conclusion: The best exploration parameter seems to be a good compromise.

1.2.4 Network architecture

In this section, we study the effect of the network architecture.

1. One linear layer with 20 neurons
2. One linear layer with 200 neurons
3. Two linear layers with 200 neurons

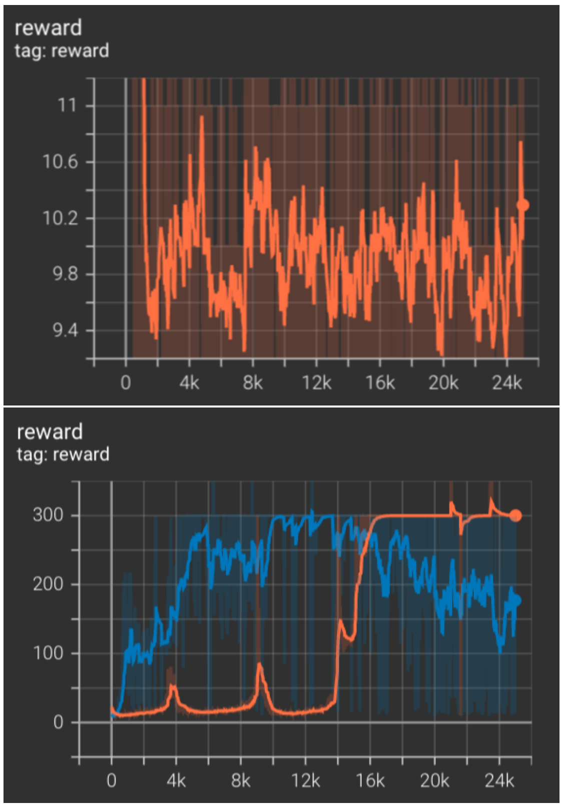


FIGURE 1.5: Comparison of network architectures
20 neurons (top orange), 200 neurons (bottom orange), 2 layers with 200 neurons (blue)

With 20 neurons, the agent does not converge. With 2 layers of 200 neurons, it converges with less iteration but it is much less stable.

Conclusion: The model needs enough neurons in order to learn the complex optimal trajectory, but not too much otherwise it is unstable.

1.3 Update frequency of the target network

In this section, we compare the update frequency

1. Update every iteration (= without target network)
2. Update every 100 iterations
3. Update every 1000 iterations

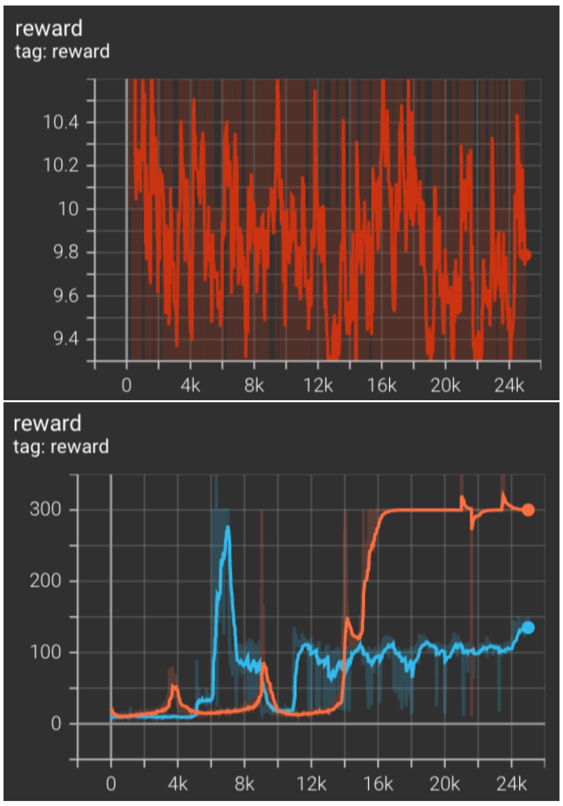


FIGURE 1.6: Comparison of the update frequency *1 (orange up), 100 (orange bottom), 1000 (blue*

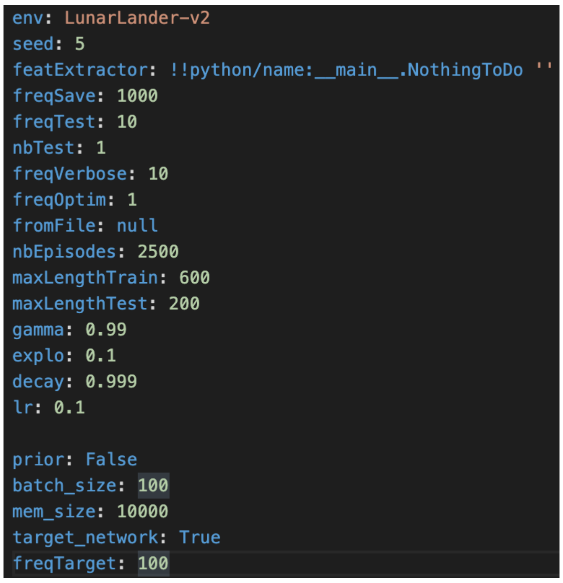
With an update every iteration, the instability is too big so the agent does learn an optimal strategy. With an update every 1000 iterations, the agent seems to be stuck on an under optimal strategy (maybe it will eventually converge and be more stable).

Conclusion: The update frequency is very important for the convergence and 100 seems to be a good compromise.

1.4 Lunar lander

We will study the impact of the target network on Lunar Lander.

We will use the following hyperparameters by default:



```
env: LunarLander-v2
seed: 5
featExtractor: !!python/name:__main__.NothingToDo ''
freqSave: 1000
freqTest: 10
nbTest: 1
freqVerbose: 10
freqOptim: 1
fromFile: null
nbEpisodes: 2500
maxLengthTrain: 600
maxLengthTest: 200
gamma: 0.99
explo: 0.1
decay: 0.999
lr: 0.1

prior: False
batch_size: 100
mem_size: 10000
target_network: True
freqTarget: 100
```

FIGURE 1.7: Hyperparameters by default for Lunar Lander

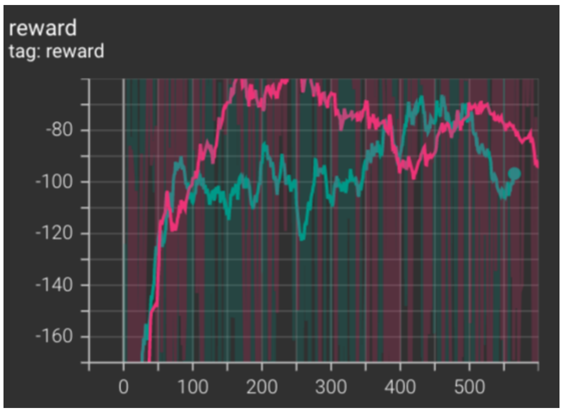


FIGURE 1.8: Impact of the max length in train
600 (rose), 1000 (green)

The satellite keeps its balance but does not touch the grown before the end of the episode, so we increased maxLengthTrain from 600 to 1000 and maxLengthTest to 500, but the results were similar, so the satellite would just stay in the air as long as possible and not land. It seems like the satellite is stuck on an under optimal trajectory.

So we tried increasing the exploration parameter from 0.1 to 0.3.

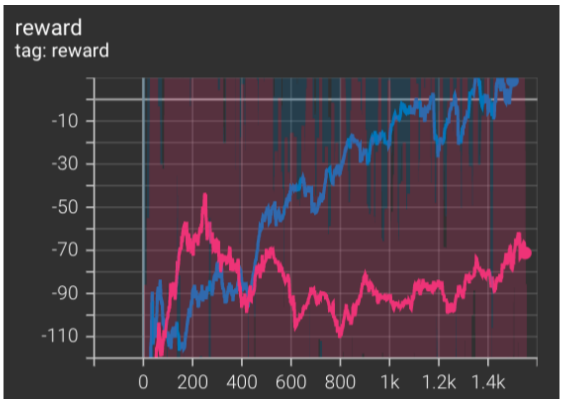


FIGURE 1.9: Impact of the exploration parameter
0.1 (rose), 0.3 (blue)

The results is better with a higher exploration mode, because the agent is not stuck on the sub-optimal strategy of staying in the air.

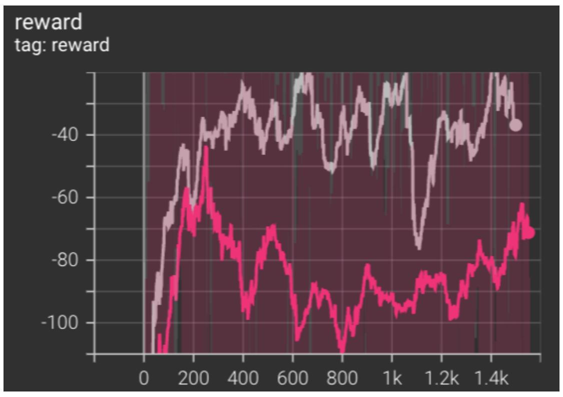


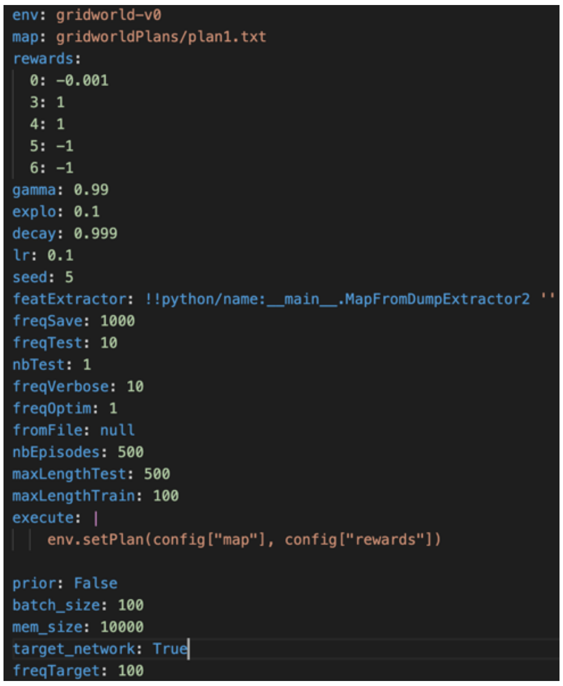
FIGURE 1.10: Impact of the target network
without (light rose), with (rose)

Without the target network, the result seems a little bit better, maybe because the agent is less stuck on the same sub-optimal trajectory, but the satellite is still mainly keeping its balance without going down.

1.5 Gridworld

We will study the impact of the target network on the plans 1 and 5 of Gridworld.

We will use the following hyperparameters:



```
env: gridworld-v0
map: gridworldPlans/plan1.txt
rewards:
  0: -0.001
  3: 1
  4: 1
  5: -1
  6: -1
gamma: 0.99
explo: 0.1
decay: 0.999
lr: 0.1
seed: 5
featExtractor: !!python/name:__main__.MapFromDumpExtractor2 ''
freqSave: 1000
freqTest: 10
nbTest: 1
freqVerbose: 10
freqOptim: 1
fromFile: null
nbEpisodes: 500
maxLengthTest: 500
maxLengthTrain: 100
execute: |
    env.setPlan(config["map"], config["rewards"])

prior: False
batch_size: 100
mem_size: 10000
target_network: True
freqTarget: 100
```

FIGURE 1.11: Hyperparameters by default for Gridworld

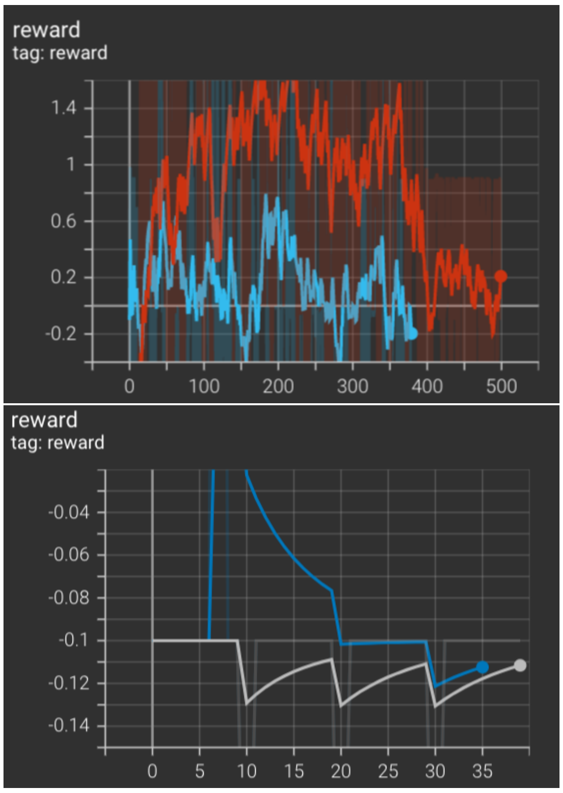


FIGURE 1.12: Impact of the target network (Plan1 top) *without (light blue and blue), with (red and grey*

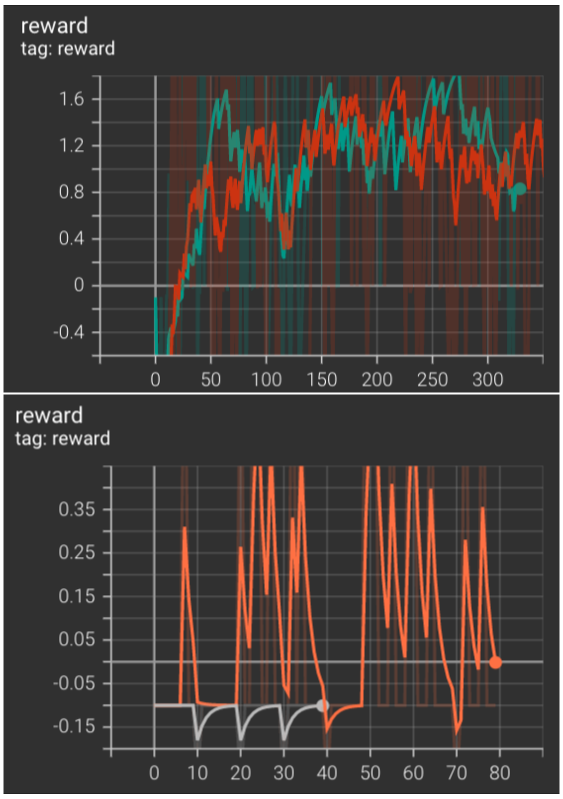


FIGURE 1.13: Impact of the exploration parameter (Plan1 top) *0.1 (red and grey), 0 (green), 0.4 (orange)*

On plan 1, the target network achieves better results, and the result is much better with an exploration parameter of 0, because otherwise the agent is stuck trying to avoid the red square. On plan 5, if the exploration parameter is too small (ex : 1), the agent never finds the green square so it remains blocked until the maximum length of an episode. So we tried with $\epsilon = 0.4$, the agent finds the green square sometimes but never finds the yellow square reward. It might find it with an even higher ϵ, but then it would difficultly be able to find the green square back.

Chapter 2

TME 5

2.1 Introduction

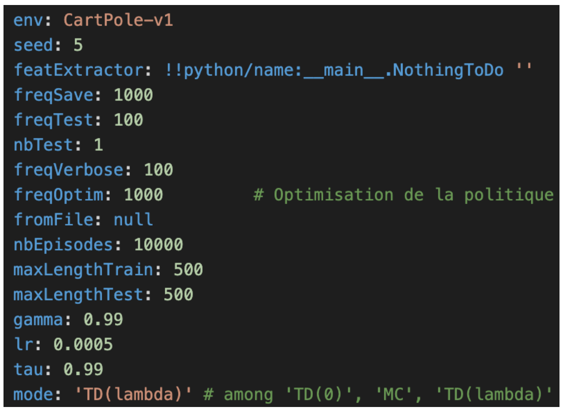
In this chapter, we will study the Actor-Critic algorithm.

2.2 Cartpole

We will study the following configurations on Cartpole:

- TD(0)
- Monte-Carlo
- TD(λ)

Unless specified, we will use the following hyperparameters:



```yaml
env: CartPole-v1
seed: 5
featExtractor: !!python/name:__main__.NothingToDo ''
freqSave: 1000
freqTest: 100
nbTest: 1
freqVerbose: 100
freqOptim: 1000         # Optimisation de la politique
fromFile: null
nbEpisodes: 10000
maxLengthTrain: 500
maxLengthTest: 500
gamma: 0.99
lr: 0.0005
tau: 0.99
mode: 'TD(lambda)' # among 'TD(0)', 'MC', 'TD(lambda)'
```

FIGURE 2.1: Hyperparameters by default for Cartpole

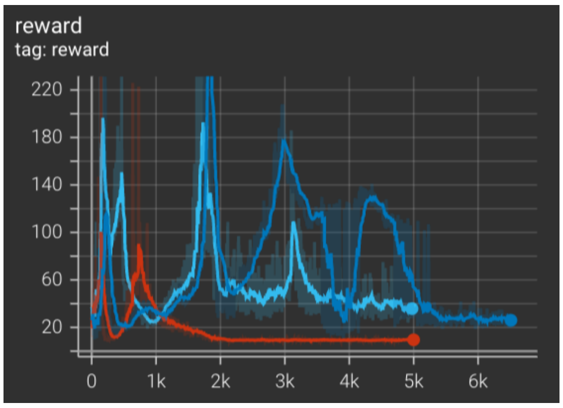


FIGURE 2.2: Comparison between TD(0), Monte-Carlo and TD(λ)
TD(0) blue, MC red and TD(λ) light blue

The cumulative reward is less than with DQN. With Monte-Carlo, the agent does not learn. TD(0) achieves better results.

After trying different hyperparameters, we achieved better results by decreasing the learning rate from 0.001 to 0.005 and decreasing the optimization frequence from 40 to 1000

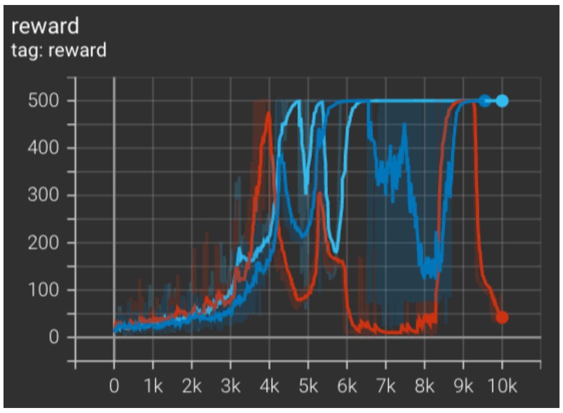


FIGURE 2.3: Comparison between TD(0), Monte-Carlo and TD(λ)
MC blue, TD(0) red, TD(λ) light blue

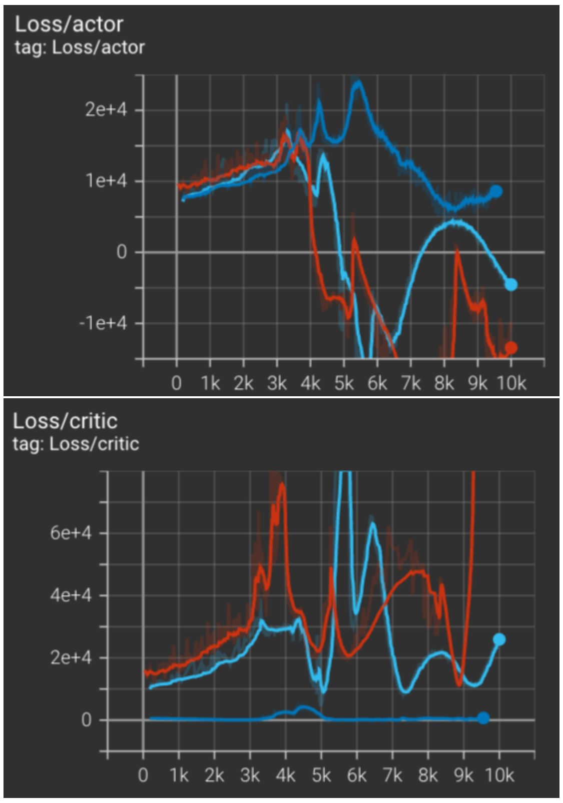


FIGURE 2.4: Comparison of loss functions between TD(0), Monte-Carlo and TD(λ)
MC blue, TD(0) red, TD(λ) light blue
Actor top, critic bottom

MC takes a little bit more time to find an optimal strategy than TD(0) but is more stable, maybe because the propagation of the reward is too slow to have a good critic with TD(0). TD(lambda) seems to be the best compromise and more stable.

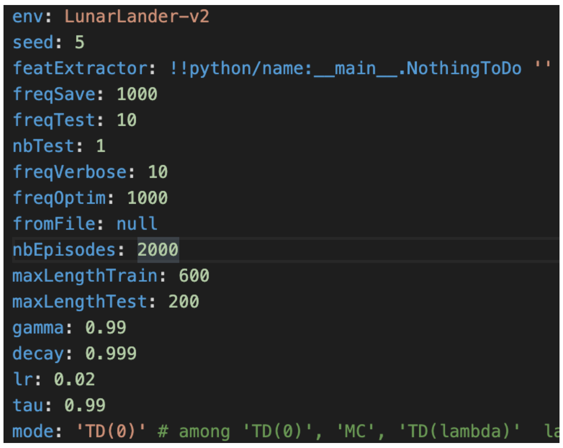
We also note that the inference time during the agent's choice of action is much shorter than in the DQN case. This is because there is no need to evaluate the criticism but simply to sample according to π.

2.3 Lunar

We will study the following configurations on Lunar:

- TD(0)
- Monte-Carlo
- TD(λ)

Unless specified, we will use the following hyperparameters:



```
env: LunarLander-v2
seed: 5
featExtractor: !!python/name:__main__.NothingToDo ''
freqSave: 1000
freqTest: 10
nbTest: 1
freqVerbose: 10
freqOptim: 1000
fromFile: null
nbEpisodes: 2000
maxLengthTrain: 600
maxLengthTest: 200
gamma: 0.99
decay: 0.999
lr: 0.02
tau: 0.99
mode: 'TD(0)' # among 'TD(0)', 'MC', 'TD(lambda)'  la
```

FIGURE 2.5: Hyperparameters by default for Lunar

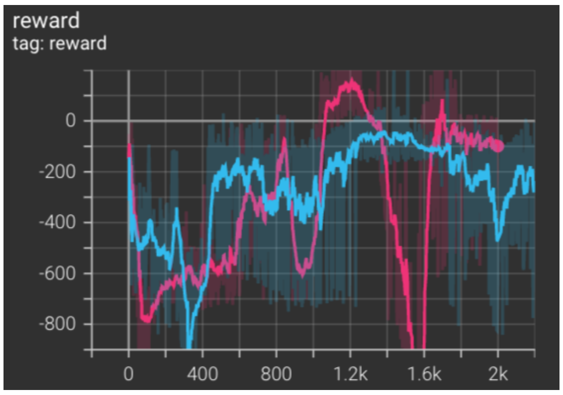


FIGURE 2.6: Comparison between TD(0) and Monte-Carlo on Lunar
TD(0) rose, MC light blue

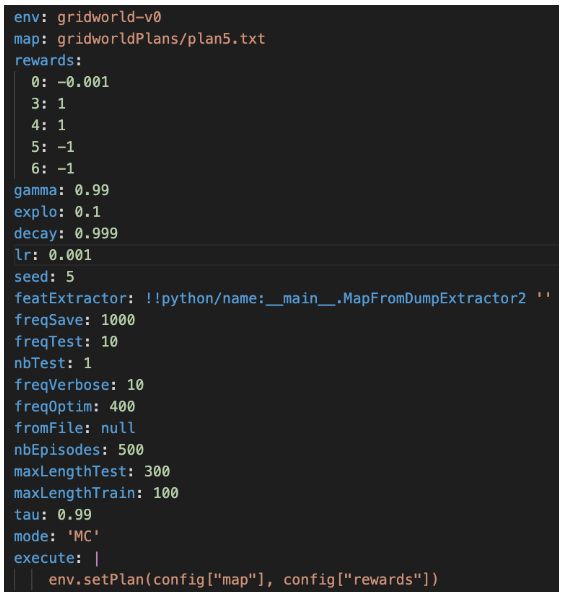
TD(0) and MC achieve similar results but MC seems more stable. The results are better with DQN.

2.4 Gridworld

We will study the following configurations on the plans 1 and 5 of Gridworld:

- TD(0)
- Monte-Carlo
- TD(λ)

Unless specified, we will use the following hyperparameters:



```
env: gridworld-v0
map: gridworldPlans/plan5.txt
rewards:
  0: -0.001
  3: 1
  4: 1
  5: -1
  6: -1
gamma: 0.99
explo: 0.1
decay: 0.999
lr: 0.001
seed: 5
featExtractor: !!python/name:__main__.MapFromDumpExtractor2 ''
freqSave: 1000
freqTest: 10
nbTest: 1
freqVerbose: 10
freqOptim: 400
fromFile: null
nbEpisodes: 500
maxLengthTest: 300
maxLengthTrain: 100
tau: 0.99
mode: 'MC'
execute: |
    env.setPlan(config["map"], config["rewards"])
```

FIGURE 2.7: Hyperparameters by default for Gridworld

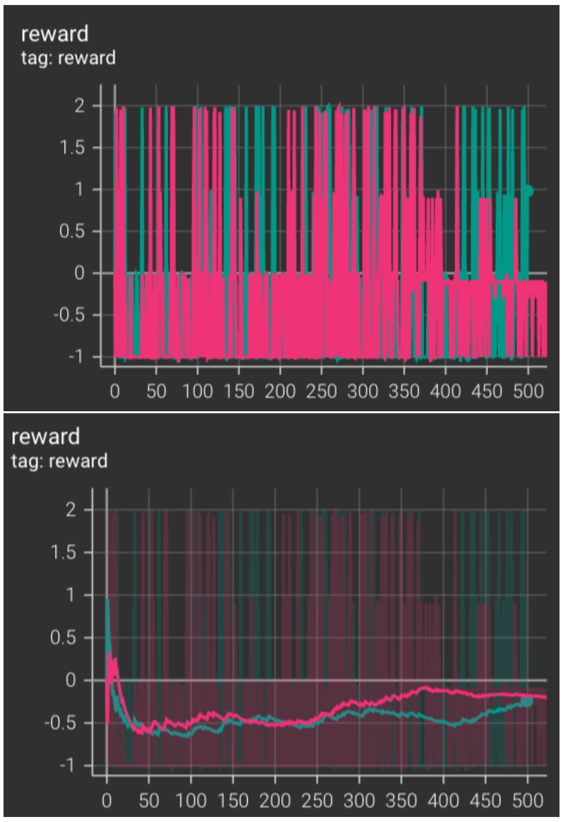


FIGURE 2.8: Comparison between TD(0), Monte-Carlo and TD(λ) on plan 1
MC rose, TD(0) green
No smooth top, smooth 0.99 bottom

The results look slightly better for MC, but in both case the learning is very unstable. DQN performs better overall because it is more stable.

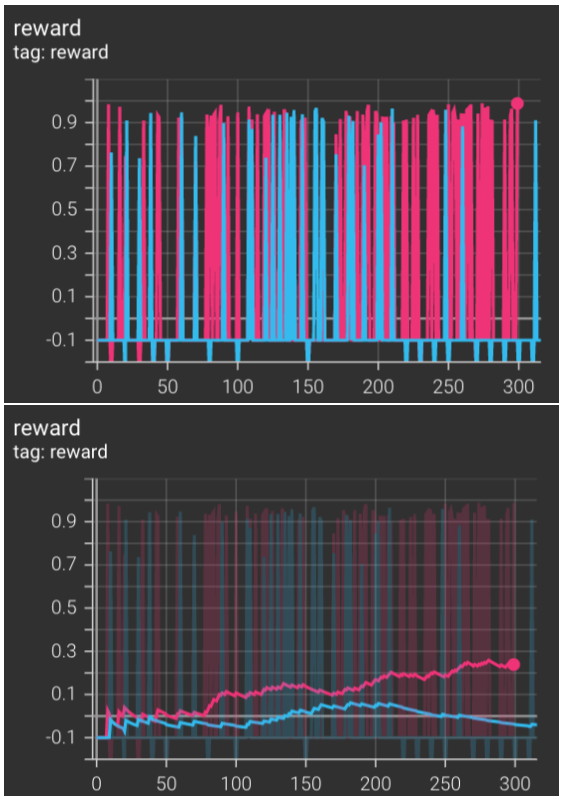


FIGURE 2.9: Comparison between TD(0), Monte-Carlo and TD(λ) on plan 5
MC light blue, TD(0) rose
No smooth top, smooth 0.99 bottom

Once again, the results are very similar, but the agent never manages to go down to get the reward at the bottom of the map.

Chapter 3

TME 6: Proximal Policy Optimization

3.1 Introduction

TRPO is a first attempt at doing a policy gradient algorithm with off-policy learning. It does so by introducing a Kullback-Leibler divergence counstraint on the learned policy to the previous one to define a trust region in the space of policies.

TRPO uses second order derivatives with conjugates gradients. PPO achieves similar performance and is much simpler to implement and to tune. It uses a KL penalty instead of counstraint.

There are two versions, one with a clipping of the Importance ratios. And one with a learned KullBack-Leibler regularization.

3.2 Cartpole

We use the following set of hyper parameters where gamma is the discount factor, tau is the discount faster of the Generalized Advantage Estimation, clip is the clipping factor, c1 is a weighting parameters in front of the critic loss, and c2 is the weighting term in front of the entropy regularization term.

freqOptim	maxLengthTrain	gamma	optim steps	gae tau	clip	batch size	hidden dim	critic lr	actor lr	c1	c2
32	300	0.98	10	0.8	0.2	32	64	3e-4	1e-4	0.5	0.0

Table 3.1: PPO clipped parameters

For the PPO-KL we use a $\delta = 1$ and $kl_adapt = 1$, which correspond respectively to δ, β_0 in the paper.

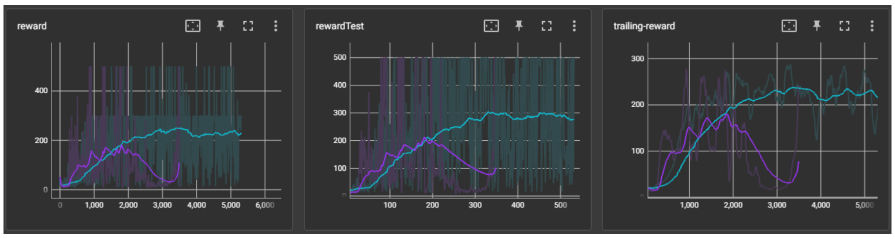


FIGURE 3.1: PPO clipped (blue) vs KL (purple)

1. PPO-clipped reaches a mean reward of 230, where the max of actions taken is 300 outperforming by a large margin PPO-KL on this map.
2. Indeed PPO-KL starts by a fast convergence, but does some catastrophic forgetting, before reaching back PPO-clipped.
3. The learning of both versions is quite noisy even after convergence

3.3 Lunar Lander

We use the following set of hyper parameters

freqOptim	maxLengthTrain	gamma	optim steps	gae tau	clip	batch size	hidden dim	critic lr	actor lr	c1	c2
20	300	0.999	4	0.8	0.2	32	64	3e-4	1e-4	0.5	0.0

TABLE 3.2: PPO clipped parameters

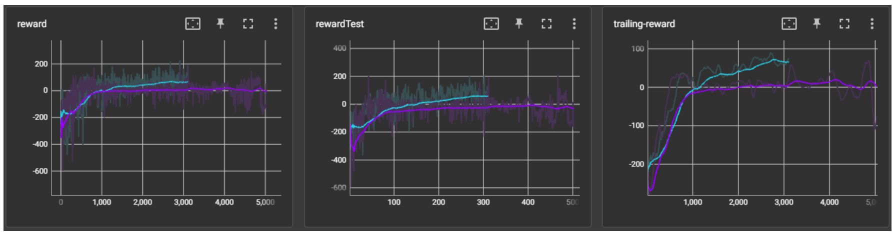


FIGURE 3.2: PPO clipped (blue) vs KL (purple)

1. The KL version plateau around a mean reward of zero.
2. The clipped version plateau around a mean reward of 100 outperforming once again the KL version by far.

The clipped version manages to successfully land the rocket most of the time between the flags, even though we can remark the learning curves are quite noisy. But the KL version simply doesn't.

3.4 Grid world

We used the following hyper parameters

freqOptim	maxLengthTrain	gamma	optim steps	gae tau	clip	batch size	hidden dim	critic lr	actor lr	c1	c2	delta	$\text{kl}_a dapt$
32	300	0.999	10	0.8	0.2	8	64	3e-4	3e-4	0.5	0.0	0.01	1

TABLE 3.3: PPO Hyper-parameters

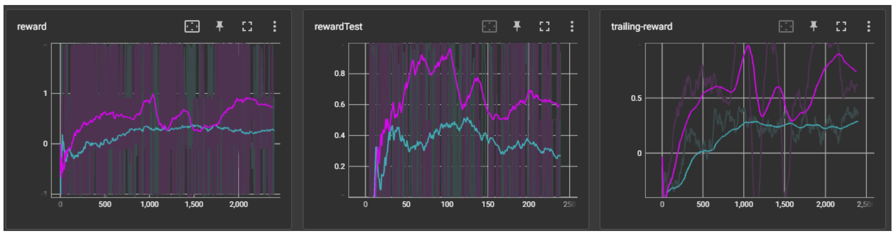


FIGURE 3.3: GridWorld plan 1: PPO clipped (blue) vs KL (purple)

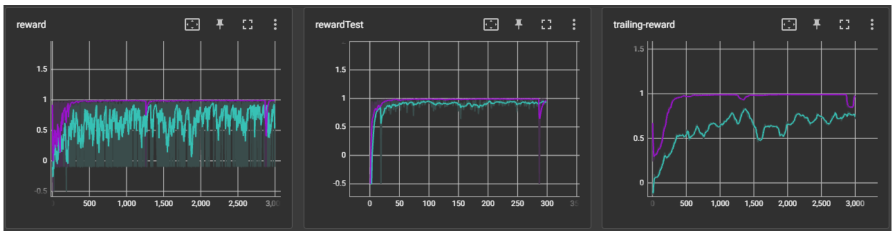


FIGURE 3.4: GridWorld plan 5: PPO clipped (blue) vs KL (purple)

On this map, PPO-KL outperforms PPO-clipped on both maps. It is also very stable compared to PPO-clipped on Gridworld plan 5

We notice on plan 1 that both PPO manage to retrieve the second reward before going back to the first, they don't have a constant behavior, nor do they manage to converge towards this solution.

Chapter 4

TME 7: Deep Deterministic Policy Gradient

4.1 Introduction

In this chapter, we will study the the Deep Deterministic Policy Gradient approach. It belongs to actor-critic policy gradient methods. It uses off-policy data and the Bellman equation to learn the Critic in a smilar fashion to DQN. It is especially designed for continuous action spaces, by trying to learn a deterministic policy $a = \mu_\theta(s)$. The gradient of the policy boils down to $\nabla_\mu J_\beta(\theta) = E_{s-\rho^\beta}[\nabla_a Q(s,a)_{|a=\mu_\theta(s)} \nabla_\theta \pi_\theta(s)]$

1. Since we use a deterministic policy, we add an Ornstein-Uhlenbeck noise to the actions at training time, for exploration.

2. We use soft update of the target critic network and target policy network controlled by a parameter τ

4.2 Hyper-parameter search

This section highlights our experiment on DDPG hyper-parameters on the map Pendulum v-0. I proceeded by initially using the settings recommended by the OpenAI baseline. And I experimented around these values on isolated hyper-parameters.

4.2.1 Normalization

Reading a few articles, we notice that the use of Batch Normalization is often contested and Layer Normalization is preferred. Therefore we wanted to study the effects of Layer Normalization over Batch Normalization. Here is the set of hyper-parameters we used :

freqOptim	maxLengthTrain	gamma	noise	optim steps	batch size	hidden dim	soft tau	m
100	300	0.99	0.2	50	128	128	0.01	

Hyper-parameters

We obtained the following curves.

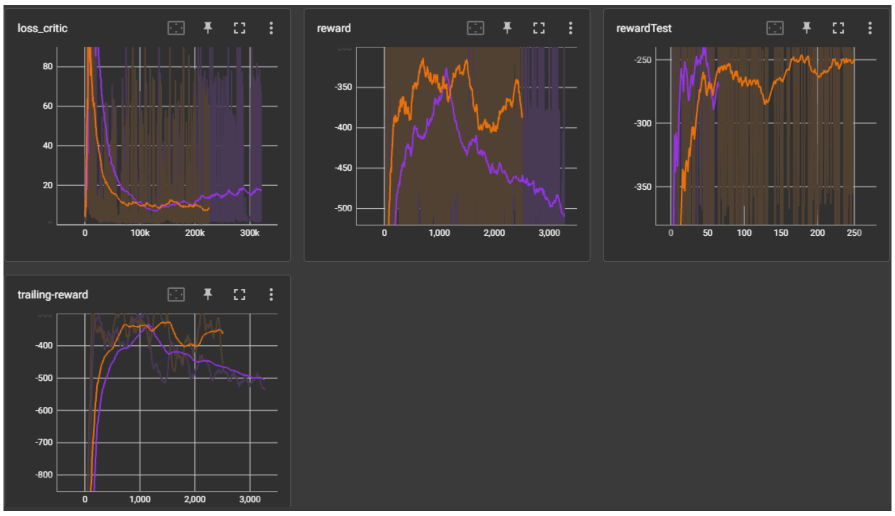


Figure 4.1: Comparison of the Layer Normalization mechanism
(a) In orange the network using Layer Normalization (b) In purple without

We notice large improvements in rewards both at training and testing time. The critic loss seems more stable with normalization, as opposed to the other, the network does not lose capacity over time, as shown by the increasing cost curve at the end and the falling reward.

Overall performance with DDPG are good compared to the literature on Pendulum-v0. A mean reward of -200 is generally the best we can expect on this map.

Conclusion: Layer normalization improve convergence and stabilize learning over time.

4.2.2 Batch size

In this section, we will compare the results using various batch-size, we use the previous hyper-parameters table 4.3.

We are gonna experiment with :

1. (a) A batch size of 128, 50 optimization steps.
2. (b) A batch size of 1000, 5 optimization to keep the ratio of observations seen by the network the same.
3. (c) A batch size of 1000, 20 optimization steps

Experiment (b) shows that with a very few optimization steps even with large batch size, the network struggles to converge quickly enough. Experiment (c) and (a) confirm that larger batch size do not help reaching better convergence or better performance, nor does it remove the excess of stochasticity that we see in the not-smoothed curves. We will investigate this last issue by tuning the soft update parameter ”tau” later on.

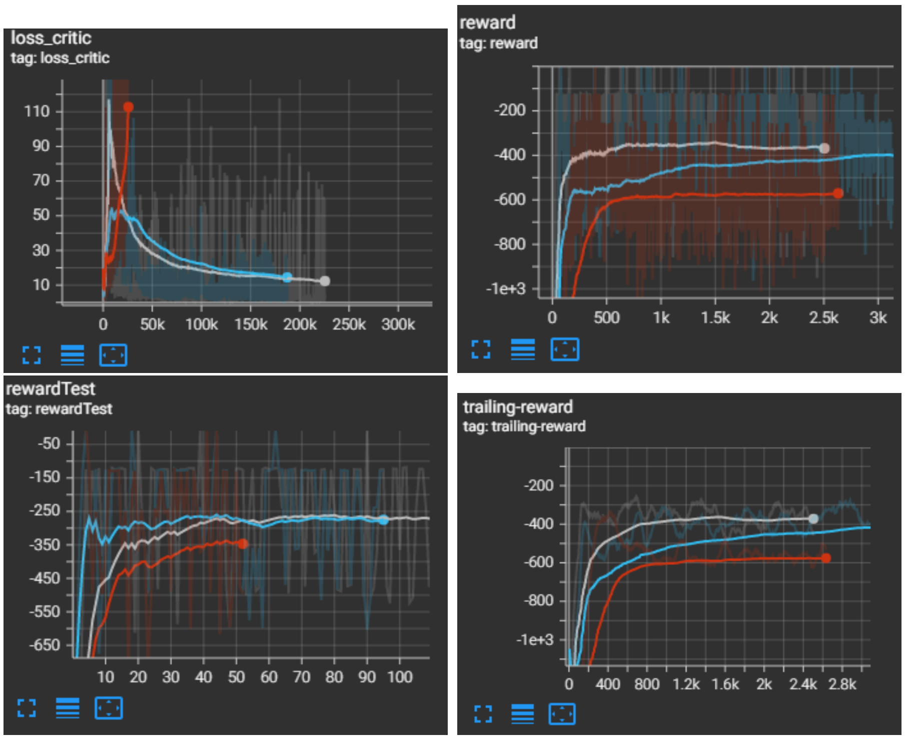


FIGURE 4.2: Comparison of the Layer Normalization mechanism
(a) Grey (b) Red (c) Blue

Conclusion: We will prefer using smaller batch sizes for the profit of more optimization steps.

4.2.3 Hidden layers

In this section, we compare the impact of the network complexity by studying two scenarios using previous hyper-parameters 4.3

1. (a) A 3 layer network with 128 hidden units

2. (b) A 3 layer network with 32 hidden units

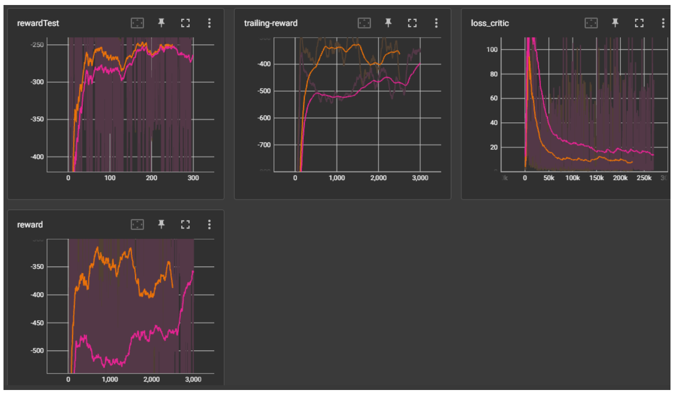


FIGURE 4.3: Network complexity analysis
(a) Orange (b) Purple

The shallower network has inferior results overall. It converges slower, and the critic seems to reach a bigger loss value. It is likely that the network might have converged to the same level would we have pushed the training further.

Conclusion: Using a Critic and Actor network with greater complexity improves overall performance.

4.2.4 Soft update

In this section, we study the effect of the soft update parametrized by which controls how much of the updated parameter do we keep to update the target network. We use the same hyper-parameters as previously 4.3.

1. (a) A soft update with a parameter = 0.01
2. (b) A soft update with a parameter = 0.3

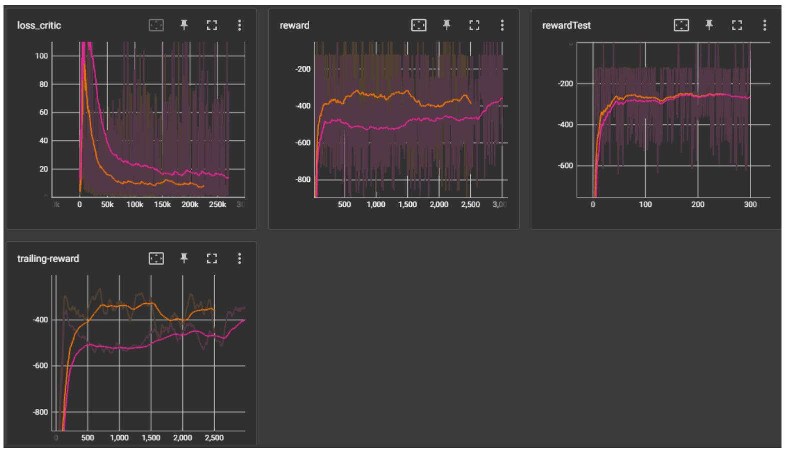


FIGURE 4.4: Soft update of target networks analysis
(a) Orange (b) Purple

There is a great discrepancy in results at training time, but little difference at test time. We notice the same amount of noise during training while we could have expected more noise for case (b) because of more rash update of the target. In that regard, we also could have expected faster convergence for case (b) because the target network could be optimized further, but we notice the opposite for the loss function.

Conclusion: I recommend using lower value for the τ parameter, because of a more stable convergence.

4.3 Lunar-landing

Lunar Lander necessited quite a lot of hyper-parameter tuning. Indeed, since we receive a penalty of -100 if the Lander crashes and of + 100 if it lander and comes to rest.

Initially the learned Lander tends to levitate with the minimum use of thruster and can be stuck in this state for quite some time.

1. Only by using a network with Layer Normalization and a 3 layer network with 400 hidden dimension did we manage to obtain good results.
2. Another important parameter to set was the use of a discount factor of 0.99. Using lower gamma proved less efficient, because we wanted to take into account this possible reward of landing inside the zone, even if it comes at great risk. (estimated target could be biased or noisy)
3. We initially tried a batch size of 500 and 1000, but it reduced stochasticity too much and prevented us from leaving this local minima so we went on with 128.

freqOptim	maxLengthTrain	gamma	noise	optim steps	batch size	hidden dim	soft tau	mem size	critic lr	actor lr
100	300	0.99	0.1	50	128	400	0.01	10k	1e-3	1e-4

TABLE 4.2: Hyper-parameters of LundarLander experiment

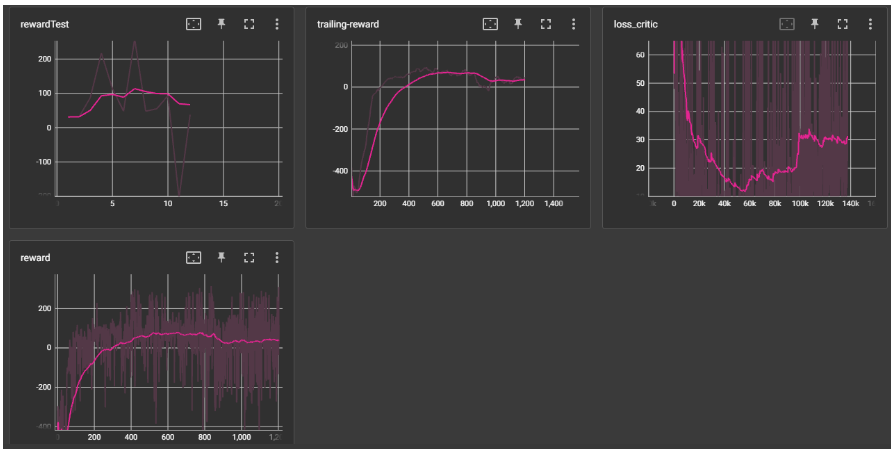


FIGURE 4.5: LunarLanderContinuous-v2 training curves with DDPG

After 200 episodes, we reach an average reward of 100. Looking at instantaneous reward, we see that DDPG is quite stochastic on this map, with a mean reward of 100 but spanning at +200 and -200. Looking at Gym-OpenAI documentation, we can expect a reward in the [100-140] range on this map for an agent landing successfully on the launching pad, meaning that on average, our agent lands on the pad, but tends to be either excellent or very poor. Remember that we added noise to our actions during training, which very likely induce this stochasticity. Indeed if we were to look at the reward at test time, it is "always" positive.

We thought that using a noise of standard deviation 0.2 instead of 0.1 would help increase generalization. It sort of did the opposite. Convergence around a mean reward of 100 came later on, and after some time, we did some sort of catastrophic forgetting with the loss of the critic increasing.

4.4 Mountain-car

freqOptim	maxLengthTrain	gamma	noise	optim steps	batch size	hidden dim	soft tau	mem size	critic lr	actor lr
100	300	0.99	0.2	50	128	400	0.01	10k	1e-3	1e-4

Table 4.3: Hyper-parameters of mountain-car experiment

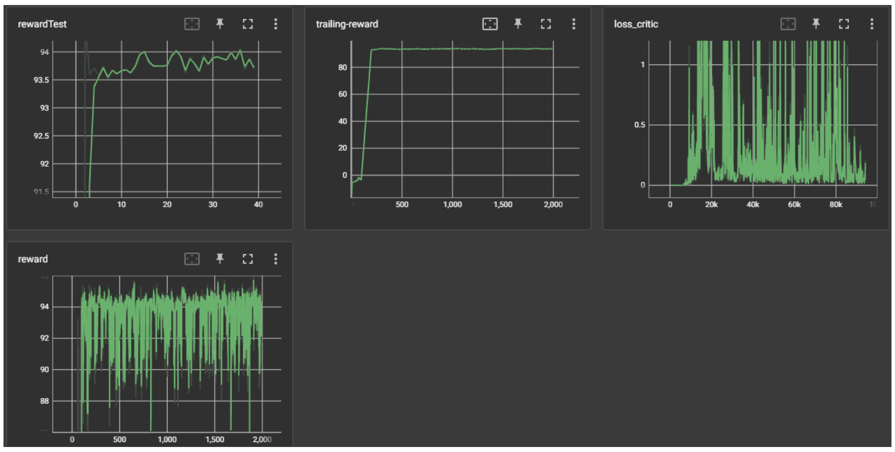


Figure 4.6: MountainCarContinuous-v0 training curves with DDPG

We notice that after 500 episodes, DDPG converges towards optimal reward of 100. Looking at the trailing reward over 100 episodes, we notice that DDPG is quite stable on this map. Qualitatively, the car reaches the top of the mountain in under 100 actions on average.

Chapter 5

TME 8: Soft Actor-Critic (SAC)

5.1 Introduction

Soft Actor Critic (SAC) is an attempt to get the best of two worlds:

1. TRPO and PPO approach, where you have a stochastic policy and you apply policy gradient methods, but on-policy, resulting in a very stable learning of low sample efficiency

2. DDPG, TD3 approach, where you have a deterministic policy and you apply policy gradient methods, but off-policy, resulting in a very unstable learning of better sample efficiency

SAC uses a stochastic policy with an entropy regularization term to prevent the policy to converge towards determinism. This term changes the usual objective functions seen before and need the use of a "soft" critic target and "soft" policy target. Everything is learned off-policy for better sample efficiency.

The implemented code presents two version, one with automatic tuning of the temperature parameter controlling the entropy regularization term, and one with an arbitrary choice of this hyper-parameter. Note that the automatic version still requires to set a target entropy parameter, so it isn't really automatic. Authors of SAC recommend to use an target entropy of $-dim(ActionSpace)$ for continuous action space. I did experiment with this version, but in the following sections, we use the non-automatic version to study the effect of this regularization term.

5.2 Hyper-parameter search

This section highlights our experiment on SAC hyper-parameters on the map Pendulum v-0. I proceeded by initially using the settings recommended by the OpenAI baseline in their "model zoo". And I experimented around these values on isolated hyper-parameters.

5.2.1 Batch-size and number of optimization steps

In this section, we conduct an experiment on batch-size and frequence/number of optimization steps with the following hyper-parameters :

1. The Critic is a 2 layer network without layer normalization nor batch-normalization.
2. The Actor is a 2 layer network for each parameter of the gaussian, again no layer normalization nor batch-normalization.
3. Activation functions are LeakyRelu for stability.

In this first section, we set the temperature to 0.002 like openAI did, and did not use adaptive temperature.

freqOptim	maxLengthTrain	gamma	temperature	optim steps	batch size	hidden dim	soft tau	mem size	learning rate
50	300	0.99	0.001	20	128	64	0.005	50k	1e-3

TABLE 5.1: SAC Hyper-parameters

We tried 2 versions to assess the best value for batch size.

1. (a) A 20 steps optimization with batches of size 128 every 50 events.
2. (b) A 20 steps optimization with batches of size 1000 every 100 events.
3. (c) A 50 steps optimization with batches of size 1000 every 1000 events.
4. (d) A 10 steps optimization with batches of size 1000 every 1000 events.

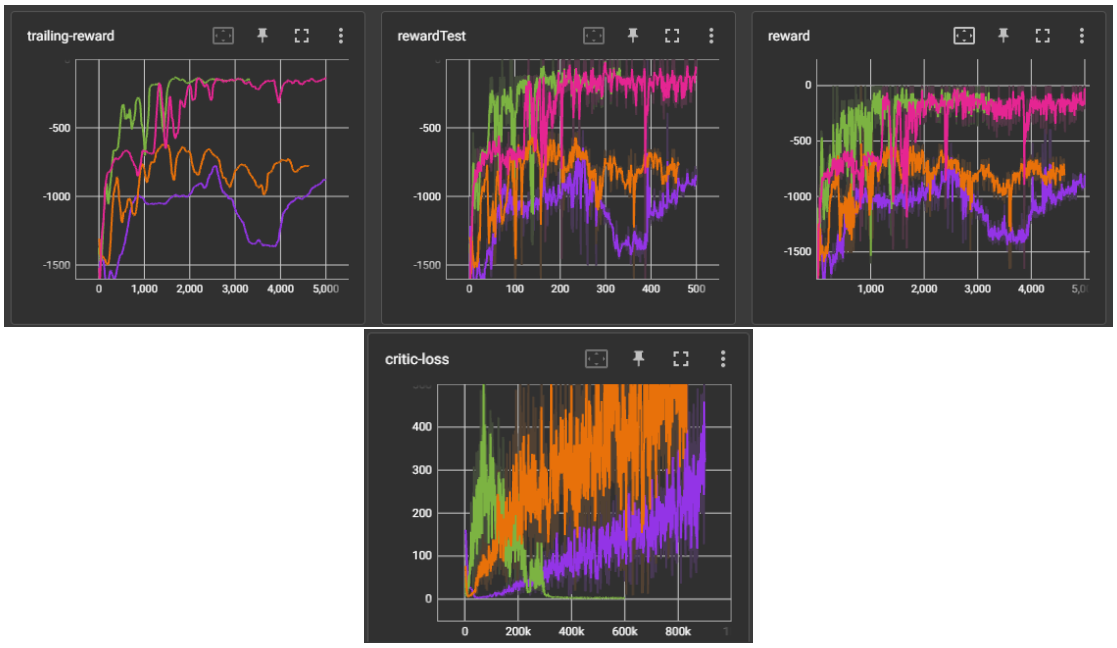


FIGURE 5.1: Comparison of the optimization frequency
(a) Pink (b) Green (c) Orange (d) Purple

Optimization frequency Experiments (a) and (b) show clearly the importance of having a sufficiently high frequency of optimisation steps when face to the others. They both converge quickly to the optimal value -200, whereas (c) and (d) do not and for the reason highlighted by the cost function plot, they don't have sufficient and frequent enough optimization step for the critic to converge. If we look at (c) vs (d), this shows that having an inadequate optimization frequency, and yet a lot of optimization steps, the critic diverges more clearly, and thus increasing the number of steps is not a solution to mitigate the insufficiency of frequency, even if it seems to help reward-wise.

Batch-size Looking at (b) vs (a), we see clearly that a bigger batch size ensures faster convergence, but also greater stability beyond convergence. Yet (a) still manage to reach optimal performance.

Conclusion: It is very important to tune the frequency of optimization to around a few dozen events. Larger batch size help reaching faster and more stable convergence.

Here are the retained hyper-parameters:

freqOptim	maxLengthTrain	gamma	temperature	optim steps	batch size	hidden dim	soft tau	mem size	learning rate
100	300	0.99	0.001	20	1000	64	0.005	50k	1e-3

TABLE 5.2: SAC Hyper-parameters

5.2.2 Temperature

We try to optimize the policy toward high positive Q values, but we added an entropy regularization term weighted by the coefficient "temperature". The greater the coefficient, the more we regularize. If we increase the temperature, we incite the networks to explore more. So we can expect a slower convergence.

We use hyper-parameters for experiment (a) 5.2 with a $Temperature = 0.001$ and for experiment (b) we use $Temperature = 0.1$.

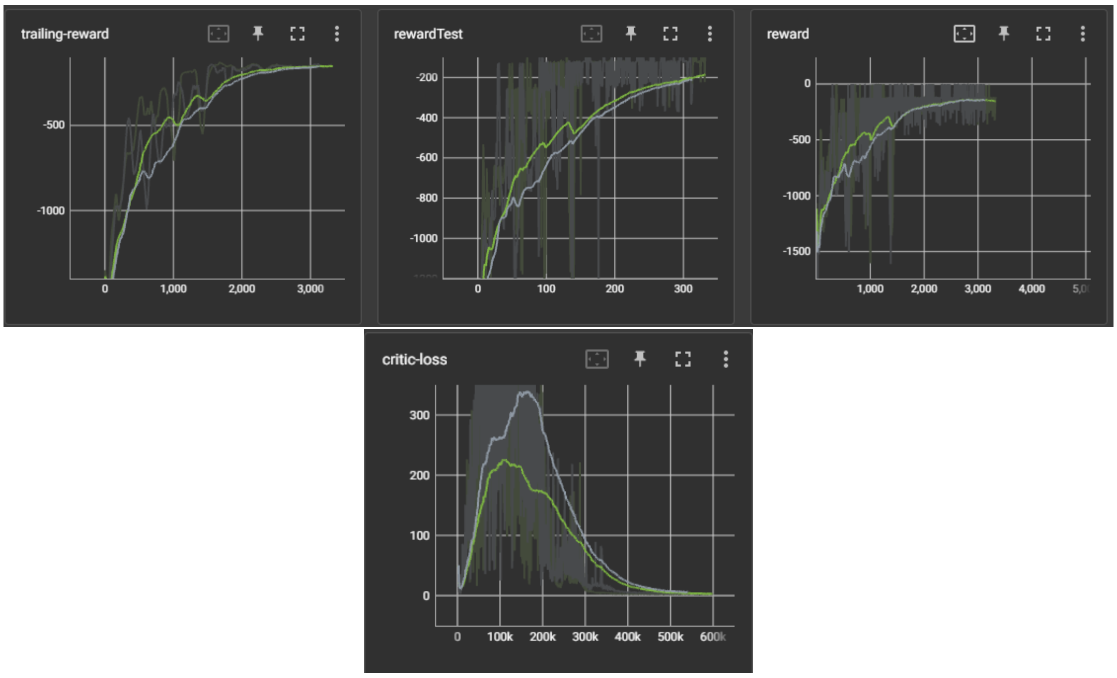


FIGURE 5.2: Effet of the temperature coefficient
(a) Green (b) Grey

Fig 5.2 indicates that the critic network has indeed a harder time to converge (b), but results remain consistent with (a), probably because the effects of exploration on this map are mitigated.

5.2.3 Soft-update

The soft update is slightly different for SAC than for DDPG because we do not use a policy target, but two critic target to avoid the over-estimation bias in the expected target. This was first introduced in TD3. Therefore we will study once again the impact of this hyper-parameter, especially for the critic convergence.

We use hyper-parameters for experiment (a) 5.2 and for experiment (b) we change $\tau = 0.3$. We can expect the critic to learn in a very noisy way. Looking at below learning curve, we confirm that it is beneficial to update target parameters less rashly for greater stability.

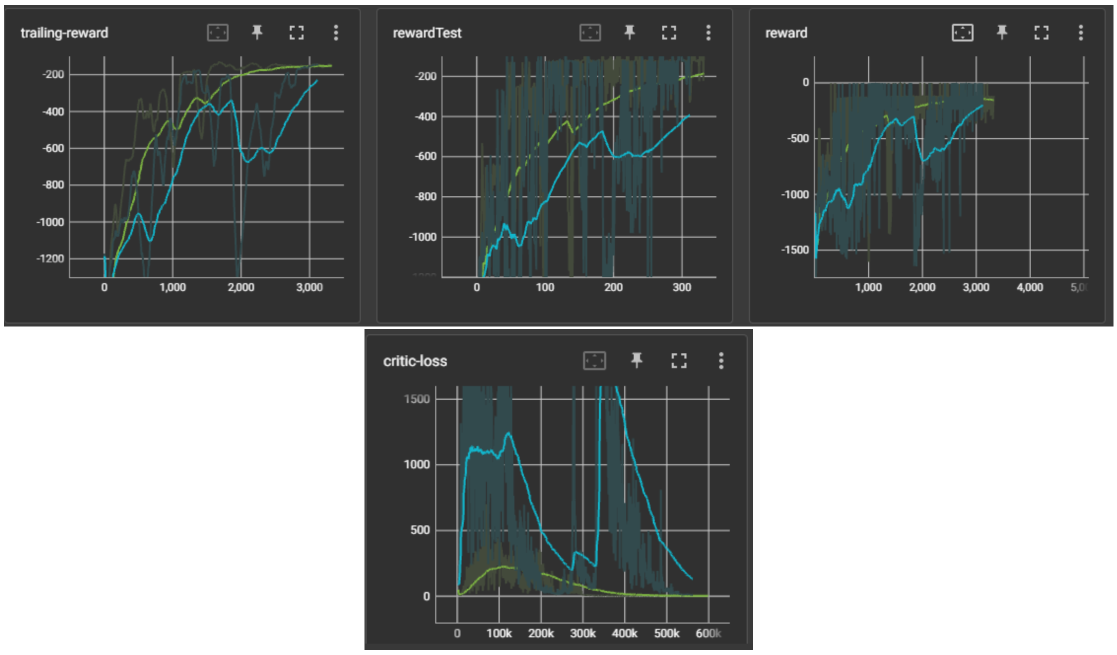


FIGURE 5.3: Effects of soft update for SAC
(a) Green (b) Grey

5.2.4 Automatic entropy

In this section, we compare the behavior of the algorithm using a learned entropy regularization term or by setting it.

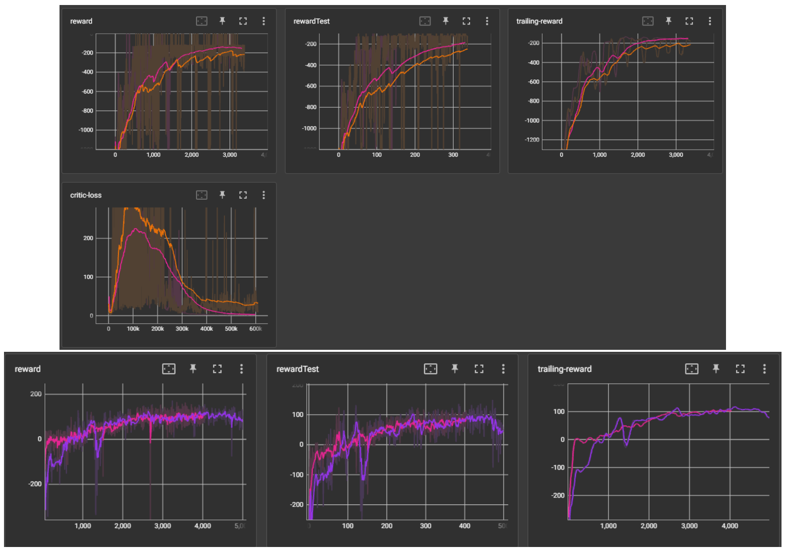


FIGURE 5.4: Automatic entropy regularization term vs set
(a) Pendulum map (b) Lunar Lander Map

For both maps, the above curve corresponds to the regularization term manually set. It was set at $temperature = 0.002$. And as previously stated in the Introduction, we use for the learned approach a target entropy of $-dim(ActionSpace)$.

We notice equivalent performance for both approach, with the indubitable advantage of not having to choose any hyper parameters for the learned approach. After experimenting with all these Reinforcement Learning algorithm on various maps, this is a substantial relief. Hyper parameters are a pain...

5.3 Lunar Lander

We use the set of hyper-parameters 5.2 but with a batch size of 128 instead of 1000, because training time is lower for equivalent performance.

Do not that we use a network with 2 hidden layers of 64 hidden dimension for SAC.

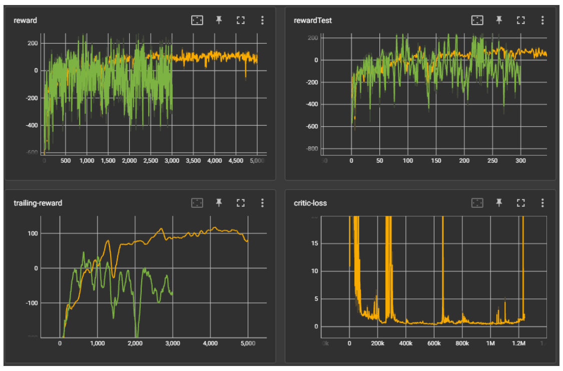


FIGURE 5.5: Comparison of SAC (yellow) vs DDPG (green)

1. SAC converges to a mean reward of 120 shown by the trailing reward wheras DDPG struggles maintaining a mean positive reward.
2. SAC converges with much more confidence/stability than DDPG. DDPG learning is much noisier around its mean reward.
3. Nonetheless, SAC never reaches as good rewards as DDPG does which hits reward of 200 from time to time.

Conclusion: SAC seems more powerful than DDPG on Lunar Lander, reaching a mean reward of 120, and maintaining this performance over time. SAC lands correctly the Rocket every single time, whereas DDPG can do a perfect land and the following episode crash it ridiculously.

5.4 Mountain Car

We use the set of hyper-parameters 5.2 but with a batch size of 128 instead of 1000, because training time is lower for equivalent performance.

Using SAC as presented in the paper I never managed to reach good performance and the car was stuck in a local optimum. I looked at pytorch forums, where someone explained that SAC can't converge on MountainCar except if you add a similar noise

(Orsntein-Uhlenbeck). So I implemented the OU-noise on SAC for this map, and use the same standard deviation as DDPG.

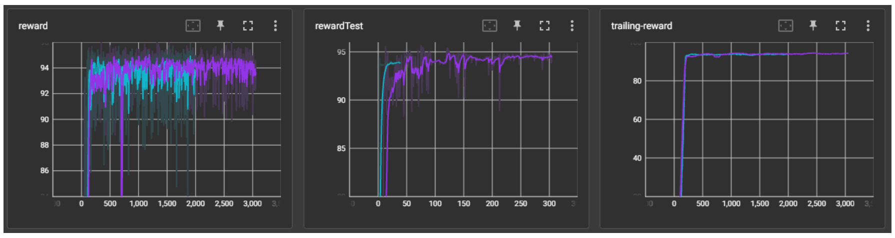


FIGURE 5.6: Comparison of SAC (purple) vs DDPG (blue)

On this map, we notice similar performance for both algorithm. It is still noticeable that SAC converges with much more confidence, and that DDPG learning tends to be noisy.

5.5 Pendulum

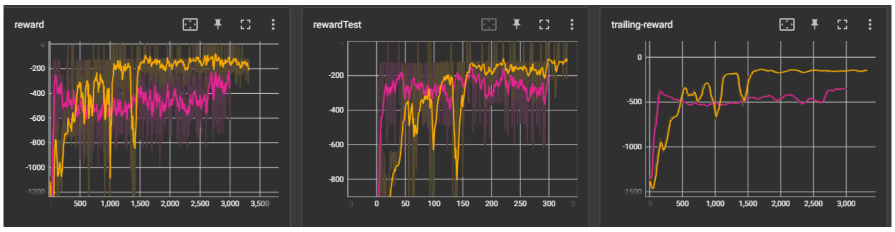


FIGURE 5.7: Comparison of SAC (yellow) vs DDPG (Pink)

We come to the same conclusions. We can add that DDPG converges faster, but SAC converges with much more confidence and to a higher plateau with a mean reward of -120 versus -400 for DDPG.