# RL mini-project : Mountain Car environment

Authors : Harfouch Rayan & Tara[1] Fjellman

Spring 2024

## 1 Introduction

To be written after the rest of the report is done.

## 2 Environment

The Mountain Car environment is a classic reinforcement learning problem. The agent is placed in a valley between two hills and has to learn to reach the top of the right hill. The agent can only apply a force of -1, 0 or 1 to the car. The car has a maximum speed of 0.07 and a maximum position of 0.6. The agent receives a reward of -1 at each time step, and a reward of 0 when it reaches the top of the hill. The episode ends when the agent reaches the top of the hill or after 200 steps. The state space is continuous and has two dimensions, the position and the velocity of the car. The action space is discrete and has three dimensions, the force applied to the car. The environment is implemented in the OpenAI gym library.

## 3 Random agent

To analyse the performance of the agent, one can analyse the duration of the episodes as a function of time. Indeed, if the agent learns the task, the duration of the episodes should decrease in time. This because at the start the agent doesn't know the task, and so the episode is truncated at 'full_ep_len=200', while as it learns it, it should manage to finish the task before reaching the step limit. Running the the agent for 100 episodes, we obtain fig ... . Looking at it, it is clear that the agent does not learn the task, as all the durations coincide with the truncation time. Actually the agent just oscillates around the minimum. This is due to the sparsity of the reward.

## 4 DQN

### 4.1 Vanilla version

Running 1000 episodes and computing their duration along with the cumulated reward per episode, we get fig (maybe not relevant)... . Looking at it we see that the behaviour is the same as for the random agent, i.e. the agent does not learn the task. This is due again to the sparsity of the reward.

### 4.2 Training metrics

To analyse further the training behaviour of our agents we can look at the loss and reward per episode. We can moreover look at the cumulative (cumulated) reward and cumulative successes over the episodes.

### 4.3 Heuristic reward

#### 4.3.1 Heuristic reward function

To alleviate the sparsity of the reward, we introduce a heuristic reward function which can be used as auxiliary reward for the DQN agent. Intuitively, we want this function to

[1]My administrative name is Tobia, so you will find me under that name in EPFL related databases.

make the agent learn it should reach the top of the hill, which is placed on the right of the environnement. We therefore decide to take a heuristic function that assigns a reward if the agent is on the right of its (average) starting point. To make sure the agent tries and reach higher and higher on the hill, we decide to make this function be a growing one with the position. A simple and versatile proposal is to take

$$f(x) = A\mathbb{I}[x > \bar{x}\_0](x - \langle x_0 \rangle)^n$$

for $A > 0$ some constant and a given $n \in \mathbb{N}^\star$. In practice we most often take $n = 3$ and $A = 0.1$. To make extra sure that for all values of the scale factor $A$, the agent does not get tempted to stay and collect the heuristic reward, we decide to offset the function by its maximal value (i.e. $A$). This way, the heuristic reward is always negative, and the agent is encouraged to reach the top of the hill.

#### 4.3.2 Auxiliary reward scaling

For the auxiliary reward to be useful, it should intuitively not be too small. Otherwise the situation would be almost the same as when no auxiliary reward is granted. As for the upper bound, things are more subtle. If the auxiliary reward is too large, it will dominate the environment reward. For the specific choice of heuristic reward, this should not impact the learning of the task, as by itself it would already lead the agent to the reward. However, it if one choses the reward unwisely, we can imagine that the agent could learn to stay on the right of the environment to collect reward rather than finishing the episode.

This intuition is confirmed by the results reported in fig ... .

## 5 RND reward

### 5.1 Normalisation

As recommended by the hint, we standardise the input and output of the RND network. The input standardisation is done since we want to output to vary as a function of how different the observed state is compared to the previously visited ones. Indeed, subtracting the typical state (i.e. the mean one) provides a good idea of how different the new input state is. The normalising by the typical distance from the mean state (i.e. the standard deviation) is done to make sure the input is of order 1, which is a good practice for neural networks to avoid instabilities.

The standardisation of the output is instead done to make sure that we can control the characteristics of the RND reward attributed to the states. Indeed, by subtracting the mean we make sure that typical states are penalised while new ones are rewarded (so we promote exploration); while normalisation make it possible to tweak the the order of magnitude of the reward (as normalised variables are of order 1). The clamping is done to avoid problems with the non-linearity of the neural network.

### 5.2 Reward-factor scale

Mostly the same argument applies as for the heuristic reward. This time however, if the factor is taken too large, we expect the agent to prefer exploring instead of ending the episode by reaching the top of the hill. Indeed, when the agent will have reached the top of the hill a few times, the states that will have allowed it to do so will have a low RND reward. This is because the RND network will have seen them many times, and so the prediction error will be low. * right size ? * crude scale : over an episode the rewards should be comparable ?? * handling gradients * they are set to zero after each update * we use target network to only include part of the loss in the differenciation * dataloaders are usually used for training (to shuffle data and create batches) * can't add samples to it, and should not re-define it from scratch * we just keep a list instead, sampling with random indices * we only start training once the replay buffer is full * implementation of target net was done with help of chat gpt (command to import the weights from another net)

## 6 Dyna

## 7 Conclusion

## 8 Appendix