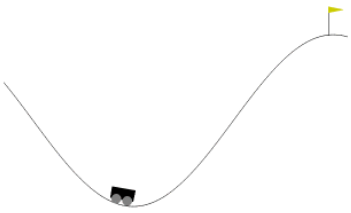
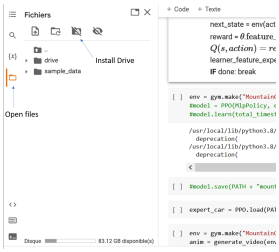
Lab Inverse Reinforcement Learning

1 Mountain Car

The Mountain Car MDP is a deterministic MDP that consists of a car placed stochastically at the bottom of a sinusoidal valley, with the only possible actions being the accelerations that can be applied to the car in either direction. The goal of the MDP is to strategically accelerate the car to reach the goal state on top of the right hill. However the car does not have enough energy to climb the hill in one go, therefore it has to make us of gravity by going left first and then use momentum to climb the hill in the right.



We will try to solve the task without making use of the environment's reward. It will only be used for evaluation. As an expert we will train an agent using stable baselines. Alternatively you can make use of a pretrained model which weights are downloadable from moodle. For this you will first need to install google drive on you colab environment.



The objective of the lab is to learn a reward function that would allow the agent to solve the task.

2 MaxEnt IRL

Inverse Reinforcement Learning is a hard problem first because it is ill posed. This means that many reward functions can explain the expert behaviour. Moreover, even degenerate solutions like the null reward are always solutions of the problem. This makes it difficult to know which reward to look for and which one could generalise to new states or environment dynamics that the expert has never encountered.

The maximum entropy solution helps to solve that problem. Instead of looking directly for a reward function, one could try to optimise an agent so it can explore the same distribution of trajectories as the expert. This would mean that either following the agent or the expert, one would always observe the same trajectories. Therefore the Maximum Entropy algorithm is based on minimizing the difference between emprical state distributions observed and expert state distributions. this way, we, according to the maximum entropy principal, one can converge towards a reward that is not degenerate and that introduce the least bias possible.

The distribution (from either agent or expert) can be represented as:

$$\sum_{\text{trajectory}\tau} P(\tau_i) f_{\tau_i}$$

Where P is the probability of trajectory τ and f the features of a state.

What are these state features? It can be about anything that helps distinguish states from each others. **Feature vector**: We can represent any state by a feature vector f. In the context of this lab the feature vector is simply a one hot encoded vector. This means that if we have n different possible states. The feature vector of state 1 will be a vector of length n with a 1 in the first entry and 0 everywhere else. **Feature matrix**: It is the matrix that compiles the feature vectors of all states. In our case it is an identity matrix of size n. **Feature count**: Considering a trajectory τ, the feature count of τ is defined as the sum of feature vectors of all states in the trajectory: $f_\tau = \sum_{s \in \tau} f(s)$

The reward function (parameterized by θ) can then be considered as a linear combination of the features:

$$\begin{aligned} R_\theta(\tau) &= \sum_{s \in \tau} \theta^T f_s \\ &= \theta^T f_\tau \end{aligned} \tag{1}$$

The probability that a trajectory appears in the demonstrations should be proportional to the reward it generates:

$$p(\tau) = \frac{\exp(R_\theta(\tau))}{Z}$$

With $Z = \sum_\tau \exp(R_\theta(\tau))$

The objective is therefore to maximize $p(\tau)$ the probability to choose trajectories that earn highest rewards:

$$\theta^* = \arg\max_\theta L(\theta) = \arg\max_\theta \sum_{\text{examples}} \log P(\bar{\zeta}|\theta, T)$$

TODO: Derive the gradient of the loss to make easily computable.

Derivating this loss, you should end with the following gradient:

$$\nabla_\theta L(\theta) = \hat{f}_\tau - \sum_s p(s|\theta) f_\tau$$

Therefore the gradient is easily obtained by substracting the agent feature count from the expert's.