University of Liège
School of Engineering

Reinforcement Learning in a Continuous Domain

INFO8003-1: Optimal decision making for complex problems

Julien Gustin, Joachim Houyon

March 18, 2022

1 Implementation of the domain

1.1 Policy selection

For our experiment, we have decided to implement an `always accelerate` policy. Formally, it gives:

$$\mu(p, s) = 4 \ \ \forall (p, s) \in X$$

1.2 Results of the policy

The image below represents a simulated trajectory of 10 one-steps in the domain. The initial state is $(p_0, 0)$, where p_0 $U([-0.1, 0.1])$. Each line represents a one-step transition $((p_t, s_t), u_t, r_t, (p_{t+1}, s_{t+1}))$ at time t.

```
[[(p=-0.04, s=0.00) 4 0 (p=-0.05, s=-0.27)]
 [(p=-0.05, s=-0.27) 4 0 (p=-0.09, s=-0.55)]
 [(p=-0.09, s=-0.55) 4 0 (p=-0.16, s=-0.81)]
 [(p=-0.16, s=-0.81) 4 0 (p=-0.25, s=-1.02)]
 [(p=-0.25, s=-1.02) 4 0 (p=-0.36, s=-1.08)]
 [(p=-0.36, s=-1.08) 4 0 (p=-0.46, s=-0.91)]
 [(p=-0.46, s=-0.91) 4 0 (p=-0.53, s=-0.51)]
 [(p=-0.53, s=-0.51) 4 0 (p=-0.56, s=-0.02)]
 [(p=-0.56, s=-0.02) 4 0 (p=-0.54, s=0.48)]
 [(p=-0.54, s=0.48) 4 0 (p=-0.47, s=0.89)]]
```

Figure 1. Trajectory of 10 one-steps in the domain

2 Expected return of a policy in continuous domain

To estimates the expected return of a policy for the car on the hill problem rather than computing it for one single state, we exploit the Monte-Carlo principle. We used a set of K[1] i.i.d. initial states where $p_0 \sim U([-0.1, 0.1])$ and $s_0 = 0$, and computed the the average expected return of the stationary policy. We define the set of initial state by X^i.

$$J_\infty^\mu = \frac{\sum_{x \in X^i} J_\infty^\mu(x)}{K} \tag{1}$$

$J_\infty^\mu(x) \ \ \forall x \in X^i$ had been estimated by simulating the policy starting by initial state x for N steps. We will name this approximation by J_N^μ.

$$J_N^\mu(x) = \sum_{t=0}^{N-1} \lambda^t r(x_t, \mu(x_t)) | x_0 = x$$

[1] 50 in our case

[2] We can find a lower bound of N by fixing the error of approximation ϵ, where $\epsilon = ||J_N^\mu - J_\infty^\mu||_\infty$.

$$\begin{aligned} \epsilon &\leq \frac{\gamma^N}{1-\gamma} Br \\ \frac{\epsilon(1-\gamma)}{Br} &\leq \gamma^N \\ \log_\gamma \left(\frac{\epsilon(1-\gamma)}{Br} \right) &\leq N \end{aligned}$$

Given $\gamma = 0.95$, $Br = 1$ and $\epsilon = 10^{-3}$ we have:

$$N = \left\lceil \log_\gamma \left(\frac{\epsilon(1-\gamma)}{B_r} \right) \right\rceil = 194. \tag{2}$$

The result for the *always accelerate* policy can be seen in **Fig.2** and is quite interesting and expected. Indeed if the agent always try to always accelerate it will keep oscillating and never reach a terminal state.

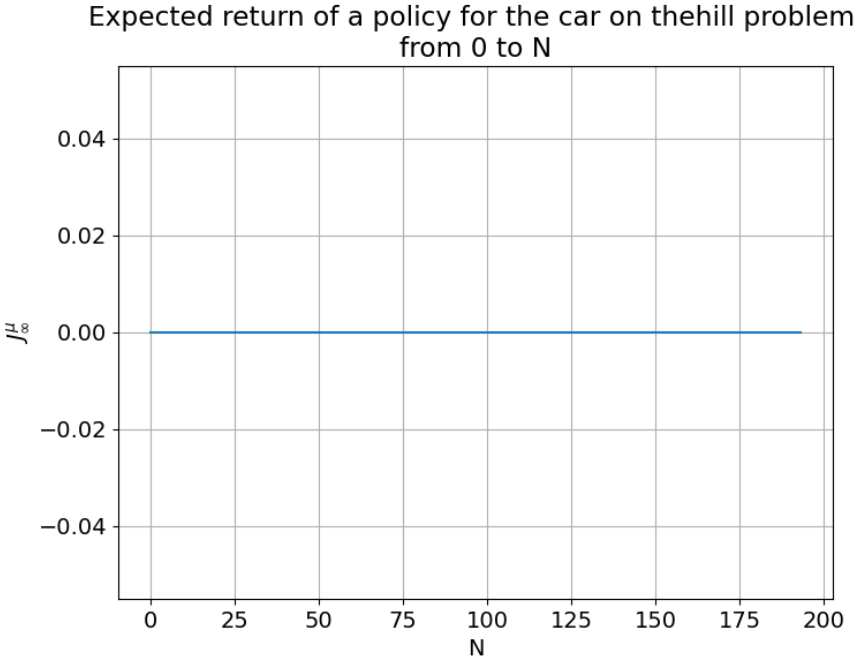


Figure 2. Expected return of *Always accelerate* policy, over N

[2]Note that car on the hill problem is deterministic.

3 Visualization

Given a trajectory and the policy defined in the section 1, we save, at each one-step transition, the frame associated to it. The result gives us the video of our simulation on the domain. Since the gap between two time-steps is 0.1 seconds, the video is recorded at a rate of $\frac{1}{0.1} = 10$ frames per second.

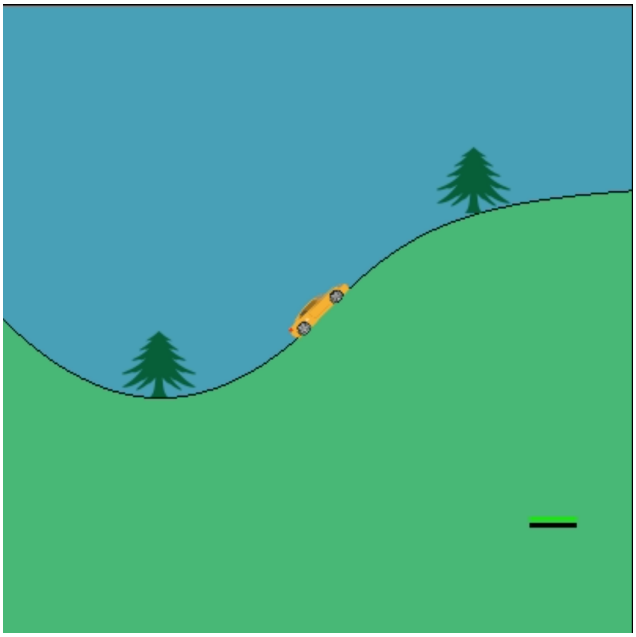


Figure 3. A frame of a simulation on the domain

In the *videos/* directory we have put 3 videos,

1. Using a random policy
2. Using a only accelerate policy
3. Using fitted Q learning with Totally Randomized Trees

4 Fitted-Q-Iteration

The fitted Q iteration computes from h_t (one step transitions) the functions $\hat{Q}_1, \hat{Q}_2, ..., \hat{Q}_N$ approximation of $Q_1, Q_2, ..., Q_N$. The idea is at each iteration one build a training set using h_t and the approximation of the precedent Q iteration. More formally

- $N = 1$ the algorithm determine $\hat{Q}_1$ of $Q_1(x,u) = E_{w\sim P_w(.|x,u)}[r(x,u,x)]$ on the training set
$$TS = \{((x_k, u_k), r_k)\}_{k=0}^{t-1}$$
- $N > 1$ the algorithm outputs a model $\hat{Q}_N$ of $Q_N(x,u) = E_{w\sim P_w(|x,u)}[r(x,u,x) + \gamma max_{u'\in U}(x_{k+1}, u')]$ by using

$$TS = \{((x_k, u_k), r_k + max_{u'\in U}\hat{Q}_{N_1}(x_{k+1}, u'))\}_{k=0}^{t-1}$$

4.1 One step transition generator

To build our dataset we have chosen two one-step transitions generator

1. **Random generator**: The idea of this generator is to perform M simulation of length K until having D[3] one step transition. Using M random initial value where $p_0 \sim U([-0.1, 0.1])$ and $s_0 = 0$.
2. **Discretized exhaustive list**

   Since we know that the domain is deterministic, we can interact with the domain with every state-action pair $((p,s),u)$, giving us all the possible outcomes. However, since the state space is continuous, we need to discretize:

   Let $(p'_1, p'_1, ..., p'_n)$ and $(s'_1, s'_1, ..., s'_m)$ respectively be each possible value of p and s (where $(p,s) \in X$) such that n and m are big enough in order to get a good approximation of the continuous state space. Every states (p'_i, s'_j), $i \in [1,n], j \in [1,m]$, represent all the possibles starting points for a one-step transition.

   This approach will generate $n * m * 2$ one-step transitions.

In order to compare them in the best manner we decided to generate exactly the same number of one step transition meaning $n * m * 2$.

4.2 Stopping rules

We had to propose two stopping rules, for that point we looked at *Tree-Based Batch Mode Reinforcement Learning*[4] and implemented the two stopping conditions they propose.

1. Define a priori a maximum number of iterations. By using the error bound on the sub-optimally with μ^*_N, such that the number of iterations is given by the following equation

[3] $D = n * m * 2$

[4] https://www.jmlr.org/papers/volume6/ernst05a/ernst05a.pdf

$$||J^{\mu^*_N} - J^{\mu^*}||_\infty \leq 2\frac{\gamma^N Br}{(1-\gamma)^2}$$

2. The second solution is to stop the iterative process when the infinity norm between $\hat{Q}_N$ and $\hat{Q}_{N-1}$ drop below a certain value.

Finally we compared three supervised

4.3 Supervised learning techniques

1. Linear regression
2. Extremely Randomized Trees
3. Neural network. For the architecture of the neural network we look at this paper [1] that perform similar experiment the mountain car environment. The neural network has 2 hidden layers of 5 neurons each, all equiped with relu activation function. And the weights of the network are randomly initialized within [-0.5, 0.5]. This neural network had been trained with Adam for 10 epoch at each iteration.

4.4 Results

It is worth noticing that the extracted policy using linear regression is not optimal: this is actually the same policy we defined in the first section.

Furthermore, we can see a difference between the random one-steps and the discretized exhaustive list trajectories. Indeed, by checking figures 9a and 9b, we see that the trajectory sampled from random one-steps suggests that the car should accelerate near the end of the hill when having a low velocity, which is not correct because the car will not have enough velocity to reach its goal, also using the discretized exhaustive list, there is less uncertainty in some states. In overall, listing all the possibles state-action pairs is better for the learning algorithm in order to reach the optimal policy.

Regarding the stopping condition, the difference between those two is barely noticeable when using the tree algorithm. However, during the process, we noticed that the distance condition does not converge but oscillates around a value and never reaches the desired threshold. Therefore, we would rather consider using the theoretical bound, which ensures that the algorithm will stop and has better chances to be closer to the real Q table.

Finally concerning the models, Linear regression and NN perform really poorly compared to the Extremely Randomized Trees. Indeed, using Linear regression, we make the assumption of linear relationship between the state and the Q-function, which does not seem to be the case. About the neural network, it looks like it doesn't have enough representation capacities, it may be due by the lack of neurons at each layer or the number of layers, or even how we trained it. In that environment using Totally Randomized Tree, which is known and tested that the policy derived from it is very good, gives the best results by looking at the expected returns **Table3**. We can conclude that Linear regression and NN have failed to converge towards an optimal policy.

	Bound stopping condition	Distance stopping condition
Random one-steps generator	0.0	0.0
Exhaustive state generator	0.0	0.312

Table 1. Estimations of the expected returns of $\hat{\mu}_N^*$, where $\hat{Q}_N$ had been estimated using Linear Regression

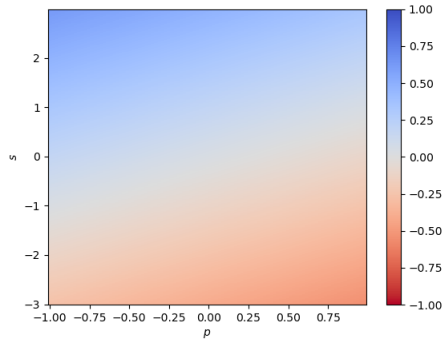
	Bound stopping condition	Distance stopping condition
Random one-steps generator	0.417	0.418
Exhaustive state generator	0.418	0.418

Table 2. Estimations of the expected returns of $\hat{\mu}_N^*$, where $\hat{Q}_N$ had been estimated using Extremely Randomized Trees

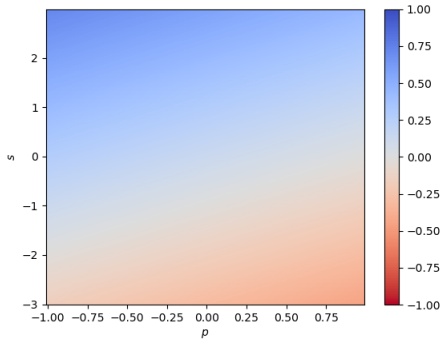
	Bound stopping condition	Distance stopping condition
Random one-steps generator	0.104	0.0
Exhaustive state generator	0.0	0.0

Table 3. Estimations of the expected returns of $\hat{\mu}_N^*$, where $\hat{Q}_N$ had been estimated using a Neural Network

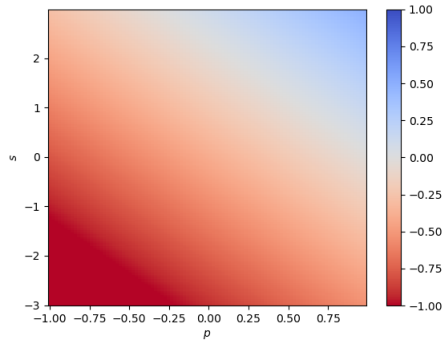
The following figures had been computed with ≈ 80000 trajectories.



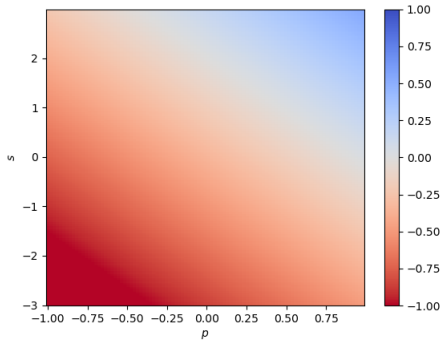
(a) random one-steps, bound, u=-4



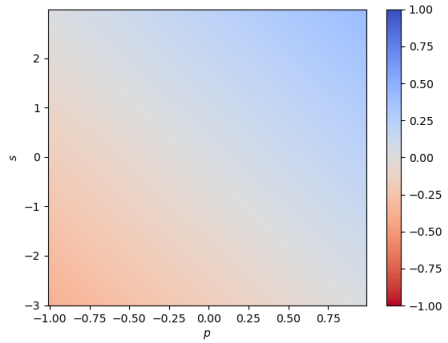
(b) random one-steps, bound, u=4



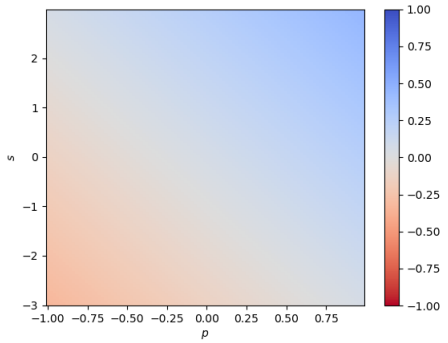
(c) discretized exhaustive list, bound, u=-4



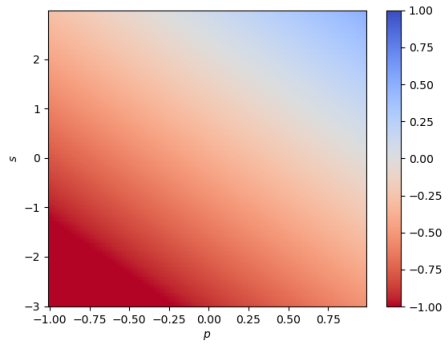
(d) discretized exhaustive list, bound, u=4



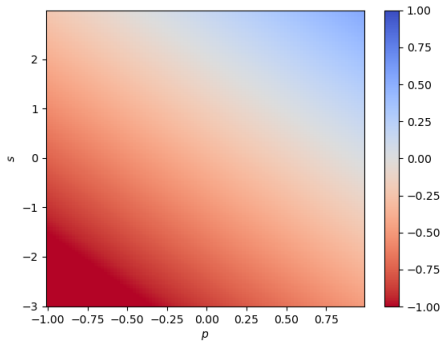
(e) random one-steps, distance, u=-4



(f) random one-steps, distance, u=4

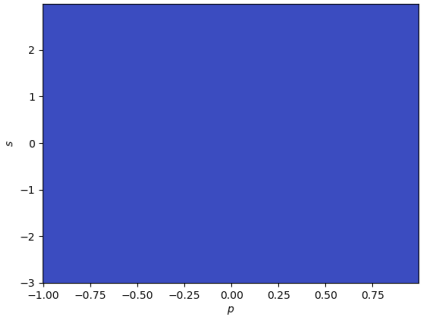


(g) discretized exhaustive list, bound, u=-4

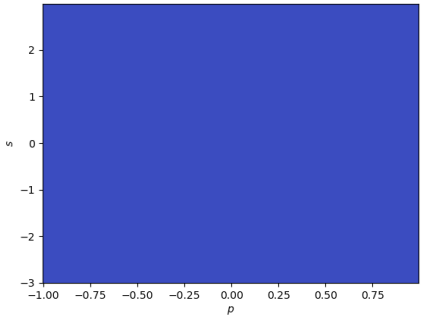


(h) discretized exhaustive list, bound, u=4

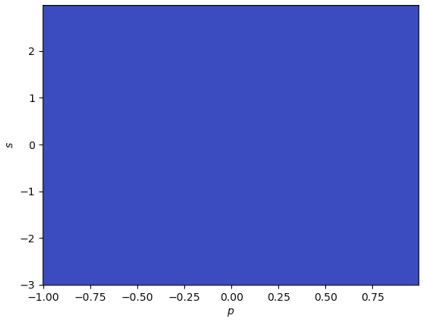
Figure 4. $\hat{Q}_N$ in a 2D grid (red for action u=-4 and blue for action u=4) using Linear Regression



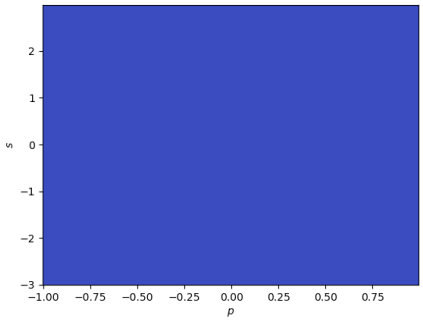
(a) Random one-steps, bound



(b) Random one-steps, distance

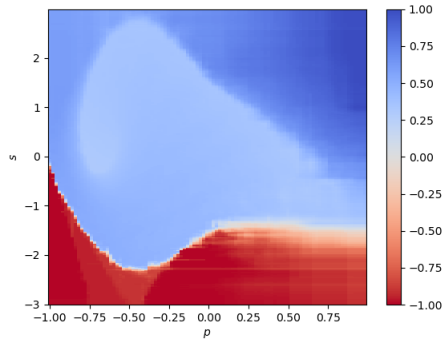


(c) Discretized exhaustive list, bound

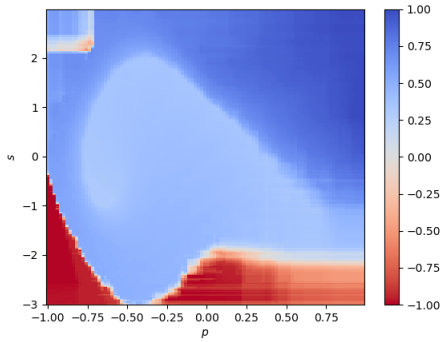


(d) Discretized exhaustive list, distance

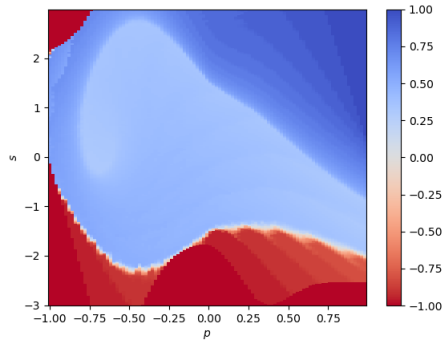
Figure 5. Extracted optimal policy $\hat{\mu_*}_N$ from $\hat{Q}_N$ in a 2D grid (red for action u=-4 and blue for action u=4) using Linear Regression



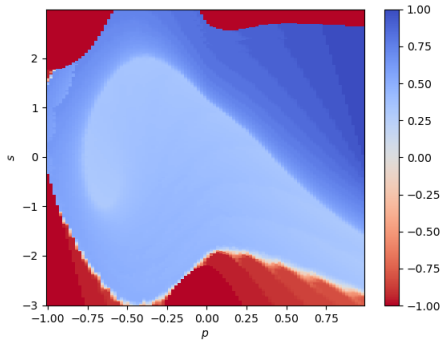
(a) random one-steps, bound, u=-4



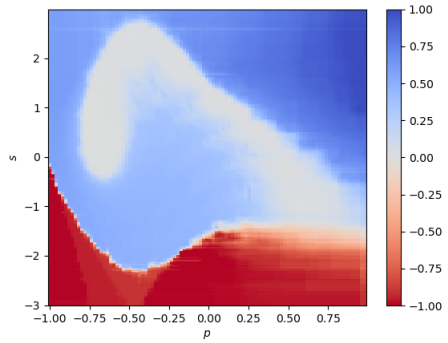
(b) random one-steps, bound, u=4



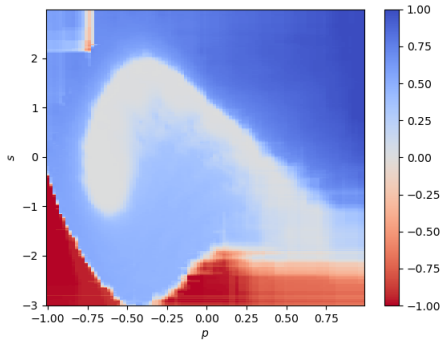
(c) discretized exhaustive list, bound, u=-4



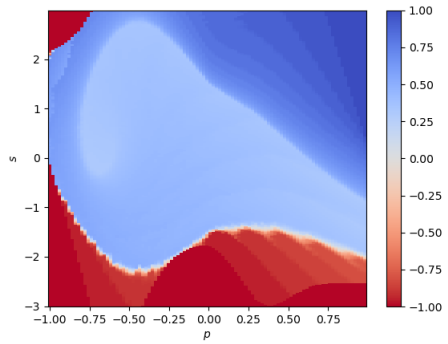
(d) discretized exhaustive list, bound, u=4



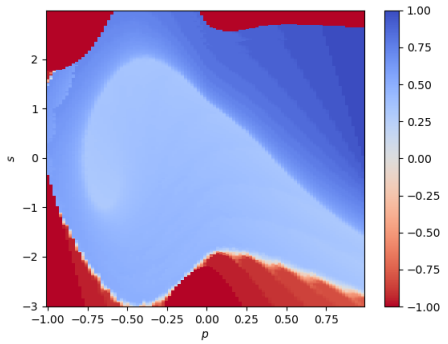
(e) random one-steps, distance, u=-4



(f) random one-steps, distance, u=4

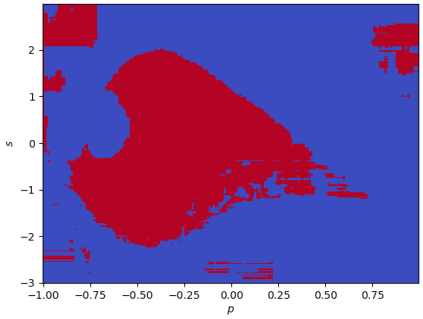


(g) discretized exhaustive list, bound, u=-4

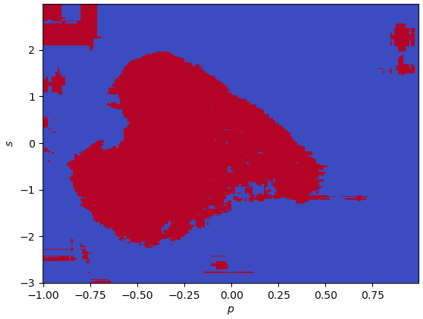


(h) discretized exhaustive list, bound, u=4

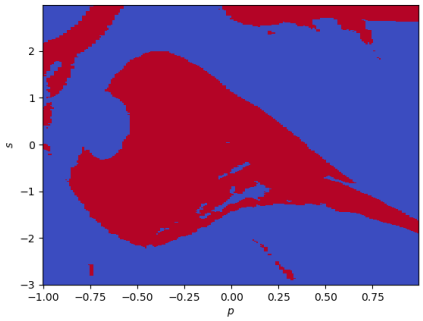
Figure 6. $\hat{Q}_N$ in a 2D grid (red for action u=-4 and blue for action u=4) using Extremely Randomized Trees



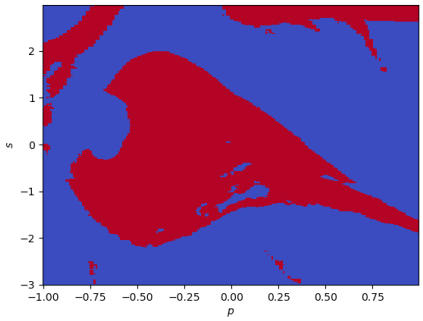
(a) Random one-steps, bound



(b) Random one-steps, distance

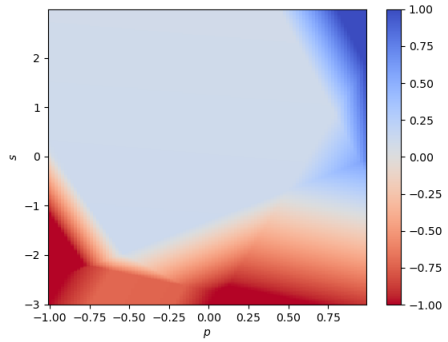


(c) Discretized exhaustive list, bound

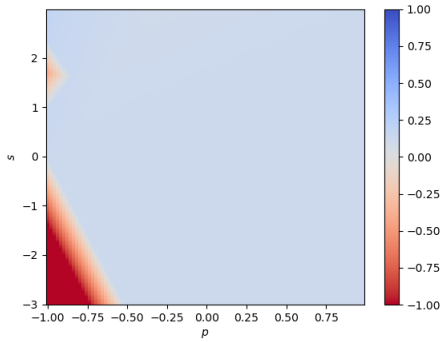


(d) Discretized exhaustive list, distance

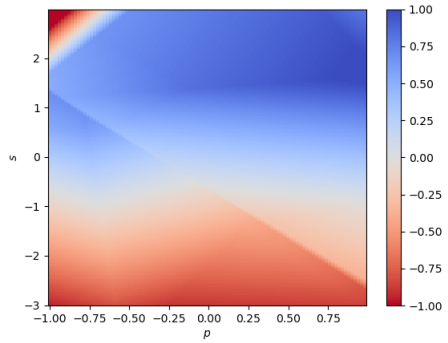
Figure 7. Extracted optimal policy $\hat{\mu_*}_N$ from $\hat{Q}_N$ in a 2D grid (red for action u=-4 and blue for action u=4) using Extremely Randomized Trees



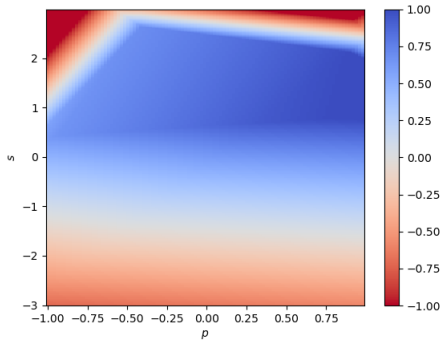
(a) random one-steps, bound, u=-4



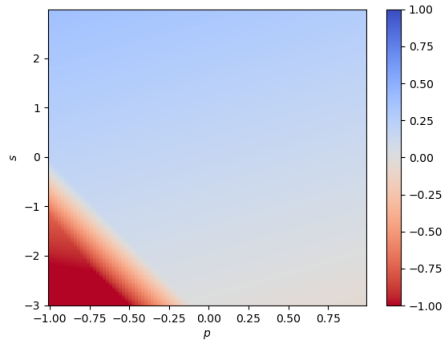
(b) random one-steps, bound, u=4



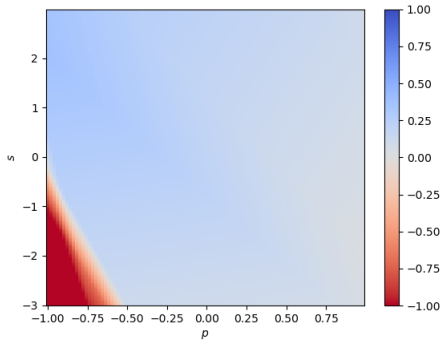
(c) discretized exhaustive list, bound, u=-4



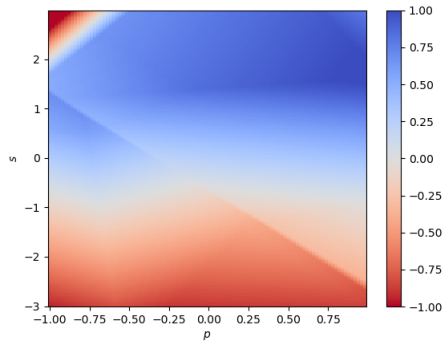
(d) discretized exhaustive list, bound, u=4



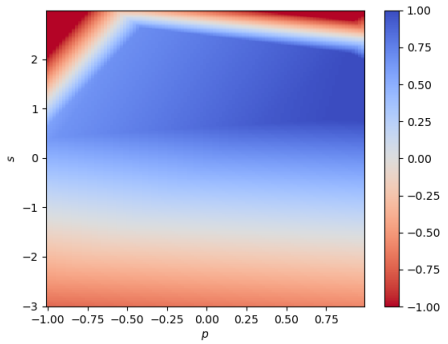
(e) random one-steps, distance, u=-4



(f) random one-steps, distance, u=4

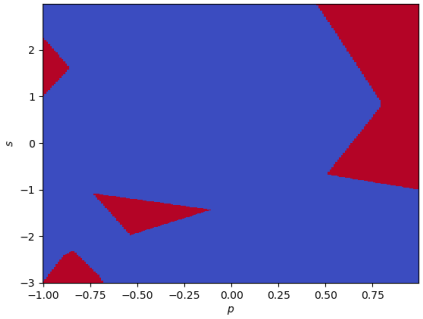


(g) discretized exhaustive list, bound, u=-4

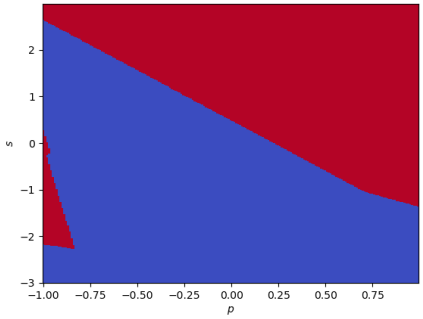


(h) discretized exhaustive list, bound, u=4

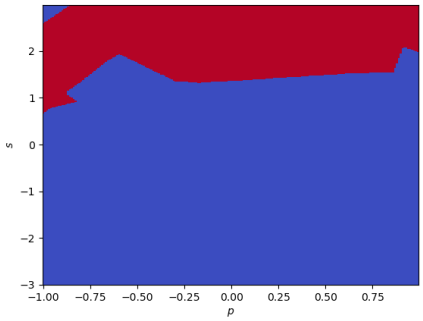
Figure 8. $\hat{Q}_N$ in a 2D grid (red for action u=-4 and blue for action u=4) using a Neural Network



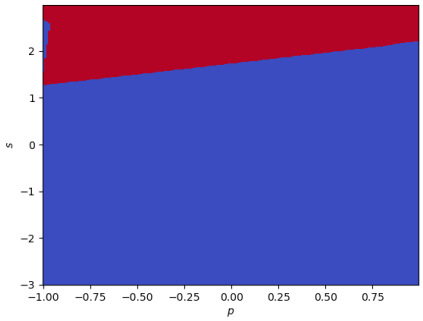
(a) Random one-steps, bound



(b) Random one-steps, distance



(c) Discretized exhaustive list, bound



(d) Discretized exhaustive list, distance

Figure 9. Extracted optimal policy $\hat{\mu}_{*N}$ from $\hat{Q}_N$ in a 2D grid (red for action u=-4 and blue for action u=4) using a Neural Network

5 Parametric Q-Learning

The parametric Q-learning algorithm is an extension of the algorithm where a parametric Q-function of the form $\tilde{Q}(x, u, a)$ is used. In our case we use a neural network as the approximation architecture, therefore the parameters a are the weights of that neural network. [2]

The update of $\tilde{Q}(x_t, u_t, a)$ is defined by

$$\delta(x_t, u_t) = r_t + \lambda \max_{u \in U} \tilde{Q}(x_{t+1}, u, a) - \tilde{Q}(x_t, u_t, a)$$

after observing (x_t, u_t, r_t, x_{t+1}) It follow the following change in parameters:

$$a \leftarrow a + \alpha \delta(x_t, u_t) \frac{\partial \tilde{Q}(x_t, u_t, a)}{\partial a}$$

This is directly derived from

$$a \leftarrow a - \alpha \frac{\partial l_a(x_t, u_t, x_{t+1})}{a}$$

where

$$l_a(x_t, u_t, x_{t+1}) = (\hat{Q}(x_t, u_t, a) - r_t - \lambda \max_{u \in U} \hat{Q}(x_{t+1}, u, a))^2$$

$$\frac{\partial l_a(x_t, u_t, x_{t+1})}{\partial a} = -\delta(x_t, u_t) \frac{\partial \hat{Q}(x_t, u_t, a)}{\partial a}$$

The chosen architecture is the same the one defined at 3, we have made this choice in order to compare at best the two approaches (FQI and PQ), which is an architecture inspired by [1].

The following result had been computed using 80000 trajectory sampled using **4.1**, where our neutral network had been trained using batch gradient descent of 32 samples, optimized with Adam for 100 epochs. Unfortunately by lack of time we were not able to all the experiments over different seed and average the result. Therefore, these results should be taken with a grain of salt.

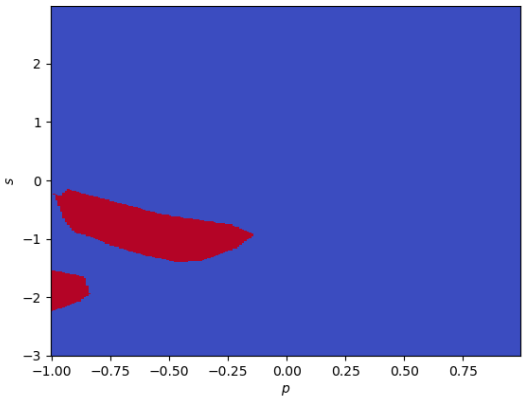


Figure 10. Policy $\hat{\mu}_*$ derived from $\hat{Q}$ using Parametric Q-Learning with a neural network of 2 hidden layer of 5 neurons

And this policy give an expected return of 0.055, in average across 10 different seeds with a standard deviation of 0.107

We have also made the experiment of a wider network using this time 2 hidden layers of 200 neurons,

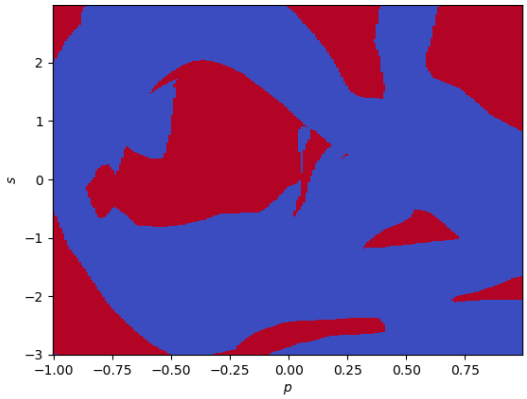


Figure 11. Policy $\hat{\mu}_*$ derived from $\hat{Q}$ using Parametric Q-Learning with a neural network of 2 hidden layer of 200 neurons

This visually give a result closer to the one computed with fitted-Q-iteration with extremely randomized trees, and gives an expected returns of 0.305 in average across 10 different seeds, with a standard deviation of 0.09.

5.1 Protocol

For comparing FQI and parametric Q-learning, we design an experimental protocol where we will be able to respectively see the expected values of our algorithms.

A few remarks:

- The number of epochs is fixed to $5N$, where N is the number of iterations needed to have an error bound of maximum 10^{-2} on the sub-optimality, we fixed to $5N$ because we perform N iterations of the fitted Q iteration with 5 epochs. This allow more fairness on the experiment
- The trajectories will be exactly the same for both algorithms, so that we don't include any bias. Furthermore, the experiments will evaluate both algorithms for the following trajectory sizes: $[5000, 10000, 20000, 50000, 100000, 500000]$
- The experiments can be replayed, a seed is fixed.

Figures 13 represents the expected reward of the policy $\hat{\mu}_*$ with respect to the number of one-step transitions, all sampled using **4.1** with different algorithm.

In overall parametric Q-learning with that architecture gives *bad results*. However, when looking at some videos produce by this policy the derived policy from the neural network looks ok but not enough *complex*, meaning that it knows that in order to reach its objective it need to have some momentum and then accelerate, but at some point it struggle to know if it is in a state that need acceleration to reach the top or need to decelerate to take momentum. Also we had been confronted to some scenario where it decelerate or accelerate too fast and therefore reach a terminal state without even reaching a loosing nor wining position, one way to explain it is that this is likely that the architecture didn't

Figure 12. Comparison of the expected result of the fitted-q iteration (FQI), the parametric Q-learning (PQ) and the online parametric Q-learning with normalization (PQ-Online) across different trajectory length, using the same neural network of 2 hidden layers of 5 neurons

learn that it should not go over a velocity of 3 as it is very unlikely to sample trajectories leading to this kind of states. Therefore, at some cases, the car will take too much momentum in order to reach its goal, leading to a loosing state. On the other hand, the car will eventually reach the winning state with the right velocity.

This being said, we have some ideas for improvement. A first solution in order to address this problem could be the double QLearning algorithm, this would allow to not use the same network to choose the action and evaluate value. A second improvement could be to try another way than random to sample trajectories. And finally change the architecture of our network, indeed, it seems not enough *complex* and does not have enough representation capacities.

To check if the issue comes from the architecture of our neural network or not, we have set the number of neurons in each layer to 200 instead of 5. And this give the following results:

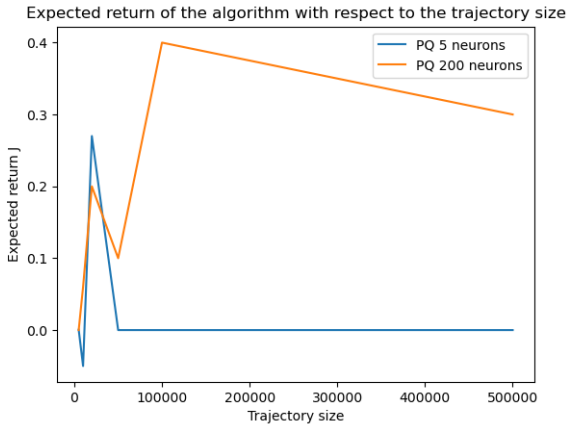


Figure 13. Comparison of parametric Q-learning using 2 hidden layers of 5 neurons against one of 2 hidden layers of 200 neurons

In that experiment it proves that having a wider network gives better result than when using only 5 neurons, which may again comes from the lack of representation capacities of the the neural network with 5 neurons. It could be interesting to try that architecture with fitted-Q iteration, however, we are limited by time.

6 Normalised Parametric Q-Learning

For that part we had to implement the online parametric Q-Learning. One issue was to deal with terminal states, but we solved that problem by restarting to a new random initial state every time we reach a terminal one.

Moreover we had to normalised the update term of the Q-functions, according to [3] normalizing gradient descent allow solving the issue of vanishing gradient near stationary points, which may indeed be an explication of why our neural network seems not to learn.

The update rule is not define by

$$a \leftarrow a + \alpha\delta(x_t, u_t)\frac{\frac{\partial\tilde{Q}(x_t,u_t,a)}{\partial a}}{\left|\left|\frac{\partial\tilde{Q}(x_t,u_t,a)}{\partial a}\right|\right|_2}$$

We use norm 2 because it is what had been advice on [3]. We then performed the protocol define in 5.1 using as trajectory length the same as the other algorithms.

However by looking at **Fig.12** results are quite disappointing. Indeed, it again struggle to converge to a policy that eventually reach a wining state. However these result should be taken with a grain of salt because these experiment had been performed only one time. A better way could be to average the result among X experiment across different seeds.

References

[1] Martin Riedmiller. "Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method". 2005. URL: https://ml.informatik.uni-freiburg.de/former/_media/publications/rieecml05.pdf (pages 5, 13).

[2] Shipra Agrawalr. "Lecture 3: Large-scale Q-learning". 2010. URL: https://ieor8100.github.io/rl/docs/Lecture%5C%203%5C%20-%5C%20Q-learning%5C%20function%5C%20approximation.pdf (page 13).

[3] Jermwatt. "A.3 Normalized Gradient Descent". 2020. URL: https://jermwatt.github.io/machine_learning_refined/notes/3_First_order_methods/3_9_Normalized.html (page 17).