UNIVERSITY OF LIÈGE

MASTER IN ENGINEERING

March 18, 2022

INFO8003-1
Optimal decision making for complex problems

Assignment 2

Maxime Amodei - **s171830**
Cédric Siboyabasore - **s175202**

1 Implementation of the domain

We define the policy μ_h (implemented by the function `policy_heuristic`) as follows.

$$\mu_h(p,s) = \begin{cases} -4 & \text{if} \quad |p| < 0.4 \text{ and } |s| < 0.8 \\ 4 & \text{otherwise} \end{cases}$$

This policy was designed to gain momentum by first going left and using the potential energy to climb the hill in the right direction.
The simulation of this policy, starting at $p = -0.009$, is shown below. We can see that the car make two oscillations because the heuristic-based policy is not well-designed, however the rightmost terminal state is reached and gives a reward of +1. After this state, the reward signal stays at zero, as expected.

```
x_t=(-0.063, +0.000), u=-4.0, r=+0.0 x_t+1=(-0.099, -0.733)
x_t=(-0.099, -0.733), u=-4.0, r=+0.0 x_t+1=(-0.213, -1.586)
x_t=(-0.213, -1.586), u=+4.0, r=+0.0 x_t+1=(-0.384, -1.779)
x_t=(-0.384, -1.779), u=+4.0, r=+0.0 x_t+1=(-0.552, -1.479)
x_t=(-0.552, -1.479), u=+4.0, r=+0.0 x_t+1=(-0.669, -0.832)
x_t=(-0.669, -0.832), u=+4.0, r=+0.0 x_t+1=(-0.718, -0.133)
x_t=(-0.718, -0.133), u=+4.0, r=+0.0 x_t+1=(-0.696, +0.566)
x_t=(-0.696, +0.566), u=+4.0, r=+0.0 x_t+1=(-0.605, +1.254)
x_t=(-0.605, +1.254), u=+4.0, r=+0.0 x_t+1=(-0.453, +1.738)
x_t=(-0.453, +1.738), u=+4.0, r=+0.0 x_t+1=(-0.276, +1.714)
x_t=(-0.276, +1.714), u=+4.0, r=+0.0 x_t+1=(-0.120, +1.365)
x_t=(-0.120, +1.365), u=+4.0, r=+0.0 x_t+1=(-0.004, +0.970)
x_t=(-0.004, +0.970), u=+4.0, r=+0.0 x_t+1=(+0.079, +0.697)
x_t=(+0.079, +0.697), u=-4.0, r=+0.0 x_t+1=(+0.115, +0.003)
x_t=(+0.115, +0.003), u=-4.0, r=+0.0 x_t+1=(+0.080, -0.692)
x_t=(+0.080, -0.692), u=-4.0, r=+0.0 x_t+1=(-0.022, -1.393)
x_t=(-0.022, -1.393), u=+4.0, r=+0.0 x_t+1=(-0.185, -1.883)
x_t=(-0.185, -1.883), u=+4.0, r=+0.0 x_t+1=(-0.393, -2.191)
x_t=(-0.393, -2.191), u=+4.0, r=+0.0 x_t+1=(-0.599, -1.813)
x_t=(-0.599, -1.813), u=+4.0, r=+0.0 x_t+1=(-0.743, -1.032)
x_t=(-0.743, -1.032), u=+4.0, r=+0.0 x_t+1=(-0.808, -0.271)
x_t=(-0.808, -0.271), u=+4.0, r=+0.0 x_t+1=(-0.799, +0.462)
x_t=(-0.799, +0.462), u=+4.0, r=+0.0 x_t+1=(-0.714, +1.239)
x_t=(-0.714, +1.239), u=+4.0, r=+0.0 x_t+1=(-0.552, +1.983)
x_t=(-0.552, +1.983), u=+4.0, r=+0.0 x_t+1=(-0.337, +2.178)
x_t=(-0.337, +2.178), u=+4.0, r=+0.0 x_t+1=(-0.137, +1.772)
x_t=(-0.137, +1.772), u=+4.0, r=+0.0 x_t+1=(+0.015, +1.302)
x_t=(+0.015, +1.302), u=+4.0, r=+0.0 x_t+1=(+0.134, +1.092)
x_t=(+0.134, +1.092), u=+4.0, r=+0.0 x_t+1=(+0.236, +0.957)
x_t=(+0.236, +0.957), u=+4.0, r=+0.0 x_t+1=(+0.328, +0.884)
x_t=(+0.328, +0.884), u=+4.0, r=+0.0 x_t+1=(+0.416, +0.889)
```

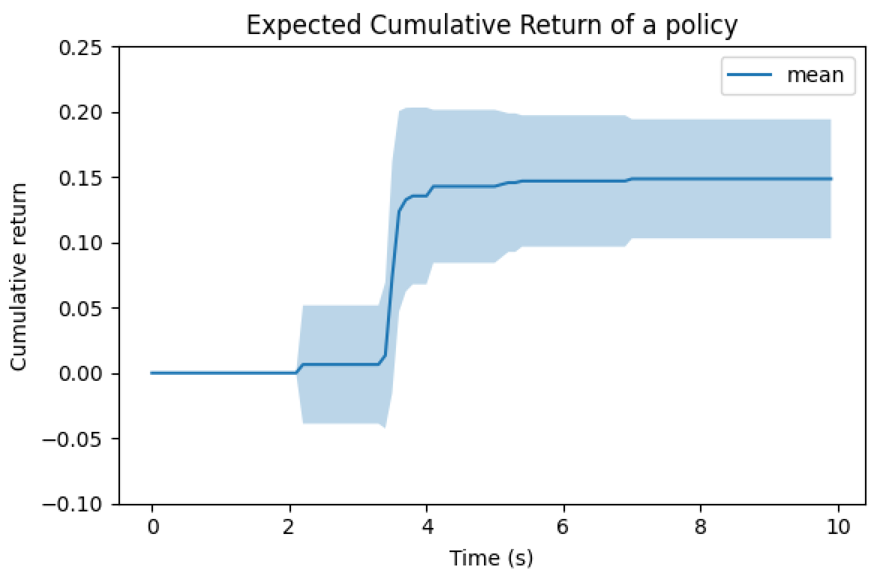


Figure 1: Expected return after 10s (100 time steps, for 50 estimations)

```
x_t=(+0.416, +0.889), u=+4.0, r=+0.0 x_t+1=(+0.508, +0.979)
x_t=(+0.508, +0.979), u=+4.0, r=+0.0 x_t+1=(+0.614, +1.151)
x_t=(+0.614, +1.151), u=+4.0, r=+0.0 x_t+1=(+0.741, +1.391)
x_t=(+0.741, +1.391), u=+4.0, r=+0.0 x_t+1=(+0.894, +1.685)
x_t=(+0.894, +1.685), u=+4.0, r=+1.0 x_t+1=(+1.079, +2.018)
x_t=(+1.079, +2.018), u=+4.0, r=+0.0 x_t+1=(+1.079, +2.018)
```

2 Expected return of a policy in continuous domain

The expected return of a stationary policy μ when the system starts from $x_0 = x$ is given by

$$J^{\mu}(x) = \lim_{T\to\infty} \underset{w_0,w_1,\ldots,w_T}{E}\left[\sum_{t=0}^{T} \gamma^t r\left(x_t, \mu\left(x_t\right), w_t\right) \mid x_0 = x\right]$$

We can instead use a Monte Carlo approach : we start from the initial state and we estimate the expectation by drawing N trajectories of length T and taking as an empirical estimate of the expectation the average value of the cumulative rewards of the drawn trajectories.
The expected return of a policy for the problem is therefore estimated by

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T}\gamma^t r_{nt}$$

We chose $N = 100$, which we consider large enough to approximate well the infinite time horizon of the policy because the expected return converges even before iteration 100. The plot can be seen in Figure 1.

3 Visualization

Our implementation uses the `save_caronthehill_image` function implemented in this Python Script to save the frames resulting from the simulation of the policy in a folder *frames*. It then assembles these frames to produce a GIF file displaying the simulation of the policy.

4 Fitted Q-Iteration

4.1 Dataset generation

The Fitted Q-Iteration works on random 4-tuples that need to be generated before training. Two solutions are proposed to build the basis of the training set.
The first idea is to use all the tuples in a trajectory as the training set. This has the advantage that the more common (and useful) states will be seen more often, but there are some parameters to adjust. First is the policy to use: a policy that always take the shortest path to the goal could give good results relatively fast for these states but as the other states will never be seen, the generalization may not work. A purely random policy will eventually see many more states, however, there is no guarantee that the reward $r = 1$ for the good terminal state is seen even once. If the training set doesn't account for it, it will be impossible to learn a valid Q function. We then decided to use a purely random policy to generalize better and will monitor the amount of time the reward $r = 1$ is seen. The second parameter is the trajectory length : especially on a random policy, we may need to wait a long time before we can observe the good reward. To find a good trajectory length, we simulated 1000 trajectories of a length of 1000. We found that out of 1000 trajectories, only 15 reached the goal, with the longest one taking 121 steps. We decided to set the length at 200 to include a safety margin. We decided to use 100 different trajectories of 200 steps, mainly because of the training time : using a larger dataset increase the time needed to compute each iteration.
The second idea is to generate states and actions from a uniform distribution, then compute the one-step transitions on those states-actions to build the training set. This approach is simpler : the only parameter is the size of the training set. To increase the number of successful trajectories in the training set size, we generated near-terminal states because their rewards are non-zero and added them to the training set.

4.2 Stopping criterion

The first criterion we chose is to settle for a fixed number of iterations. The idea is very simple, but finding a good number of iteration is not trivial : we saw from the policy in Section 1 that it is entirely possible to reach the goal in less than 100 time steps from the initial states, but we have no idea about the all the other states. Since those states are not as much interesting, but we didn't feel like completely ignoring them, we settled for a heuristic of 500 iterations.

Our second criterion looks at the policy and stops the iteration we the policy stops to change. Of course, for a continuous state space, this is impossible to do in practice. We decided to uniformly sample S_p points (as tuples (p, s)) and evaluate the policy on all these points, at each iteration. When the policy remains the same on all these S_p points for at least T_t iterations, we say that the learning is done, and we can stop iterating. We chose $S_p = 100$ and $T_t = 10$. We do not have a proper justification for these values : setting them too low would increase the False Positive Rate, and settings them too high would increase the False Negative Rate. We felt like these were good enough experimentally, but did not take the time to find optimal values. We also need to be careful with this policy : sometimes the policy extracted from the model assigns the same action to all states, and this can last for a few iterations. When this happens the criteria would stop incorrectly. Instead, we also check that the prediction is not uniform over the whole sample size (which could again be problematic and never stop if all points have the same action in the optimal policy).

As a third criterion[1], we could a priori define the maximum number N of iterations by setting the bound on the suboptimality of optimal policies μ_N^*

$$\left\| J^{\mu^*} - J^{\mu_N^*} \right\|_\infty \leq \frac{2\gamma^N B_r}{(1-\gamma)^2}$$

where $B_r = 1$ and $\gamma = 0.95$.

By choosing a bound of $\frac{2\gamma^N B_r}{(1-\gamma)^2} = 0.1$, we could then derive $N = \frac{\log\left\{0.1\frac{(1-\gamma)^2}{2B_r}\right\}}{\log\gamma} \approx 177$ and stop after this number of steps. We could also choose a bound of 0.01 which cuts the training later, as $N \approx 222$.

We chose to keep the two first proposed stopping rules for the computation of the $\widehat{Q}_N$-functions sequences.

4.3 Neural Network Structure

We could not find an inductive bias for the structure of a neural network appropriate for this task, and evaluating which complexity would give the best results is a rather complex task. We turned to the scientific literature and found [Rie05] which describe roughly the same problem of a car on a hill. We settled for their architecture of 2 hidden layers of 5 neurons each, all using sigmoid activation function. For the learning procedure, we differed from their proposition and used the default settings from sci-kit learn, except the initialization. This includes the Adam optimizer, with an initial learning rate of 0.001 and a maximum of 200 iterations to perform gradient descent on each dataset. The main difference with the paper is that instead of using a brand-new network for each dataset, we reused the same one for the whole problem. The network is initialized with a Glorot-Normal strategy (following the recommendation of sci-kit learn) and then the weights and biases are reuse for the next iteration of Q. We chose to do this because the network was hard to train : multiple times, sci-kit learn issued a warning stating that the network did not converge in 200 iterations. Since the datasets should be close to each other between iteration, we thought that a network fitted with the last iteration could be a good starting point. Indeed, the training time was considerably shortened, although we will not report any time measurement since it is not the goal of this assignment.
As shown in the results, this architecture was not working well in our cases. We also looked at the default parameters from the sci-kit learn library which recommends a single hidden layer of 100 neurons, all activated via a ReLU unit. For good measure, we also implemented this neural network. This network is more complex than the first one, and we hope that it will be able to learn the function better.

[1]D.Ernst, P.Geurts, L. Wehenkel : Tree-Based Batch Mode Reinforcement Learning

4.4 Linear Regression

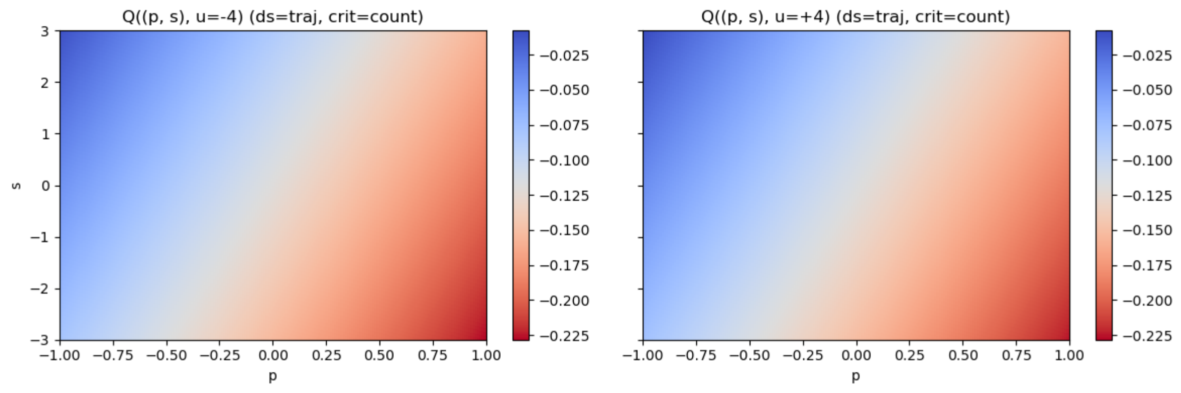


Figure 2: Q-function from Linear Regression on random trajectories, fixed number of iterations

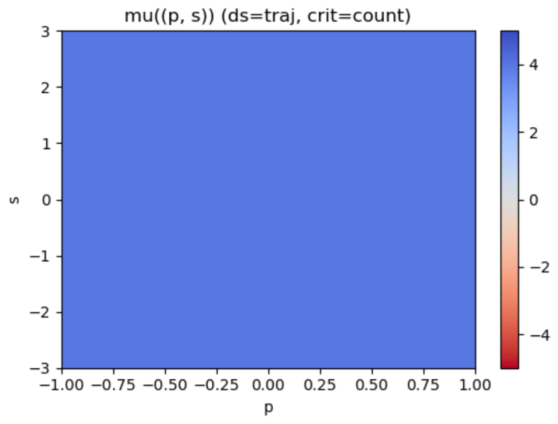


Figure 3: Extracted policy from Q-function from Linear Regression on random trajectories, fixed number of iterations

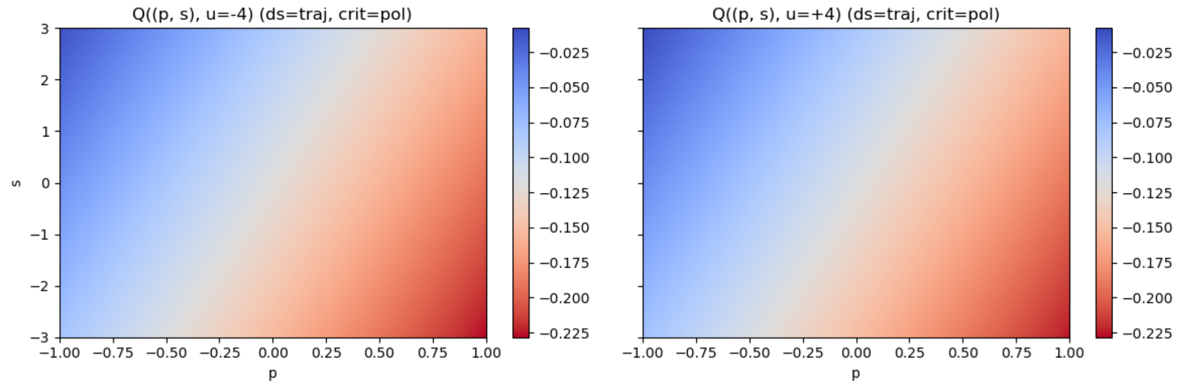


Figure 4: Q-function from Linear Regression on random trajectories, policy-based criterion

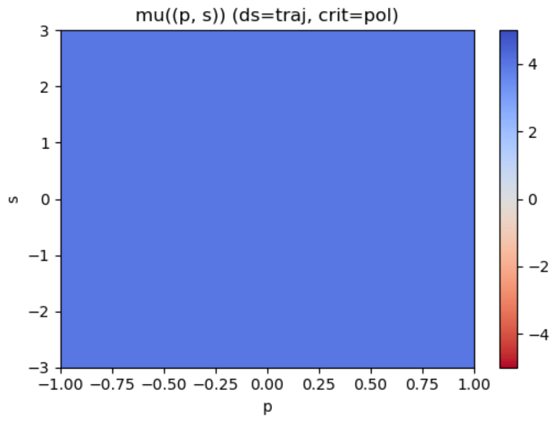


Figure 5: Extracted policy from Q-function from Linear Regression on random trajectories, policy-based criterion

We can notice that there is no difference between the results obtained with the two stopping strategies, as we fixed the maximum number of iterations to 500 and the policy-based stopping condition is not met before iteration 500.

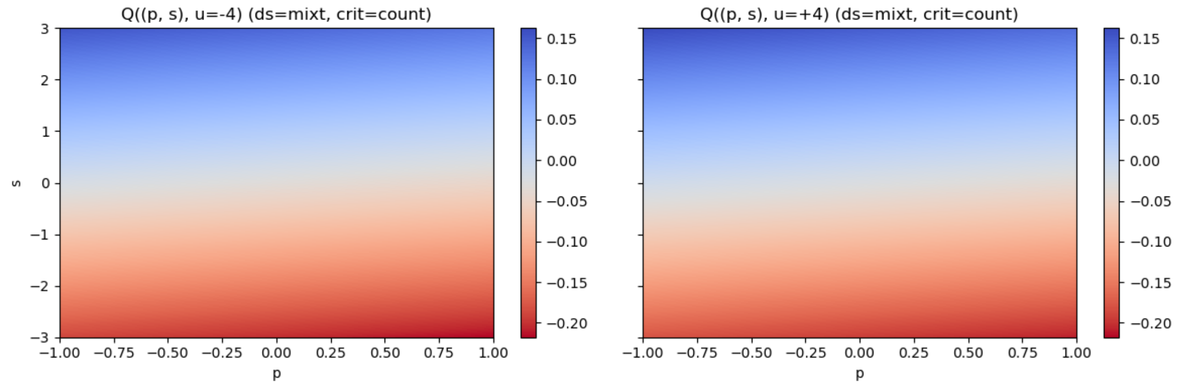


Figure 6: Q-function from Linear Regression on random trajectories mixed with other random transitions, fixed number of iterations

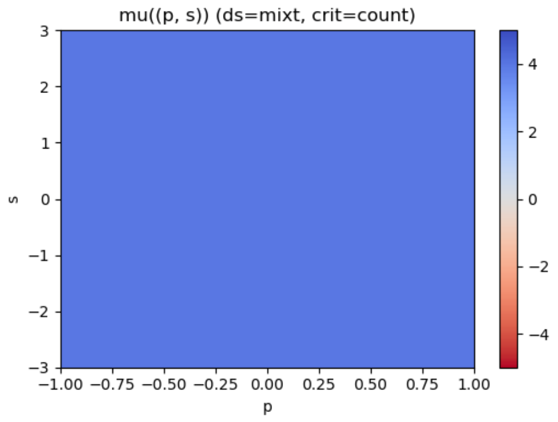


Figure 7: Extracted policy from Q-function from Linear Regression on random trajectories mixed with other random transitions, fixed number of iterations

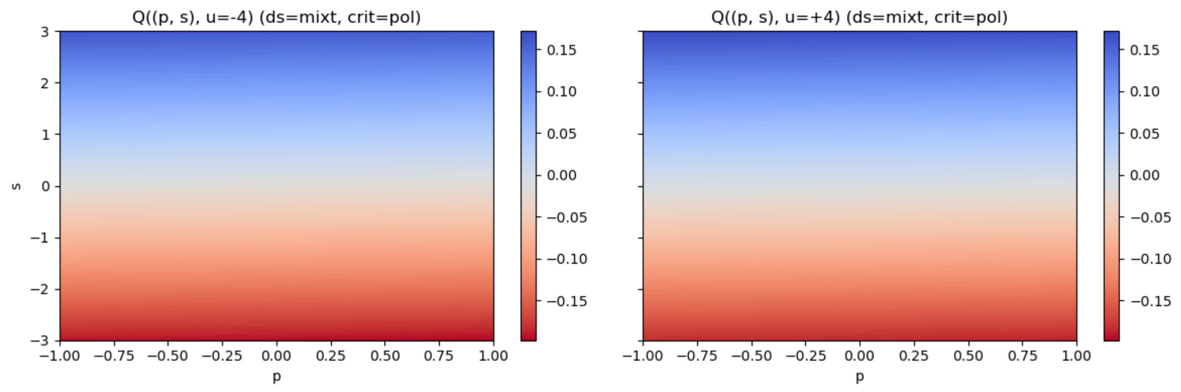


Figure 8: Q-function from Linear Regression on random trajectories mixed with other random transitions, policy-based criterion

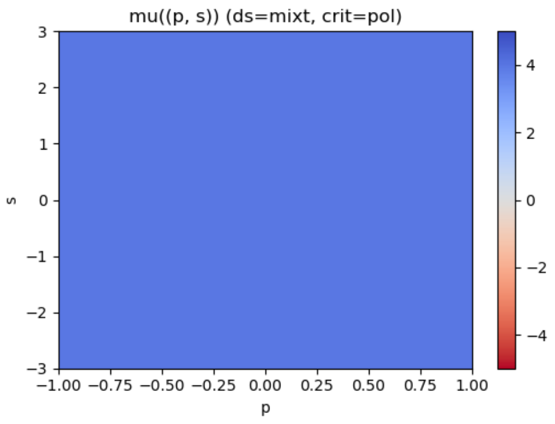


Figure 9: Extracted policy from Q-function from Linear Regression on random trajectories mixed with other random transitions, policy-based criterion

We can notice that there is no difference between the results obtained with the two stopping strategies, as we fixed the maximum number of iterations to 500 and the policy-based stopping condition is not met before iteration 500.

4.5 Extremely Randomized Trees

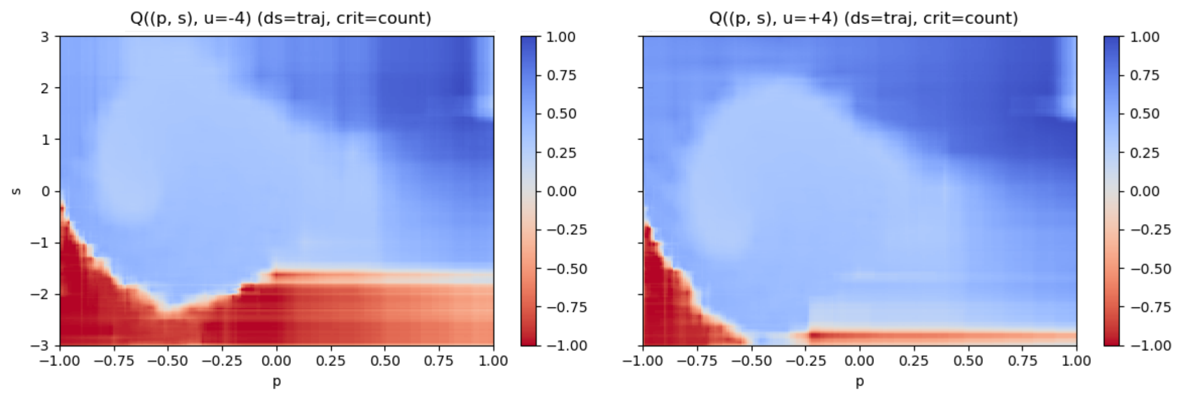


Figure 10: Q-function from Extremely Randomized Trees on random trajectories, Fixed number of iterations

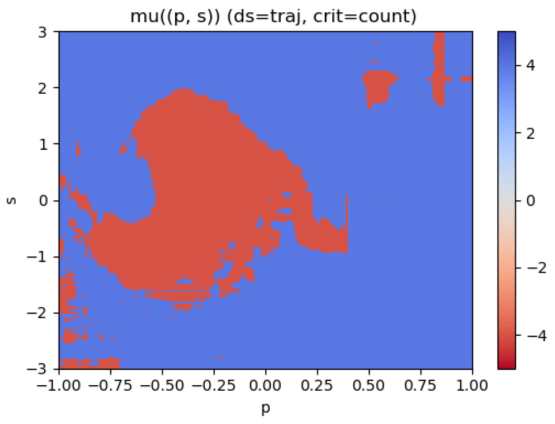


Figure 11: Extracted policy from Q-function from Extremely Randomized Trees on random trajectories, Fixed number of iterations

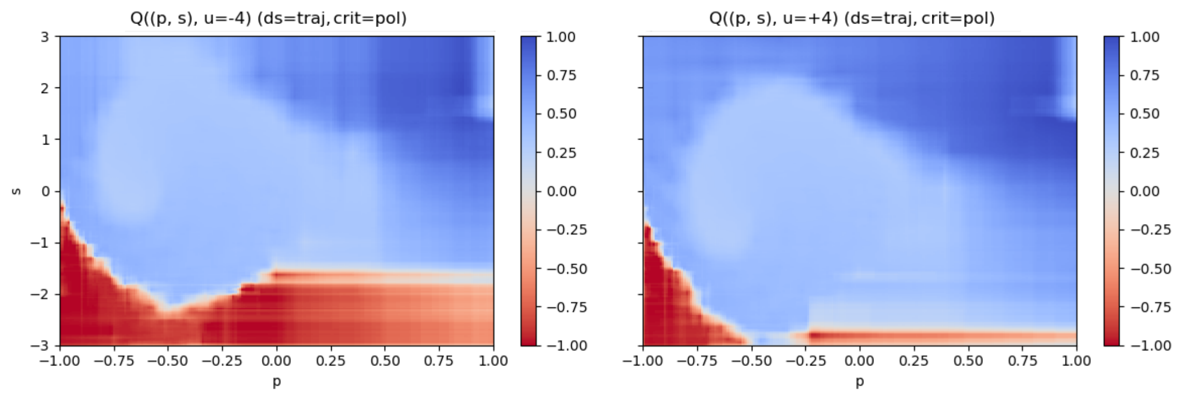


Figure 12: Q-function from Extremely Randomized Trees on random trajectories, Policy-based criterion

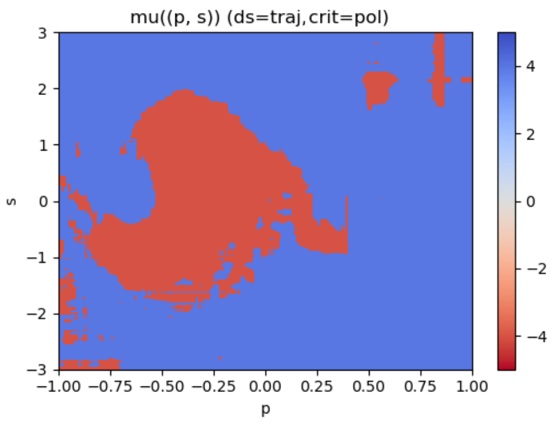


Figure 13: Extracted policy from Q-function from Extremely Randomized Trees on random trajectories, Policy-based criterion

We can notice that there is no difference between the results obtained with the two stopping strategies, as we fixed the maximum number of iterations to 500 and the policy-based stopping condition is not met before iteration 500.

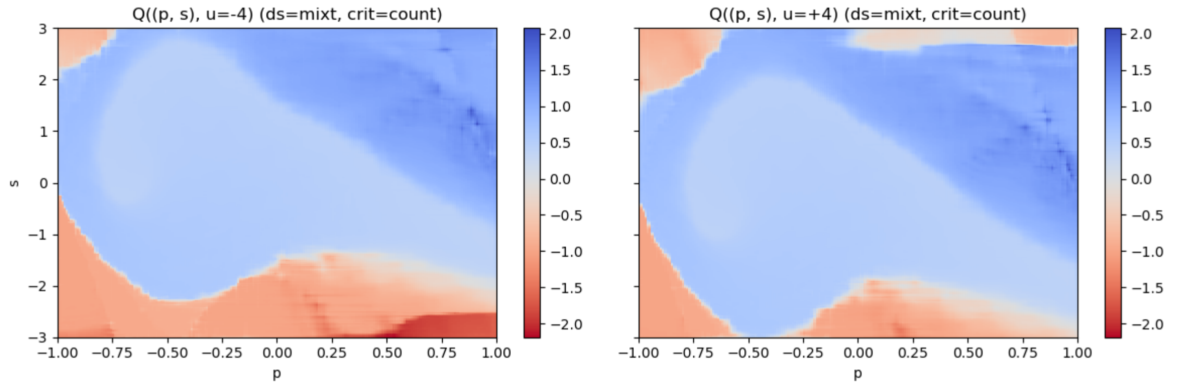


Figure 14: Q-function from Extremely Randomized Trees on random trajectories mixed with other random transitions, fixed number of iterations

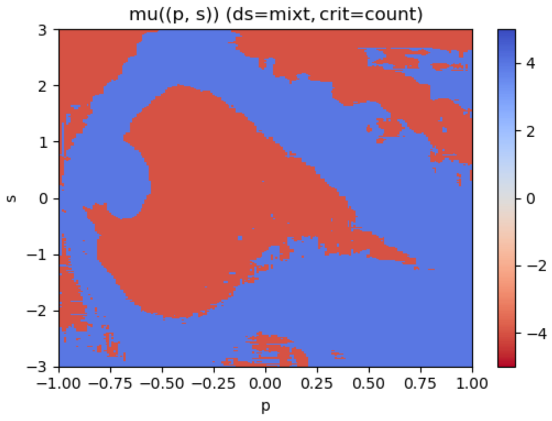


Figure 15: Extracted policy from Q-function from Extremely Randomized Trees on random trajectories mixed with other random transitions, fixed number of iterations

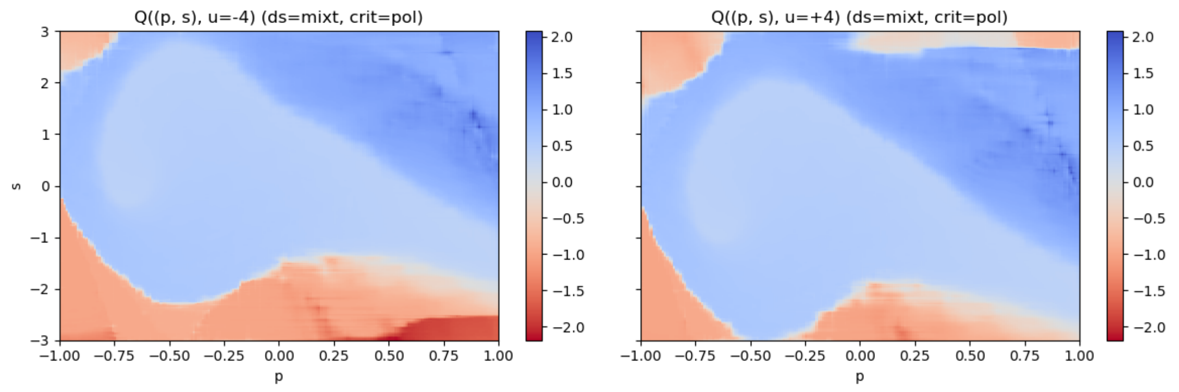


Figure 16: Q-function from Extremely Randomized Trees on random trajectories mixed with other random transitions, policy-based criterion

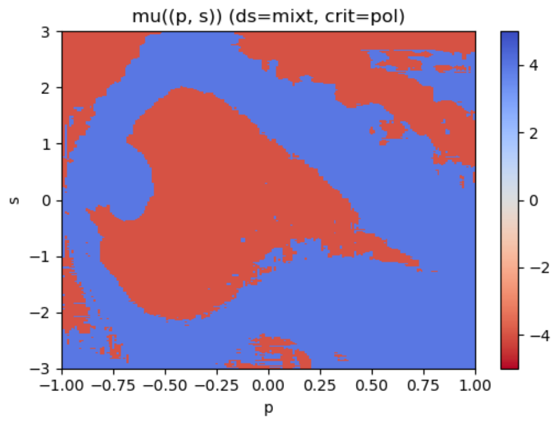


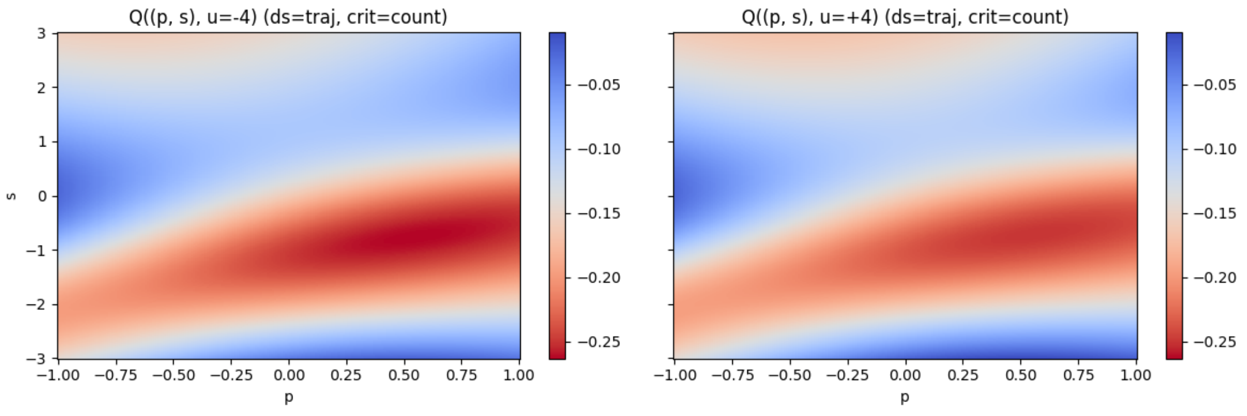
Figure 17: Extracted policy from Q-function from Extremely Randomized Trees on random trajectories mixed with other random transitions, policy-based criterion

We can notice that there is no difference between the results obtained with the two stopping strategies, as we fixed the maximum number of iterations to 500 and the policy-based stopping condition is not met before iteration 500.

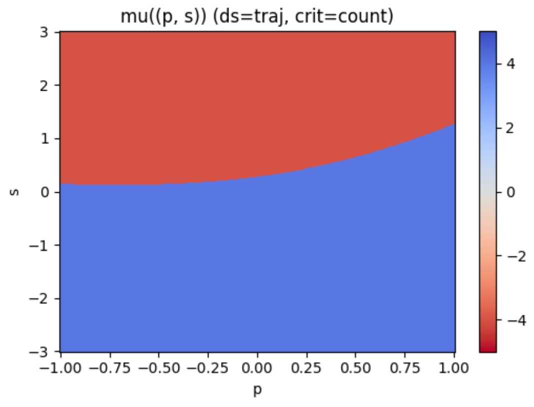
4.6 Neural networks

4.6. (a) Architecture from [Rie05]

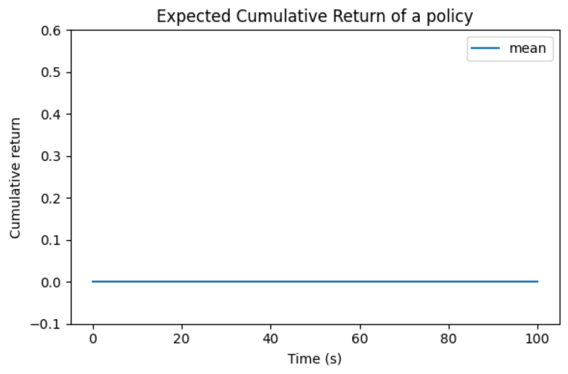
Sadly, all Figures from (29a, 19c, 20c and 21c) show that the model was never able to learn the required feature to solve the problem : the expected cumulative reward is always zero as the car never manages to reach out of the valley. It is worth noting that the criterion stopping when the policy stops changing was never applied : the convergence was never fully reached and therefore the limit of 500 iterations stopped the algorithm too soon. Therefore both situation are actually equivalent. We can see that some aspects of the decision boundary look somewhat similar to the ones from the tree-based methods, but it is clearly insufficient.
It is worth noting that these results are `not` a sign of failure from the neural to learn the data-set at any given iteration, but rather an accumulation of errors during all the iteration that destroyed the



(a) Q-function for the two action : decelerate (left) or accelerate (right)



(b) Policy



(c) Expected Cumulative Reward

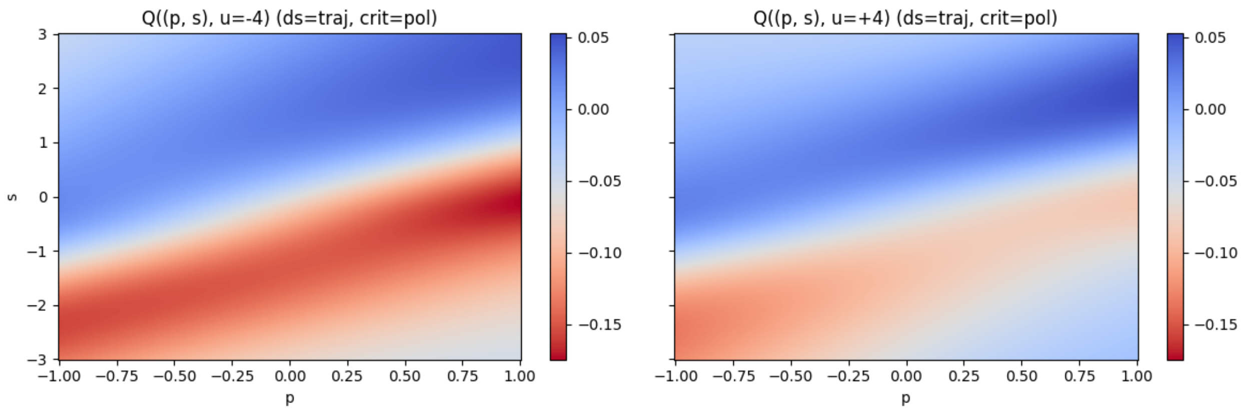
Figure 18: Q-function, extracted policy and its expected cumulative reward, all learnt from a Neural Network (small) on random full trajectories, with 500 fixed iterations

features from the learning set and caused the training to fail. At each iteration, convergence checks are performed automatically by sci-kit learn and there is no reason to believe that they have failed. This motivates our second approach, of a larger networks, with more connections.

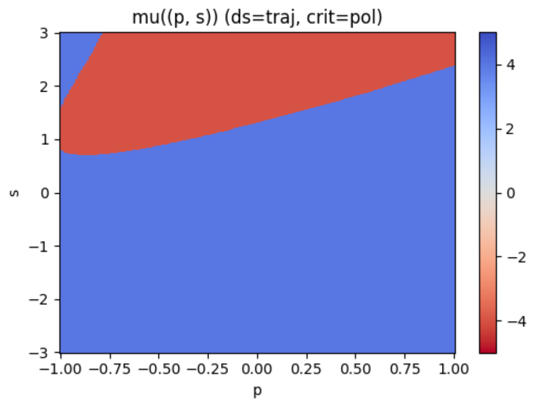
4.6. (b) Larger Network

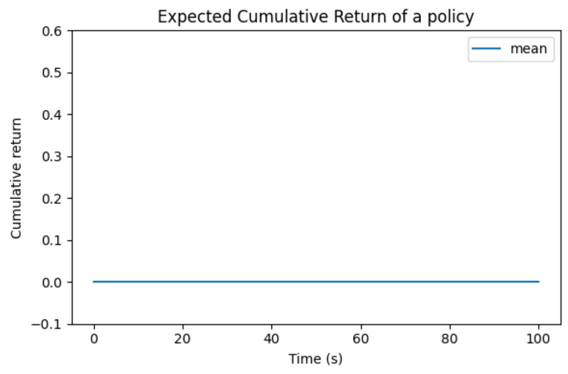
Figures 22 are sadly very similar to the previous values of the smaller network, showing a lack of convergence to the desired output, although the decision boundary on the policy is closer to what we could expect (comparing to the tree based methods). These closer results let use think that with more data or longer training, we could have reached a better approximation.
Fortunately, Figure 23 shows a much better results, the expected cumulative reward is finally positive in Figure 23c. The contour plot of the state-action value function is also much closer to the ones obtained in the tree-based method.
We also evaluated the criterion stopping at policy convergence. However it failed to stop early and gave the same results (up to statistical errors). To avoid cluttering this report, there are not shown.

4.7 Summary



(a) Q-function for the two action : decelerate (left) or accelerate (right)

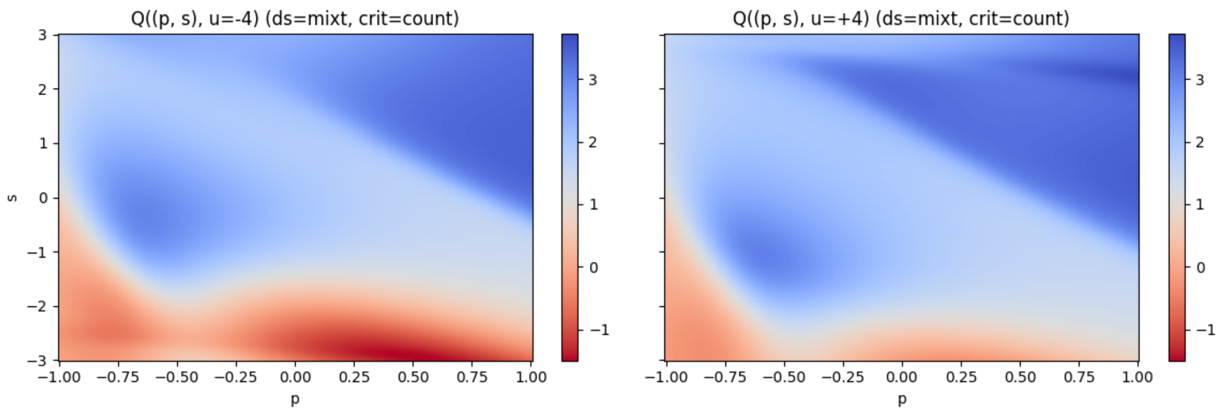




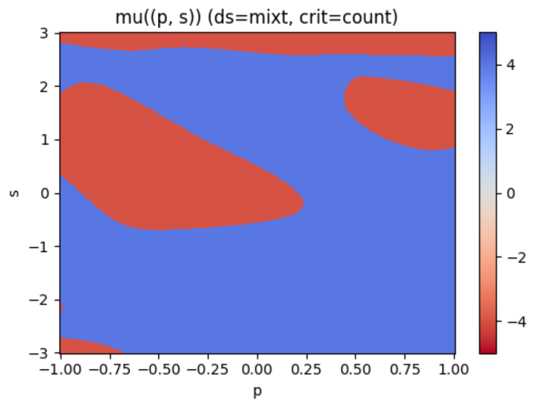
(b) Policy

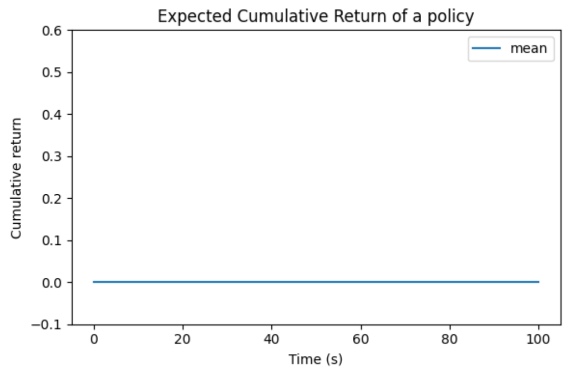
(c) Expected Cumulative Reward

Figure 19: Q-function, extracted policy and its expected cumulative reward, all learnt from a Neural Network (small) on random full trajectories, stopping when the policy stops to change (see discussions)



(a) Q-function for the two action : decelerate (left) or accelerate (right)

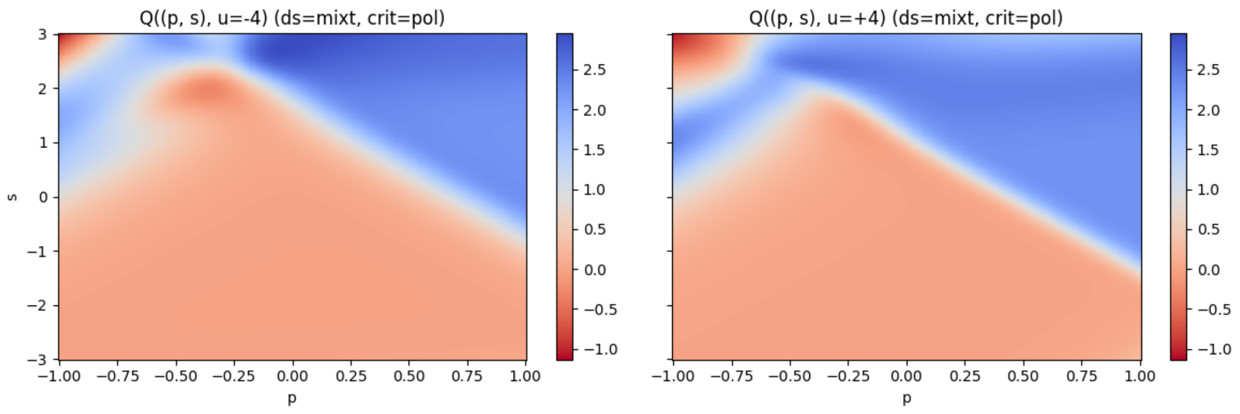




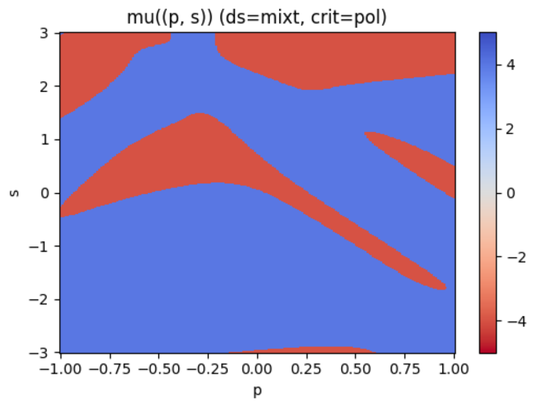
(b) Policy

(c) Expected Cumulative Reward

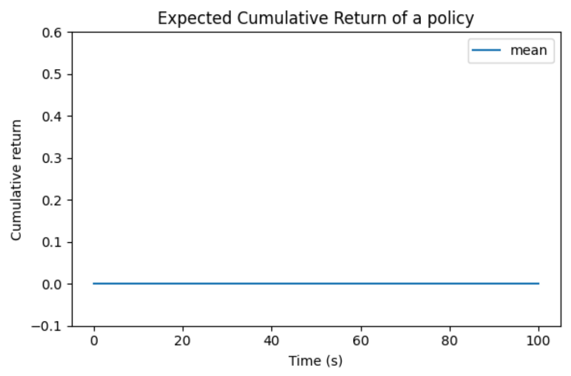
Figure 20: Q-function, extracted policy and its expected cumulative reward, all learnt from a Neural Network (small) on random trajectories mixed with other random transitions, with 500 fixed iterations



(a) Q-function for the two action : decelerate (left) or accelerate (right)

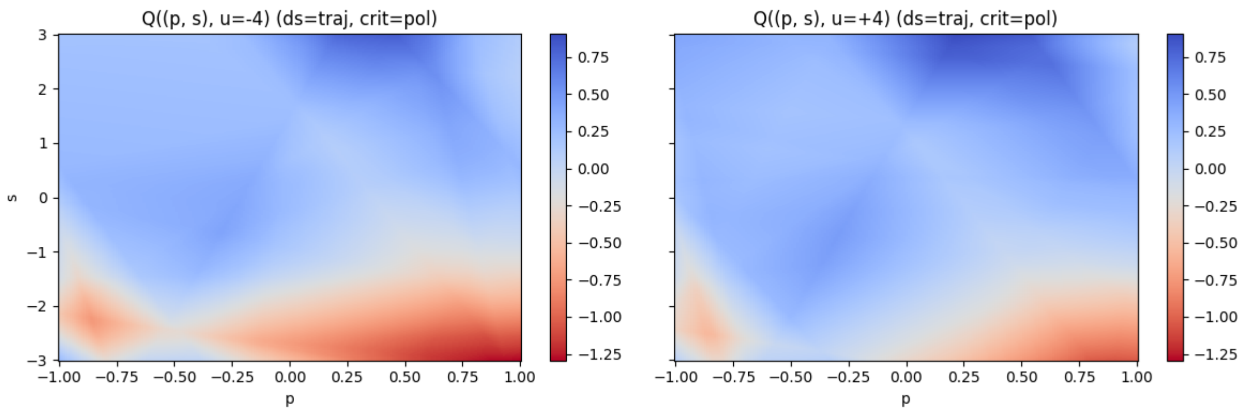


(b) Policy

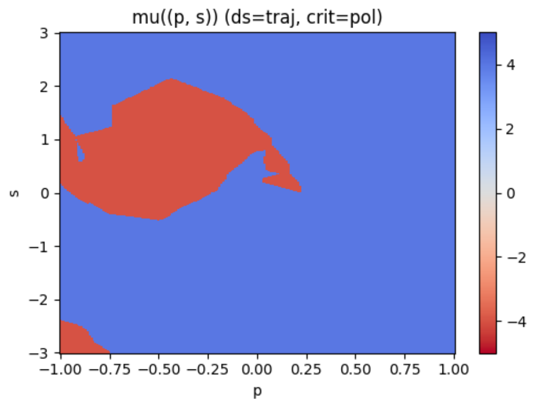


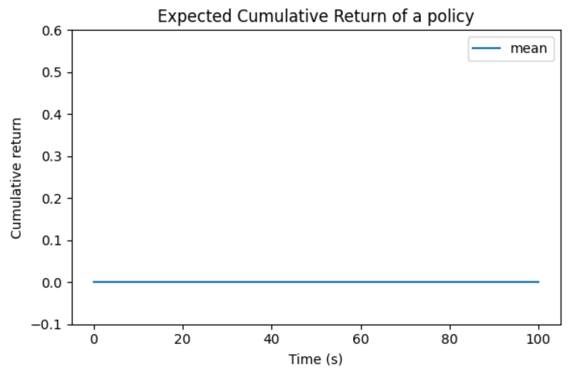
(c) Expected Cumulative Reward

Figure 21: Q-function, extracted policy and its expected cumulative reward, all learnt from a Neural Network (small) on random full trajectories mixed with other random transitions, stopping when the policy stops to change (see discussions)



(a) Q-function for the two action : decelerate (left) or accelerate (right)

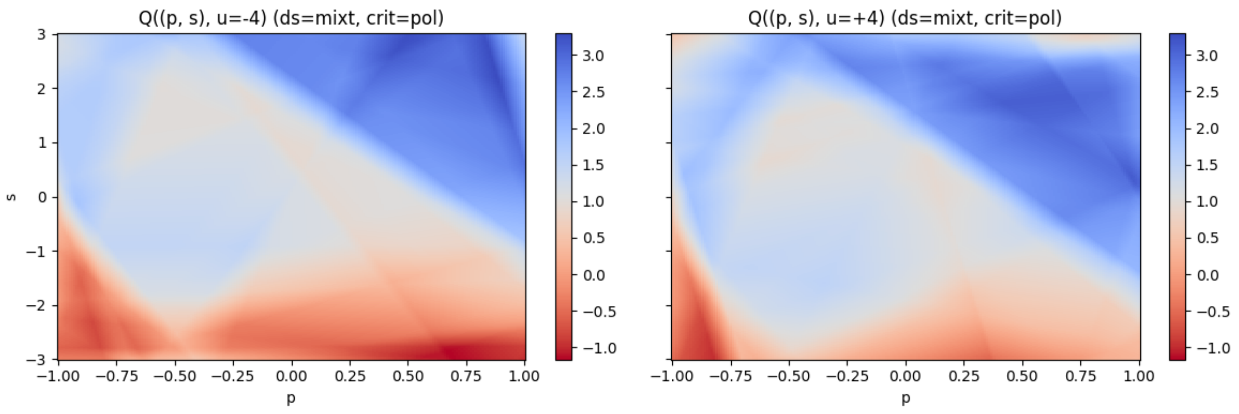




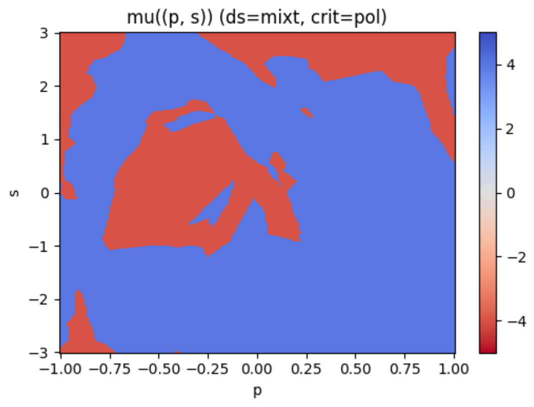
(b) Policy

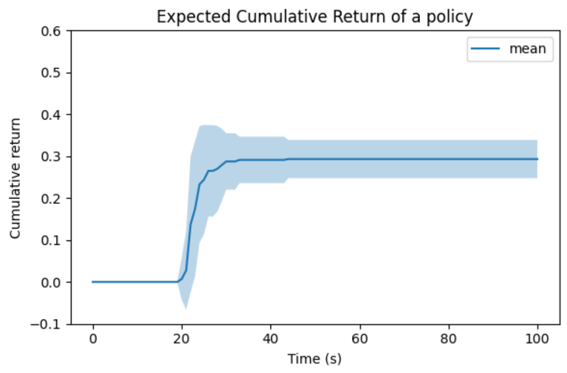
(c) Expected Cumulative Reward

Figure 22: Q-function, extracted policy and its expected cumulative reward, all learnt from a Neural Network (larger) on random full trajectories, with 500 fixed iterations



(a) Q-function for the two action : decelerate (left) or accelerate (right)





(b) Policy

(c) Expected Cumulative Reward

Figure 23: Q-function, extracted policy and its expected cumulative reward, all learnt from a Neural Network (larger) on random full trajectories mixed with other random transitions, with 500 fixed iterations

Table 1 shows the detailed comparison between all the data generation algorithms, the criteria, and the supervised learning algorithms. It is now clear that the neural network from [Rie05] was too small for this task, perhaps we overlooked some differences between the two problems. The criteria we designed to handle the convergence of the policy was not working properly : the branch avoiding too-early exit (making sure that the policy is not uniform of the sample) prevented the criterion to stop soon enough and there were no difference with the counting criterion. The comparison between the standard trajectory generation and the mixed source would be very interesting given the results, but those situation vary too much from one to the other. The mixed generation is not only different in type, it is also more than 10 times as large. Thus we cannot estimate whether the differences in results are due to the kind of data (more information on the terminal states) or only the size of the data-set.

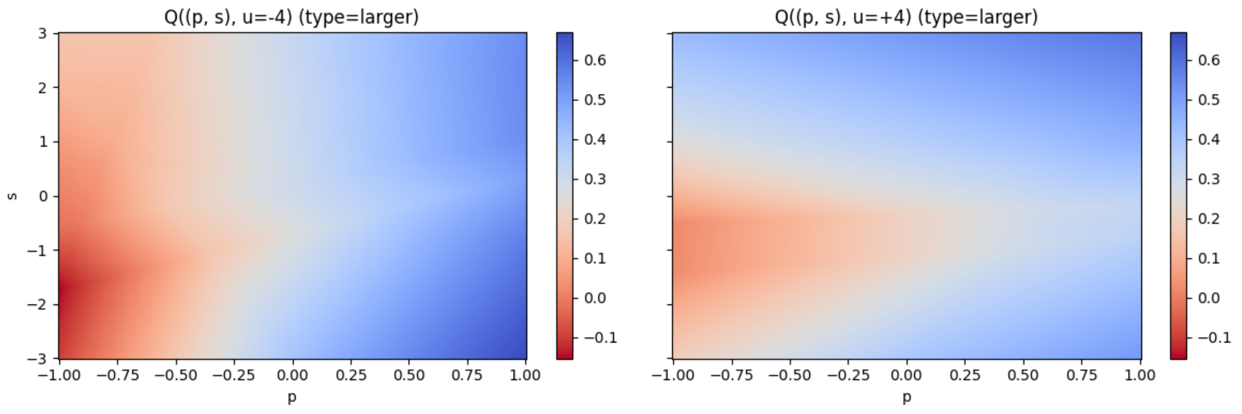
	Count-based crit.		Policy-based crit.	
	simple traj.	mixed traj.	simple traj.	mixed traj.
Linear Regression	0	0	0	0
E.T.	0.28	0.33	0.28	0.33
NN (5, 5)	0	0	0	0
NN (100,)	n/a	n/a	0	0.29

Table 1: Comparison of the different parameters

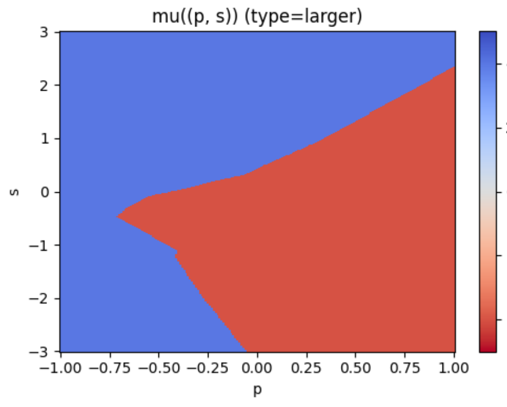
5 Parametric Q-Learning

To implement the Q-Learning algorithm, we chose to do a semi-online approach using a greedy policy : we iterate a trajectory and learn the Q-function based on the transitions observed. Following our results from the first assignment, we saw that using a replay buffer was an effective strategy, and we implemented it in our method `train_greedy`. The training routine requires specifying a **number of epochs** : this is the number of trajectories used to do the whole training and will be the parameter of interest for our protocol. We set the size of the replay buffer and the length of the trajectories to **200 transitions**. After each transition is observed, we draw **10 samples** at random from the replay buffer, and do the update steps based on these transitions.
To build the neural-network, we reflected on the previous attempt we made for the fitted Q-iterations : this network possibly has too much complexity by taking the action as a variable. Even though it's a binary variable, it is treated as a real-valued variable. We propose to use a two-headed structure following this inductive bias : since the Q may be very different in both case, we should rather have different network (or at least first layers) than specify it as a variable. For the structure of the network, we decided to stay close to the previous architecture with a few differences : since Q is bounded by -1 and 1, the last layer is fed to a tanh activation function. Moreover, ReLU have been increasingly popular, so we replaced remaining sigmoid activations by ReLUs. Finally, we increased the width of the hidden layers to increase the complexity of the model, hoping it would be better than the last one.
The above-mentioned training routine begs the question : when do we stop the learning algorithm ? A first strategy could be to look at the norm of the update term : we tried to print it every 100 update, but saw little to no variation over time (these results are reported below). Another valid strategy is to compute the expected cumulative return on a few short trajectories and stop when it reaches a threshold (we found that 0.3 gives decent results). Unfortunately, this strategy doesn't really allow controlling the size of the training set, which is prohibitive for this assignment. Another important aspect of this criterion is that it is not stable : if we remove it, we see that the ECR (expected cumulative return) is sometimes above the threshold but shortly after it may drop to a negative value or stay at zero (for a strategy that fails to reach any of the end) as can be seen in the training dump below (although in the update given, the ECR is not negative). This tells us that the model isn't actually learning properly, but we haven't found a solution that gives better results.

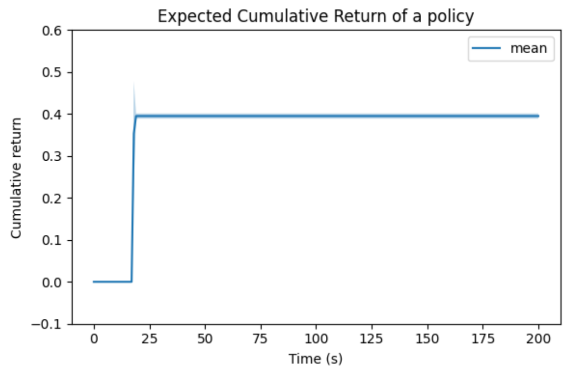
```
epoch=   0/100 t=   9/200 update: max_update=4.459E-02    ECR: mean[-1]= +0.13
epoch=   0/100 t=  19/200 update: max_update=1.475E-02    ECR: mean[-1]= +0.28
epoch=   2/100 t=   1/200 update: max_update=3.903E-03    ECR: mean[-1]= -0.77
epoch=   3/100 t=   3/200 update: max_update=5.918E-03    ECR: mean[-1]= +0.00
epoch=   3/100 t=  13/200 update: max_update=5.071E-03    ECR: mean[-1]= +0.14
...
epoch=  21/100 t=  71/200 update: max_update=1.081E-03    ECR: mean[-1]= +0.00
epoch=  21/100 t=  81/200 update: max_update=1.014E-03    ECR: mean[-1]= +0.00
epoch=  21/100 t=  91/200 update: max_update=1.152E-03    ECR: mean[-1]= +0.00
epoch=  21/100 t= 101/200 update: max_update=9.836E-04    ECR: mean[-1]= +0.00
epoch=  21/100 t= 111/200 update: max_update=7.583E-04    ECR: mean[-1]= +0.00
...
epoch=  99/100 t= 122/200 update: max_update=7.407E-04    ECR: mean[-1]= +0.00
epoch=  99/100 t= 132/200 update: max_update=4.761E-04    ECR: mean[-1]= +0.00
```



(a) Q-function for the two action : decelerate (left) or accelerate (right)



(b) Policy



(c) Expected Cumulative Reward

Figure 24: Q-function, extracted policy and its expected cumulative reward, extracted from a model using Q-Learning with a greedy policy ($\varepsilon = 0.25$) that was stopped after its ECR reached 0.3.

```
epoch=  99/100 t= 142/200 update: max_update=8.095E-04    ECR: mean[-1]= +0.00
epoch=  99/100 t= 152/200 update: max_update=8.897E-04    ECR: mean[-1]= +0.00
...
```

Figure 24 shows that this strategy yields a model with decent capabilities. In the observed trajectories, the car gains momentum to the left before accelerating and reaching the right side in a few steps. We cannot make any comment regarding the optionality of the model, since we do not know any bound for the Q function, but the model doesn't seem to be far. While the $Q-$function seems a bit blurry, the policy has clear boundaries, at least clearer than most previous models.
The stopping criterion is computed on a few small trajectories, which may severely under- or over-estimate the ECR and explains why Figure 24c shows an ECR of 0.4 while the training stopped after it only reached 0.3. Moreover, the criterion is only checked very few steps, so we do not expect to always have an ECR of 0.3.
For the experimental protocol, we decided to approximate the number of one-step transitions by the number of epochs times the maximal length of said trajectories. We impose a limit of 200 transitions, but if a terminal state is reached the trajectory is stopped : this metrics therefore overestimate the number of transitions. We decided to evaluate the ECR after **1, 5, 10, 50, and 100 epochs**. Of course, the experiment is repeated 10 times and the mean is reported.

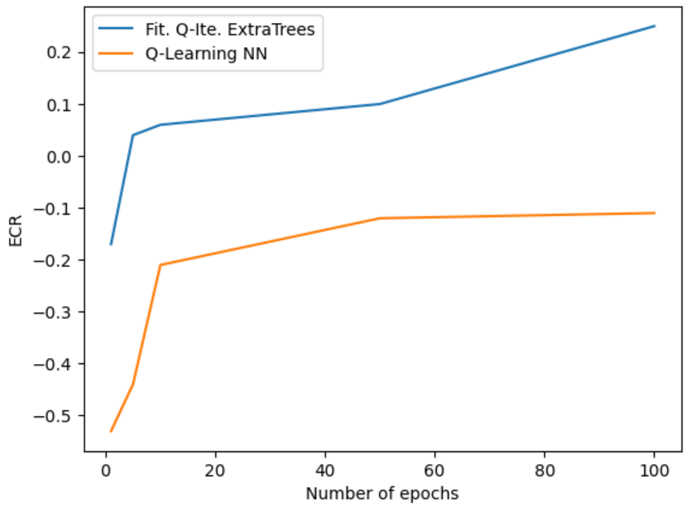


Figure 25: Comparison between fitted Q-iteraitons on ExtraTrees with parametric Q-Learning on a neural network

To do the comparison, we used q-learning with a greedy policy on **n_epochs**, with trajectories of at most 200 transitions each, as described earlier. The dataset size can be approximated by **traj_len** $\times$ **n_epochs**. On the other hands, ExtraTrees were trained using offline fitted Q-Iterations on **n_epochs** trajectories containing exactly **traj_len** transitions each (terminal states are repeated), for 200 epochs. There is slightly more data in the second case, although the transitions after termination may not be informative. Moreover, the first algorithm doesn't generate trajectories from a totally random policy like the second one, but using a greedy policy, which can be more effective. The comparison between the two is thus not perfect.
Figure 25 shows the comparison between the two. As explained previously, the stopping criterion is not good, for any of the two models, but it is the only one that allows to compare easily the two approaches. As such, the performances of the Q-Learning are severely under-estimated. We can still see that the performances improve with respect to the size of the data, although we expected a more impressive growth.
Retrospectively, we should have trained our model differently : we could have kept the same criterion and count the number of transitions, which would not underestimate its performance, but we lacked the time to do that.

6 Normalized Q-learning

In this section, we re-run the experimental protocol we have designed in Section 5 but with a normalized update term, which means that the parameter update becomes

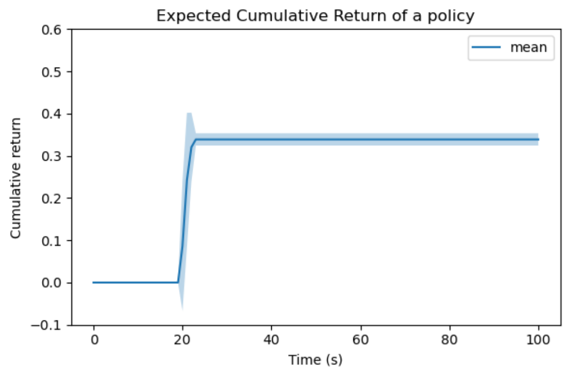
$$a \leftarrow a + \alpha \delta\left(x_t, u_t\right) \frac{\frac{\partial \widetilde{Q}(x_t, u_t, a)}{\partial a}}{\left\| \frac{\partial \widetilde{Q}(x_t, u_t, a)}{\partial a} \right\|}$$

It makes sense to normalize the gradient, as we want to step in a direction of magnitude equal to the

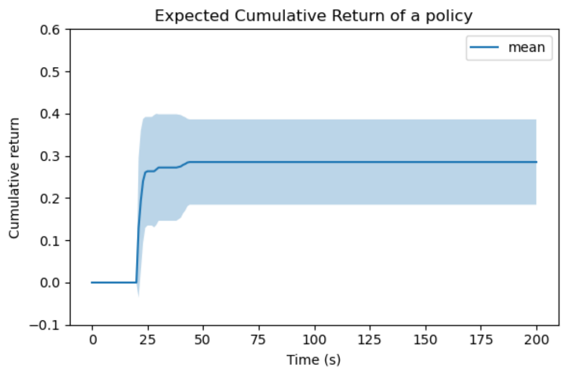
error- that we want to correct.
We tested several definitions of the normalization term : we tested the L-2 norm $\left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\| = \left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_2$ and the infinite norm $\left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\| = \left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_\infty$ of the gradient.

As a mean of comparison with the Fitted-Q-iteration (FIQ), we will again only compare our results with those obtained with the Extremely Randomized Trees, as it was the only model that yielded to non-zero expected returns. We take the graph of the expected cumulative return we obtained in section 4 when running on the mixed trajectories dataset. As a reminder, the two stopping criteria we had chosen in that section resulted in the same graph.

With our stopping criterion from section 5 consisting in cutting the learning when the expected cumulative return reaches 0.3 for normalization term $\left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_2$, we have Figures 26 and 27. The same stopping criterion is used with normalization term $\left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_2$ and yields the graphs in Figures 28 and 29.

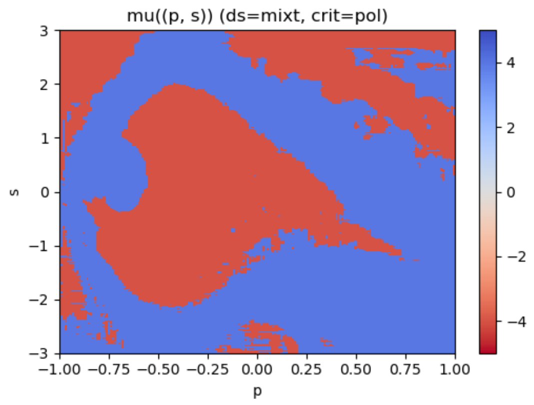


(a) Expected Cumulative Reward of Extremely Randomized Trees with the FIQ

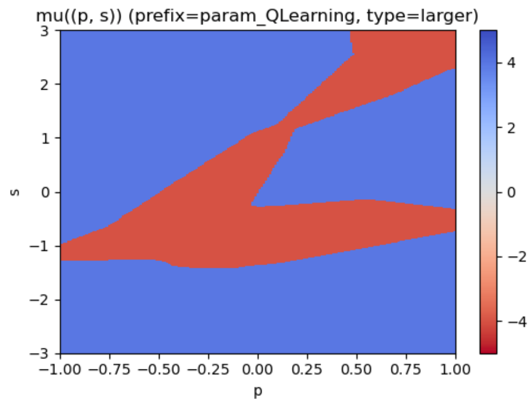


(b) Expected Cumulative Reward of our Neural Network with the normalized PQL

Figure 26: Comparison between the expected cumulative returns obtained with the FQI (a) and the normalized PQL with normalization term $\left\|\frac{\partial \widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_2$ (b)

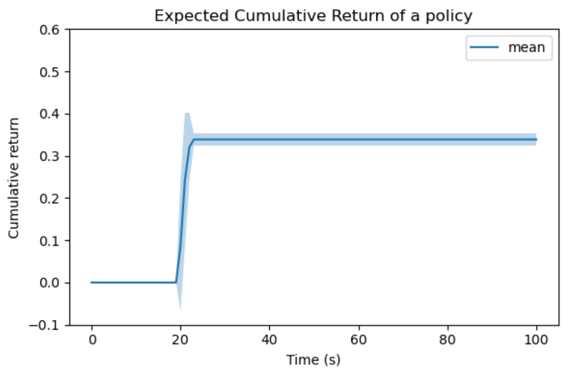


(a) Extracted policy from Q-function from Extremely Randomized Trees with FIQ

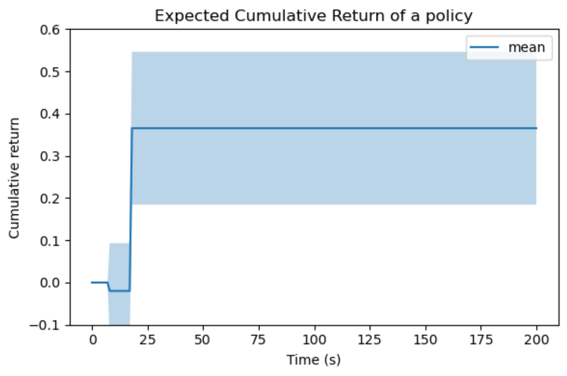


(b) Extracted policy from our Neural Network with the normalized PQL

Figure 27: Comparison between the extracted policies $\widehat{\mu}_*$ obtained with the FQL (a) and the normalized PQL with normalization term $\left\|\frac{\partial \widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_2$(b)

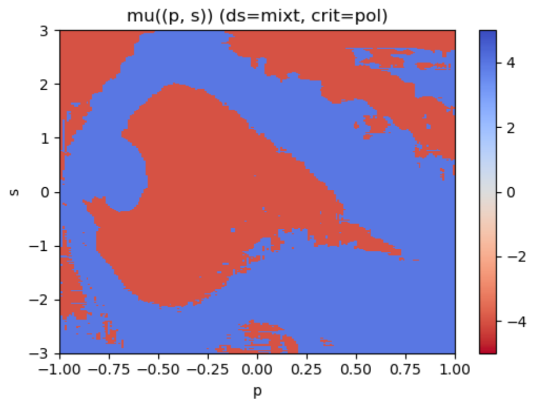


(a) Expected Cumulative Reward of Extremely Randomized Trees with the FIQ

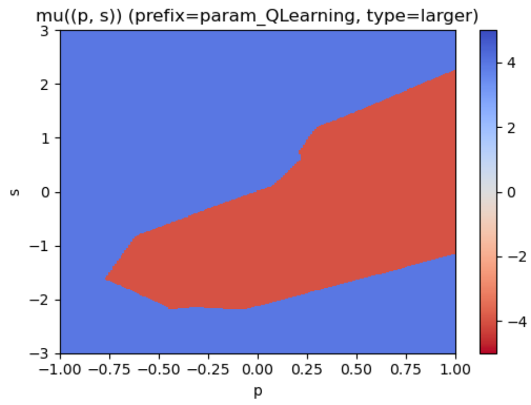


(b) Expected Cumulative Reward of our Neural Network with the normalized PQL

Figure 28: Comparison between the expected cumulative returns obtained with the FQI (a) and the normalized PQL with normalization term $\left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_\infty$ (b)



(a) Extracted policy from Q-function from Extremely Randomized Trees with FIQ



(b) Extracted policy from our Neural Network with the normalized PQL

Figure 29: Comparison between the extracted policies $\widehat{\mu}_*$ obtained with the FQL (a) and the normalized PQL with normalization term $\left\|\frac{\partial\widetilde{Q}(x_t,u_t,a)}{\partial a}\right\|_\infty$(b)

It is difficult to draw any conclusions from these figures : depending on the initial state, we may need the same number of one-step system transitions to reach our goal. Re-running the experiment multiple times lead to models with varying standard deviations, which makes it impossible to infer any consistent comparison with the results of the fitted-Q-iteration.

References

[Rie05] Martin Riedmiller. “Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method”. In: *Machine Learning: ECML 2005*. Springer Berlin Heidelberg, 2005, pp. 317–328. DOI: `10.1007/11564096_32`. URL: `https://doi.org/10.1007/11564096_32`.