Final project : Load balancing
Introduction to reinforcement learning

Léo Meissner and Héloïse Lafargue

Computer Science Department - Third year
Imaging & multimedia
2023-2024

1 Introduction : Load balancing

Consider the following simple model for a dispatcher in a data center.

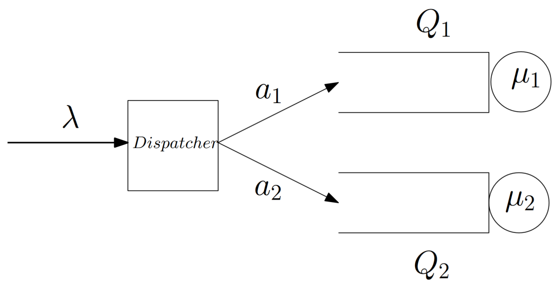


FIGURE 1 – A dispatcher shares the load between two servers

Time is discrete. Let Q1 and Q2 denote the total number of jobs in the server 1 and 2, respectively. Every time slot, the dispatcher observes (Q1, Q2) and takes the action a_i, i = 1, 2, that dispatches a potential new incoming job to server i. The cost in every time slot is Q1 + Q2, independently of the action. It has been chosen $\mu = 0.2$, $\mu 2 = 0.4$ and $\lambda = 0.3$, and the discounting factor is $\gamma = 0.99$.

2 MDP

2.1 Policy Evaluation

We assume that the random policy dispatches every job with probability 0.5 to either Q1 and Q2.

- Bellman equation that characterizes the value function for this policy :

$$V_\Pi(s) = 0.5 * r(s, a_1) + \gamma \sum_{s' \in S} p(s'|s, a_1) V_\Pi(s') + 0.5 * r(s, a_2) + \gamma \sum_{s' \in S} p(s'|s, a_2) V_\Pi(s')$$

with the reward (opposite of the cost) : $r(s, a_i) = -(Q1 + Q2)$.

- In Figure 2, the depicted results showcase the value function obtained through Iterative Policy Evaluation. This process adheres to the contraction principle during initialization and employs a stopping criterion based on the difference between consecutive iterations.

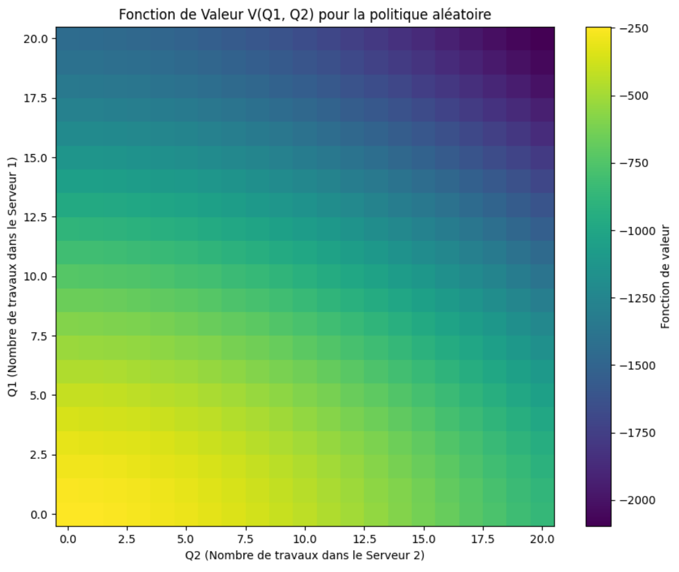


FIGURE 2 – Value function of the random policy, using iterative policy evaluation

The value function V(Q1, Q2) represents the expected value of each state based on the server capacities. The graph demonstrates a decrease in V(Q1, Q2) as both Q1 and Q2 increase, indicating a declining trend in the value of states. As the servers' workload intensifies, the value becomes more negative, illustrating the amplified cost or penalty linked with larger queues or a higher number of pending jobs awaiting processing. The rapid decline in value for Q2 suggests a swifter reduction in the second server's capacity. This pattern corresponds to the scenario where $\mu_2 > \mu_1$, indicating a higher departure rate from the second server.

2.2 Optimal control

The objective of this part is to find the optimal policy to dispatch incoming jobs.

- Bellman optimal equation that characterizes the optimal policy :

$$V_*(s) = \max\{r(s, a_1) + \gamma \sum_{s' \in S} p(s'|s, a_1)V_*(s'); r(s, a_2) + \gamma \sum_{s' \in S} p(s'|s, a_2)V_*(s')\}$$

- We solved numerically the optimality value function by Value Iteration Algorithm and here is the representation on the plane the optimal action as a function of the state (Q1, Q2) :

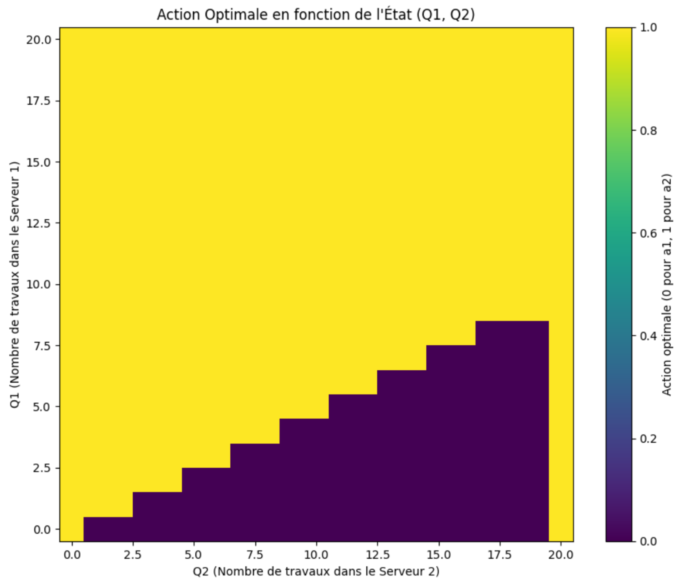


FIGURE 3 – Optimal action as a function of the state (Q1, Q2), using value iteration algorithm

In the graph representing actions under the optimal policy (fig.3), a clear trend is observed where the predominant action is a_2 (dispatch a potential new incoming job to server Q2). The trend of stepped structure for for a_1 signifies an initial equilibrium in action preference, gradually shifting towards favoring a_2 as both Q1 and Q2 become more occupied. This behavior underscores a tendency to prioritize a_2, likely due to the higher departure rate from Q2, aligning with the system's dynamics.

- In Figure 4 there is a comparison of the performances obtained with the policies.

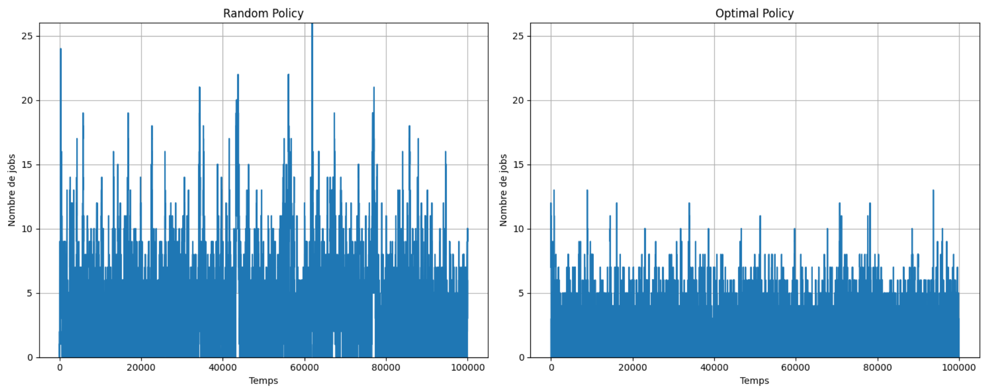


FIGURE 4 – Comparison between Random Policy and Optimal Policy in terms of jobs number

We simulated both the Random Policy and the Optimal Policy using the ServerEnvironment over 100,000 iterations. The average number of jobs observed under the Random Policy was 3.6, while under the Optimal Policy, it decreased significantly to 1.3. This difference in the average number of jobs can be attributed to the efficiency in job handling.

The Random Policy tends to generate a higher number of jobs on average compared to the Optimal Policy. This outcome signifies a more effective management and distribution of jobs under the Optimal Policy, resulting in a reduced workload and better server utilization. The Optimal Policy demonstrates a more balanced and efficient approach, leading to a lower average number of jobs over time compared to the Random Policy.

The Random Policy assigns an equal probability to each job to be sent to either of the servers, without considering the current load on the servers or the available capacity. It is not optimizing the utilization of available resources and not balancing the load effectively. The Optimal Policy is one that takes into account the current load on the servers when making decisions about the assignment of new jobs. It aims to maximize the cumulative value of states (reducing the load) while considering the servers' capacities.

To conclude, the optimal policy performs better than the random policy. The optimal policy manages to distribute the load efficiently between servers and maintain a balance between capacity and demand.

• Carrying out a one-step improvement on the random policy, we observe that a simple change to this policy affects performance (fig.5). Adjustments to the policy are made by taking into account the current load on the servers and the arrival and departure rates of jobs, in order to optimise the distribution of jobs between the servers. This improves operational performance, server load and reduces the cumulative value of states.

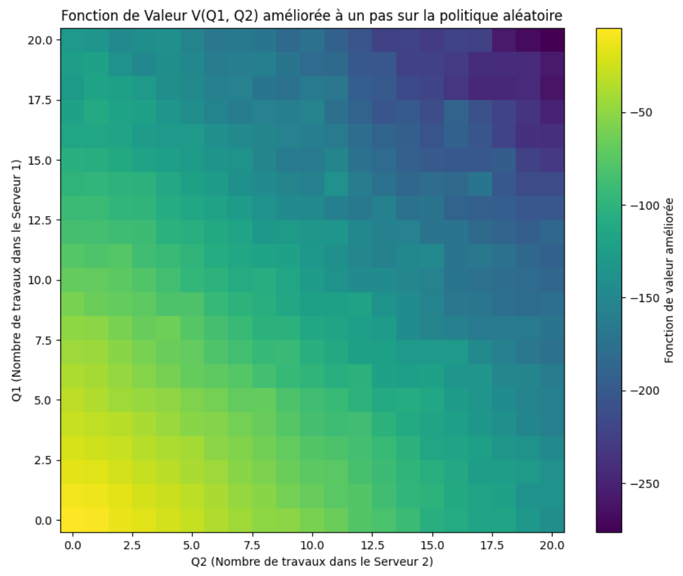


FIGURE 5 – Value function of the random policy with a one-step improvement

3 Tabular Model-Free control

3.1 Policy Evaluation

Assume the random policy that dispatches every job with probability 0.5 to either queue 1 and 2. For the learning parameter you can use $\alpha_n = 1/n$.

• We implement TD(0) which is a simple temporal-difference learning algorithm :

$$V(S) < -V(S) + \alpha(R + \gamma V(S') - V(S))$$

with the TD target : $R + \gamma V(S')$ and the TD error $\sigma = R + \gamma V(S') - V(S)$.

We experienced some problem, to implement the TD(0) method to have a good exploration with the parameters given in the subject we noticed that the servers almost never overload and the states visited concentrated between (0,0) and (10,10) usually. To have a better view of every state, we tried to reset the server state randomly at the beginning of each episode. The next paragraphs shows the results we obtain with this environment implementation.

• In Figure 6, we have the value function with TD0 and $\alpha_n = 1/n$. The value function does not look like the ones calculated in the first part and convergence does not seem to be good. Although we can still notice that there is a higher cost for the algorithm to load server 1 than server 2 probably because server 2 is more efficient.

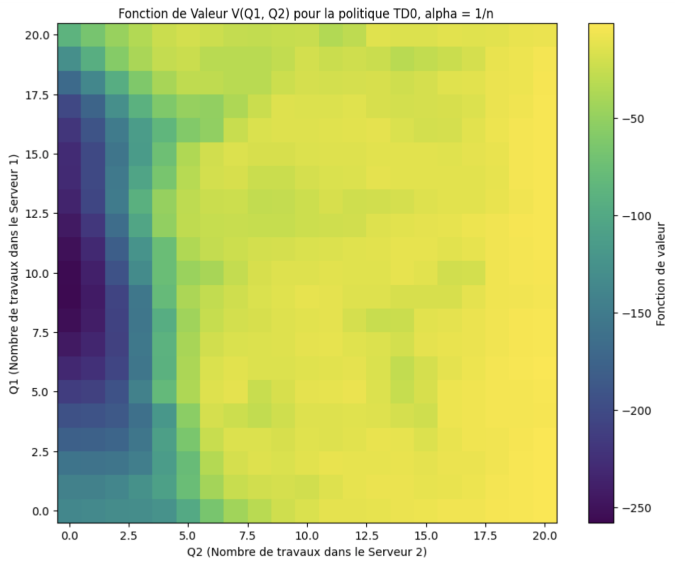


FIGURE 6 – Value function of TD(0) policy with $\alpha_n = 1/n$

• We explored other alternatives for α_n, like :

— α_n constant=, with $\alpha_n = 0.01$ in Figure 7. The convergence is really better here and the value function look more like the value functions calculated in section 1.

— $\alpha_n = 1/n^\gamma$, to see if convergence improves. With $\alpha_n = 1/n \cdot 5$ convergence improves but the value function is still different than in section 1. Although, we notice that the algorithm estimate that the cost to load server 1 is worst than the cost to load server 2.

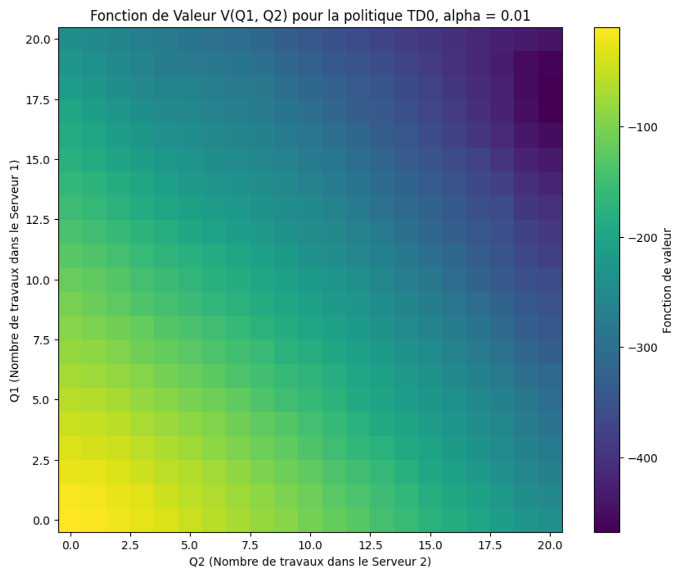


FIGURE 7 – Value function of TD(0) policy with $\alpha_n = 0.01$

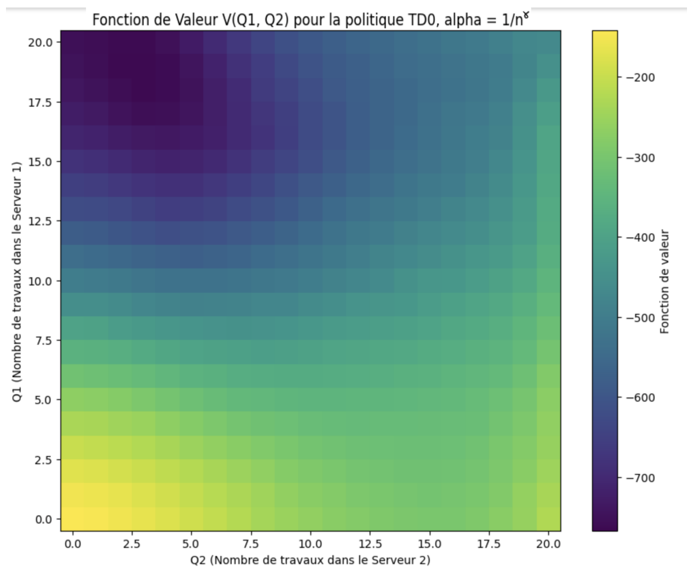
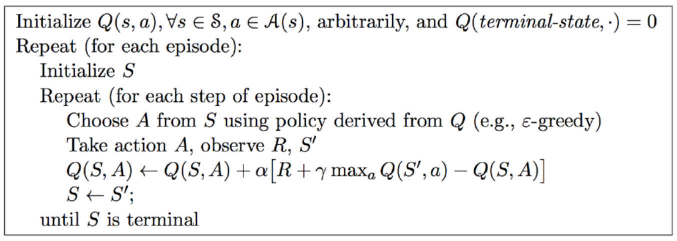


FIGURE 8 – Value function of TD(0) policy with $\alpha_n = 1/n{\cdot}5$

3.2 Optimal control

In this part you are asked to find the optimal policy to dispatch incoming jobs.

• We implemented Q-learning method, we choose to use a learning rate constant because the value function from TD(0) algorithm was the closest to section 1 and we choose the UCB method because the previous TPs showed us that it was the best method :



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$;
  until $S$ is terminal
```

• In Figure 9, is represented on the plane the optimal action as a function of the state (Q1, Q2). It is really different from the optimal policy of Section 1. We can still notice that the action to load server 2 is more represented than the action to load server 1 and when the server 1 is full but the server 2 is empty, it understands well that it needs to load server 2.

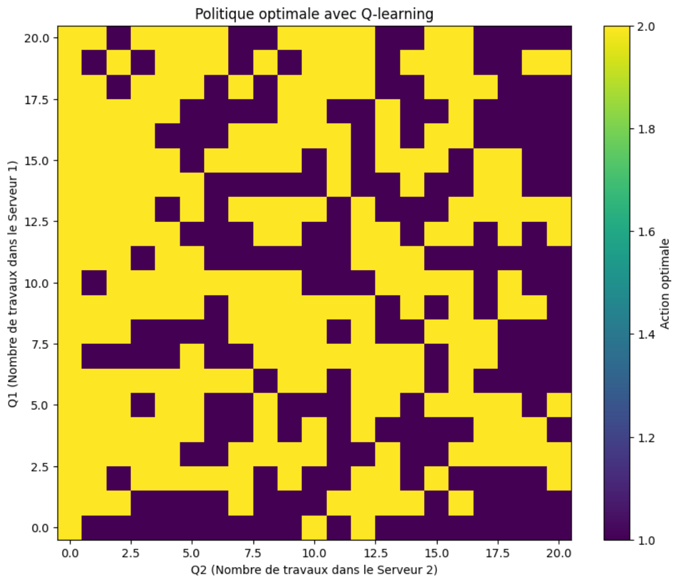


FIGURE 9 – Optimal action as a function of the state (Q1, Q2), using Q-learning algorithm

The actions learned from the q-learning method is better than the random policy but, we can also see that the number of jobs is still higher than with optimal policy of Section 1, with the Figure 10, and a mean number of jobs of 1.4.

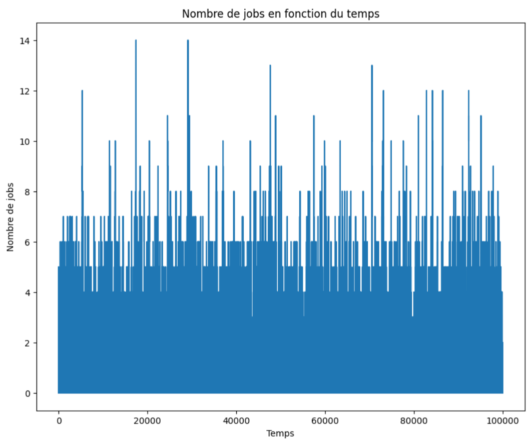


FIGURE 10 – Jobs number with Q-learning algorithm

4 Model-free control with Value Function/Policy approximation

Our objective is to devise a scheme that can learn the optimal policy as found in Section 1 more efficiently than in Section 2 using a non-tabular approach. Q-learning is not very efficient in learning the optimal policy for this problem.

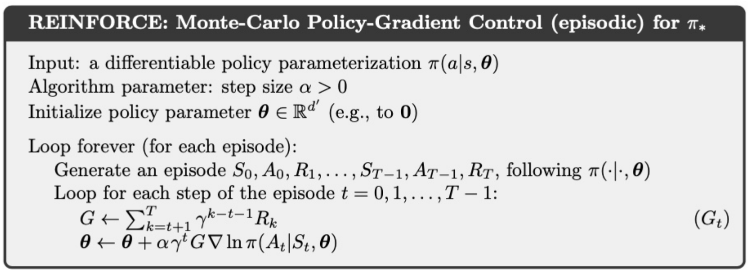
We have decided to use the softmax parametrization method for policy approximation in reinforcement learning. This technique involves representing the policy using a softmax function over action preferences or action values. The softmax function converts the values into a probability distribution over the actions. The parameters of the softmax function are adjusted through learning to achieve better performance.

$$\pi_\theta(a|s) = \frac{e^{\theta^T.x(s,a)}}{\sum_c e^{\theta^T.x(s,c)}}.$$

Then, we follow a Stochastic Gradient Ascent method, the gradient is calculated with :

$$\nabla_\theta log(\pi_\theta(a|s)) = x(s,a) - E^{\pi_\theta}(x(s,A)) = (s) * (a - \theta^t.x(s))$$

$x(s)$ is the softmax probabilities for the state s.



REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

```
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
        $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$                    $(G_t)$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$
```

FIGURE 11 – Algorithm of Monte-Carlo Policy-Gradient Method

We update the policy parameters using the reinforce update rule :

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta log(\pi_\theta(A_t|S_t))G_t$$

G_t is calculated with $G_{t+1} = G_t + \gamma * Reward_t$

It was really challenging to implement this method. Once we had an algorithm that was working a little bit, we noticed the importance of the hyperparameters. The learning was unstable and was sometimes producing good results such as figure 12, and sometimes less good such as figure 13. We can see that the algorithm usually understands the importance to balance the load between the 2 servers but has some trouble to determine the good border. We tried to modifiy the different hyperparameters such as number of episodes, learning rate, length of the trajectory but the algorithm still does not converge every time to figure 12 witch is the closest result we have to the optimal actions.

To conclude, we found that Monte-Carlo Policy-Gradient Control was the best solution to approximate the optimal actions even though our implementation is still unstable sometimes.

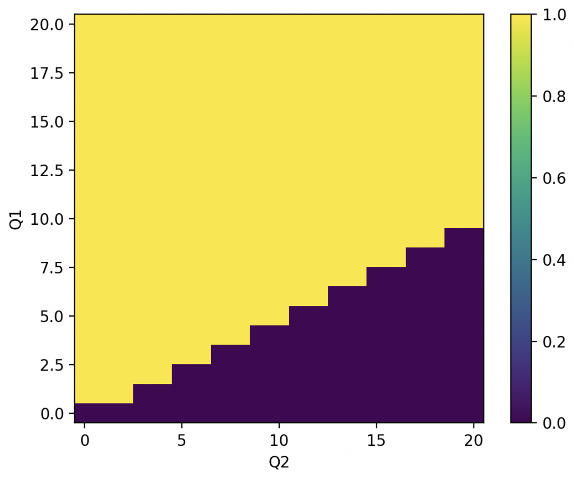


Figure 12 – Optimal action as a function of the state (Q1, Q2), using Softmax Policy Parametrization

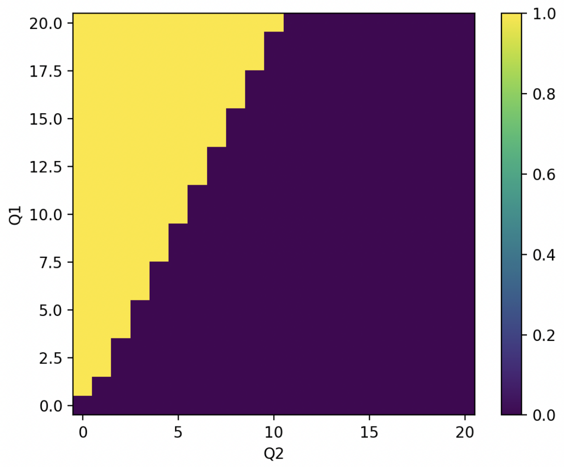


Figure 13 – Optimal action as a function of the state (Q1, Q2), using Softmax Policy Parametrization

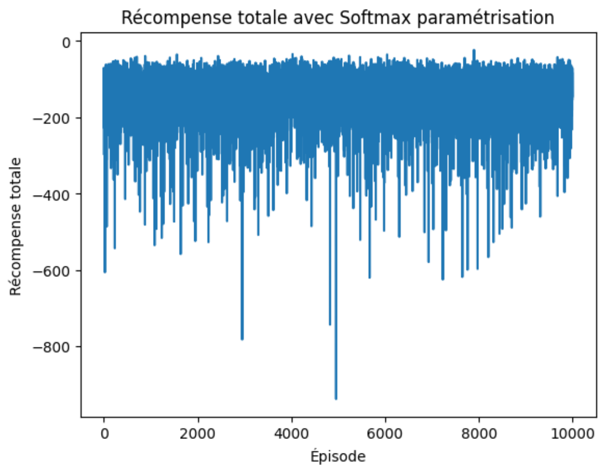


Figure 14 – Total reward over episodes