Report 09/06 : Reinforcement learning for Cache-Friendly Recommendations

Clément Bernard and Jade Bonnet

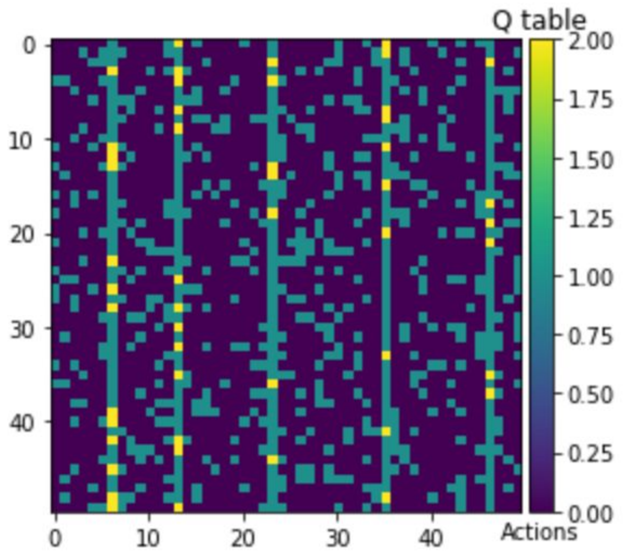
We’ve spent times to check if they were mistakes in our code, and we didn’t find clear mistakes.
Therefore, we think what was wrong is the hyperparameters choice like the learning rate.
The previous learning rate we used was **1e-3**, and the loss was decreasing, but not the rewards.
That’s why we tried to reduce the learning rate and tried it with either a linear model or a fully connected layer (like the one used previously with 100 neurons in the hidden layer).

Previous Q Learning results

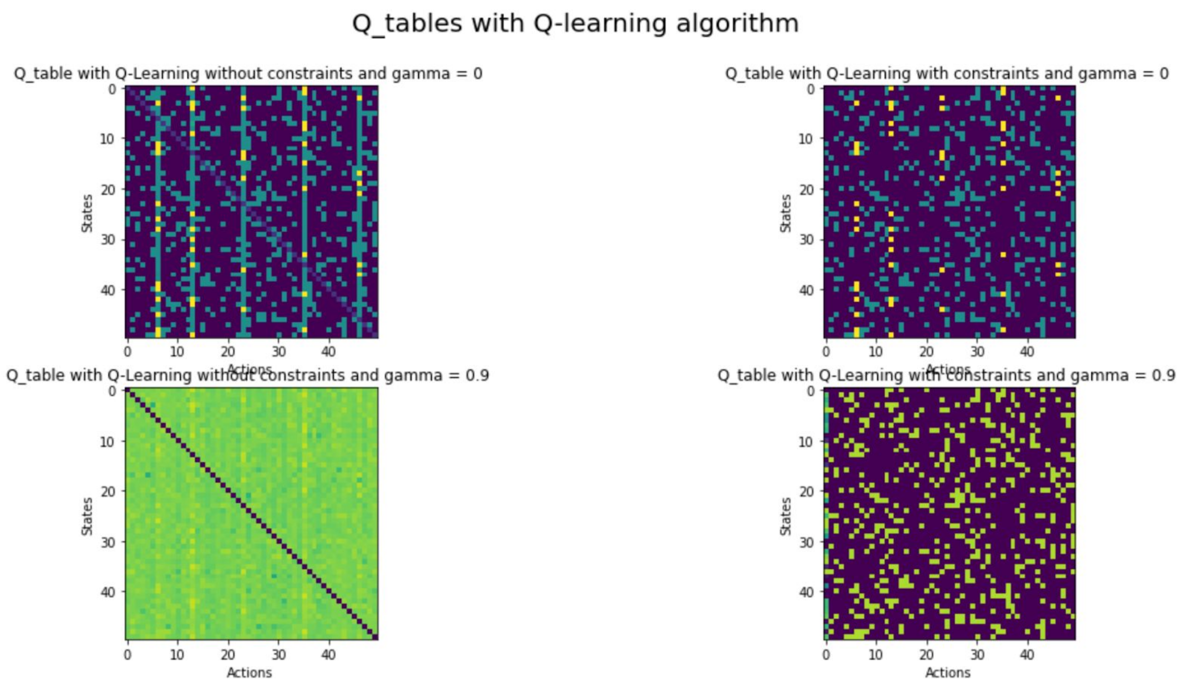
To have a good way of comparison, we generated a environment with a U matrix a ran the algorithm with the classic Q-Learning algorithm.
The hyperparameters for the environment are the following :

- <u>Catalogue size</u> : **50**
- <u>Alpha</u> : **0.6** (probability that he listens to our suggestions)
- <u>Number of episodes</u> : **10** (probability to leave = 0.1)
- <u>Number of related contents</u> : **10**
- <u>Number of cached contents</u> : **5**
- <u>Rewards</u> : **[2, 1 , 1 , 0]**

Here are the results we got from these algorithms :

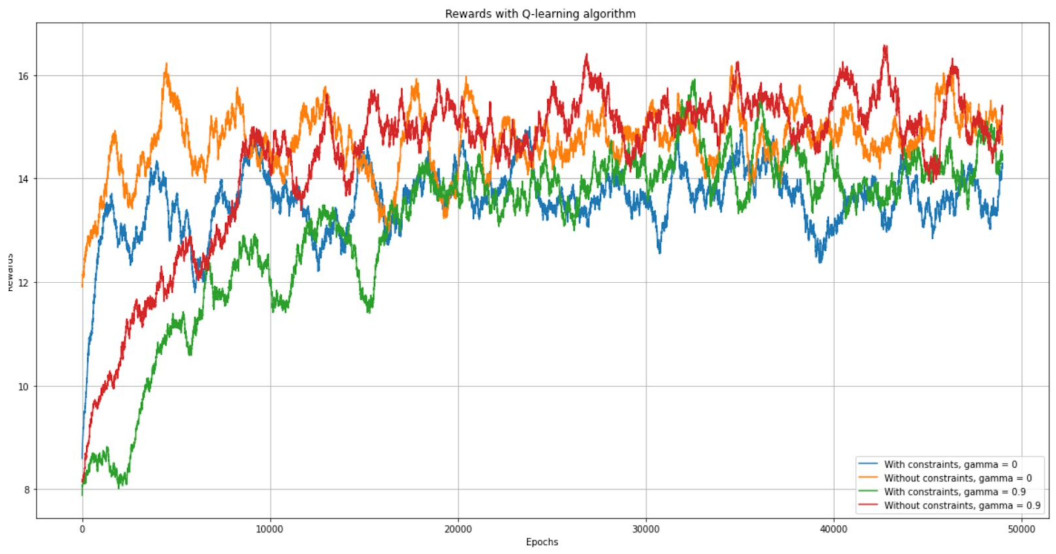


<u>Rewards matrix</u>



Q table with either gamma = 0 or 0.9 with Q Learning algorithm

Note : the q table for gamma = 0 and without constraints is exactly the reward matrix.
When gamma = 0.9 the values are quite close to each other but we can see the vertical lines of the cached contents.
Then the rewards through the epochs :



Rewards with Q Learning algorithm

So the goal of the deep q learning algorithm is to have a reward per epoch around **15**.

Deep Q learning

We compared this with either a Linear model or a fully connected model.
To do so, we set the hyperparameters of these models as follows :

- Memory size : **50**
- Epsilon-Greedy : **0.1**
- Learning rate : **1e-4**
- Batch-size : **10**

1) Linear Model

We tried the exact same algorithm with this architecture :

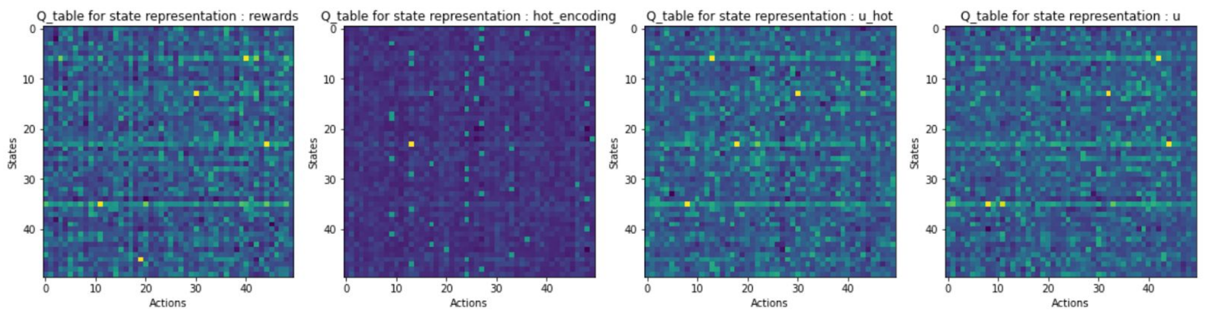
$$Q(S_t, A_t) = \sum_i \phi(S_t) w_i$$

We tried with gamma = 0 and gamma = 0.9 and then compared the loss and rewards.
We used four different representations of the states : the **one hot encoding**, the **U matrix (hot encoded and not)** and the **rewards** representation.
We remind here the representation formulas :

- **Hot encoding :** $\Phi(S_t = i) = [0, \ldots, 1, \ldots, 0]$
- **Full U matrix :** $\Phi(S_t = i) = u_i$
- **Hot encoded matrix :** $\Phi(S_t = i) = (\delta_{i,j})_{1 \le j \le n}$ where $\delta_{i,j} \begin{cases} 1 & \text{if } u_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$
- **Rewards :**

$$\Phi(S_t = i) = (\delta_{i,j})_{1 \le j \le n} \text{ where } \delta_{i,j} \begin{cases} 2 & \text{if j is cached and related} \\ 1 & \text{if j is cached and not related or related and not cached} \\ 0 & \text{otherwise} \end{cases}$$

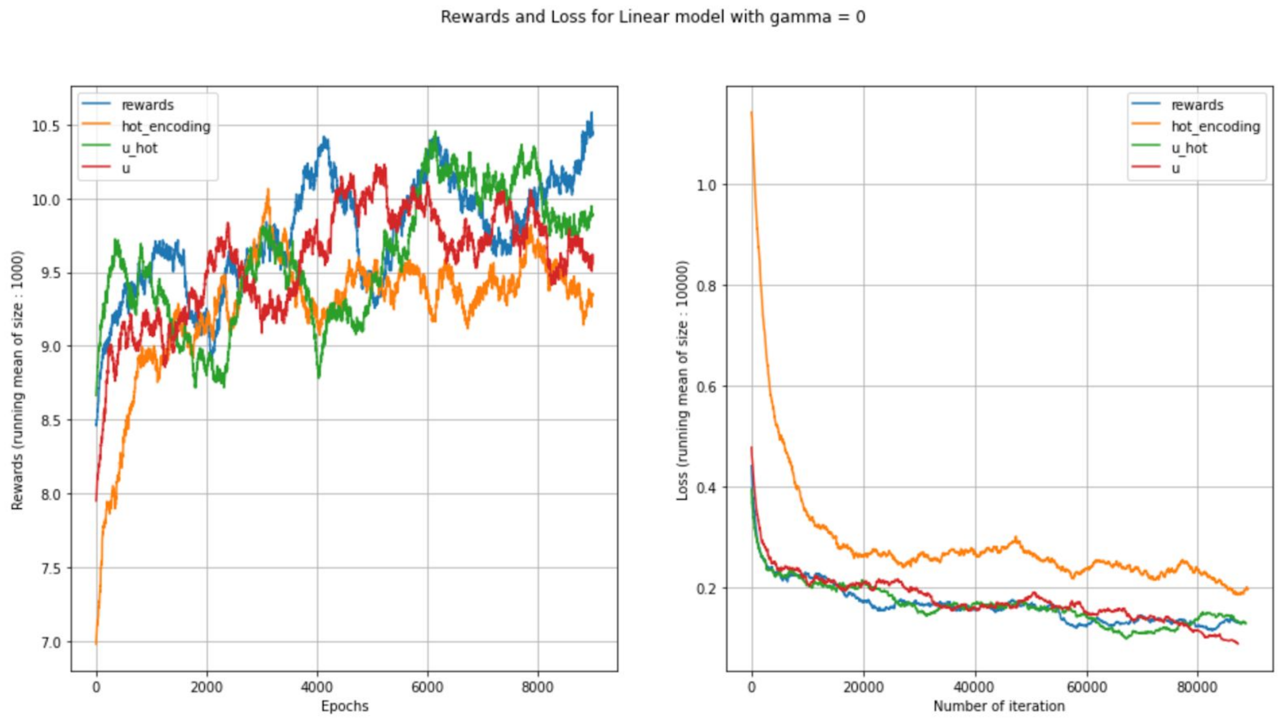
a) *Gamma = 0 and learning rate = 1e-4*



Q_tables for different states representations with deep q learning algorithm

We infer from these plots that horizontal lines have appeared for three algorithms. They have learnt almost the same q table at the end. It corresponds to the states that are cached, which means that with a linear model, it learns the fact that almost each action from a cached state will lead to high current rewards (gamma = 0 here).

Furtheremore, the rewards and the loss have a behaviour which makes sense : the loss is decreasing and the reward is increasing :

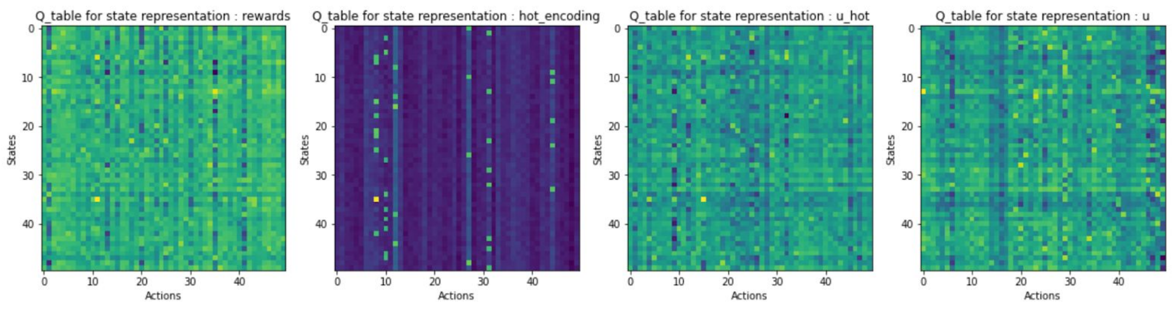


Rewards and Loss for different states representations with deep q learning algorithm

We see that the rewards are around 10.5 and not 15 as with the Q learning algorithm.

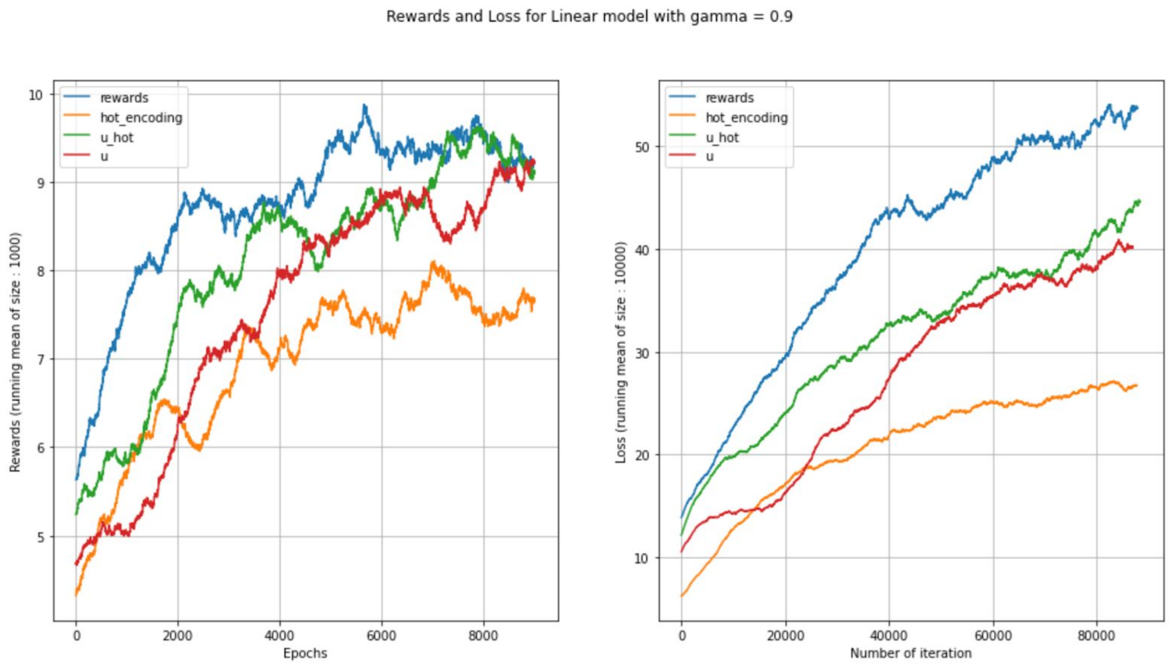
b) *Gamma = 0.9 and learning rate = 1e-4*

Then, we tried to learn the same q table but with gamma = 0.9.
The behaviour is very different and the algorithms didn't learn well.



Q_tables for different states representations with deep q learning algorithm

Just like before, the results for the hot encoding states are not consistent with the other algorithms.
The final q tables are quite messy and don't learn well. Is it because this is just a linear model ?



<u>Rewards and Loss for different states representations with deep q learning algorithm</u>

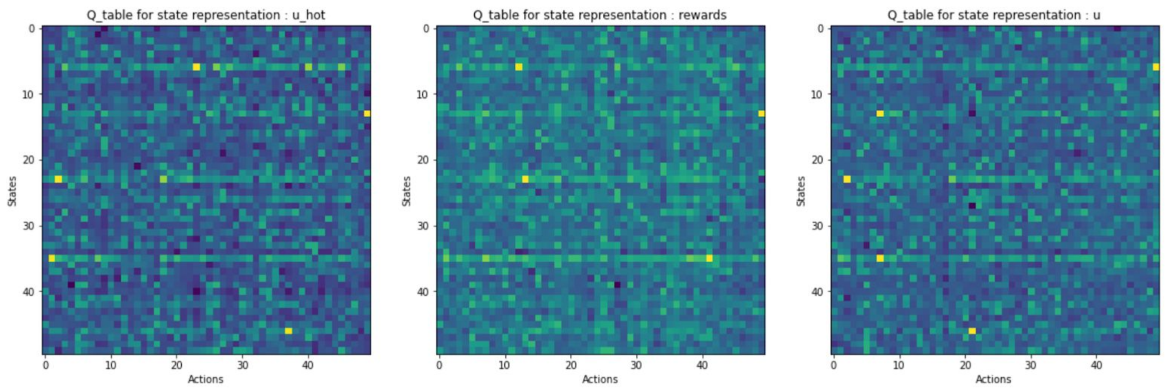
We can see with the rewards and the loss that the rewards per epoch are increasing for each algorithm, but the loss is also increasing. Such behavior doesn't really make sense. Maybe it can be explained by a learning rate either too high or too low ?

Then, we tried again with a fully connected architecture.

2) Fully connected model

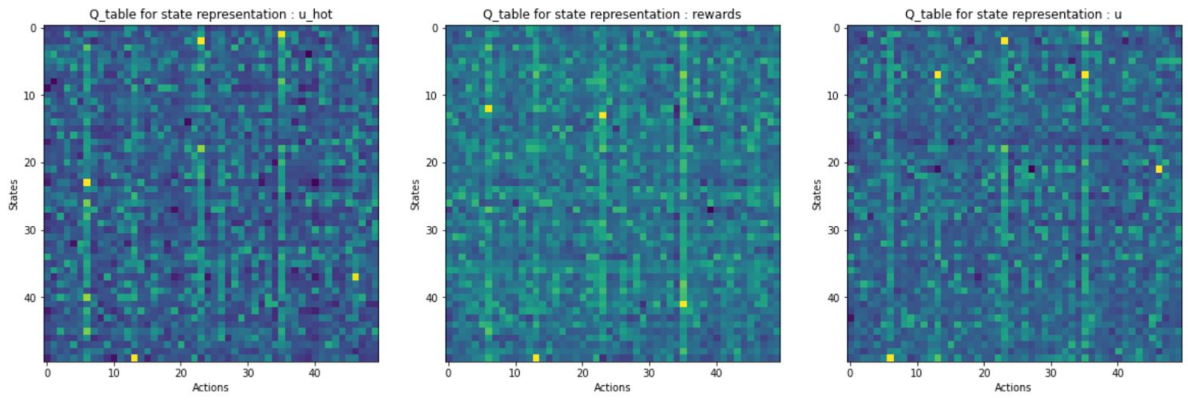
a) <u>Gamma = 0 and learning rate = 1e-4</u>

We did the same algorithm as before but with a fully connected layer between the states and the outputs. We did it with three representation of the states : the **U matrix** (one hot encoded or not) and the **rewards**.
Here are the results for gamma = 0 and a learning rate of 1e-4 :

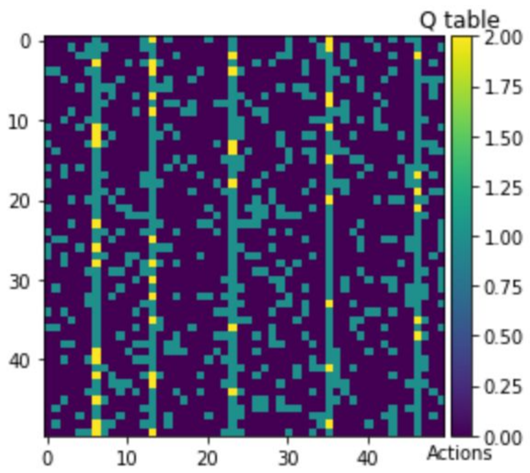


Q_tables for different states representations with fully connected deep q learning algorithm

It seems that it has learnt the same Q values than a linear model ! Furthermore, it really looks like the reward matrix if we take the transpose :

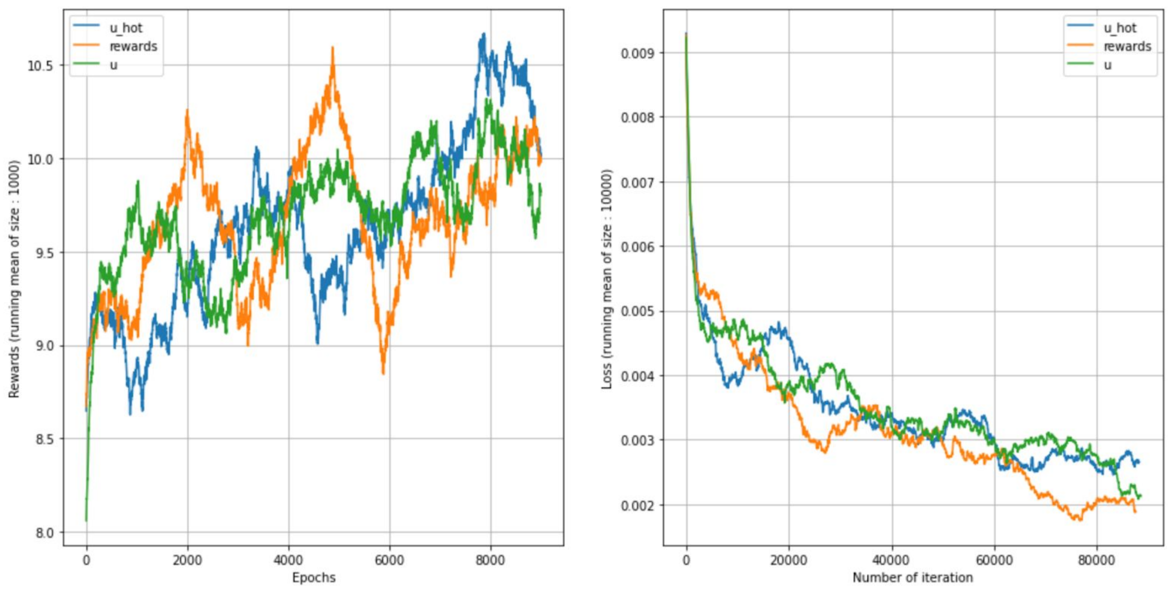


Transposed q tables for fully connected model for different states representation



Reward matrix

Furthermore, the behavior of the rewards and the loss does the same thing than the linear model used before :



Rewards and Loss for different states representations with fully connected deep q learning algorithm

In fact the rewards tend to be the same (around 10) but the loss is slightly different : it starts from 0.009 and ends around 0.003 (the learning rate is equal to 1e-4, which makes sense). We don't know if we should consider a higher learning rate, which could tend to end to forget the states and predict the same thing for each state.
To have an idea of the values we got, here is a comparison of the Q values for the state 13 (a cached one) and the for 10 first actions (we took the transpose q tables):

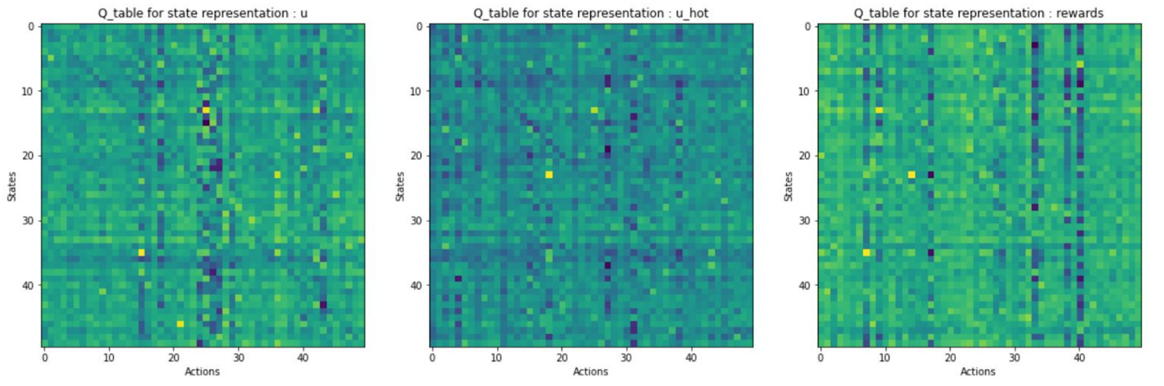
	u_hot	rewards	u	Q with constraints	Q without constraints	Reward
0	0.363220	0.965272	0.961583	1.0	1.0	1.0
1	-0.054580	0.205780	0.101139	0.0	0.0	0.0
2	-0.059671	-0.277395	-0.136941	0.0	0.0	0.0
3	0.034383	0.454041	0.141947	0.0	0.0	0.0
4	0.029950	0.081186	-0.034178	0.0	0.0	0.0
5	0.493785	0.680766	0.551183	1.0	1.0	1.0
6	0.659651	0.667273	0.707874	2.0	2.0	2.0
7	0.502488	0.232933	0.509695	1.0	1.0	1.0
8	0.073611	0.094853	0.194129	0.0	0.0	0.0
9	-0.148849	0.299082	0.023642	0.0	0.0	0.0

10 first Q values for state 13 with q learning algorithm and with fully connected deep q learning algorithm

It seems that it hasn't learnt enough to have the expected outputs.

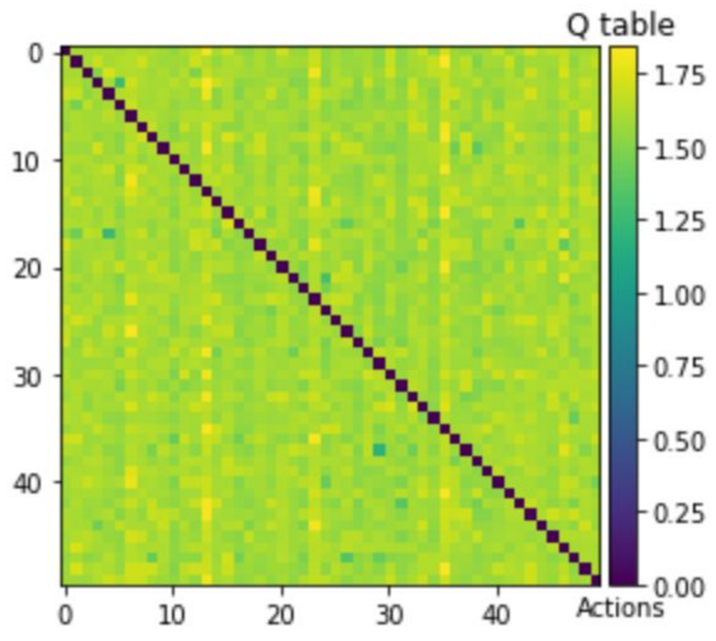
c) *Gamma = 0.9 and learning rate = 1e-4*

Finally, we tried with gamma = 0.9. In fact, the behaviour of the algorithm is almost the same as with a linear architecture :



Q_tables for different states representations with fully connected deep q learning algorithm and gamma = 0.9

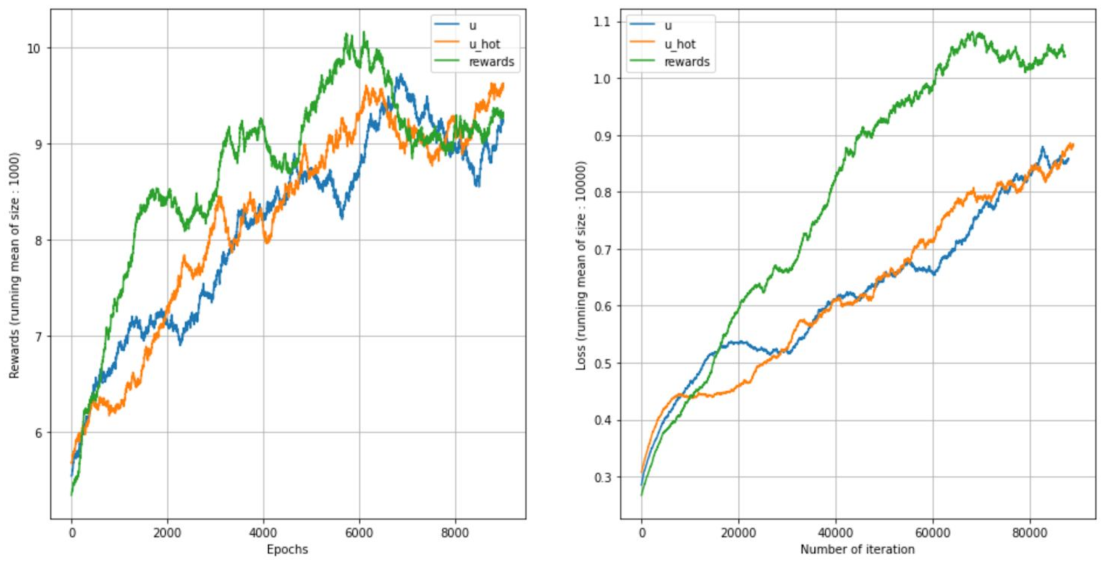
We remind the expected output (given by the q learning algorithm without any constraint) :



Q tables for gamma = 0.9 with q learning algorithm

We can see that with the deep q learning algorithm, the diagonal tends to be equal to 0 (which makes sense). Then, some vertical lines have appeared, but not for the good actions.

Furthermore, another strange behaviour is that the rewards are increasing, just like the loss :



Rewards and Loss for different states representations with fully connected deep q learning algorithm

The rewards tend to be around 10, but the loss is increasing (whereas the learning rate is equal to 1e-4, so the scale of loss is really greather than the learning rate).
Finally, the Q values are at least consistent for the different state representations :

	u	u_hot	rewards	Q without constraints	Q with constraints	Reward
0	6.922681	6.761707	7.434882	1.675830	1.754500	1.0
1	7.183261	6.320674	8.194822	1.575607	0.000000	0.0
2	5.957031	5.622552	7.683443	1.566995	0.000000	0.0
3	6.598090	6.673605	8.524837	1.607597	0.000000	0.0
4	6.607598	5.882981	7.605851	1.610434	0.000000	0.0
5	6.283659	6.978783	7.230176	1.615704	1.748817	1.0
6	6.566412	6.576182	7.794570	1.732465	1.755284	2.0
7	7.135455	5.504116	6.097948	1.659105	1.753532	1.0
8	6.276192	6.053266	7.587946	1.580943	0.000000	0.0
9	7.282988	6.808311	9.563796	1.622672	0.000000	0.0

10 first Q values for state 13 with q learning algorithm and with fully connected deep q learning algorithm (gamma = 0.9)

The q values are all around 7 for the deep q learning algorithm. These values are far from the expected rewards that could be around 1.6-1.7 according to the classic q learning algorithm. It could be explained by the loss that is increasing, but doesn’t make sense with the reward that is increasing.