Report 16/06 : Reinforcement learning for Cache-Friendly Recommendations

Clément Bernard and Jade Bonnet

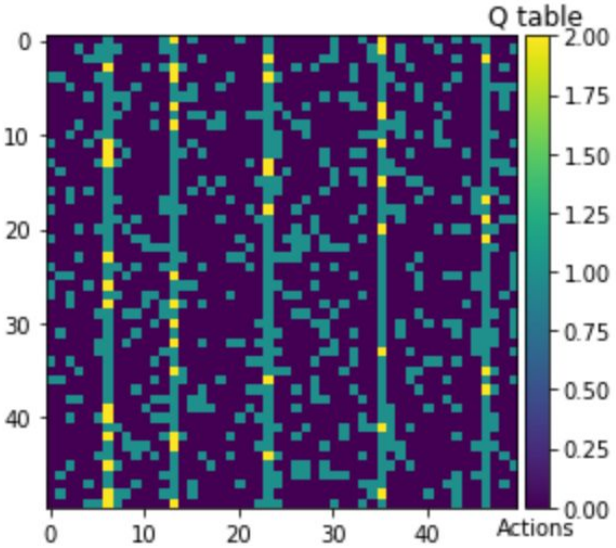
The code available is in our github :
https://github.com/clementbernardd/network_friendly/blob/master/code/deep_q_learning/rendu1606.ipynb
We've spent times to check if they were mistakes in our code, and we didn't find clear mistakes.
Therefore, we think what was wrong is the hyperparameters choice like the learning rate.
The previous learning rate we used was **1e-3** with **Adam** as optimizer, 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

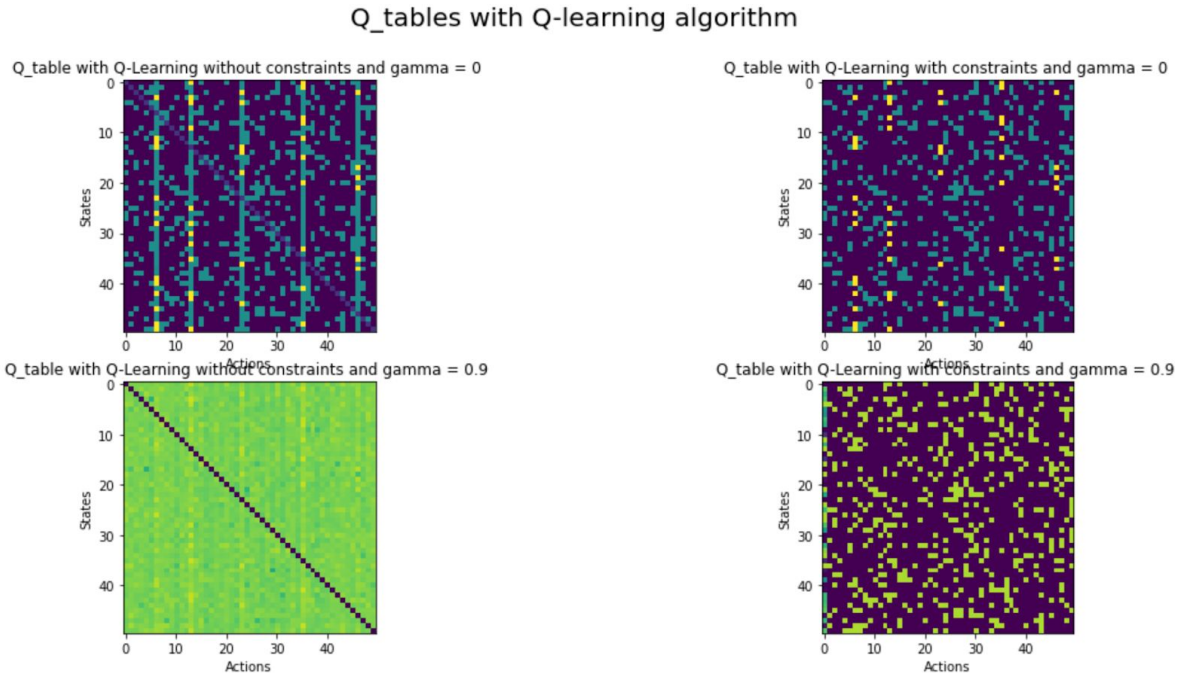
To have a good way of comparison, we generated a environment with a U matrix a ran the algorithm with the classic Q-Learning algorithm.
The hyperparameters for the environment are the following :

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

Here are the results we got from these algorithms :

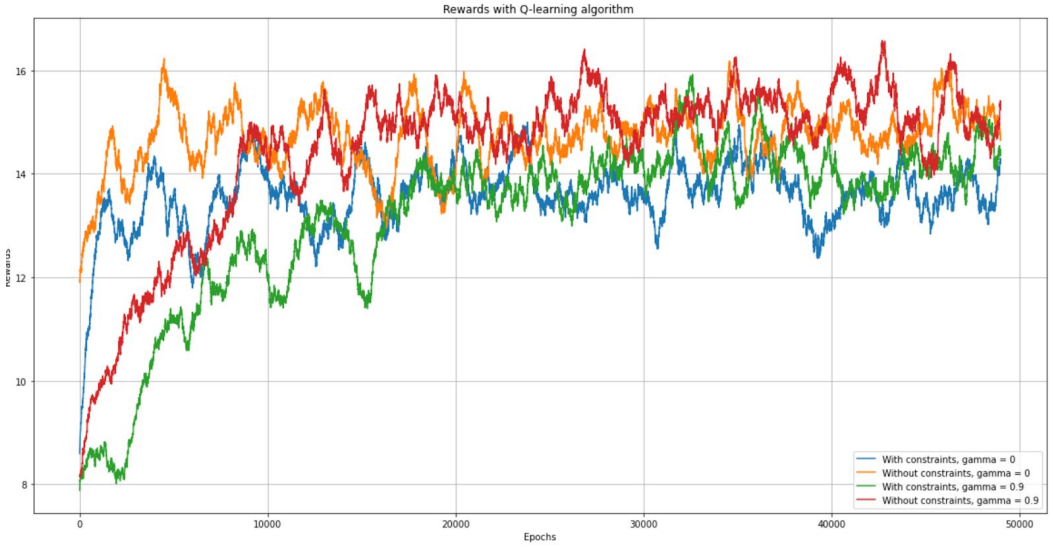


Rewards matrix



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.
We used four different representations of the states : the **one hot encoding**, the **U matrix (hot encoded)**, the **rewards** and the **valuable** representation.
We remind here the representation formulas :

- **Hot encoding :** $\Phi(S_t = i) = [0, \dots, 1, \dots, 0]$
- **Hot encoded matrix :**

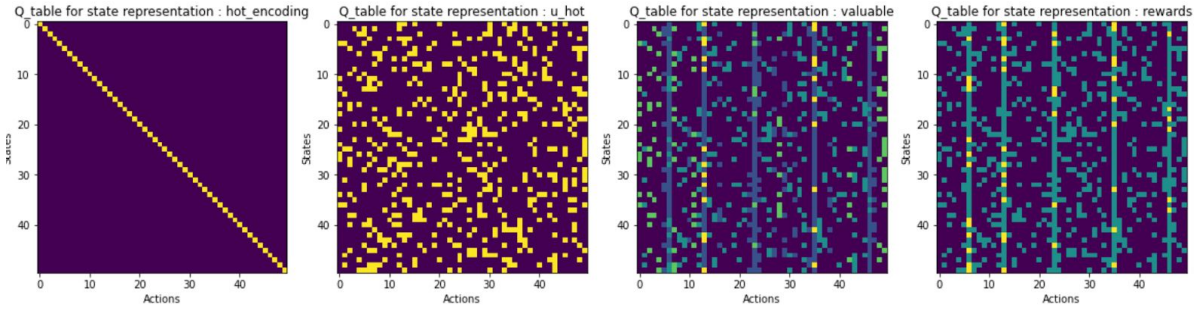
$$\Phi(S_t = i) = (\delta_{i,j})_{1 \leq j \leq n} \text{ where } \delta_{i,j} \begin{cases} 1 & \text{if } u_{i,j} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Rewards :**

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

- **Valuable :** It replaces each state by an array where each actions is :
  - +1 if this is related to the state
  - +1 if this is cached
  - +1 if the next state it leads to has related contents cached

Here is the representation of the states in a plot :



Different representation of the states

1) Linear Model

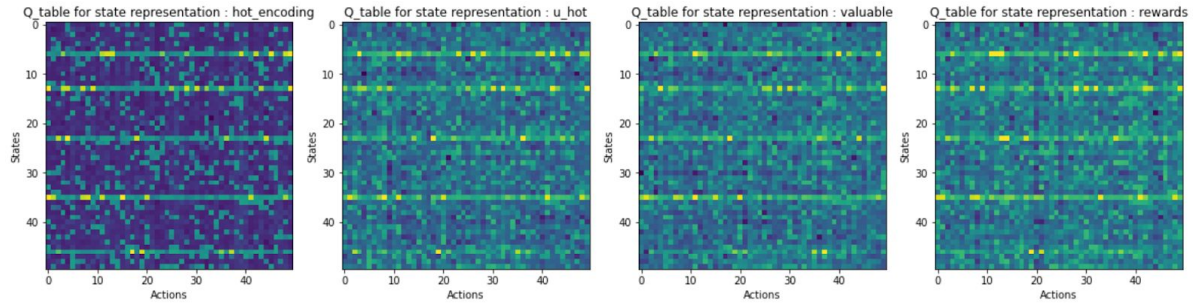
We used the hyperparameters as follows :

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

We tried the exact same algorithm with a simple Linear class from the states and the outputs.

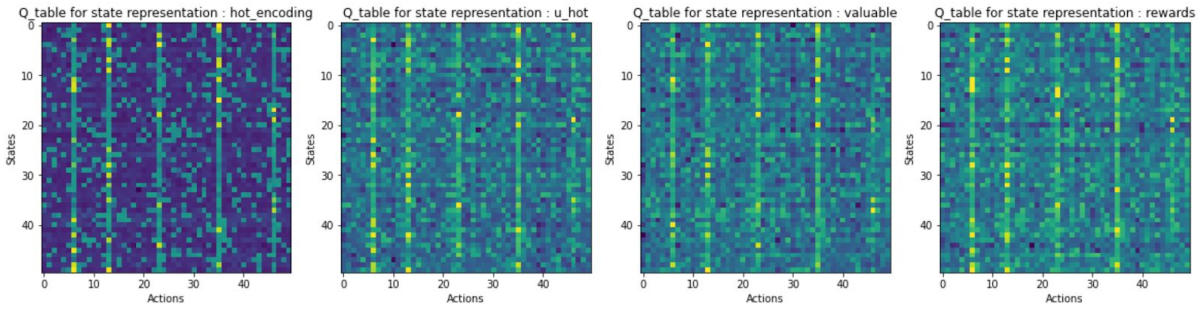
We tried with gamma = 0 and gamma = 0.9 and then compared the loss and rewards.

a) Gamma = 0, SGD optimiser and learning rate = 5e-3



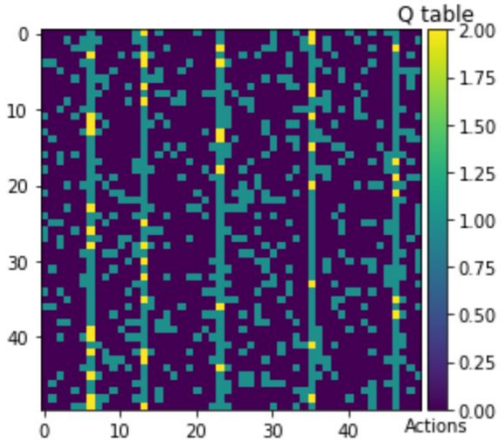
Q_tables for different states representations with deep q learning algorithm

As we replace a state by a representation of the actions, the algorithm seems to learn a representation of the actions, and not the states. Here is the results of the transpose of the q_tables :



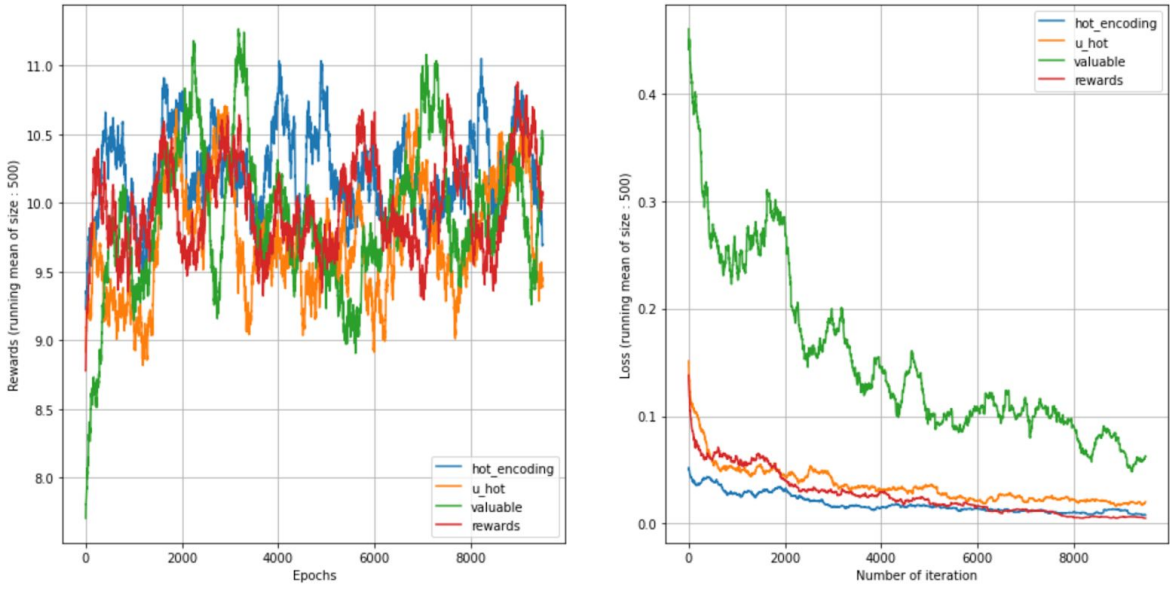
Transpose q_tables for different states representations with deep q learning algorithm

To have a comparison, we remind the rewards matrix :



Rewards matrix

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

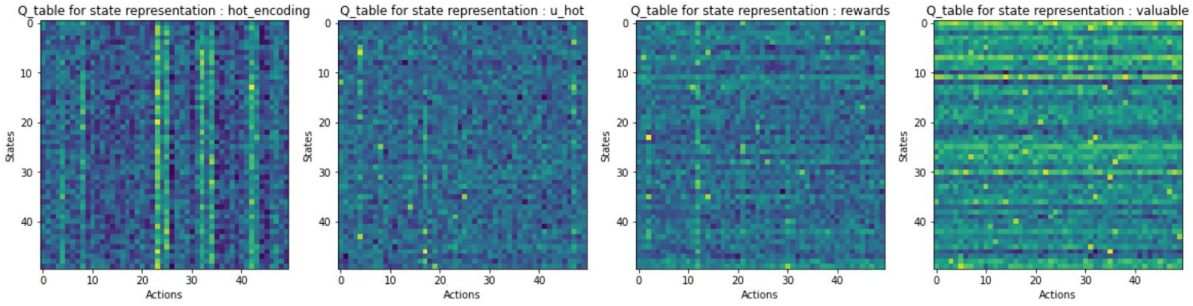


Rewards and Loss for different states representations with deep q learning algorithm

We see that the rewards are around 10 and not 15 as with the Q learning algorithm.
The ***valuable*** representation seems slower to learn. Indeed, the representation is quite complicated for a very simple task. On the other hand, the ***one hot encoding*** representation converges very fast and is very close to the rewards matrix.

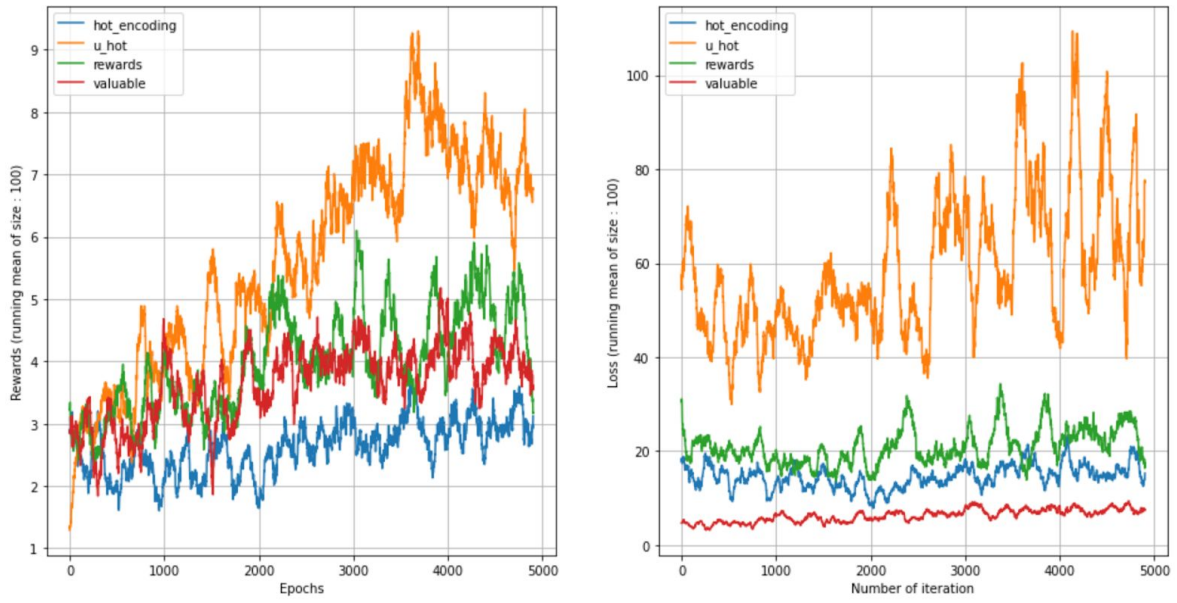
b) Gamma = 0.9, SGD optimiser and learning rate = 1e-4

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 linear model and gamma = 0.9

The results are not consistent and it doesn't seem to learn well.
Furthermore, both the rewards and the loss are increasing, which doesn't make sense for the loss.

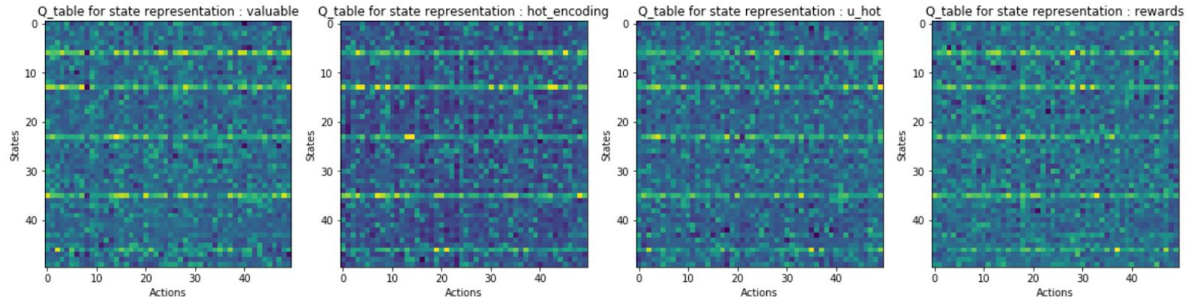


Rewards and Loss for different states representations with deep q learning algorithm

2) Fully connected model

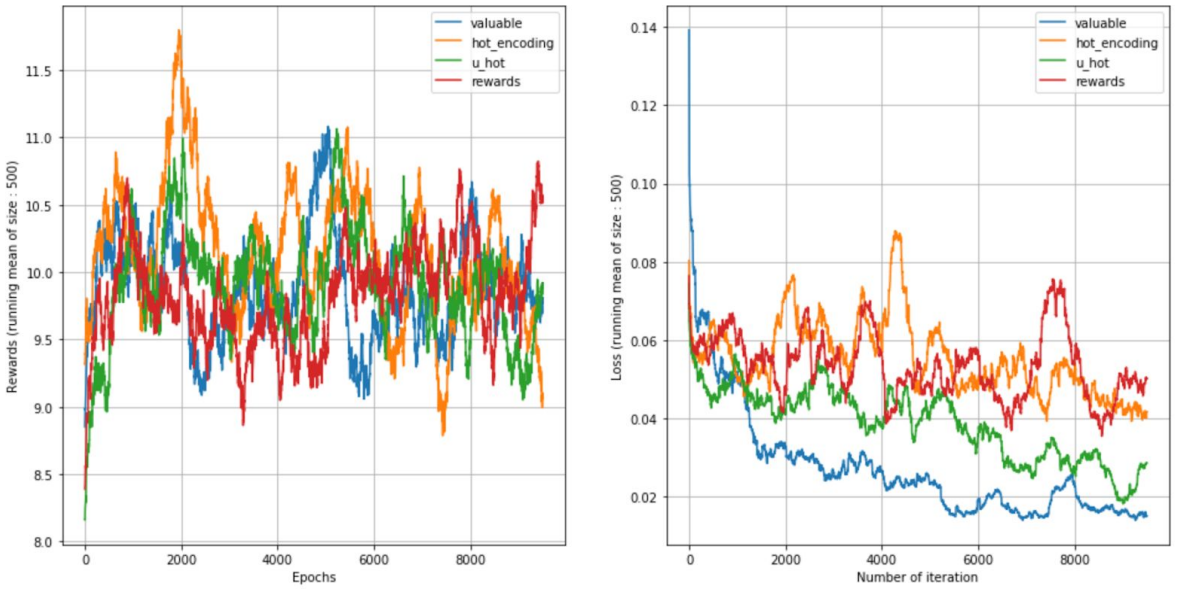
a) *Gamma = 0, Adam optimiser and learning rate = 5e-4*

We did the same algorithm as before but with a fully connected layer between the states and the outputs.
Here are the results :



Q_tables for different states representations with fully connected layers

It seems that it has learnt the same Q values than a linear model !
Furthermore, the behavior of the rewards and the loss does the same thing than the linear model used before :



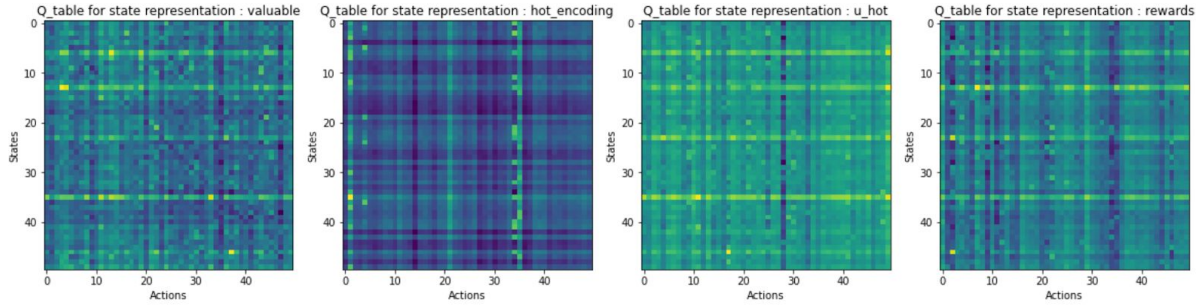
Rewards and Loss for different states representations with fully connected deep q learning algorithm

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

We used the following hyperparameters :

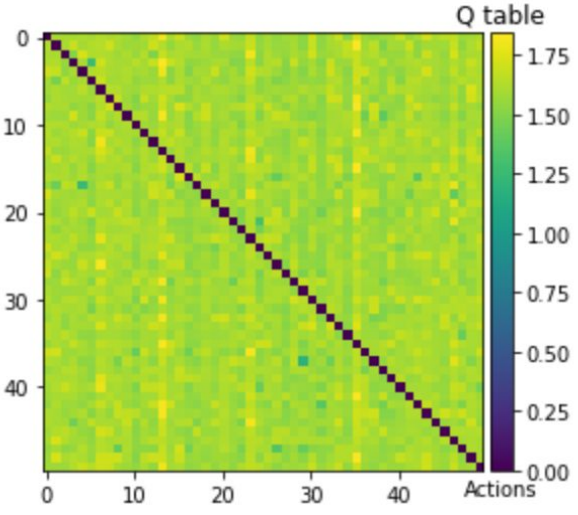
- Memory size : **200**
- Epsilon-Greedy : **0.1**
- Learning rate : **1e-4**
- Batch-size : **10**

Here are the q tables we got for 10 000 epochs :



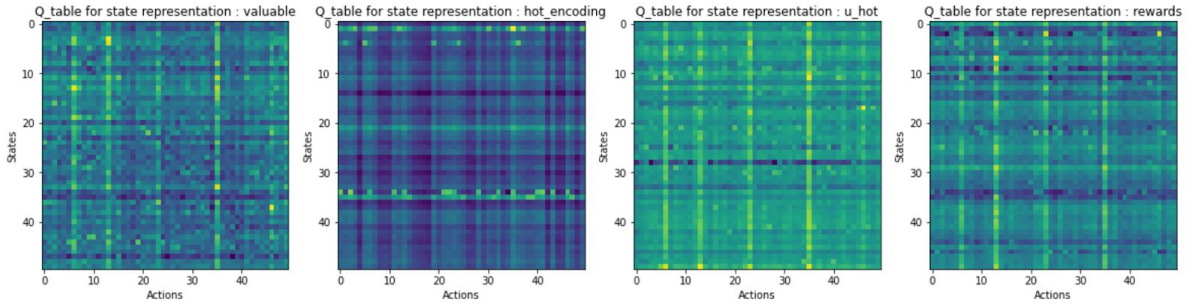
Q_tables for different states representations with fully connected deep q learning algorithm and gamma = 0.9, SGD (lr = 1e-4)

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



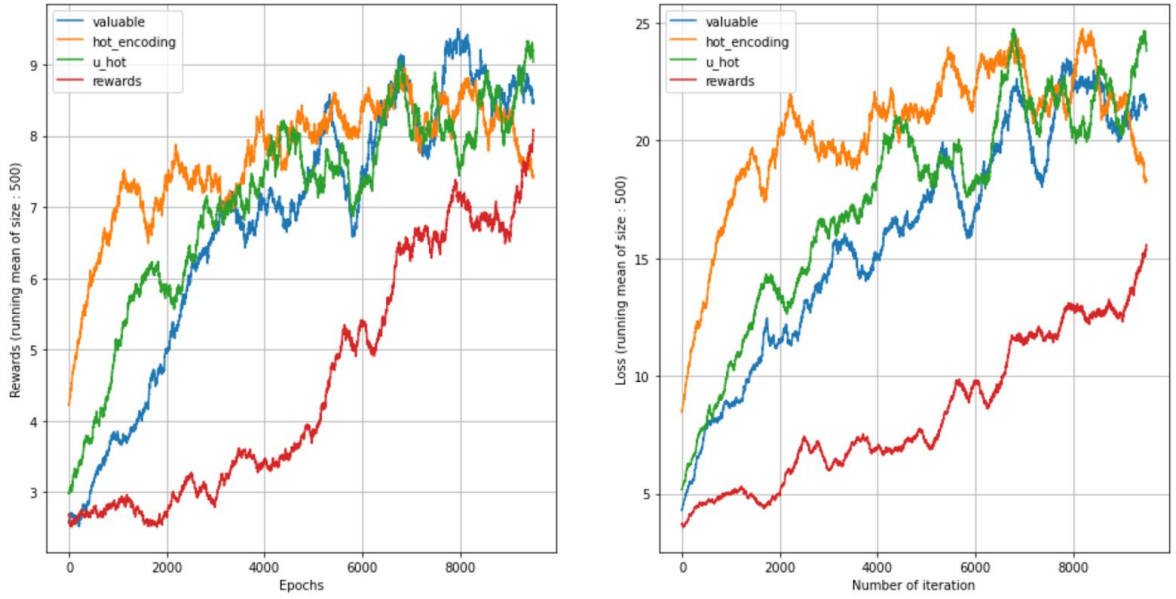
Q tables for gamma = 0.9 with q learning algorithm

Just like we did above, we compare with the transpose q-tables :



Transposed Q_tables for different states representations with fully connected deep q learning algorithm and gamma = 0.9, SGD (lr = 1e-4)

Some vertical lines have appeared (except for the one hot encoding representation), which corresponds to the ones that are cached. It seems close to the expected matrix, but not as obvious as with gamma = 0.
Furthermore, another strange behaviour is that the rewards are increasing, just like the loss :



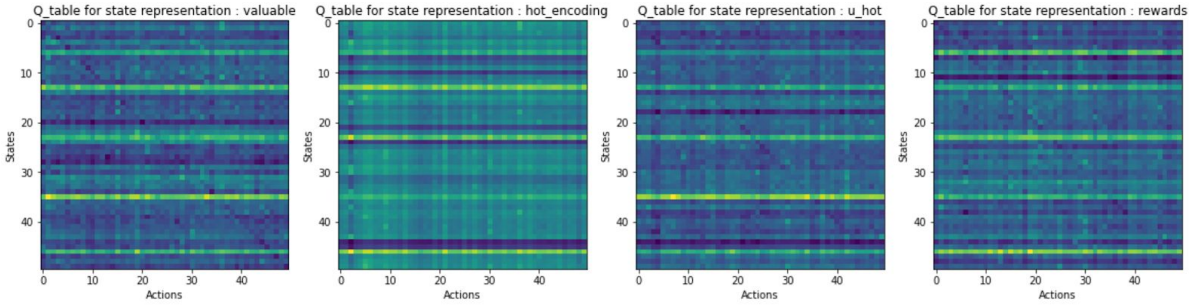
Rewards and Loss for different states representations with fully connected deep q learning algorithm (SGD and lr = 1e-4)

c) *Gamma = 0.9, Adam optimizer and learning rate = 1e-5*

We tried with another optimizer, with the following hyperparameters :

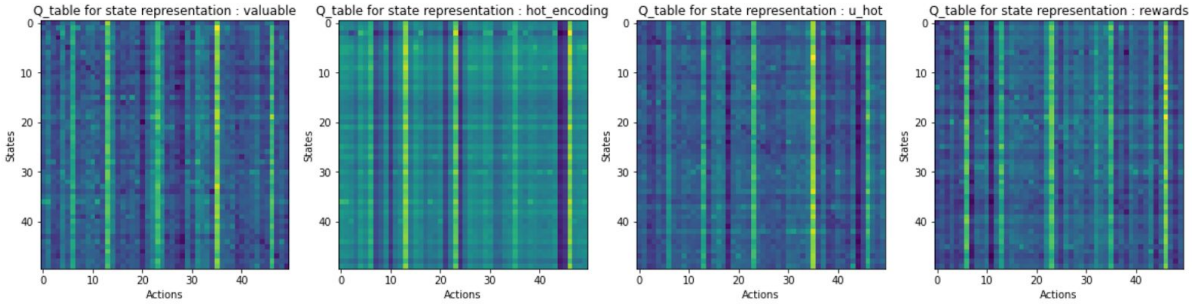
- Memory size : **200**
- Epsilon-Greedy : **0.1**
- Learning rate : **1e-5**
- Batch-size : **20**
- Epsilon-greedy : **0.4**

We did it for 5000 epochs :



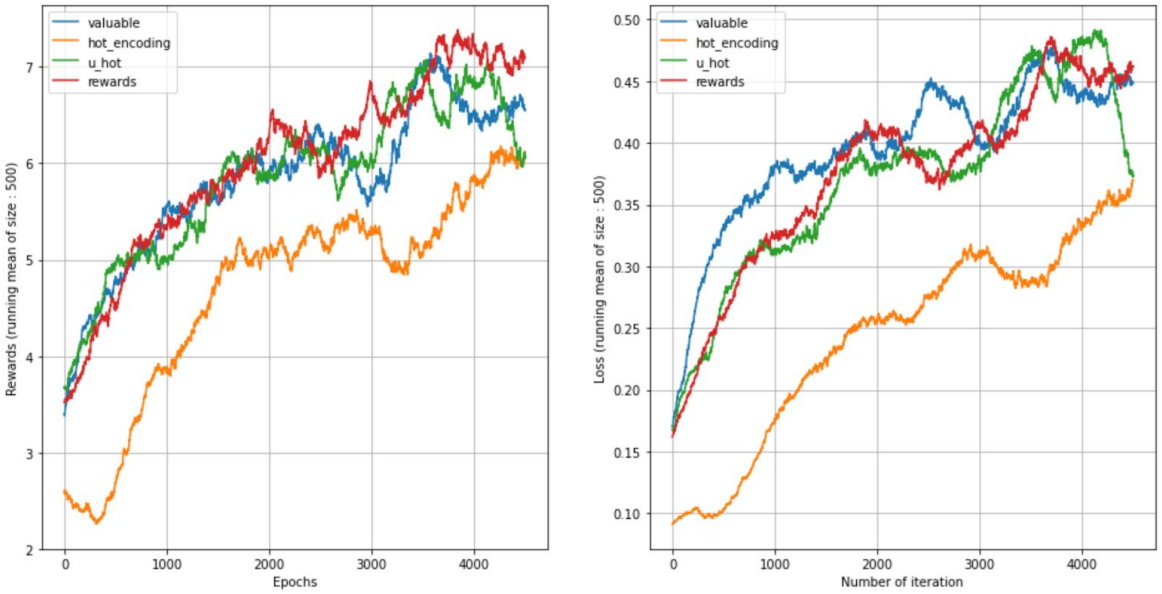
Q_tables for different states representations with fully connected deep q learning algorithm and gamma = 0.9, Adam(lr = 1e-5)

We see that the results are more consistent (the one hot encoding leads to something that makes more or less sense). Here is the transpose to compare :



Q_tables for different states representations with fully connected deep q learning algorithm and gamma = 0.9, Adam(lr = 1e-5)

Nevertheless, it remains the issue of the loss that is still increasing, even if the rewards are increasing :



Rewards and Loss for different states representations with fully connected deep q learning algorithm (Adam and lr = 1e-5)

To summarize, we think that as we represente states by information we have on contents, it tends to learn not the state representation but the actions representation (that's why we got quite good results when we take the transpose).
For gamma = 0.9, the results can make sense, but the fact that the loss is increasing is strange. The fact that the rewards are increasing seem to say that the agent is learning something, and for all representation we get something almost similar.