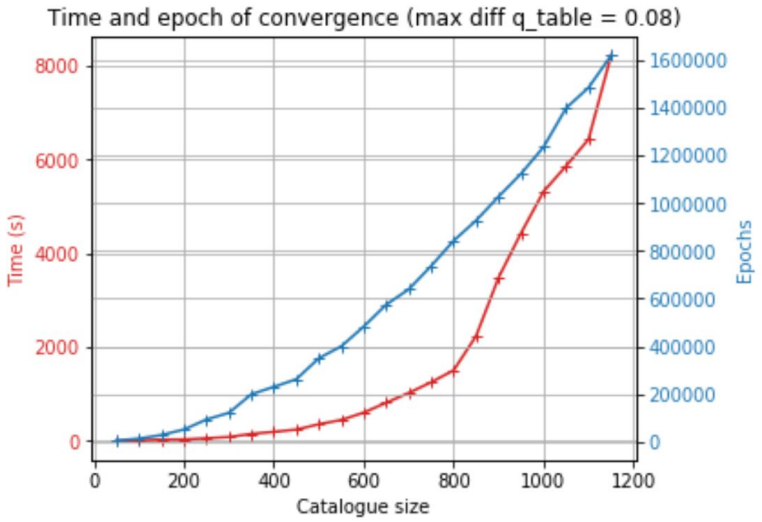
Report 02/06 : Reinforcement learning for Cache-Friendly Recommendations

Clément Bernard and Jade Bonnet

1) Q learning algorithm

a) *Time of convergence*

We updated the number of related and cached contents to be more relevant. We took therefore 5% of the total size of the catalogue related contents and 4% for the cached contents.



Time and epoch to converge for different values of gamma (constraints on related contents)

We set the threshold at 0.08. We didn't manage to go more than 1200 for the size of the catalogue.

b) *Comparison with a specific user*

We changed the behaviour of the user as follow :
When the user is in state **i**, he is likely to accept the content **j** that we suggest with the probability : $\alpha_{i,j} = \frac{u_{i,j}}{max_i(u_i)}$

We also made the following rewards :

- Related and cached : **1**
- Related and not cached : **0**
- Not related and cached : **1**
- Neither related nor cached : **0**

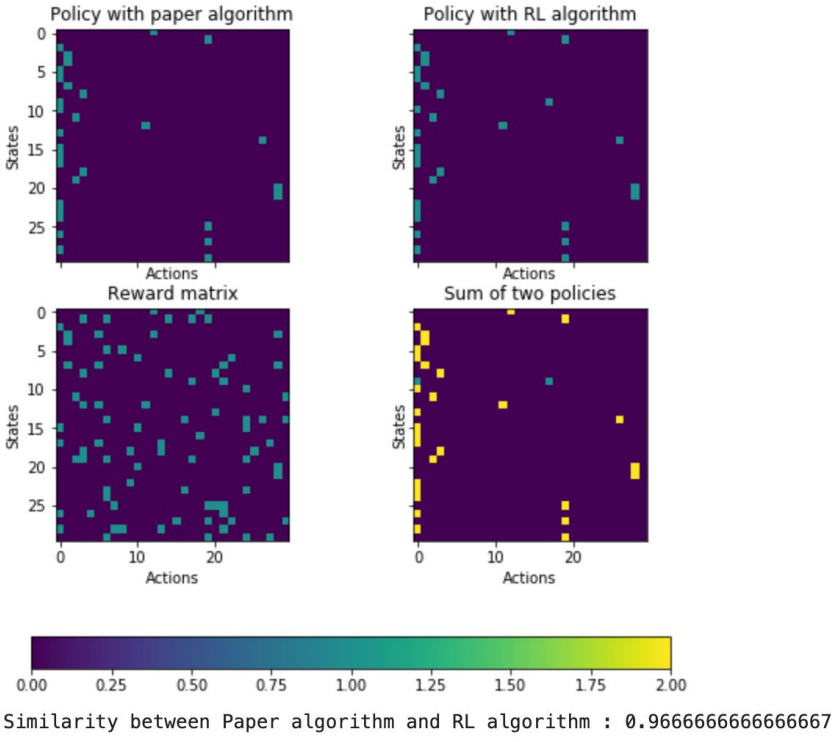
We compared our algorithm with another policy that Theo gave us.
His policy is from the following considerations :

- Catalogue size of **30**
- 3 items cached : the **3** first ones
- 100 requests in average

We used these hyperparameters :

- Probability to leave for the user : **0.01** (100 episodes in average)
- Learning rate : **0.1**
- Gamma : **0.99**
- Epsilon-Greedy : **0.1**
- Max iterations : **100 000**
- Constraints on the related contents

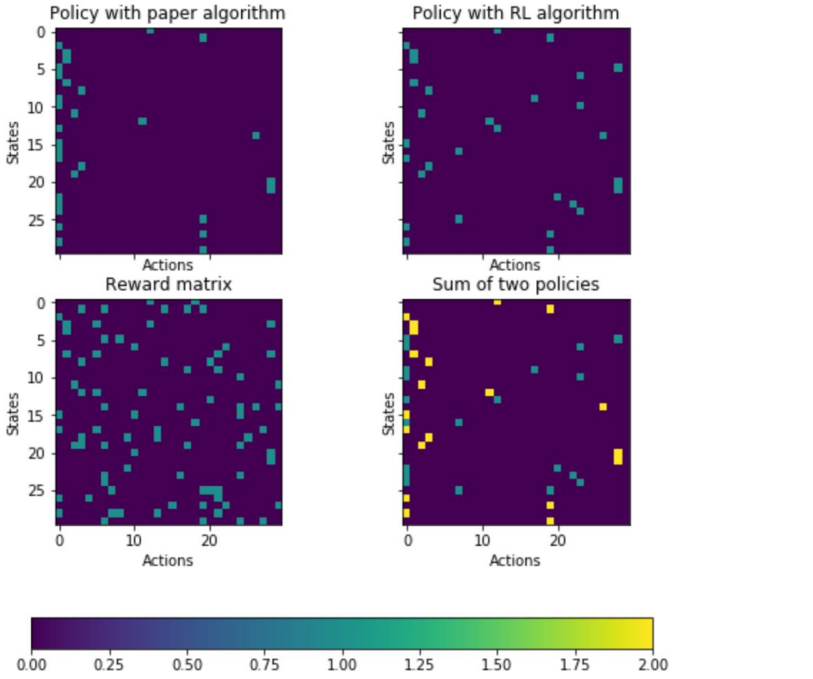
Nevertheless, to get something more similar to the algorithm, we needed to change some liens in our code.
Indeed, in our algorithm, the reward is added for an action of the agent, that is to say, when the agent suggests a content. Nevertheless, in his algorithm, this is when the user clicks.
Therefore, there is a little difference : we do add a reward for a given action whereas we should give a reward when the user chooses a real action to be similar with its policy.
We changed that behaviour by making the reward according to the user choice.
We got 96% of similarity (29 out of 30) and the two others were quite similar in the q tables :



Similarity between Paper algorithm and RL algorithm : 0.9666666666666667

Policy comparison between our policy (restricted) and Theo's policy

Then, we tried to do the same with no constraints on the action we suggest for the user.
But here is the result of our q tables :



Similarity between Paper algorithm and RL algorithm : 0.6666666666666666

Policy comparison between our policy (not restricted) and Theo's policy

And here is the q_table[0] :

```
array([0.        , 0.49225329, 0.49002856, 0.49102025, 0.49347514,
       0.4905086 , 0.49059782, 0.49136372, 0.49190191, 0.49426934,
       0.49269393, 0.49225319, 0.49500741, 0.49023851, 0.49193955,
       0.49249607, 0.49208997, 0.49295201, 0.48947882, 0.49181288,
       0.49167625, 0.49205711, 0.49042658, 0.49120269, 0.49118105,
       0.49488947, 0.49134071, 0.49352434, 0.49036252, 0.4921535 ])
```

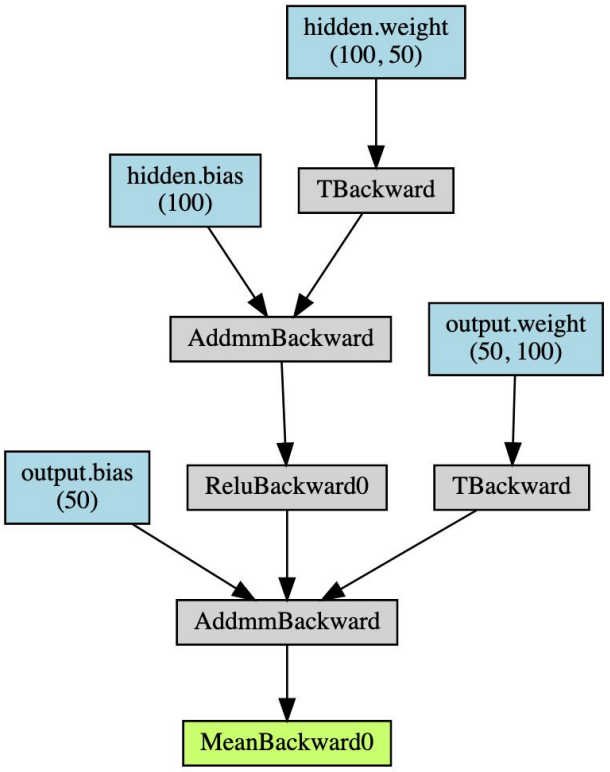
Q table without constraints for the state 0

We explain this behaviour by the user. In fact, whenever we suggest to the user an action that is not related, the alpha is equal to 0 and therefore the user will choose randomly a content among the catalogue (with probability given by p0). Then, if the content isn't among the three cached contents, he will receive a reward of 0 otherwise a reward of 1. Therefore, all the actions except from the related actions will lead to a reward of 1. This is why, we think, that all the actions are so close to each other without any constraints.

2) Deep q learning

a) Pytorch implementation

We modified our code to be adequate for the pytorch format.
The model we used is the following :



Architecture of our model : one hidden layer of size 100

b) Representation of the states

The aim of the algorithm is to find similarity in states that are close to each others in term of policy.
We tried different $\Phi(s_t)$ functions to see the behaviour of the algorithm.
Basically, we tried very basic conversion of the states to see if the algorithm learns something relevant from it.
In order to have a reliable comparison, we did the same process for each of the following representation of the states.
We ran the algorithm for 0, 100, 1000 and 10 000 epochs.

The hyperparameters for the **environment** (user behaviour) are :

- 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]**

The hyperparameters for the **deep Q agent** are :

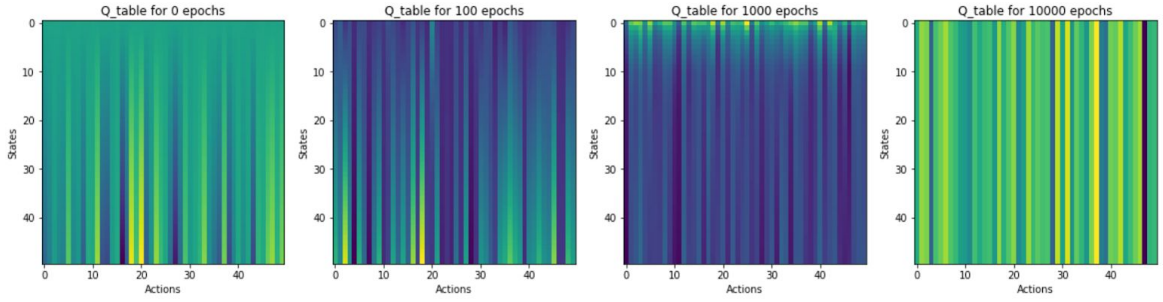
- Memory size : **50**
- Gamma : **0.99**
- Epsilon-Greedy : **0.1**
- Learning rate : **1e-3**
- Batch-size : **10**

Here are the results for the different representations of the state we chose as experimentations.

1) Identity

$$\Phi(S_t = i) = i$$

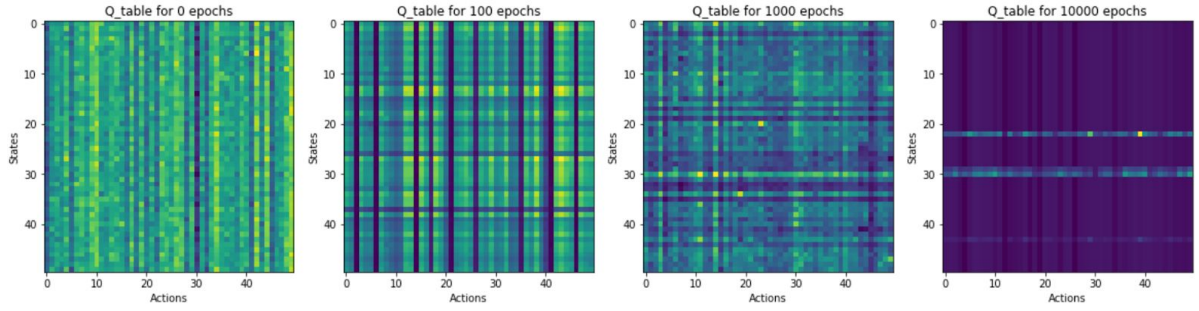
This conversion is the one we tried before. It doesn't work well.



Q table with identity states for 0,100,1000,10 000 epochs

2) One Hot encoding

$$\Phi(S_t = i) = [0, \ldots, 1, \ldots, 0]$$

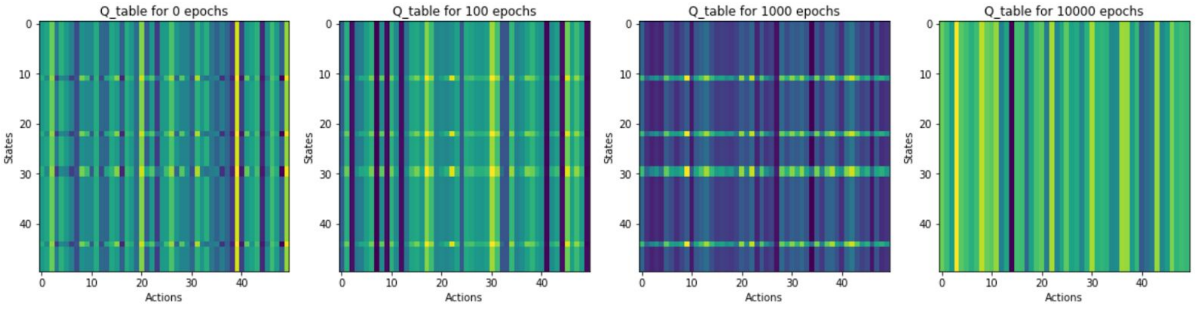


Q table with one hot encoding states for 0,100,1000,10 000 epochs

This approach is no longer better than the previous one as it doesn't give more information for the Neural Network : two states that are very similar aren't code the same.
We need something that could give as much information as we have in a given state.
Let's start with a simple discretization : is the content cached or not ?

3) Cached

$$\Phi(S_t = i) = \begin{cases} 1 & \text{if } i \text{ is cached} \\ 0 & \text{otherwise} \end{cases}$$



Q table with cached states for 0,100,1000,10 000 epochs

This conversion is just a very basic one to see the behavior of the algorithm.
Indeed, the states are in fact split into 2 different states : the cached one and the others.
We expected therefore to have 2 different type of rows. This is what we got for 0, 100 and 1000 epochs. The behavior after 10 000 epochs is still unexplained.
Nevertheless, this approach is too simple and doesn't fit our problem.
We thought about a more complex representation using the U matrix.

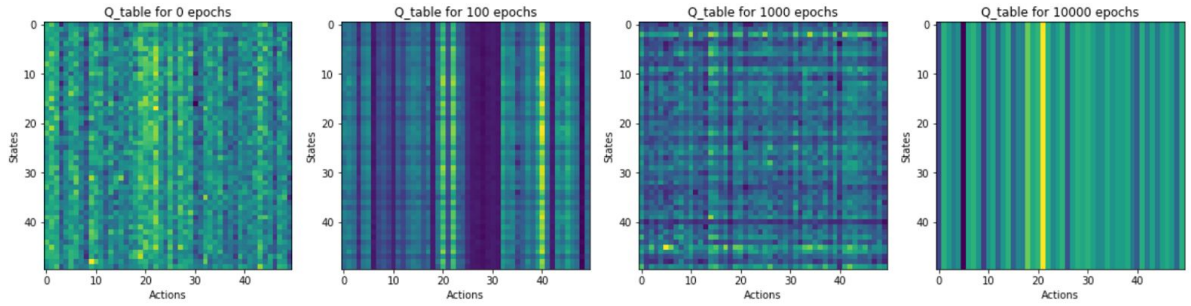
4) U matrix

Then, what came into our mind was the U matrix. Indeed, it gives all the related contents for a given state. In this case, the algorithm could perhaps infer from this matrix the related contents. Note that this approach doesn't take into account the cached contents.

1) Full U matrix

Here we just converted the state by the corresponding line of the U matrix.

$$\Phi(S_t = i) = u_i$$



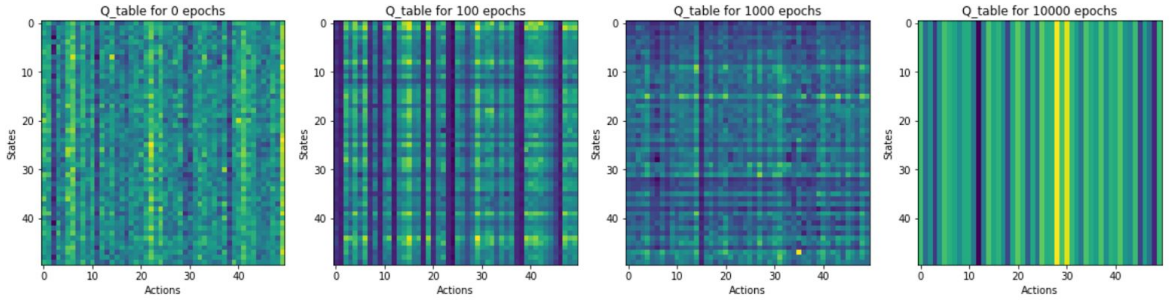
Q table with full U matrix states for 0,100,1000,10 000 epochs

We see that the q tables for 1000 epochs is closer to what we expect.

2) Binary U matrix

We also tried to convert a given state by the contents that are related. We just took the N most related contents and make a 1 for each of these.

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



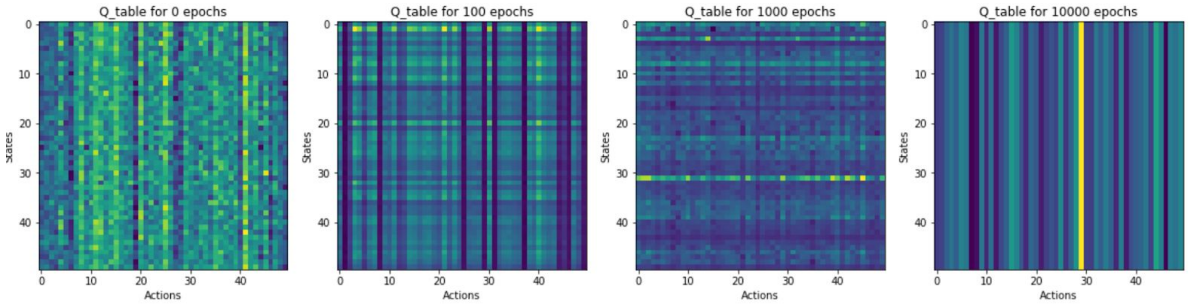
Q table with N most related states for 0,100,1000,10 000 epochs

The behaviour for 1000 epochs seem to make some sense (find similarity for states that have similarity, that is to say that have same related contents).
Note that the behaviour for 10 000 epochs is still unexplained.

5) Rewards

Finally, we tried to represent the states with the information we have : the rewards itself.

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

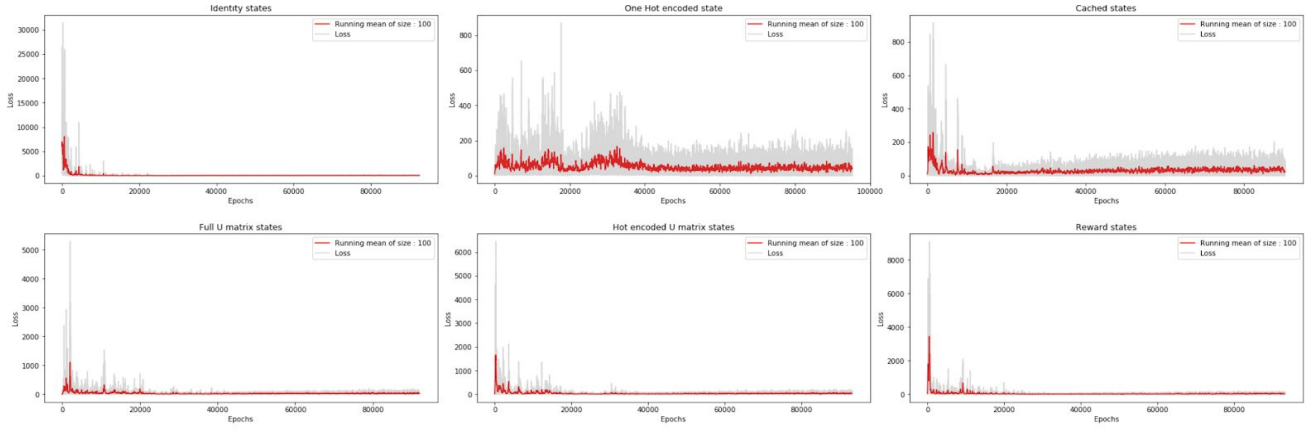


Q table with reward states for 0,100,1000,10 000 epochs

This representation goes beyond the previous use of the U matrix because it takes into account the cached contents.

c) Loss

To see if the neural network has converge or not, we can focus on the loss.
Here are our results :

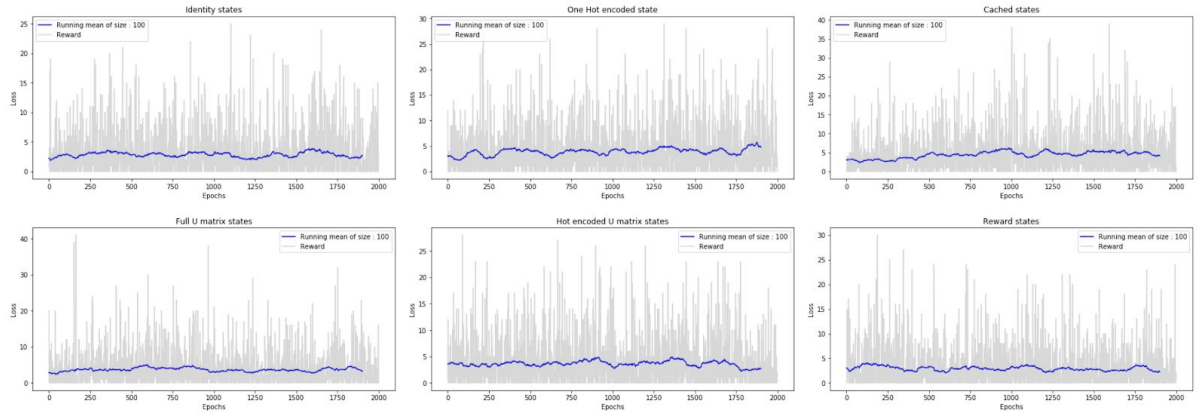


Loss for the different representations of the states

We can see that our algorithm has converged quite quickly. So the results for 10 000 epochs can be a consequence of iterations after convergence.

d) Rewards

Furthermore, one way to see if our approach is good is to look to the rewards.
We used the same hyperparameters than before but we set the max epoch at 2000.



Rewards for different representations of the states

It seems that the reward doesn’t improve a lot.