Report 07/03 : Reinforcement learning for Cache-Friendly Recommendations

Jade Bonnet and Clément Bernard

Coding

We changed a little bit our code. We tried to implement a little more the notations of the paper.

Indeed, we created a matrix U which says, for the value $u_{i,j}$ whether the content j is recommended after consuming the content i.

```
1 env.u[0]
```

```
array([0.93170979, 0.06498158, 0.88199592, 0.64072266, 0.19918211,
       0.67602785, 0.96981043, 0.4392181 , 0.3764248 , 0.2165875 ,
       0.45663475, 0.83062224, 0.43777452, 0.47524023, 0.27591998,
       0.25608642, 0.96810518, 0.36307851, 0.09188837, 0.88101028,
       0.98720857, 0.67098383, 0.99956253, 0.84215972, 0.56530964,
       0.07692582, 0.89505763, 0.41993002, 0.18651131, 0.72362251,
       0.77816459, 0.58002701, 0.55209115, 0.80361329, 0.08707156,
       0.86616058, 0.04694879, 0.99756377, 0.08863197, 0.78544886,
       0.17911188, 0.79909648, 0.13516326, 0.96516245, 0.06495555,
       0.52563413, 0.25220515, 0.62013741, 0.39227763, 0.67842086,
       0.09979203, 0.72829381, 0.4269024 , 0.92072687, 0.53525507,
       0.47007978, 0.7271575 , 0.89250979, 0.28633659, 0.02501458,
       0.80456105, 0.51983064, 0.51054291, 0.99103346, 0.30217541,
       0.69130068, 0.29076356, 0.614673  , 0.61459149, 0.610149  ,
       0.59752097, 0.66486684, 0.92281052, 0.88897026, 0.19578934,
       0.42564463, 0.90947067, 0.480839  , 0.56317387, 0.66105762,
       0.07211283, 0.18556739, 0.93610263, 0.71588583, 0.04317128,
       0.65514639, 0.22177654, 0.43415422, 0.26137624, 0.61019958,
       0.03416893, 0.40828755, 0.2800058 , 0.25691495, 0.3580778 ,
       0.47047222, 0.50275293, 0.5954626 , 0.89182137, 0.53769378])
```

We generated the values randomly. When the agent needs to choose which one to recommend, we take the n_recommended ones that are the highest (this is a first implementation, we'll need after to also consider taking contents less relevant to recommend but are cached).

Then, we considered that the cached contents are constants and don't depend on the content the user is currently consuming. Therefore, we implemented a cost array which says whether there is a cost to consum it or not (0 if cached and 1 if not cached).

```
env.cost
```

```
array([1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1])
```

We also added the probabilities to consume a content in the catalog by the user. We denote it as p_0 .

```
env.p0
```

```
[0.01853714481689089,
 0.007617474839311497,
 0.010338325663859756,
 0.019221894672283316,
 0.004308233378569737,
 0.004526416589066467,
 0.018284480254781018,
 0.014424824885359252,
 0.001179726244802063,
 0.004438283368315547,
 0.013474266271595323,
 0.0006109356267553454,
```

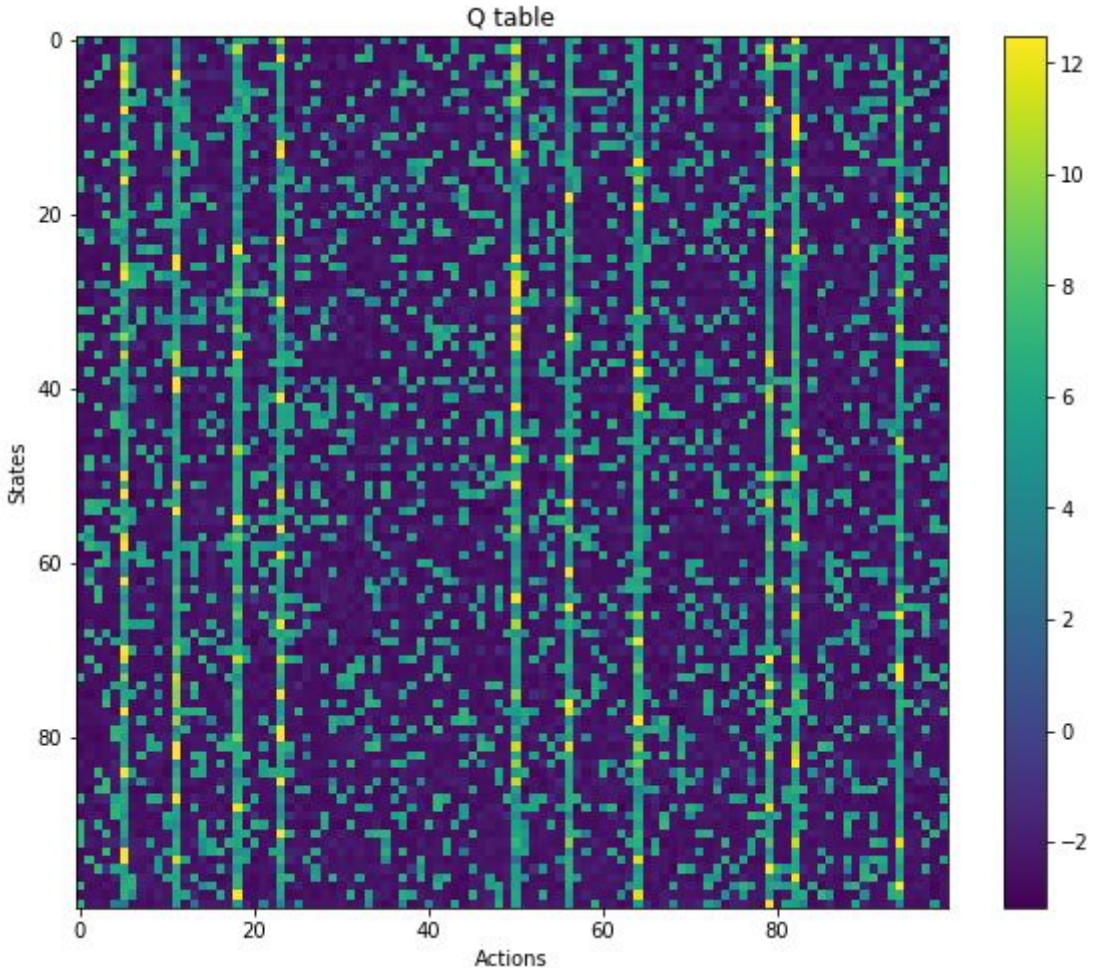
We dealt with the recommendation indexes by using the U matrix. Furthermore, we use the function “env.refresh()” to just compute a new state at each epoch.

Results

```
env = Environment(n_actions=100,n_states=100,alpha=0.6, to_leave=0.1, n_recommended=20,\
                  n_cached=10,rewards=[10,5,5,-5])
```

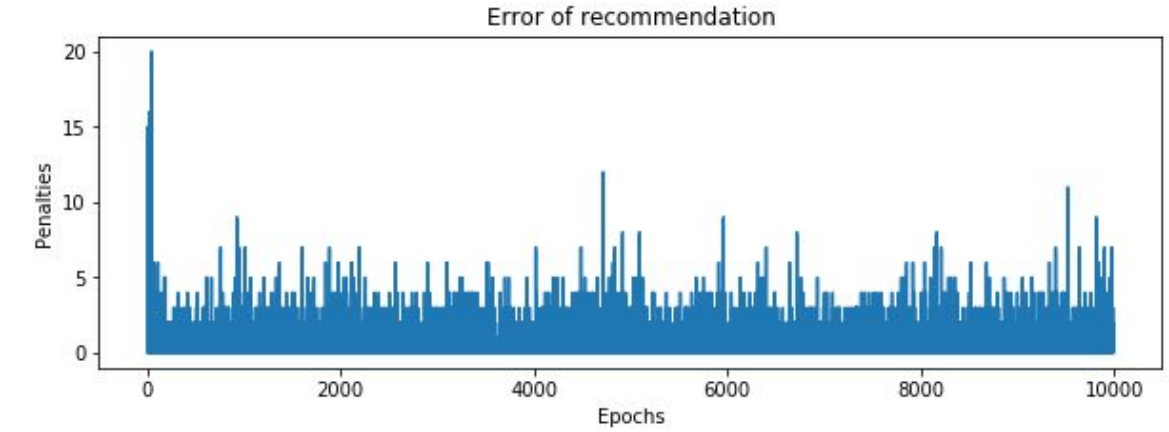
```
q_table, all_penalties, all_rewards, all_q_table = q_learning(env,alpha = 0.2,gamma = 0.2 ,\
                                    epsilon = 0.1,max_iter = 100000)
```

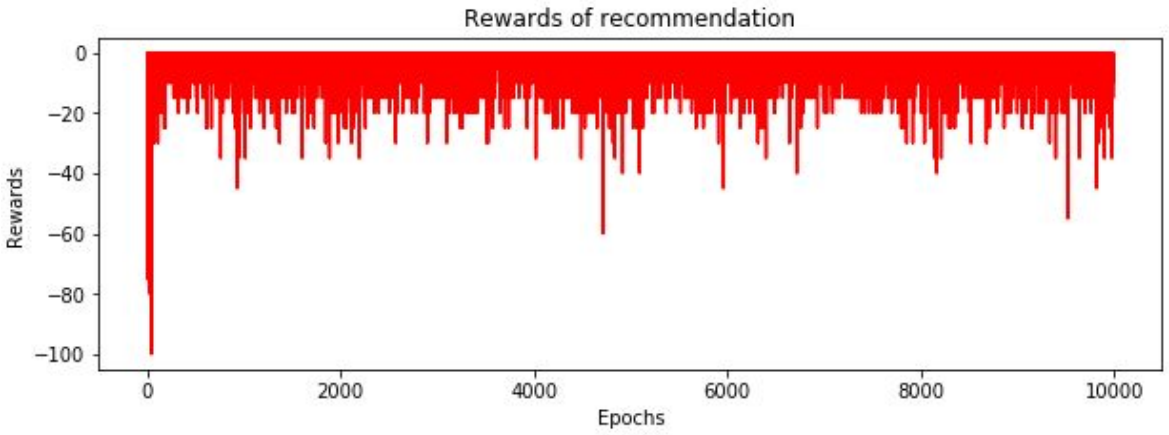
For a 100x100 states-actions with 20 recommendation and 10 cached contents, we expect that there will be high values for at least 10 columns (because 10 actions have high rewards : the cached ones) and around 20 actions per state that have high values in the q_table.
Here is the result (after 100 000 epochs) :



Q_table after 100 000 epochs

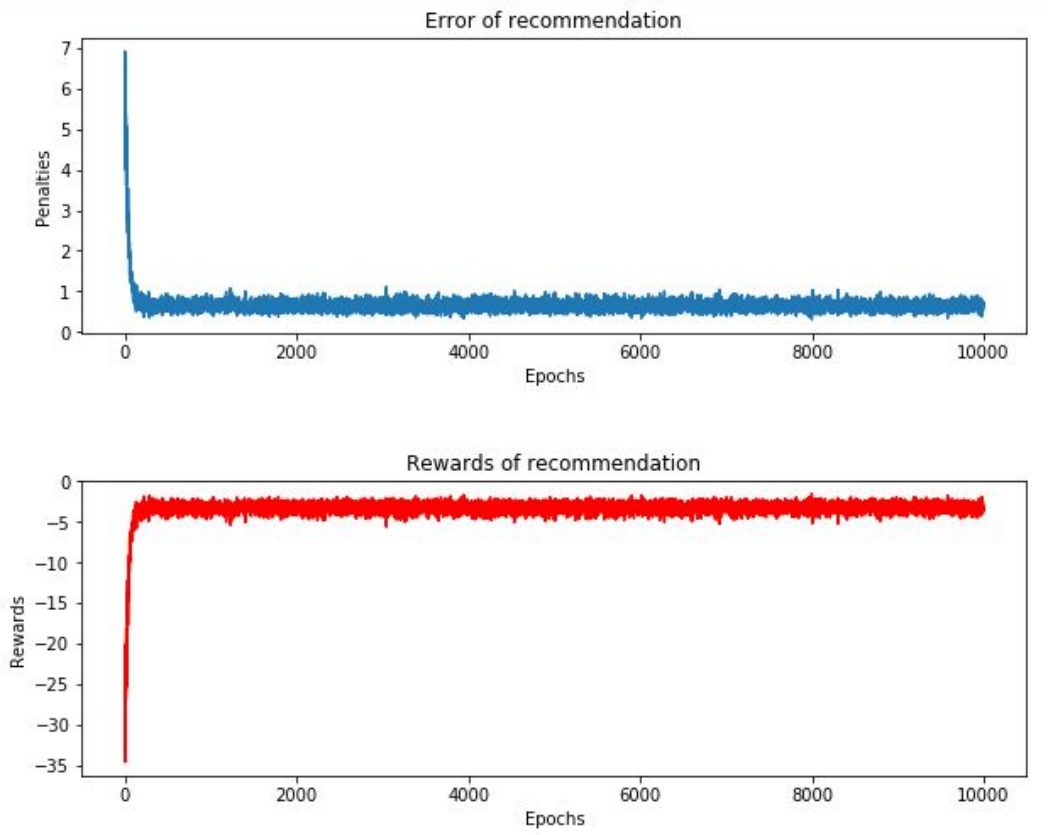
We see that this is as we expected. The yellow cases correspond to the content that is cached and recommended, the green ones correspond to either a cached or recommended content and the blue ones correspond to non-recommended and non-cached contents.
We also can see the evolution of the reward/penalties. We defined the penalty as the fact that the agent recommends a non-cached and non-recommended content.
Here is the result :





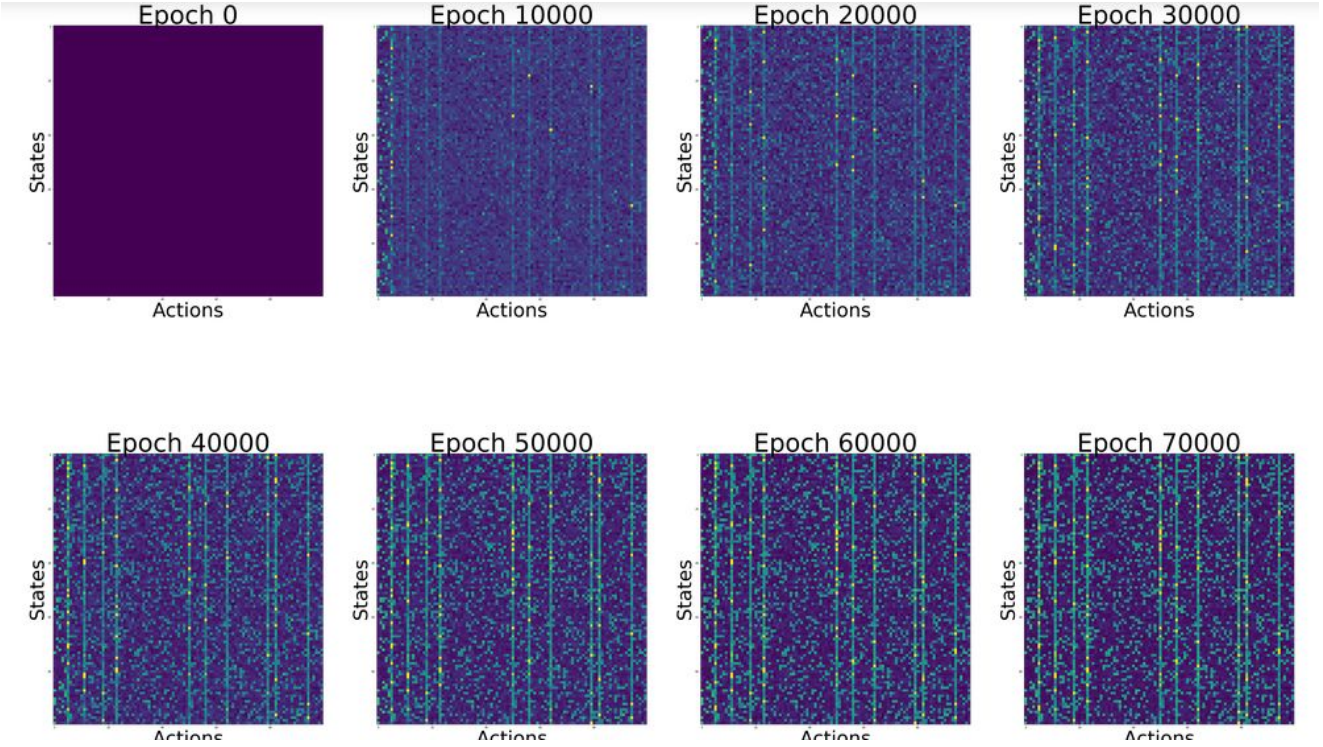
Penalties and Rewards through 10 000 epochs

We only plotted the 10 000 firsts epochs. We see that there is some noise. We decided to do the same process 100 times and then take the average. Here is the result :



Penalties and Rewards through 10 000 epochs (average over 100 simulations)

We can also see how the q_table evolved as follows :



Q_table for different epochs