Network Friendly Recommendations

Reinforcement Learning Assignment 3

Konstantinos Mylonas
Panos Michas
5 July 2023

Introduction

In the context of the course “Reinforcement Learning” we had to implement Policy Iteration, and Q-learning in order to train an agent to operate optimally in a specific environment. To be more precise, the primary focus of *Network Friendly Recommendations* is to recommend items (videos) that are cached, and relevant to the video that the user is currently watching at the same time. For this first part of the project we only recommend 2 videos at a time (N=2).

Environment

The model-based approach requires a full representation of the environment.
State (s): The video that is currently being watched. The total number of videos that are available in the platform is considered to be K.
Action (A): Each tuple of N=2 videos is considered to be one possible action. For example if a user watches video_1, one possible action would be to recommend the tuple A = (video_2, video_3). There are some limitations to what we can recommend:

1. The agent is not allowed to recommend the video that is currently being watched (e.g. $s \notin A$).

2. Each one of the recommended items must be different than the others. In other words, one can say that A should form a mathematical set, as sets don’t allow repetition. So $A = (v_1, v_2)\ s.t.\ v_1 \neq v_2\ or\ A = \{v_1, v_2\}$.

Reward (r): If the user chooses a cached video, then the agent receives a reward of +1. In any other case the reward is 0. We assume that 20% of the entire catalogue of videos is cached so we create a K-sized array of zeros and ones where the probability of 1 is 0.2 and probability of 0 is 0.8.

User Model

The user is free to choose any next video (transit to any next state) of all possible options at a time. He can either choose one of the recommended videos or navigate through the search bar. If at least one of the recommended videos is irrelevant to the current video, then the user navigates through search bar with probability 1. If all videos in A are relevant to s, then the user chooses one of the recommended videos with probability a or navigates through the search bar with probability 1-a. Relevance is given from a relevance matrix U where element $u_{ij} \in [0,1]$ and shows how relevant is content j to content i. We define a threshold u_{min} so that $u_{ij} > u_{min}$ means that content j is relevant to content i. Otherwise its considered to be irrelevant. Of course all of this is true under the condition that the user doesn't end the session, which happens with probability q. So what we have is:

$$Pr(quit) = q$$

$$Pr(continue) = 1 - q$$

$$Pr(pick\ v_i \mid continue, \exists\ irrelevant\ reccomendation) = \frac{1}{K},\ i \in \{1,2,...,K\}$$

$$Pr(pick\ from\ recommendations \mid continue, all\ relevant) = a$$

$$Pr(pick\ from\ search\ bar \mid continue, all\ relevant) = (1 - a)$$

$$Pr(pick\ v_i \mid continue, all\ relevant) = \frac{a}{N},\ v_i \in\ A$$

$$Pr(pick\ v_i\ from \mid continue, all\ relevant) = \frac{1 - a}{K},\ i \in \{1,2,...K\}$$

So the total probability to pick a recommended video given that the user continues and all recommended videos are relevant is to choose it because it is recommended or choose it through the navbar:

$$Pr(pick\ v_i \mid continue, all\ relevant) = \frac{a}{N} + \frac{1-a}{K},\ v_i \in A$$

Having said that, the total probability to continue and choose a video that is recommended, given that all recommendations are relevant is:

$$Pr(pick\ v_i, continue \mid all\ relevant) =$$
$$= Pr(continue \mid all\ relevant) \cdot Pr(pick\ v_i \mid continue, all\ relevant) =$$
$$= (1-q) \cdot (\frac{a}{N} + \frac{1-a}{K}),\ v_i \in A$$

Where we used the chain rule and the fact that the event of "user continues" is independent from the relevance of the recommended videos.

Following the same path, we end up that the probability to choose any video from the catalogue, given that the recommended videos are all relevant is

$$Pr(pick\ v_i, continue \mid all\ relevant) =$$
$$= (1-q) \cdot (\frac{1-a}{K}),\ v_i \notin A$$

Finally, given that the recommended videos are not relevant, the probability to choose any video is:

$$Pr(pick\ v_i, continue \mid \exists\ irrelevant\ recommendation) =$$
$$= (1-q) \cdot (\frac{1}{K}),\ i \in \{1,2,...,K\}$$

Those are actually the transition probabilities given action and current state.

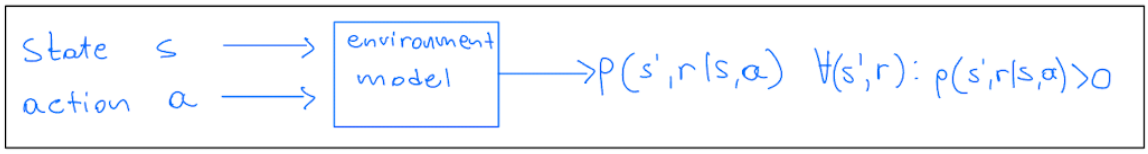
According to Sutton and Barto book, having the dynamics of an environment allows you to model it as an MDP and use model-based methods (like Policy iteration) to come up with the optimal policy and value function. In our case we do indeed have the dynamics of the environment, we have the function $p(s', r \mid s, a)$ so the next step is to implement the policy iteration algorithm.

Policy Iteration

The implementation of policy iteration is straightforward. In fact, after modelling the environment, all we had to do was to apply the Bellman update and sooner or later the algorithm would converge to the optimal policy. For the environment, we implemented a function that takes as input a state and an action and returns a set of all possible outcomes in the following form:

(probability, next state, reward, done)

That's exactly what the policy evaluation algorithm needs to iterate over, in order to calculate the value of a state. We should mention that if the agent is in a state, there is a non-zero probability to transit to any other state (while in frozen lake the next state could be only one of the neighbour blocks).



According to Sutton and Barto this is the Bellman update that we should perform in Policy Evaluation.

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a) \Big[r + \gamma v_k(s') \Big]
\end{aligned}
$$

Given that our policy is deterministic, π(a|s) = 1 only for one action so we can drop the first sum. So we end up with a sum over all s', r which is exactly what the *environment model* function returns, given s, a.

For the Policy Improvement we have to calculate Q(s, a) and then generate a new policy by mapping its state s to the action a that maximises Q(s, a), as theory suggests. According to Policy Improvement theorem this method

should generate an improved policy whose value function should be at least as good as the value function of the policy that is being improved.

Policy Iteration - Experiments

In order to see whether PI does indeed find the optimal policy we created a toy example so that we can judge if the action that the agent is picking are reasonable. For this purpose we choose K = 4 and we print the relevance matrix U, along with the vector C where C[i] = 1 if item 1 is cached. Moreover, the parameters setting is the following:

a=0.9, q=0.1, u_min = 0.2

Which is close to a real scenario in the sense that a user does indeed watch a few videos before quitting, and chooses recommendations with high probability.

The results are the following:

```
Relevance:
 [[0.  0.8 0.5 0.8]
 [0.8 0.  0.8 0.1]
 [0.1 0.7 0.8 0.8]
 [0.8 0.8 0.8 0. ]]

Cached:
 [0 1 1 0]

Optimal Policy:
[[1 2]
 [0 2]
 [1 3]
 [1 2]]
```

The policy seems to optimal because:

At state 0 the action is [1 2] which with high probability a=0.9 will bring a reward of +1 as both [1, 2] are cached. In state 1, the action [1,2] is not allowed so the agent chooses [0, 2] instead of [2, 3] because (3,1) are irrelevant so action [2,3] would lead to skipping recommendation and navigate through search bar. When this happens, the agent “loses control” and cannot affect the next choice. The same thing happens with state 2 where action is [1, 3] for the same reason. In this example the agent seems to take reasonable actions and from now on we will consider that PI converges to the optimal policy. In more complicated examples, it would be hard to explain why they are optimal because we care about the discounted

return (gamma = 0.9). So the agent might choose some actions that seem sub-optimal at a specific state but in the long run would lead to states where cached videos are also relevant so that the agent would have more impact to the user to “manipulate” him to choose cached videos.

Q learning

Model-free methods are also able to solve this problem. We implemented the Q-learning algorithm and ended up that it converges to the optimal policy. According to Sutton and Barto, in order for the q learning to find the optimal policy, it must visit each state, action pair infinitely many times. For this reason we chose to have an exploration probability of 0.2. There are for sure smarter ways to tune the exploration parameter but for this phase of the project a constant exploration probability should do the work. In general, according to the stochastic approximation theory, it is recommended to have a learning rate parameter that meets the following:

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \qquad \text{and} \qquad \sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$$

But as Sutton and Barto say, in practical applications we seldom use such learning rate. Having said that we also chose to have a constant learning rate alpha = 0.1.

In short, we have a function called “env_observe” that takes as input a state and an action and samples a next_state and reward from the distributions

that we described earlier. We use that function to create trajectories, “play episodes”.

Q-learning - Evaluation

This problem makes use of a huge action set which adds a lot of limitation. For big K, q-learning should theoretically converge but it will take a lifetime. For K < 100 we claim that the q-learning algorithm converges to the optimal policy (in a reasonable amount of training episodes). Of course there are many optimal policies, so comparing the optimal policy that PI converged to with the Q-learning is not right. They might be different, but they might still be optimal. In order to show that Q-learning finds the optimal policy, we had to play many episodes with it’s policy. We also did that with the policy that PI proposes. We played 100.000 episodes with each policy and calculated the average reward. Both policies achieve the same average reward so we have a good reason to believe that q learning is indeed optimal, given that PI is optimal (which we proved earlier).

Moreover in order to see that the Q-learning agent is indeed learning and how long does it take for him to “learn” something close to optimal, we keep track of the reward per episode. Initially average reward is low and as more and more experience is gained (agent plays more episodes) the average reward increases and becomes nearly constant at some point. With exploration probability not decaying to zero, we can’t expect it to be 100% constant but it is enough to see that the algorithm has found the optimal policy.

We can’t know in advance how many training episodes does the q-learning algorithm need in order to find the optimal policy so we simply try a reasonable number, and if it has not yet converged, we increase the number of training episodes.

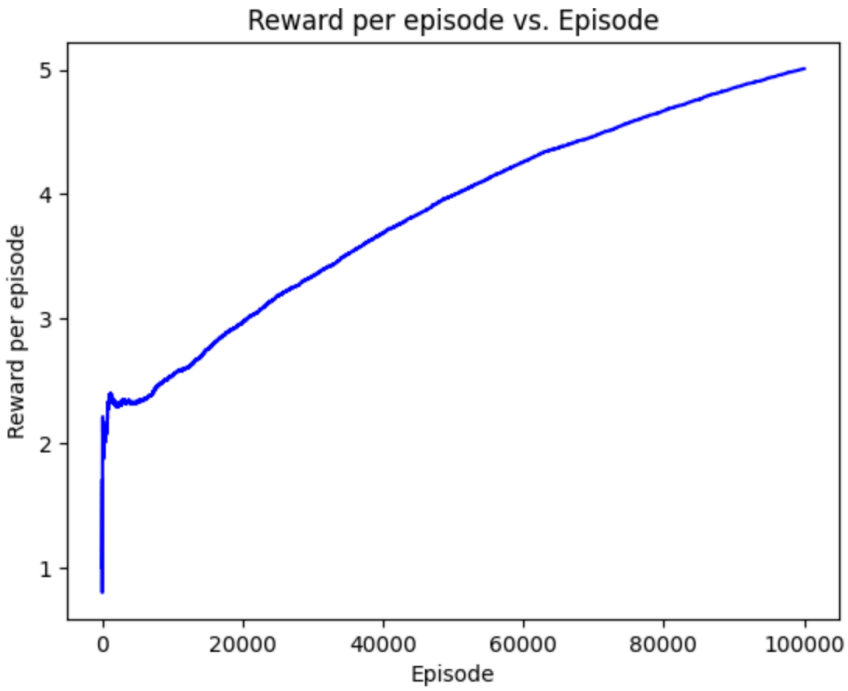
We summarise those points in the following experiments:

100.000 training episodes - 100.000 test episodes - K=20

```
POLICY ITERATION AVERAGE REWARD ON 100000 EPISODES:
8.24897
```

```
Q LEARNING AVERAGE REWARD ON 100000 EPISODES:
7.96001
```

For example in this case (K=20), we trained for 100.000 episodes and we can see an important difference in the average rewards between the optimal (Policy Iteration) and what q-learning has found. This leads to the conclusion that q-learning hasn’t still found the optimal policy. This is also shown in the following plot:



We see that the curve is not flattened yet which denotes that the agent is still learning. So 100.000 episodes for this case was not enough. We try again with 250.000 training episodes.

250.000 training episodes - 100.000 test episodes - K=20

```
POLICY ITERATION AVERAGE REWARD ON 100000 EPISODES:
8.3331
```

```
Q LEARNING AVERAGE REWARD ON 100000 EPISODES:
8.34406
```

In this case we can see the Q-learning and Policy Iteration, lead to policies that perform equally well. The following plot also shows the learning curve for q learning.

In this case, flattening has been achieved. If we played even more episodes flattening would be even more obvious. We will now try to see how will Q learning perform for K = 100. We know in advance that it's going to take many

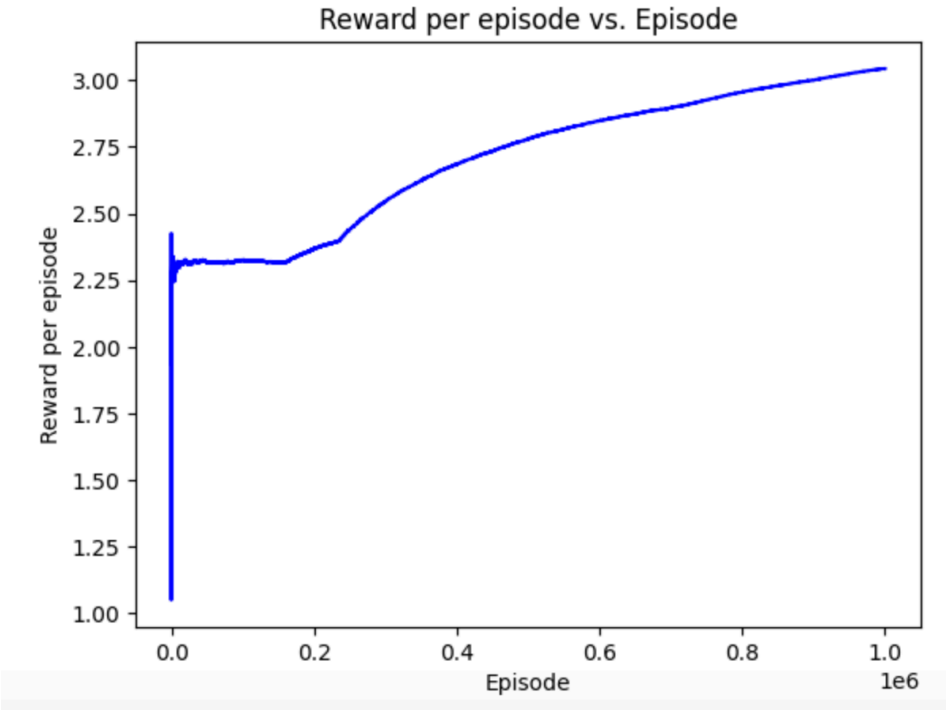
episodes for this scenario, so we will start with 1.000.000 and based on the learning curve we will decide whether we should increase them.

1.000.000 training episodes - 100.000 test episodes - K=100

```
POLICY ITERATION AVERAGE REWARD ON 100000 EPISODES:
8.32203
```

```
Q LEARNING AVERAGE REWARD ON 100000 EPISODES:
4.28526
```

In this case we can see that the policy suggested from the q-learning performs poorly.



The learning curve shows that 1.000.000 episodes were not enough so we continue with 5.000.000 episodes.

5.000.000 training episodes - 100.000 test episodes - K=100

```
POLICY ITERATION AVERAGE REWARD ON 100000 EPISODES:
8.20953
```

```
Q LEARNING AVERAGE REWARD ON 100000 EPISODES:
6.70303
```

Five million episodes and it's still learning.

We tried for 10.000.000 episodes (which run in almost one hour) and it did converge. We can now claim that for K>100, the Q-learning algorithm is not efficient.

Policy Iteration - Different parameters

Finally let's see what changes for different parameters of u_min, a and q.

u_min = 0.4 - a = 0.6 - q = 0.2 - K=20

```
POLICY ITERATION AVERAGE REWARD ON 100000 EPISODES:
1.9468
```

u_min = 0.2 - a = 0.6 - q = 0.2 - K=20

```
POLICY ITERATION AVERAGE REWARD ON 100000 EPISODES:
2.15578
```

So as we can see small *a = 0.6* leads to smaller average reward compared to 8.3 that we have seen in all previous experiments with K=20. The reason is straightforward. Small *a* means that the user will use the search bar more than it would with bigger *a*. So the actions of the agent don't affect a lot the following states and the user picks cached items with smaller probability (we have a hit if the user searches through search bar for a cached item
pr = 0.2K/K

For bigger u_min we also see poor performance and that's because as u_min increases it is less possible for cached items to be also relevant so the are not so many "good actions". Let's consider the two edge cases For u_min=0 the agent would always be able to recommend cached videos. In contrast, for

u_min = 1, no matter what the agent recommends the user will always go with the search bar.

The q parameter would affect the q learning in the following way:

Big q means that each episode has less steps (it terminates more often) so the q-learning would require more episodes to converge (but each episode would be shorter than it would be for small q).