ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

Reinforcement Learning & Dynamic Optimization

Network Friendly Recommendations

3rd Assignment - Semester Project 1st part

Georgios Frangias 2018030086
`gfrangias@tuc.gr`
Dimitrios Petrou 2018030070
`dpetrou@tuc.gr`

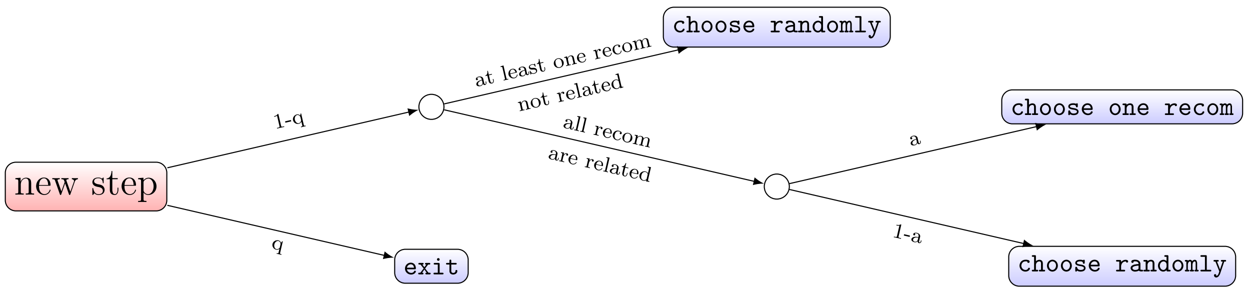
Course Professor:
Thrasyvoulos Spyropoulos

Technical University of Crete

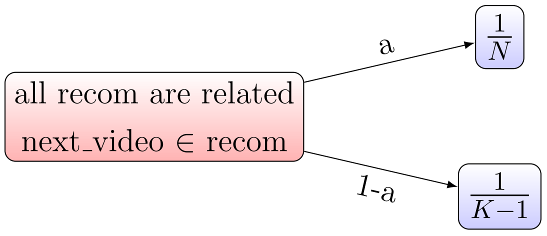
School of Electrical & Computer Engineering

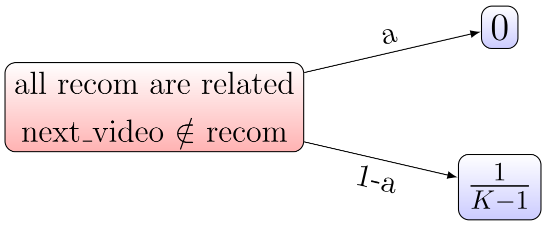
June 2023

a) Environment



The environment of the project was implemented as described in the assignment and can be simply described by the tree above. Later, for the policy iteration algorithm, there was a need to compute the probability of choosing a new state (function `env_prob`). This probability is computed based on the relevance of the recommended videos with the video that the user is currently watching. The table below describes the probability given by `env_prob`.





	$\forall$ video $\in$ recom s.t. video relevant	$\exists$ video $\in$ recom s.t. video $\neg$ relevant
next_video $\in$ recom	$\frac{a}{N} + \frac{1-a}{K-1}$	$\frac{1}{K-1}$
next_video $\notin$ recom	$\frac{1-a}{K-1}$	$\frac{1}{K-1}$

In this environment, similarly to the frozen lake game, it was necessary to matchup the terms used. So in this scenario:

- states $\rightarrow$ K videos
- actions $\rightarrow$ all possible N-lengthed tuples of recommendations to a video
- rewards $\rightarrow$ -1 for non-cached and 0 for cached videos (maximization problem)
- transitions $\rightarrow$ all "illegal" transitions were ruled out (i.e. $\text{state}_{i+1} = \text{state}_i$)
- U matrix $\rightarrow$ randomly created, the diagonal is filled with zeros and it is symmetric

b) Policy Iteration

Policy iteration is composed of two parts; policy evaluation and policy improvement. At first iteration, one random policy π_0 is picked. Then the following are done iteratively.

```
POLICY EVALUATION
  • while True:
      – $V_{i+1}(s) = \pi(a|s)\left(\mathcal{R}_s + (1-q)\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} V_i(s')\right)$
      – if V converges:  break
POLICY IMPROVEMENT
  • while True:
      – for all states, recom:
          * $q_{\pi_i}(s,a) = \mathcal{R}_s + \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} V_{\pi_i}(s')$
          * $\pi_{i+1}(s) = argmax_{a \in \mathcal{A}}(q_{\pi_i}(s,a))$
TERMINATION CONDITION
  • if  $\pi_{i+1} = \pi_i$:  break
```

- $\pi(a|s)$ is the set of actions that are allowed in this policy given state s
- $\mathcal{R}_s$ is the reward of state s
- $\mathcal{P}^a_{ss'}$ is the environment probability of moving to state s' given we are at state s and we take action a
- $V_i(s')$ is the last iteration's value function value for the next state

1. Convergence to Optimal Policy

In order to test if policy iteration is indeed converging to the optimal policy, it was needed to create a small "toy" scenario for which the optimal policy was trivially computed.

$$K = 10,\ N = 2,\ u_{min} = 0.5,\ a = 1,\ q = 0.05,\ \text{cached} = \{0, 1\},$$

$$u = \begin{bmatrix} 0.0 & 0.9 & 0.1 & 0.1 & 0.9 & 0.1 & 0.9 & 0.9 & 0.9 & 0.1 \\ 0.9 & 0.0 & 0.9 & 0.9 & 0.1 & 0.9 & 0.9 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.9 & 0.0 & 0.1 & 0.9 & 0.1 & 0.9 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.9 & 0.1 & 0.0 & 0.1 & 0.9 & 0.1 & 0.1 & 0.9 & 0.1 \\ 0.9 & 0.1 & 0.9 & 0.1 & 0.0 & 0.9 & 0.1 & 0.1 & 0.9 & 0.1 \\ 0.1 & 0.9 & 0.1 & 0.9 & 0.9 & 0.0 & 0.1 & 0.9 & 0.1 & 0.1 \\ 0.9 & 0.9 & 0.9 & 0.1 & 0.1 & 0.1 & 0.0 & 0.1 & 0.1 & 0.9 \\ 0.9 & 0.1 & 0.1 & 0.1 & 0.1 & 0.9 & 0.1 & 0.0 & 0.1 & 0.9 \\ 0.9 & 0.1 & 0.1 & 0.9 & 0.9 & 0.1 & 0.1 & 0.1 & 0.0 & 0.9 \\ 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.9 & 0.9 & 0.9 & 0.0 \end{bmatrix}$$

Next, you will see 3 columns where the first column describes the relations between the states, the second the optimal actions that were expected before running the optimal policy algorithm and the third the actual results that the optimal policy created.

$$\begin{array}{lll}
0 \xrightarrow[\text{to}]{\text{related}} 1,4,6,7,8 & \pi_*(a|s=0) = [1,4|6|7|8] & \pi(a|s=0) = [1,6] \\
1 \xrightarrow[\text{to}]{\text{related}} 0,2,3,5,6 & \pi_*(a|s=1) = [0,2|3|5|6] & \pi(a|s=1) = [0,6] \\
2 \xrightarrow[\text{to}]{\text{related}} 1,4,6 & \pi_*(a|s=2) = [1,4|6] & \pi(a|s=2) = [1,6] \\
3 \xrightarrow[\text{to}]{\text{related}} 1,5,8 & \pi_*(a|s=3) = [1,5|8] & \pi(a|s=3) = [1,5] \\
4 \xrightarrow[\text{to}]{\text{related}} 0,2,5,8 & \pi_*(a|s=4) = [0,2|5|8] & \pi(a|s=3) = [0,2] \\
5 \xrightarrow[\text{to}]{\text{related}} 1,3,4,7 & \pi_*(a|s=5) = [1,3|4|7] & \pi(a|s=5) = [1,4] \\
6 \xrightarrow[\text{to}]{\text{related}} 0,1,2,9 & \pi_*(a|s=6) = [0,1] & \pi(a|s=6) = [0,1] \\
7 \xrightarrow[\text{to}]{\text{related}} 0,5,9 & \pi_*(a|s=7) = [0,5|9] & \pi(a|s=7) = [0,5] \\
8 \xrightarrow[\text{to}]{\text{related}} 0,3,4,9 & \pi_*(a|s=8) = [0,3|4|9] & \pi(a|s=8) = [0,4] \\
9 \xrightarrow[\text{to}]{\text{related}} 6,7,8 & \pi_*(a|s=9) = [-,-] & \pi(a|s=9) = [0,1]
\end{array}$$

```
Iterations needed: 6
Optimal Policy:
[[1, 6], [0, 6], [1, 6], [1, 5], [0, 2], [1, 4], [0, 1], [0, 5], [0, 4], [0, 1]]
```

It is obvious that the policy iteration algorithm after 6 iterations ends up to the optimal policy. Every state has an optimal action. We can observe that state 6 is the only state that is related to both cached videos (0,1). This gives state 6 the third-best value function after states 0, 1, since (with the appropriate recommendation (0,1)) it leads with certainty to cached videos. This is why all states that are related to 6 add 6 to their recommended (states 0, 1, 2).

2. Parameter Tweaking

In all of our experimentation with parameter tweaking, we make use of 100000 Monte Carlo episodes after each iteration. This way, the mean loss for each iteration's policy is evaluated.
The default values for the parameters that are not tweaked are:

- K = 20
- N = 2
- $u_{min} = 0.5$
- a = 0.8
- q = 0.05

Parameter a

We wanted to observe the loss that occurs with different values of a, as well as the time it takes to converge. We assumed that a higher value of a, would achieve convergence quicker and that the mean loss would reduce further. The results verified those assumptions.

Parameter N

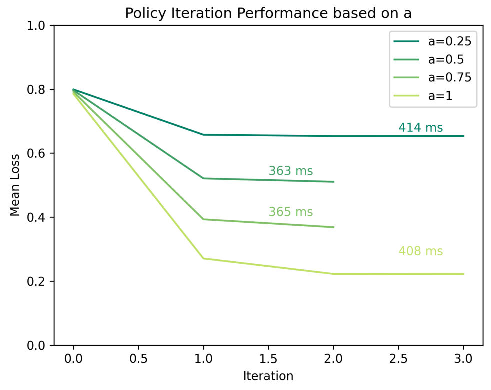
Experimenting with parameter N, it is expected that for larger N values the time needed to converge will be longer since the Q-function will have a much bigger action space to calculate. Moreover, there should be a slight loss increase because it is getting more difficult to have all recommendations be related to the currently watched video.

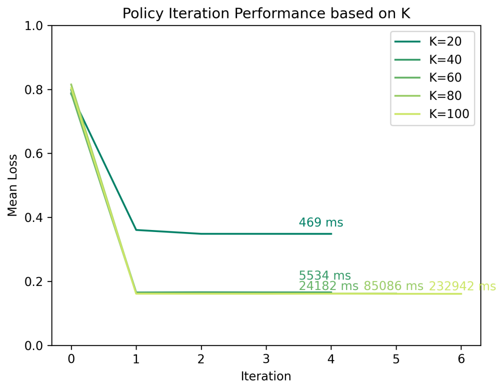
Parameter K

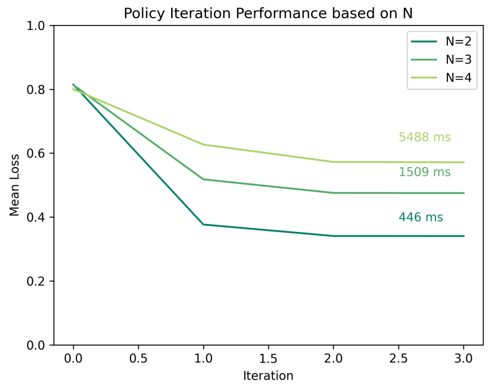
Parameter K makes it more difficult, as it decreases, to find enough relevant videos to recommend. That is why we thought, that as the parameter K increases, we would reach a plateau where there are just enough related videos to achieve the least possible loss. Where this plateau is reached depends on u_{min} as well. We delve into this later. On our example only for K=20 the plateau isn't reached which is logical, since u_{min} is set to the quite low value of 0.5. As expected, the time elapsed increases for larger K value, since with more states comes a larger Q-function.

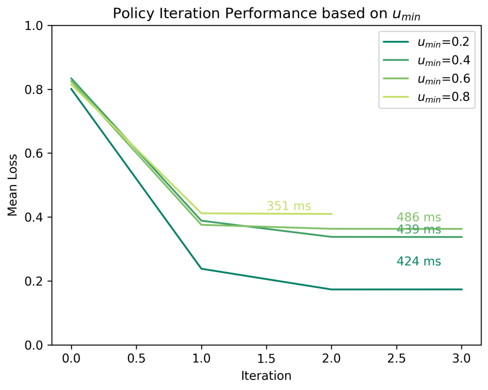
Parameter u_{min}

Tweaking the parameter u_{min}, we expected that the larger u_{min} is, the larger the loss is. Similarly with parameter K there is a plateau that we reach for a small enough u_{min}.









c) Q-Learning

Q-Learning

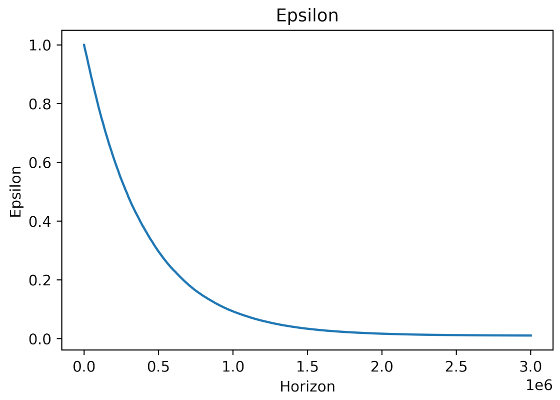
```
• Initially $Q_0(s,a) = 0 \; \forall s, a$
• Choose a random initial state
• while True:
    – sample action $a$
    – get next state $s'$
    – if $s'$ is terminal:
        ∗ $target = \mathcal{R}_s$
        ∗ sample new random initial state $s'$
    – else:
        ∗ $target = \mathcal{R}_s + (1-q)\max_{a'} Q(s', a')$
    – $Q_{i+1}(s,a) = (1-\mathfrak{a})Q_i(s,a) + \mathfrak{a} \cdot target$
    – $s = s'$
```

- *How to sample action a?*
- *Using ϵ-Greedy*

```
• With probability $\epsilon$
    – randomly choose $a$
• With probability 1-$\epsilon$
    – $a = argmax_a Q_i(s,a)$
```

In our implementation, the termination condition for Q-learning is dual. At regular intervals, specifically when the number of episodes completed is a multiple of 100, a checkpoint is reached. At this checkpoint, a comparison is made between the policy used in the current episode and the policy used in the episode that occurred 100 episodes prior. If the two are the same, we break. We also break if the current policy is the same as the result that policy iteration produced.
The ϵ parameter is given by the formula:

$$\epsilon = 0.01 + (1 - 0.01) * e^{-5 \cdot episode \cdot 10^{-5}}$$



1. Convergence to Optimal Policy

The Q-learning algorithm, when applied to the same toy example as in the policy iteration algorithm, eventually converges to the same optimal policy. However, it achieves this convergence with a significantly greater delay compared to policy iteration.

2. Parameter Tweaking and Comparison with Policy Iteration

Since running Q-Learning experiments was extremely time-consuming, we have experimented.