Project Report for Machine Learning Course

The University of Tehran

Name: Sirvan Nasiri Kia

Student ID: 250499257

Professor: Dr. Keyhanipour

Python programming language is used in this project. The matplotlib, tqdm, and time libraries should be installed using the pip command to run the program.

Note: When a graph is displayed as a plot, the program stops until it is closed. After closing, it continues.

The graph shows the available methods based on 10 actions. The number of games is 1000 and the number of repetitions is 2000.

Due to the high number of for loops (around 2,000,000 loops), running each method takes several minutes.

```
pip install matplotlib
pip install tqdm
pip install time
```

An RL class is written to execute the algorithms in this project. In the constructor of the class, the number of actions (actionsCount), the number of game stages (steps), and the number of repetitions (iterationsCount) are taken from the user. Also, the value of each action is randomly initialized from the uniform distribution in the range [0.0, 1.0].

Additionally, the Actions class is used to store the required data for each action. This class includes the following methods:

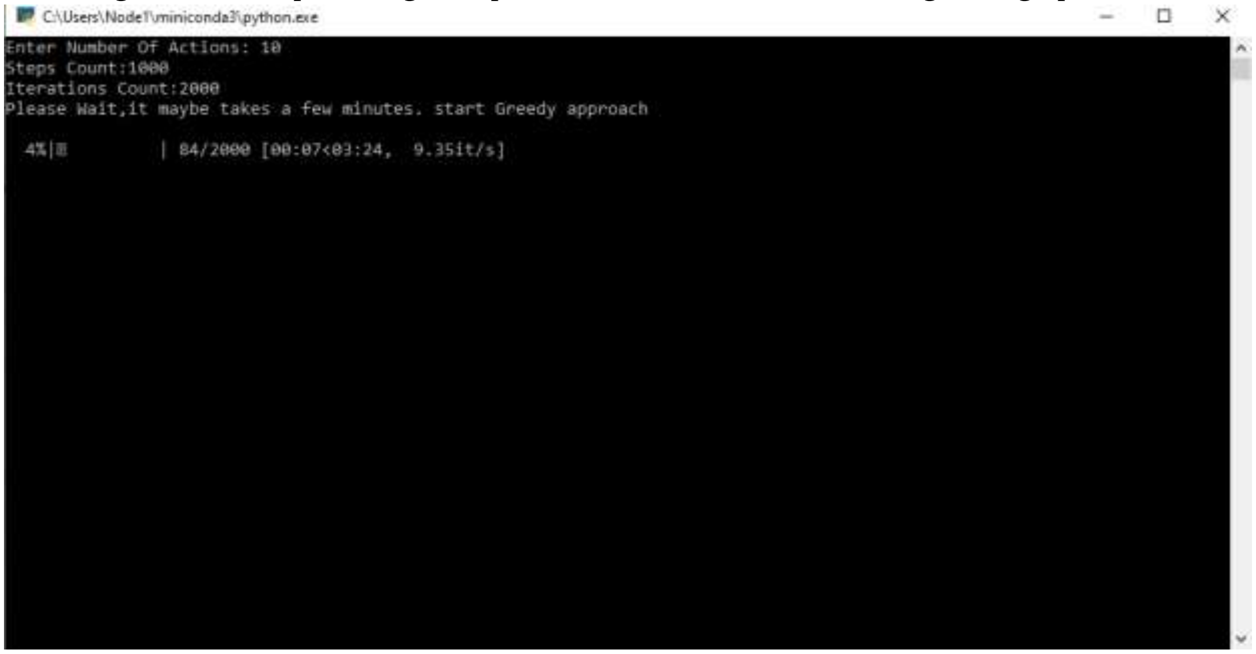
- GiveReward: This method returns a random value in the range [Q(a), 1] from the uniform distribution as r.

-update: This method takes the received reward value at time t as input. Then, it updates the value of Q(a) at time t based on the value at time t-1 (i.e., taking the average of r values up to time t).

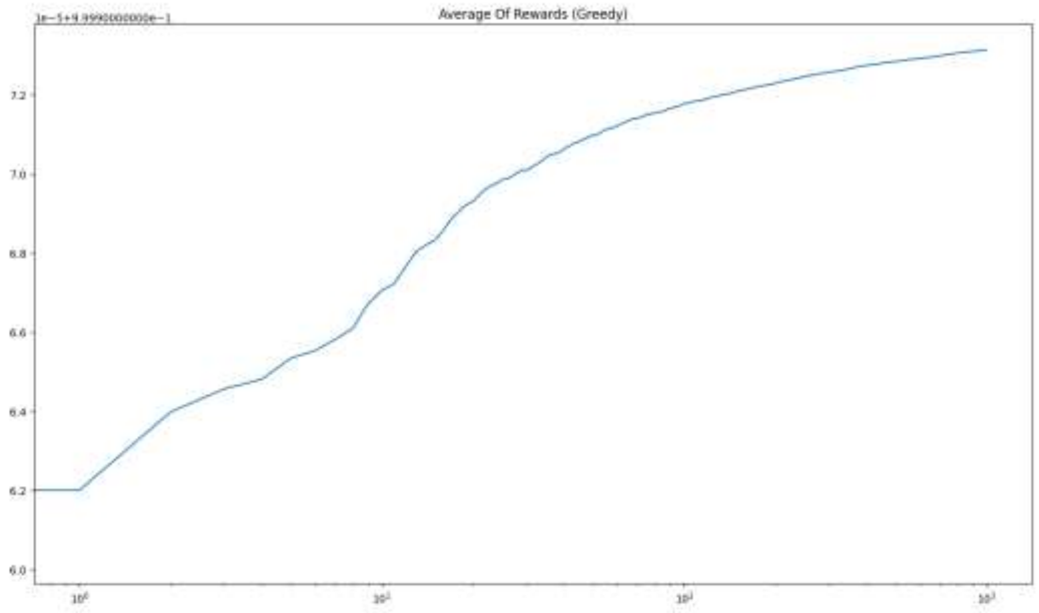
The Greedy algorithm:

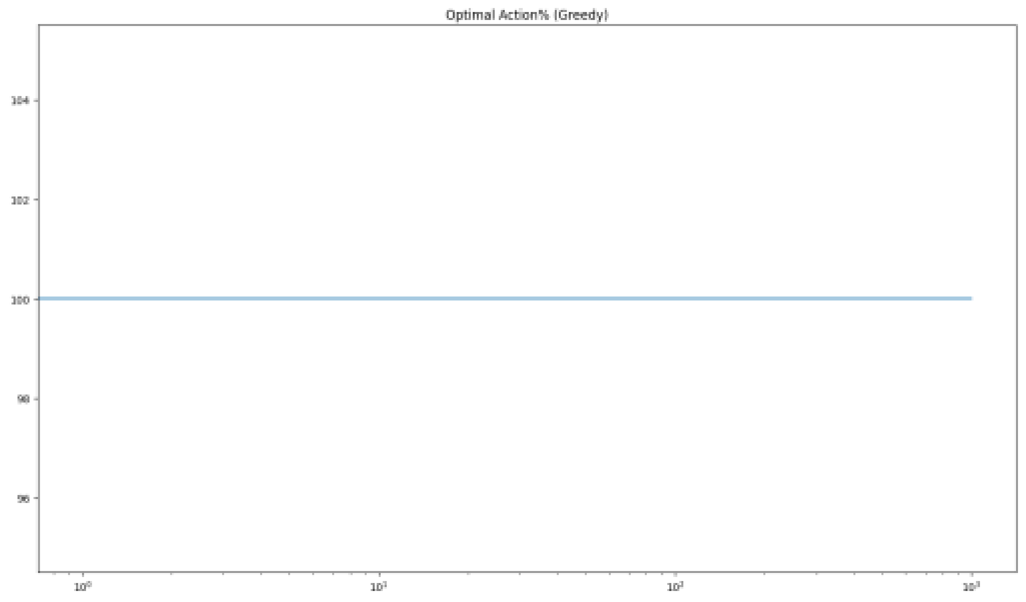
```
optimalActionIndex=np.argmax([a.quality for a in self.actions])
```

In this algorithm, we first obtain the optimal action value (based on the initial assignment of values to each action). Then, we execute the Greedy algorithm. Finally, we display the average rewards given and the percentage of optimal action selection at each stage as a graph.



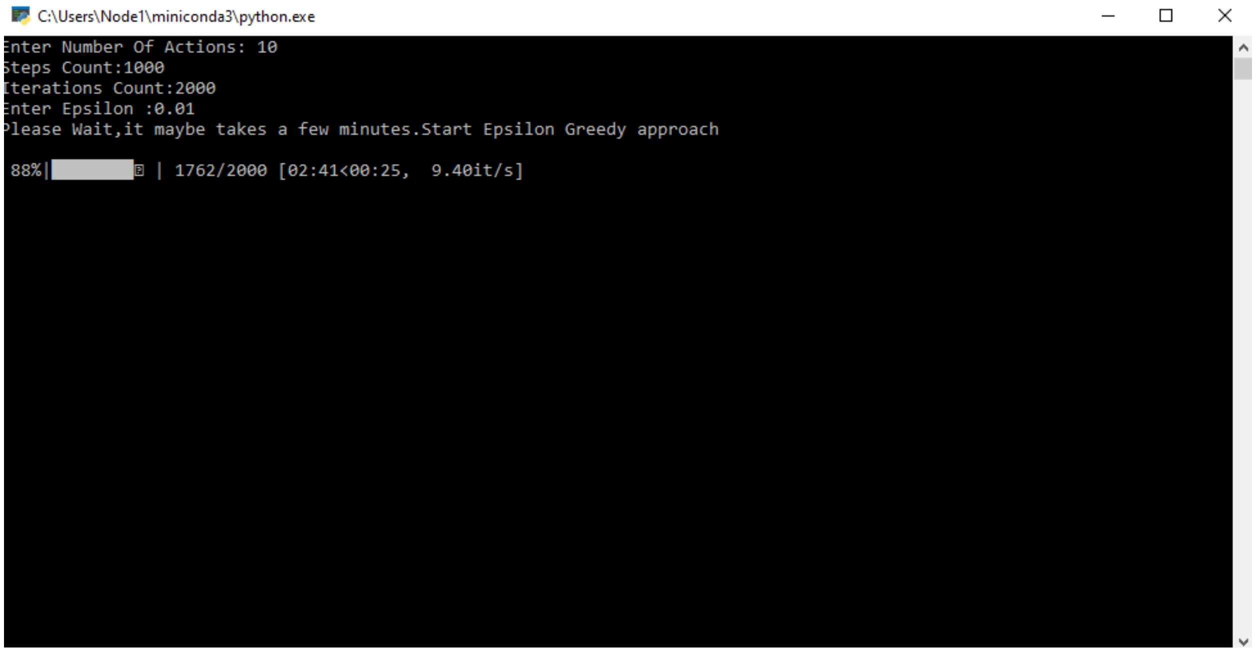
```
Enter Number Of Actions: 10
Steps Count:1000
Iterations Count:2000
Please Wait,it maybe takes a few minutes. start Greedy approach

  4%|▌         | 84/2000 [00:07<03:24,  9.35it/s]
```



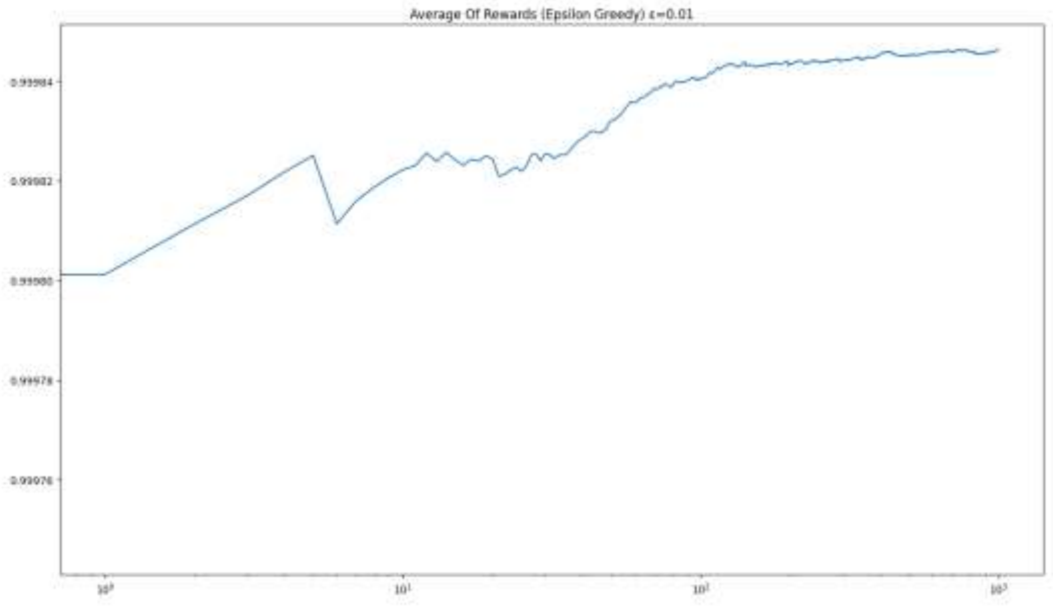


The ε-Greedy algorithm

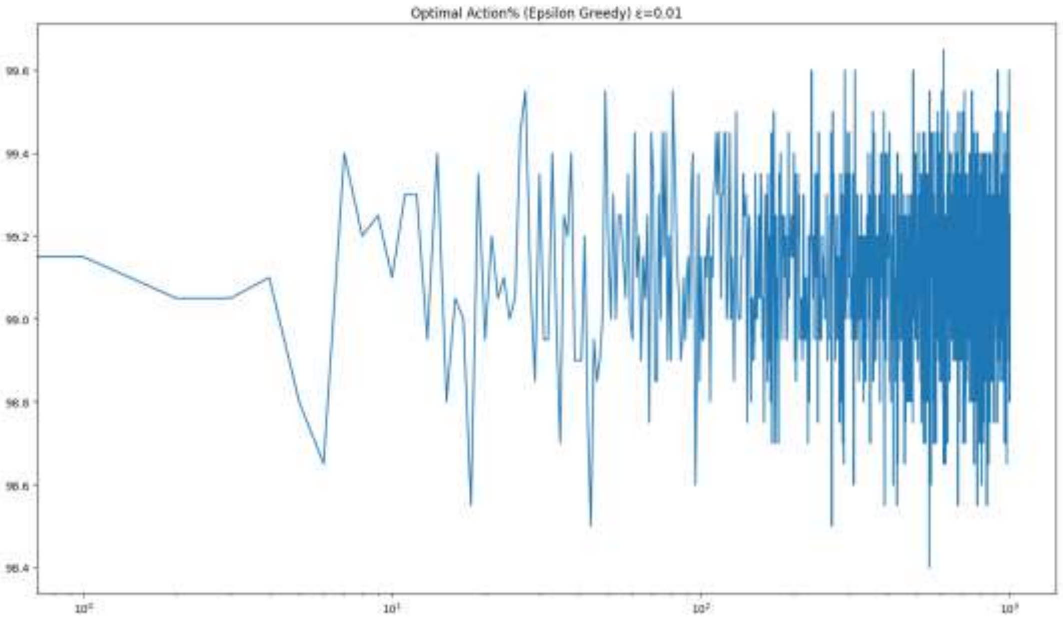
It's similar to the Greedy algorithm, with the difference that in each step, a random number between 0.0 and 1.0 is generated. Based on the value of this number and its comparison with the value of epsilon (which is received as input), we either choose the optimal action or select a random action. The average rewards given and the percentage of optimal action selection at each stage are displayed as a graph.



```
Enter Number Of Actions: 10
Steps Count:1000
Iterations Count:2000
Enter Epsilon :0.01
Please Wait,it maybe takes a few minutes.Start Epsilon Greedy approach

 88%|████████▊ | 1762/2000 [02:41<00:25,  9.40it/s]
```



Average Of Rewards (Epsilon Greedy) ε=0.01
0.99984
0.99982
0.99980
0.99978
0.99976
10^0
10^1
10^2
10^3



Optimal Action% (Epsilon Greedy) ε=0.01
99.6
99.4
99.2
99.0
98.8
98.6
98.4
10^0
10^1
10^2
10^3

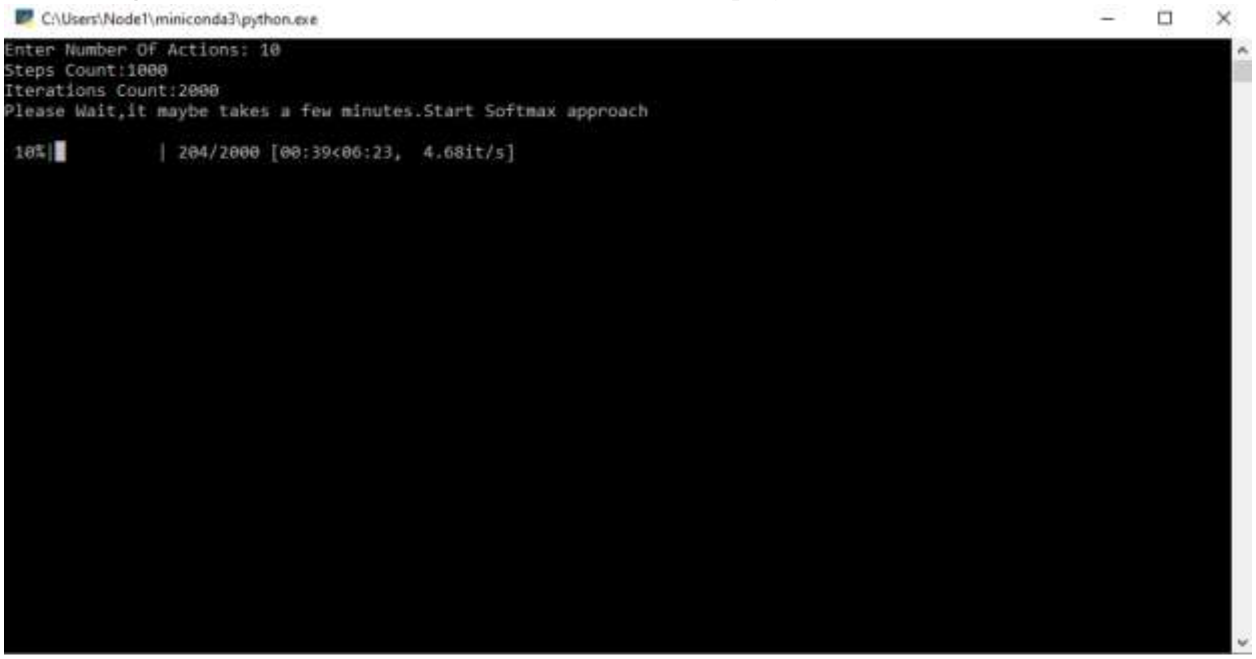
The Softmax algorithm

similar to the Greedy method, except that in this algorithm, instead of using the distribution below, the SoftmaxDis value is updated using the calculateSoftmaxDis method.

Choose action a on play t with probability

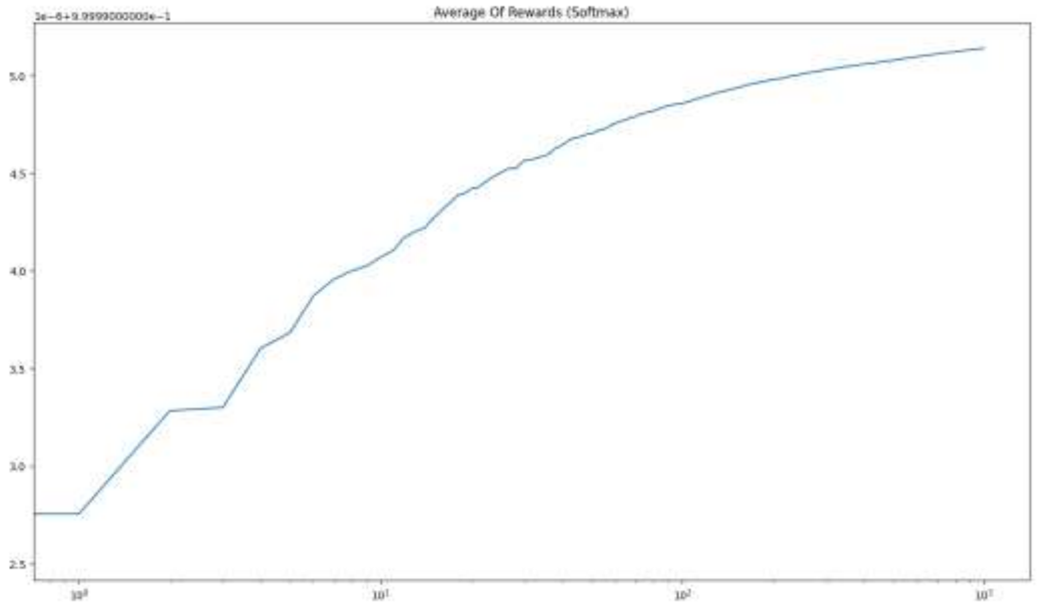
$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$

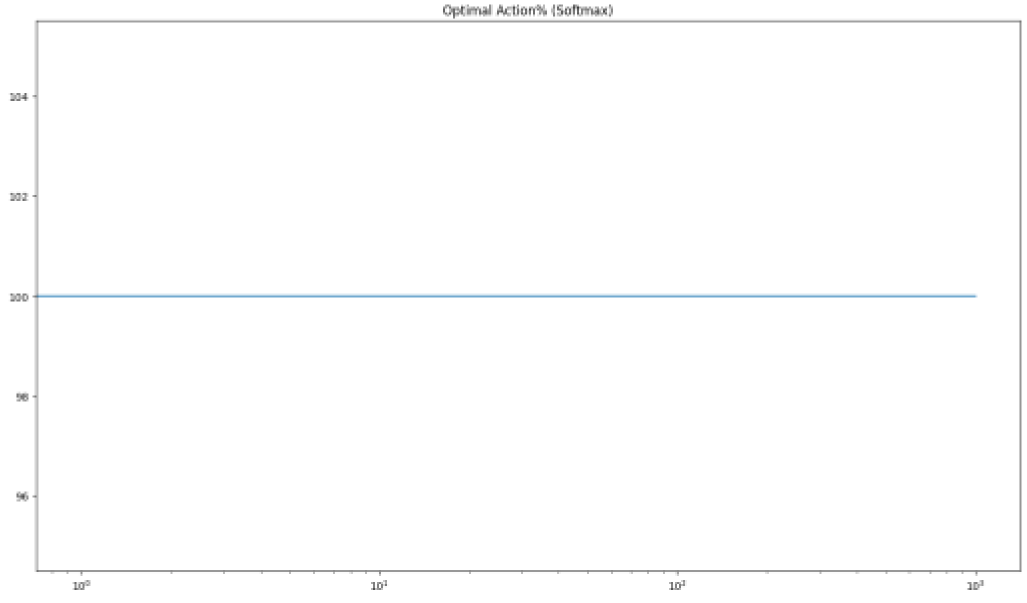
In this algorithm, the value of eta is set to 0.1, although it can be reduced as a decreasing function over time, such as 1/(step+1). Due to limitations of the exp function in Python and the out of range error, a constant value has been used in this project.



C:\Users\Node1\miniconda3\python.exe

```
Enter Number Of Actions: 10
Steps Count:1000
Iterations Count:2000
Please Wait,it maybe takes a few minutes.Start Softmax approach

 10%|█         | 204/2000 [00:39<06:23,  4.68it/s]
```


Average Of Rewards (Softmax)
1e−6+9.9999000000e−1
5.0
4.5
4.0
3.5
3.0
2.5
10^0
10^1
10^2
10^3


Optimal Action% (Softmax)
104
102
100
98
96
10^0
10^1
10^2
10^3