N-Armed-Bandit and epsilon-greedy Agent

***Abstract*—In this experiment we have to create n-armed bandit environment, epsilon-greedy algorithm/agent. And explore stationary and non-stationary rewards.**

I. Introduction

The goal of this project is to understand working n-armed bandit reinforcement learning task and epsilon greedy algorithm. N-arm-bandit is a classical Reinforcement Learning example where and agent chooses between ”n” actions and receives reward based on it. One thing to note is that in N-armed-bandit problem there is no concept of states as the agent always stays in single state. Examples of N-armed-bandit are like Doctor applying various types of treatments for disease and monitor the outcomes. In this we can either have stationary reward for an action or non-stationary reward(reward which changes with time). In these types of problems there is trade-off between exploration and exploitation.

1) Exploration

2) Exploitation

In exploration agent improves knowledge for long-term benefits. In exploitation agent exploit current knowledge for short term benefits(greedy). So we must balance exploration and exploitation to get best result. One general approach is exploit most of the time with small chance of exploring. [1]. We have simulated these problems and solved it using python in Vs-code. You can find all the links here:- (here)

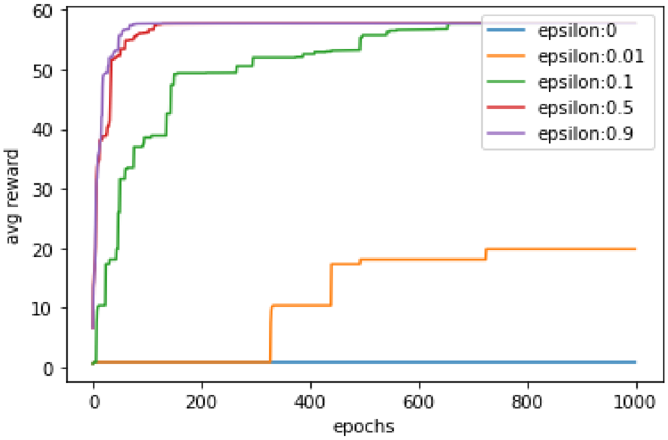
II. Theory

A. Epsilon-Greedy Action Selection

refers to probability of choosing to explore. Roll a dice if got 1 then explore so here exploring 1/6 times. [1].

$$Q_{n+1} \doteq Q_n + \alpha \left[R_n - Q_n \right] \tag{1}$$

We will see effect of epsilon on average reward:



B. Optimistic initial values

It encourages early exploration. But it has some limitations Limitations of Optimistic initial Values: (for now consider epsilon is 0)

It drives early exploration only.

It is not suited for non-stationary problem.

III. Procedure

A. Our implementation n-arm-bandit env and agent

In this we have created classes for env and separate class for agent. Agent uses epsilon greedy approach to find the next action. This action is passed to env class which returns reward which is again non stationary. as it has some mean and variance.

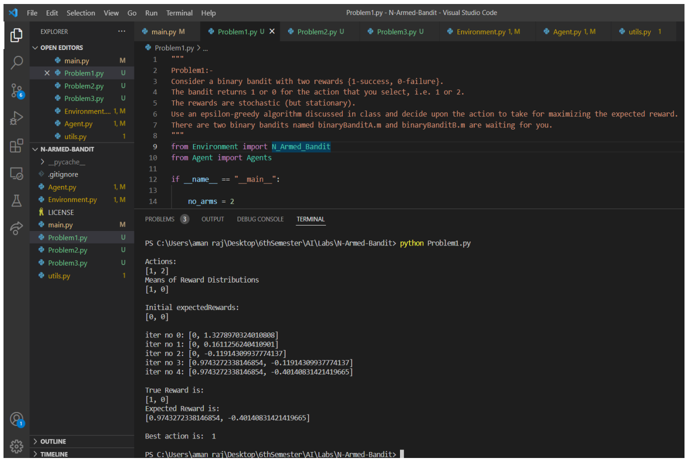
Now based on this returned reward agent updates it's expected reward.

B. Outputs:

We have total 3 problems. For each problem we have created a script named as Problem1.py, Problem2.py, Problem3.py respectively.

In first problem we have taken 2 arms which has reward of 1,0 and one respectively.

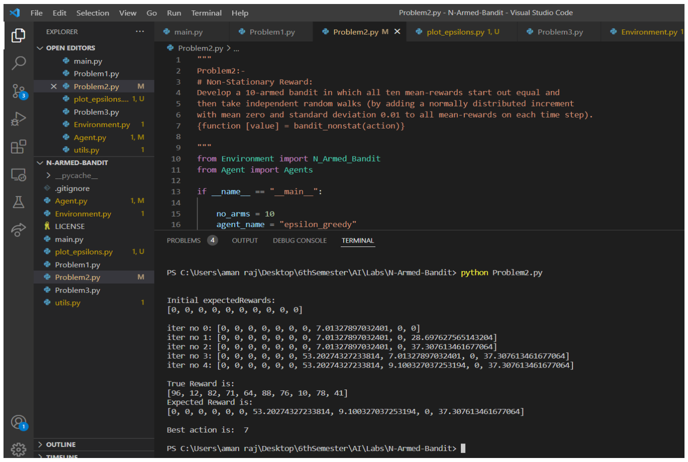
We train it for 5 epochs, then we find which action agent thinks is better and based on the expected rewards it suggest to take first action, thus our agent has learned well.



```python
"""
Problem1:-
Consider a binary bandit with two rewards {1-success, 0-failure}.
The bandit returns 1 or 0 for the action that you select, i.e. 1 or 2.
The rewards are stochastic (but stationary).
Use an epsilon-greedy algorithm discussed in class and decide upon the action to take for maximizing the expected reward.
There are two binary bandits named binaryBanditA.m and binaryBanditB.m are waiting for you.
"""
from Environment import N_Armed_Bandit
from Agent import Agents

if __name__ == "__main__":

    no_arms = 2
```

```
PS C:\Users\aman raj\Desktop\6thSemester\AI\Labs\N-Armed-Bandit> python Problem1.py

Actions:
[1, 2]
Means of Reward Distributions
[1, 0]

Initial expectedRewards:
[0, 0]

iter no 0: [0, 1.3278970324010808]
iter no 1: [0, 0.1611256240410901]
iter no 2: [0, -0.11914309937774137]
iter no 3: [0.9743272338146854, -0.11914309937774137]
iter no 4: [0.9743272338146854, -0.40140831421419665]

True Reward is:
[1, 0]
Expected Reward is:
[0.9743272338146854, -0.40140831421419665]

Best action is:  1

PS C:\Users\aman raj\Desktop\6thSemester\AI\Labs\N-Armed-Bandit>
```

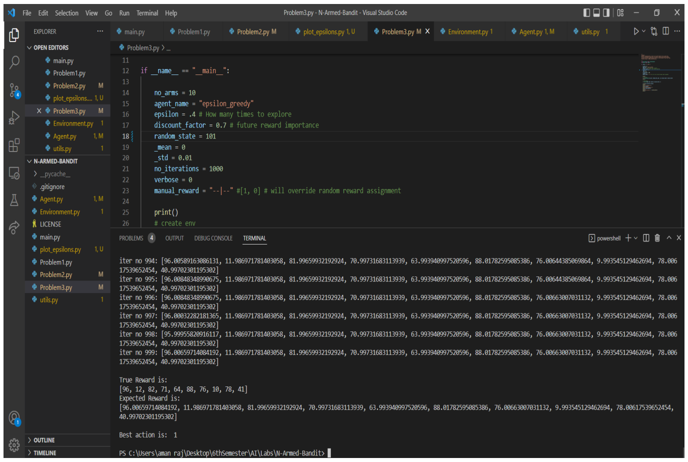
Next in Problem2 we have 10 armed problem with non-stationary reward. Reward has mean of 0 with standard deviation of 0.01. So for this problem we run it for 5 iterations. In this we realize that our agent is not able to decide properly the optimal action. In this we see that first action will return optimal reward but our agent is not able to calculate optimal point given we train it for very less no of iterations(5) and thus it returns 7 as optimal action.



```python
"""
Problem2:-
# Non-Stationary Reward:
Develop a 10-armed bandit in which all ten mean-rewards start out equal and
then take independent random walks (by adding a normally distributed increment
with mean zero and standard deviation 0.01 to all mean-rewards on each time step).
{function [value] = bandit_nonstat(action)}

"""
from Environment import N_Armed_Bandit
from Agent import Agents

if __name__ == "__main__":

    no_arms = 10
    agent_name = "epsilon_greedy"
```

```
PS C:\Users\aman raj\Desktop\6thSemester\AI\Labs\N-Armed-Bandit> python Problem2.py


Initial expectedRewards:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

iter no 0: [0, 0, 0, 0, 0, 0, 0, 7.01327897032401, 0, 0]
iter no 1: [0, 0, 0, 0, 0, 0, 0, 7.01327897032401, 0, 28.697627565143204]
iter no 2: [0, 0, 0, 0, 0, 0, 0, 7.01327897032401, 0, 37.307613461677064]
iter no 3: [0, 0, 0, 0, 0, 0, 53.20274327233814, 7.01327897032401, 0, 37.307613461677064]
iter no 4: [0, 0, 0, 0, 0, 0, 53.20274327233814, 9.100327037253194, 0, 37.307613461677064]

True Reward is:
[96, 12, 82, 71, 64, 88, 76, 10, 78, 41]
Expected Reward is:
[0, 0, 0, 0, 0, 0, 53.20274327233814, 9.100327037253194, 0, 37.307613461677064]

Best action is:  7

PS C:\Users\aman raj\Desktop\6thSemester\AI\Labs\N-Armed-Bandit> ▌
```

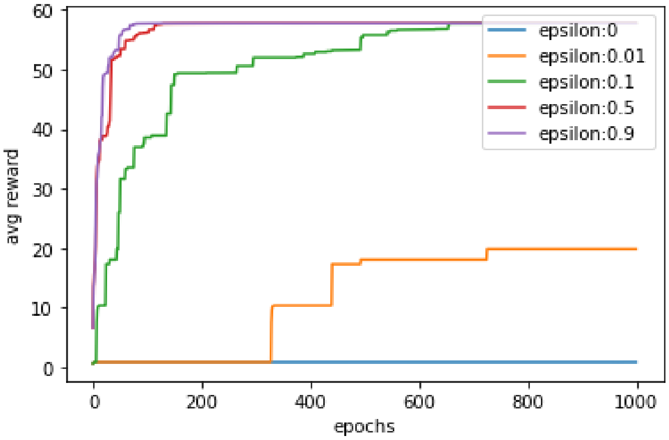
Next in Problem3 we have again same 10 arm problem with non-stationary reward. Reward has mean of 0 with standard deviation of 0.01. So for this problem we run it for 1000 iterations. This time our agent is able to decide properly the optimal action. In this we see that first action will return optimal reward but our agent is able to calculate optimal point given we train it for 1000 iterations and thus it returns 1 as optimal action.



```python

if __name__ == "__main__":

    no_arms = 10
    agent_name = "epsilon_greedy"
    epsilon = .4 # How many times to explore
    discount_factor = 0.7 # future reward importance
    random_state = 101
    _mean = 0
    _std = 0.01
    no_iterations = 1000
    verbose = 0
    manual_reward = "--|--" #[1, 0] # will override random reward assignment

    print()
    # create env
```

```
iter no 994: [96.00589163086131, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00644385069864, 9.993545129462694, 78.006
17539652454, 40.99702301195302]
iter no 995: [96.00848348990675, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00644385069864, 9.993545129462694, 78.006
17539652454, 40.99702301195302]
iter no 996: [96.00848348990675, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00663007031132, 9.993545129462694, 78.006
17539652454, 40.99702301195302]
iter no 997: [96.00032282181365, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00663007031132, 9.993545129462694, 78.006
17539652454, 40.99702301195302]
iter no 998: [95.99955820916117, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00663007031132, 9.993545129462694, 78.006
17539652454, 40.99702301195302]
iter no 999: [96.00659714084192, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00663007031132, 9.993545129462694, 78.006
17539652454, 40.99702301195302]

True Reward is:
[96, 12, 82, 71, 64, 88, 76, 10, 78, 41]
Expected Reward is:
[96.00659714084192, 11.986971781403058, 81.99659932192924, 70.99731683113939, 63.993940997520596, 88.01782595085386, 76.00663007031132, 9.993545129462694, 78.00617539652454,
40.99702301195302]

Best action is:  1

PS C:\Users\aman raj\Desktop\6thSemester\AI\Labs\N-Armed-Bandit> ▌
```

At the end we have ran different agents with different epsilon values [0,0.01, 0.1, 0.5,0.9] and plotted their average reward. And we realize that on an average epsilon of 0.5 gives better result.
We also notice that when epsilon is 0 our agent always finds next reward greedily and thus is never able to explore. Due to this as we can see that our agent is stuck in local optimal point.



IV. Conclusion

We see that exploration and exploitation trade-off has huge impact on our final agent. So we must set epsilon wisely depending on our problem statement. This experiment was very interesting I got to learn how to formulate classical Reinforcement Learning problem using n-armed-bandit and epsilon-greedy agent, I also learned about stationary and non-stationary reward environments and how to implement them.

References

[1] Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto (second edition)