**CS698R: Deep Reinforcement Learning**

# Assignment #1

**Name**: ROMIT MOHANTY
**Roll NO.**: 190720
**GitHub Repo Link**: Repository Link

## Solution to Problem 1: Multi-armed Bandits

1. 3 is globally used as a seed for all random number generation procedure. $\alpha$ and $\beta$ are the probabilities of going in the direction of action left and right respectively. If both $\alpha$ and $\beta$ are **1** then we should always get a **reward 1**. If either of them is **1**, then we should get a **reward of 1 whenever we take that action**. I took some test cases and ran them for 10 episodes respectively.

   - **test case:** [1,1](should always give a reward of 1)
   - **output:**
     Episode:1 End State: 0 Reward:1 Action:0
     Episode:2 End State: 0 Reward:1 Action:0
     Episode:3 End State: 2 Reward:1 Action:1
     Episode:4 End State: 2 Reward:1 Action:1
     Episode:5 End State: 0 Reward:1 Action:0
     Episode:6 End State: 0 Reward:1 Action:0
     Episode:7 End State: 2 Reward:1 Action:1
     Episode:8 End State: 2 Reward:1 Action:1
     Episode:9 End State: 0 Reward:1 Action:0
     Episode:10 End State: 0 Reward:1 Action:0
   - **test case:** [1,0] (should give 0 reward for every right action and 1 reward for every left action)
   - **output:**
     Episode:1 End State: 0 Reward:1 Action:0
     Episode:2 End State: 0 Reward:1 Action:0
     Episode:3 End State: 0 Reward:0 Action:1
     Episode:4 End State: 0 Reward:0 Action:1
     Episode:5 End State: 0 Reward:1 Action:0
     Episode:6 End State: 0 Reward:1 Action:0
     Episode:7 End State: 0 Reward:0 Action:1
     Episode:8 End State: 0 Reward:0 Action:1
     Episode:9 End State: 0 Reward:1 Action:0
     Episode:10 End State: 0 Reward:1 Action:0
   - **test case:** [0.5,0.5] (We can't tell much about which action will give what reward and everything is very random)
   - **output:**
     Episode:1 End State: 2 Reward:0 Action:0
     Episode:2 End State: 2 Reward:0 Action:0
     Episode:3 End State: 0 Reward:0 Action:1
     Episode:4 End State: 2 Reward:1 Action:1
     Episode:5 End State: 2 Reward:0 Action:0
     Episode:6 End State: 2 Reward:0 Action:0
     Episode:7 End State: 0 Reward:0 Action:1
     Episode:8 End State: 0 Reward:0 Action:1
     Episode:9 End State: 0 Reward:1 Action:0
     Episode:10 End State: 0 Reward:1 Action:0

2. σ and μ is the variance and the mean of the Gaussian Distribution from which our environments samples the rewards. It is clear that if $\sigma = 0$ then our reward of each action should be the respective μİ took the above mentioned value and ran it for 10 episodes. I also ran it for $\mu = 0$ and $\sigma = 1$ and the observations are:

   - **$\sigma = 0$, $\mu = 1$**
   - **Output:** All Rewards irrespective of actions are 1
     Episode:1 End State: 4 Reward:1.0 Action:3
     Episode:2 End State: 10 Reward:1.0 Action:9
     Episode:3 End State: 9 Reward:1.0 Action:8
     Episode:4 End State: 3 Reward:1.0 Action:2
     Episode:5 End State: 6 Reward:1.0 Action:5
     Episode:6 End State: 10 Reward:1.0 Action:9
     Episode:7 End State: 8 Reward:1.0 Action:7
     Episode:8 End State: 10 Reward:1.0 Action:9
     Episode:9 End State: 2 Reward:1.0 Action:1
     Episode:10 End State: 10 Reward:1.0 Action:9

   - **$\sigma = 1$, $\mu = 0$**
   - **Output:** All Rewards are random
     Episode:1 End State: 4 Reward:-3.177357456727173 Action:3
     Episode:2 End State: 10 Reward:0.40740435014008197 Action:9
     Episode:3 End State: 9 Reward:0.8374998732316016 Action:8
     Episode:4 End State: 3 Reward:1.806070531724957 Action:2
     Episode:5 End State: 6 Reward:-0.30472533709212657 Action:5
     Episode:6 End State: 10 Reward:-0.8818954449603935 Action:9
     Episode:7 End State: 8 Reward:-1.1723606244433777 Action:7
     Episode:8 End State: 10 Reward:-2.023695345942471 Action:9
     Episode:9 End State: 2 Reward:1.4188772847701494 Action:1
     Episode:10 End State: 10 Reward:-1.5782856604709783 Action:9

3. (a) **Pure Exploitation**:
   I ran this for 10 episodes and the results are self explanatory. Once the agent got a reward from Action 1 (right) we can see that it never takes the left action again.

| Episode no. | Q estimate for Action 0 | Q estimate for Action 1 | Rewards | Action Taken |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0.5 | 1 | 1 |
| 6 | 0 | 0.666667 | 1 | 1 |
| 7 | 0 | 0.75 | 1 | 1 |
| 8 | 0 | 0.6 | 0 | 1 |
| 9 | 0 | 0.5 | 0 | 1 |
| 10 | 0 | 0.42857143 | 0 | 1 |

(b) **Pure Exploration**:
I ran this for 10 episodes and the results are self explanatory. We can see that in spite getting an initial reward from taking action-1 the agent still continues to take action 0 to explore different possibilities.

| Episode no. | Q estimate for Action 0 | Q estimate for Action 1 | Rewards | Action Taken |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0.5 | 1 | 1 |
| 6 | 0 | 0.5 | 0 | 0 |
| 7 | 0 | 0.5 | 0 | 0 |
| 8 | 0 | 0.333333 | 0 | 1 |
| 9 | 0 | 0.25 | 0 | 1 |
| 10 | 0.2 | 0.25 | 1 | 0 |

(c) **Epsilon Greedy**:
I ran this for 10 episodes and the results are self explanatory. I had taken a very less $\varepsilon = 0.1$ hence it never explores after getting a reward from Action-1

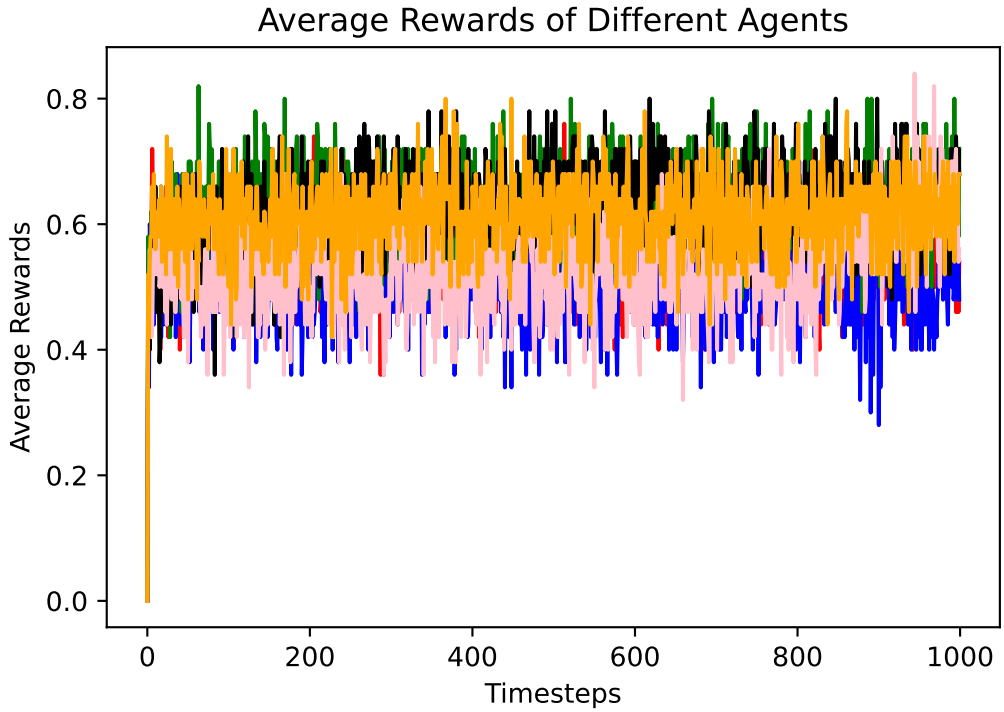
| Episode no. | Q estimate for Action 0 | Q estimate for Action 1 | Rewards | Action Taken |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0.5 | 0 | 1 |
| 5 | 0 | 0.66667 | 1 | 1 |
| 6 | 0 | 0.75 | 1 | 1 |
| 7 | 0 | 0.8 | 1 | 1 |
| 8 | 0 | 0.66667 | 0 | 1 |
| 9 | 0 | 0.57142857 | 0 | 1 |
| 10 | 0 | 0.5 | 0 | 1 |

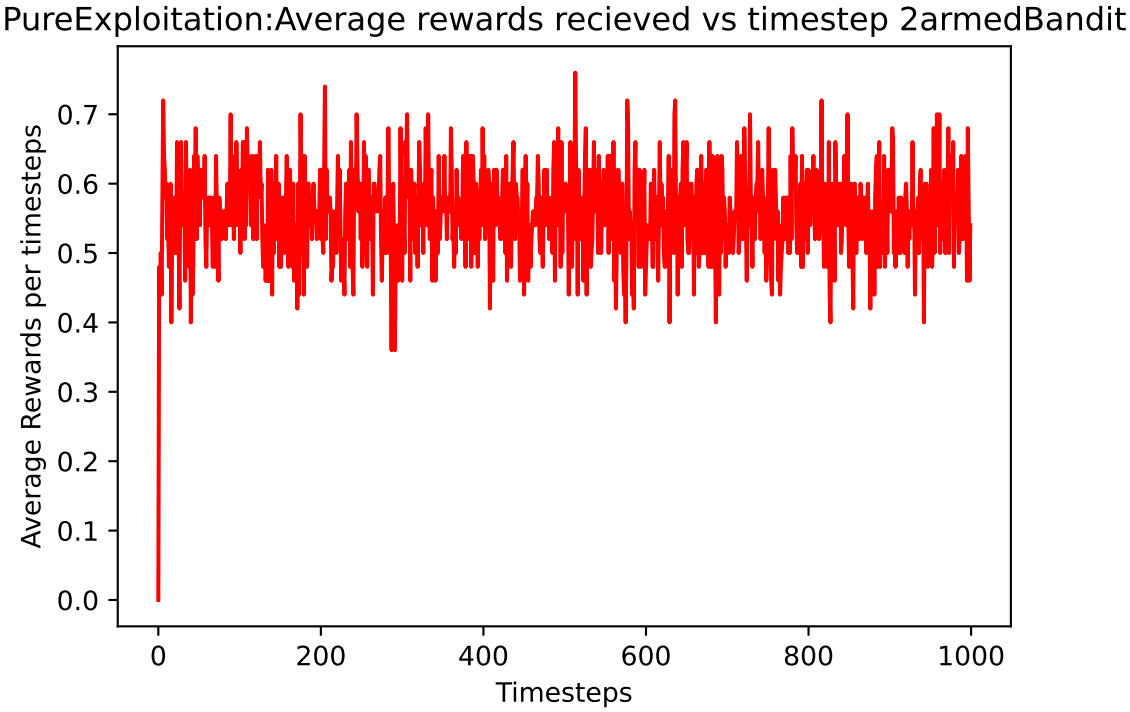
(d) **Epsilon Greedy**:
I ran this for 10 episodes and the results are self explanatory. I had taken a large $\varepsilon = 0.5$ hence it explores in some cases even after getting a reward from action -1

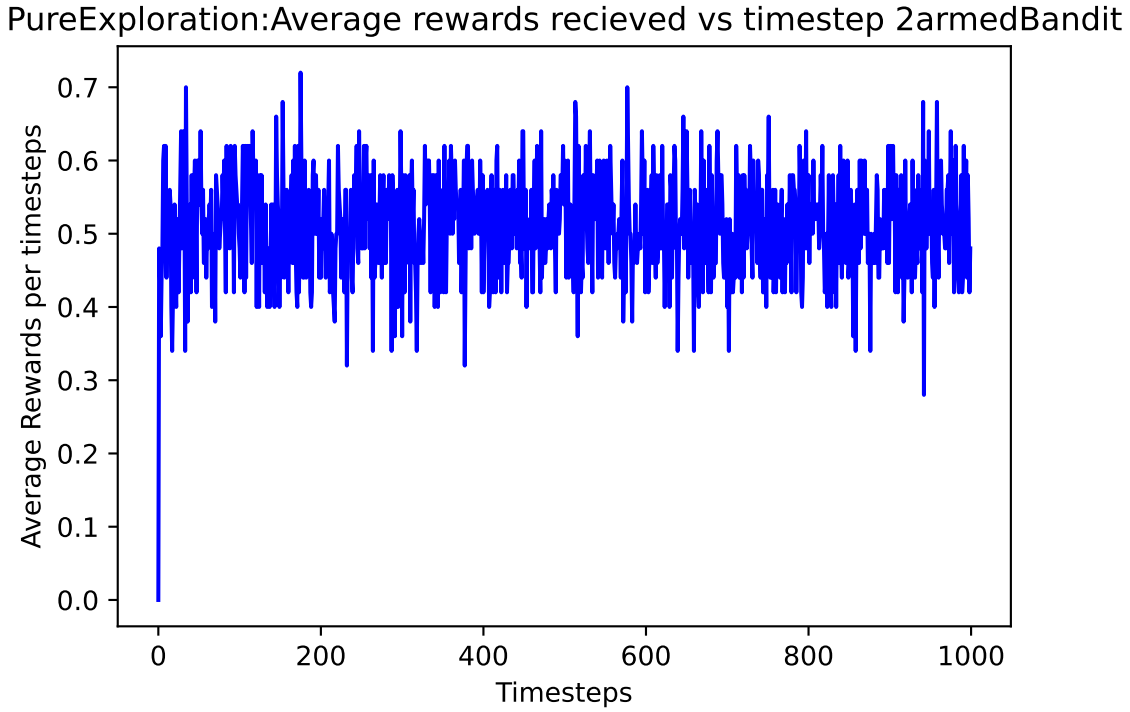
| Episode no. | Q estimate for Action 0 | Q estimate for Action 1 | Rewards | Action Taken |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0.5 | 0 | 1 |
| 5 | 0 | 0.66667 | 1 | 1 |
| 6 | 0 | 0.66667 | 0 | 0 |
| 7 | 0 | 0.75 | 1 | 1 |
| 8 | 0 | 0.6 | 0 | 1 |
| 9 | 0.33333 | 0.6 | 1 | 0 |
| 10 | 0.5 | 0.6 | 1 | 0 |

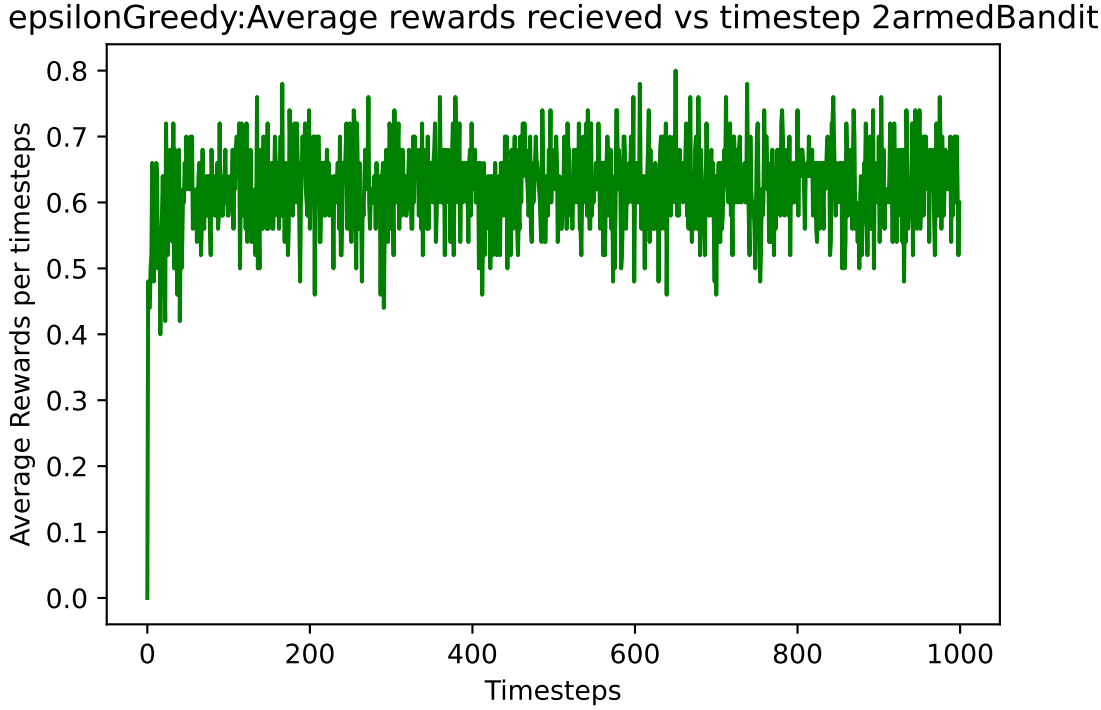
4. Since the plot has a lot of noise and for the given parameters could not converge till the end and had a high variance. So in the next pages I have individually compared different Agents using different values of different settings. Below is the legend for the below plot.We can see that epsilon greedy performs better than greedy because it explores with some small probability and then takes the best action which the pure exploitation often misses. Also due to similar lines softmax performs even better on longer run. If we compare 2 ε-greedy with different values, the one with the lower εperforms the best on the long run because once it has explored all options it takes the best action with probability more than the higher one. But its convergence is slower.UCB performs the best, but the small spikes can be concluded as a result of some exploration that it does on the longer run when its confidence parameter falls down.

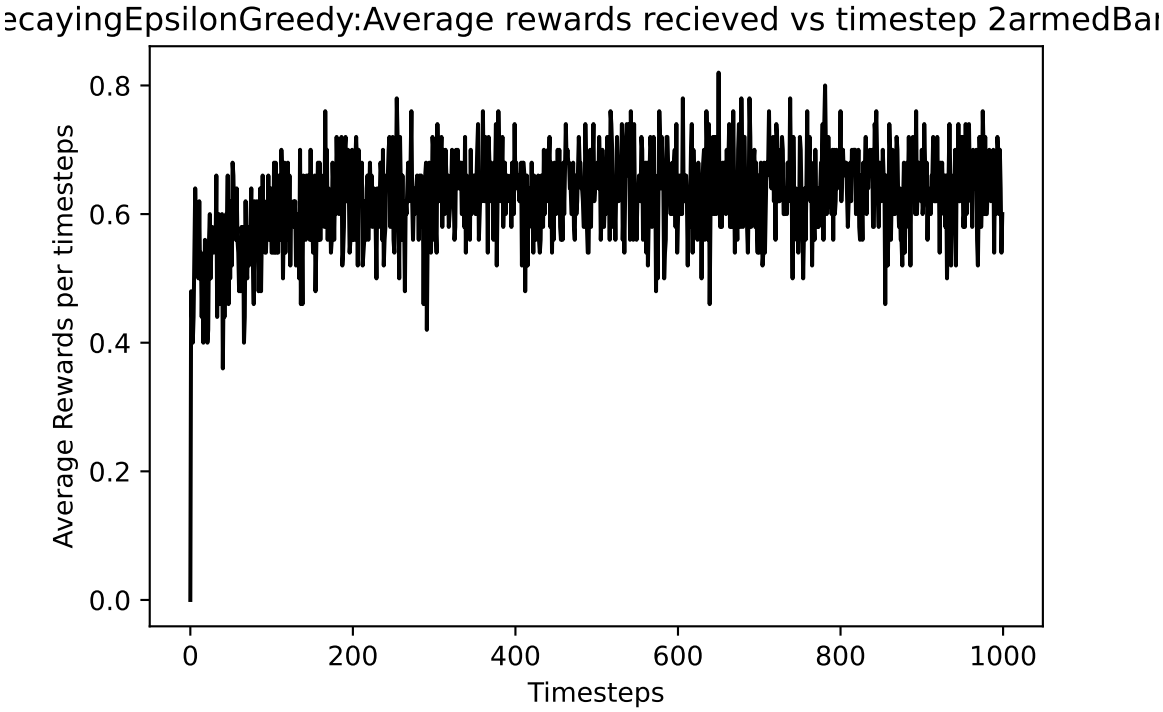
| | Color | Agent |
|---|---|---|
| | Red | Pure Exploitation |
| | Blue | Pure Exploration |
| **Legend** | Green | epsilon Greedy |
| | Black | Decaying Epsilon Greedy |
| | Pink | Soft Max Exploration |
| | Orange | UCB Exploration |

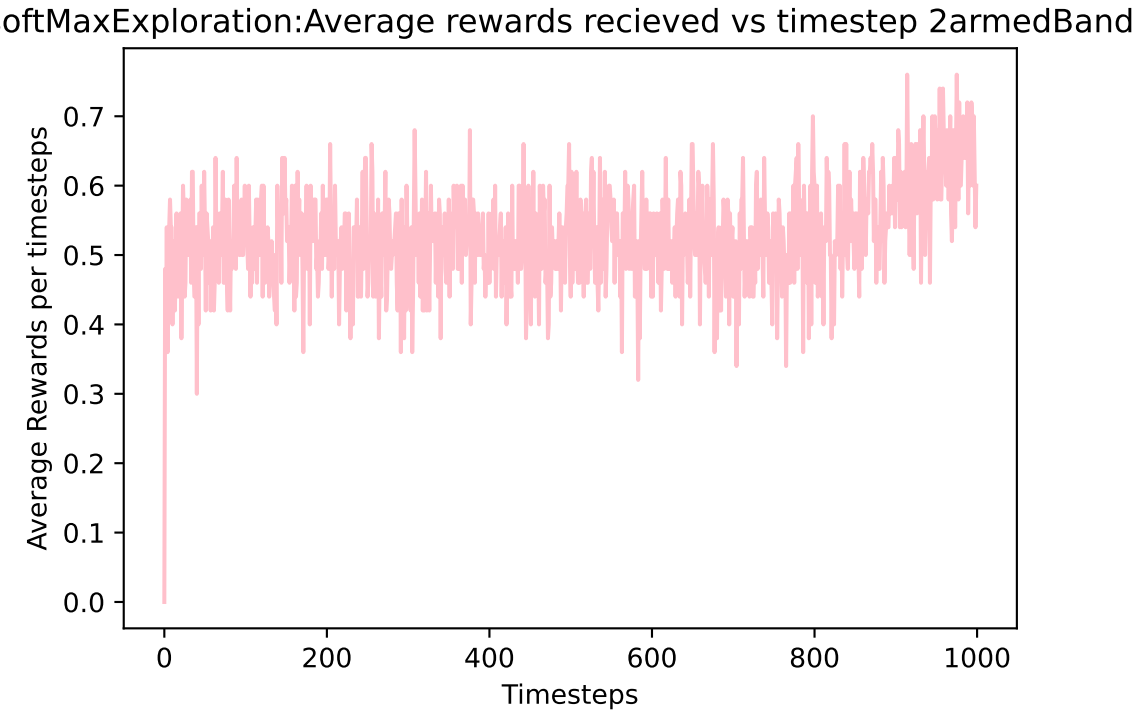


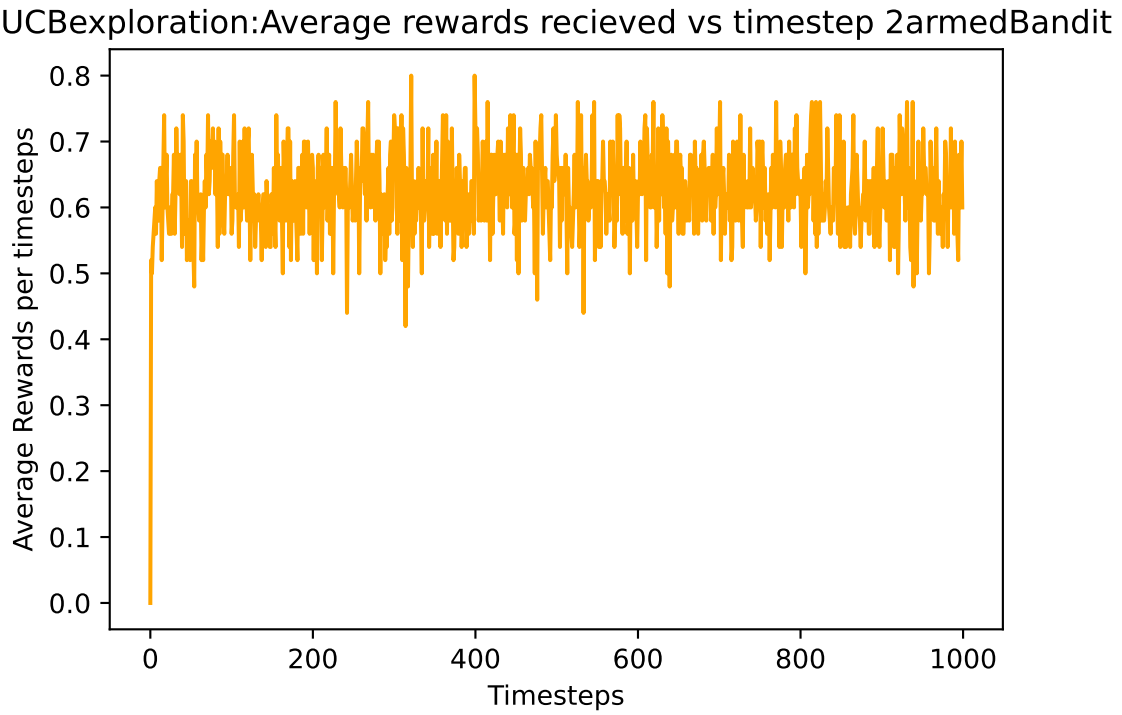

PureExploitation:Average rewards recieved vs timestep 2armedBandit
Average Rewards per timesteps
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
200
400
600
800
1000
Timesteps


PureExploration:Average rewards recieved vs timestep 2armedBandit
Average Rewards per timesteps
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
200
400
600
800
1000
Timesteps


epsilonGreedy:Average rewards recieved vs timestep 2armedBandit
Average Rewards per timesteps
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
200
400
600
800
1000
Timesteps

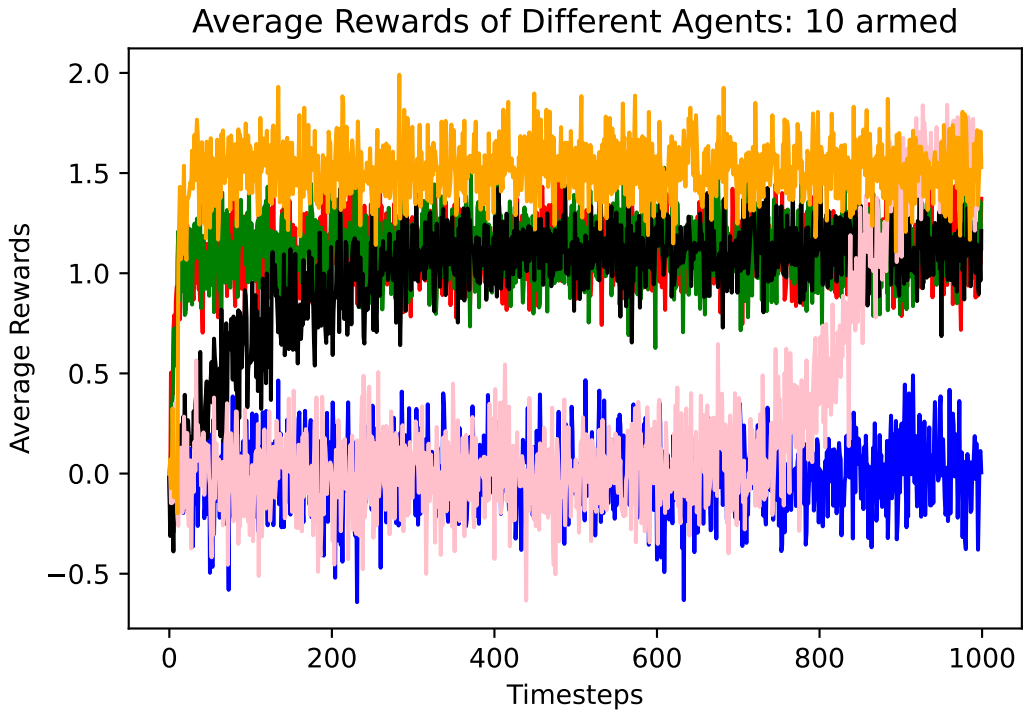

ecayingEpsilonGreedy:Average rewards recieved vs timestep 2armedBa
Average Rewards per timesteps
0.0
0.2
0.4
0.6
0.8
0
200
400
600
800
1000
Timesteps

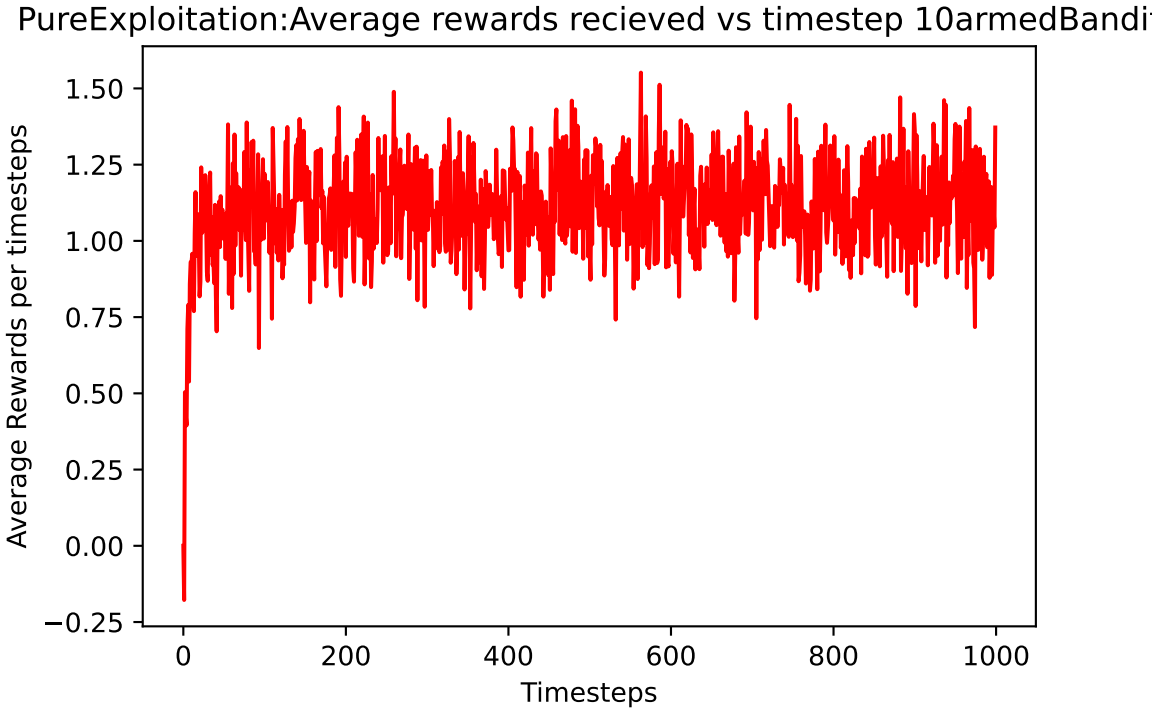

softMaxExploration:Average rewards recieved vs timestep 2armedBand
Average Rewards per timesteps
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
200
400
600
800
1000
Timesteps

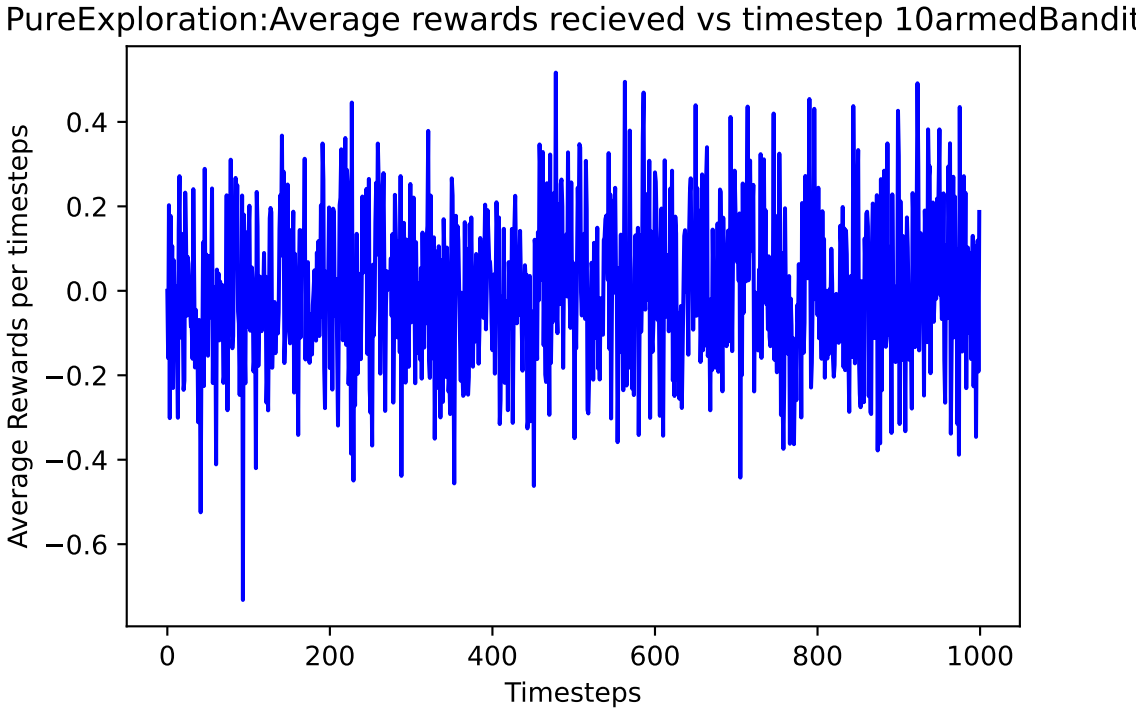

UCBexploration:Average rewards recieved vs timestep 2armedBandit
Average Rewards per timesteps
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
200
400
600
800
1000
Timesteps

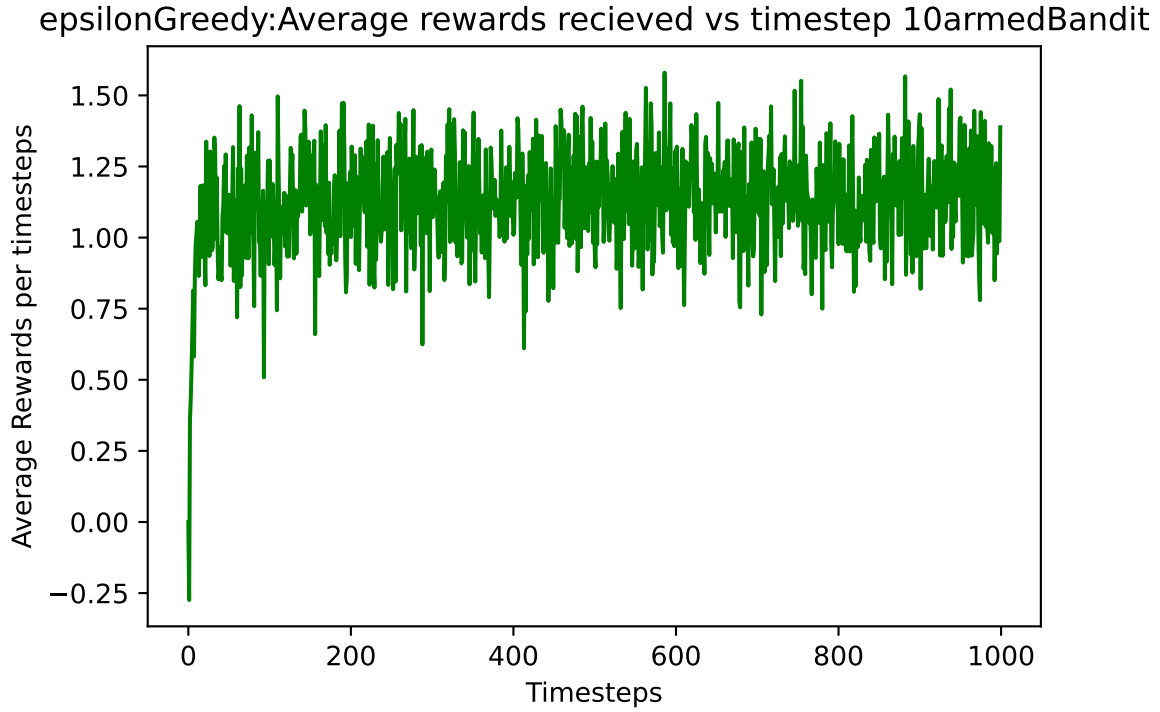
5. Since the plot has a lot of noise and for the given parameters could not converge till the end and had a high variance. So in the next pages I have individually compared different Agents in a 10 armed Bernoulli Bandit environment using different values of different settings. Below is the legend for the below plot. We can see that epsilon greedy performs better than greedy because it explores with some small probability and then takes the best action which the pure exploitation often misses. Also due to similar lines softmax performs even better on longer run. If we compare 2 ε-greedy with different values, the one with the lower εperforms the best on the long run because once it has explored all options it takes the best action with probability more than the higher one. But its convergence is slower.UCB performs the best, but the small spikes can be concluded as a result of some exploration that it does on the longer run when its confidence parameter falls down.

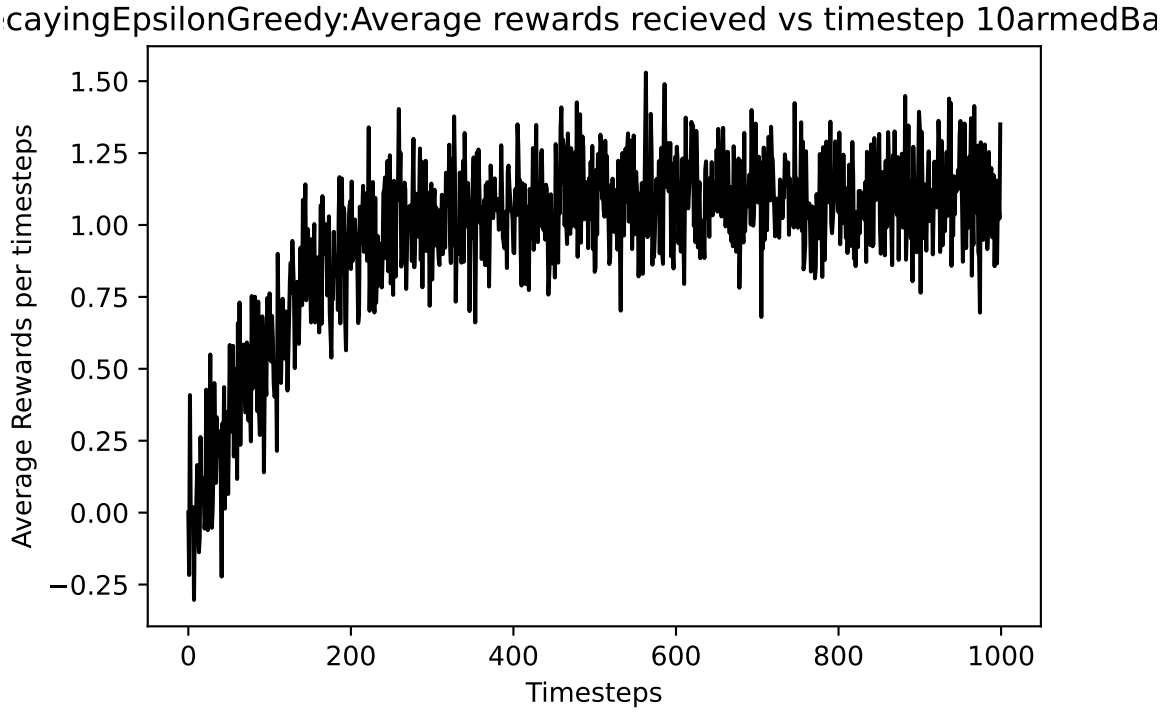
| | Color | Agent |
|---|---|---|
| | Red | Pure Exploitation |
| | Blue | Pure Exploration |
| **Legend** | Green | epsilon Greedy |
| | Black | Decaying Epsilon Greedy |
| | Pink | Soft Max Exploration |
| | Orange | UCB Exploration |

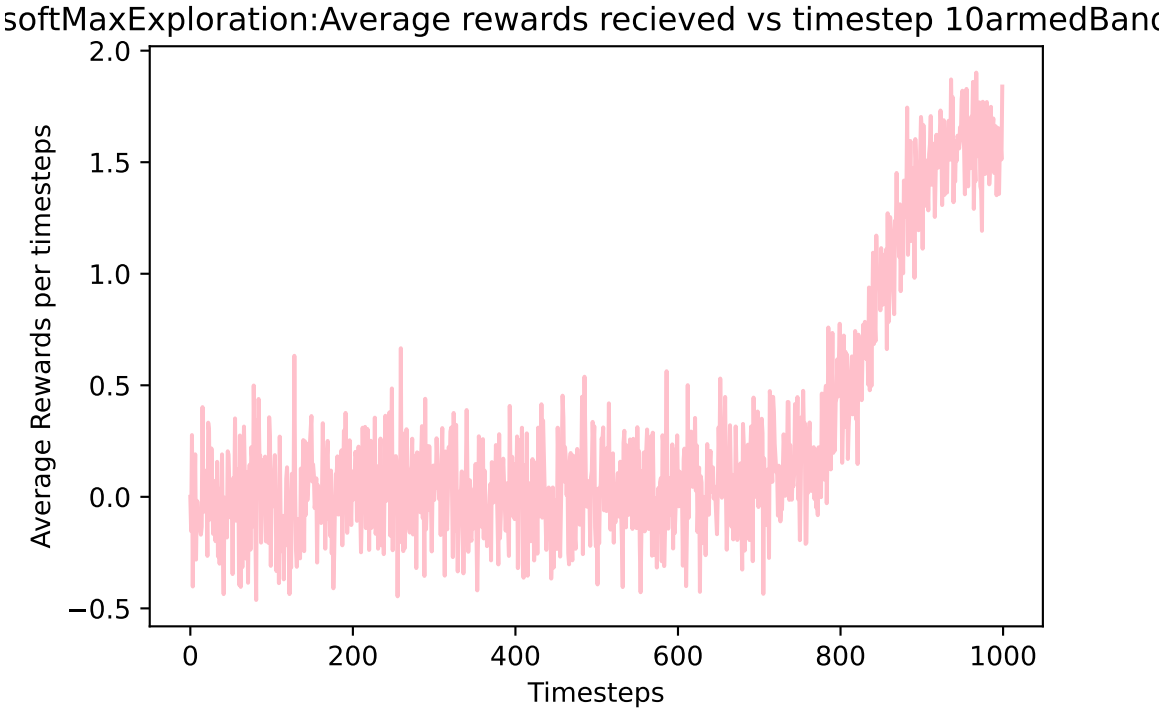


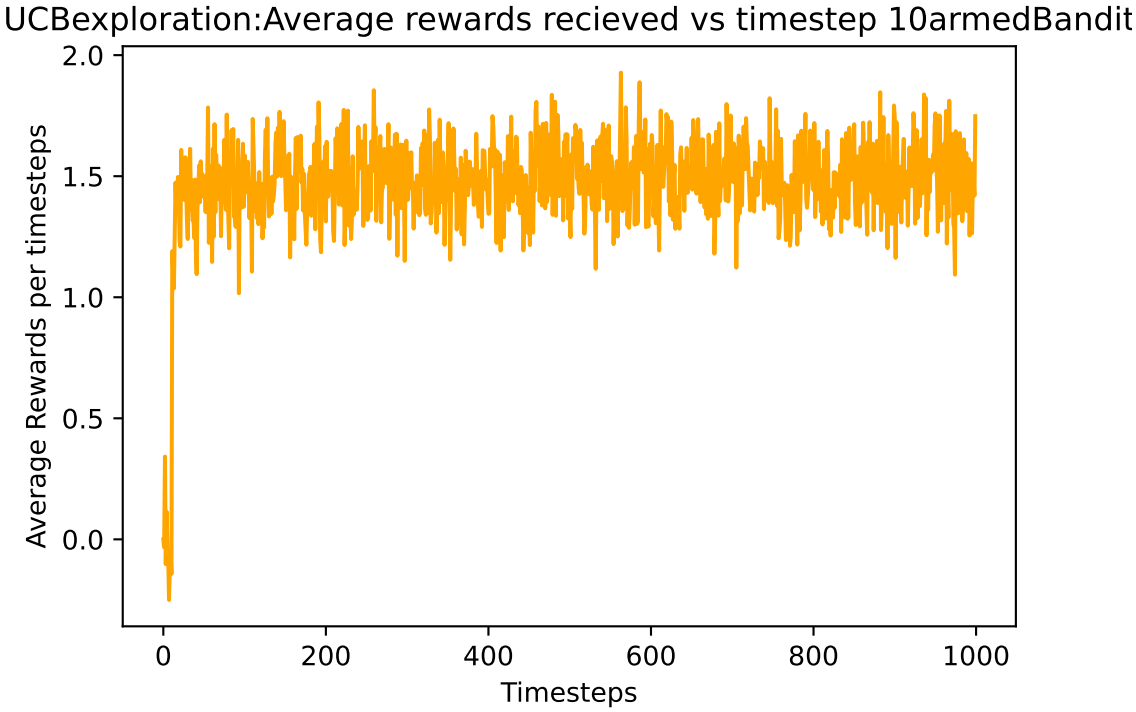

PureExploitation:Average rewards recieved vs timestep 10armedBandi
Average Rewards per timesteps
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
200
400
600
800
1000
Timesteps


PureExploration:Average rewards recieved vs timestep 10armedBandit
Average Rewards per timesteps
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
200
400
600
800
1000
Timesteps

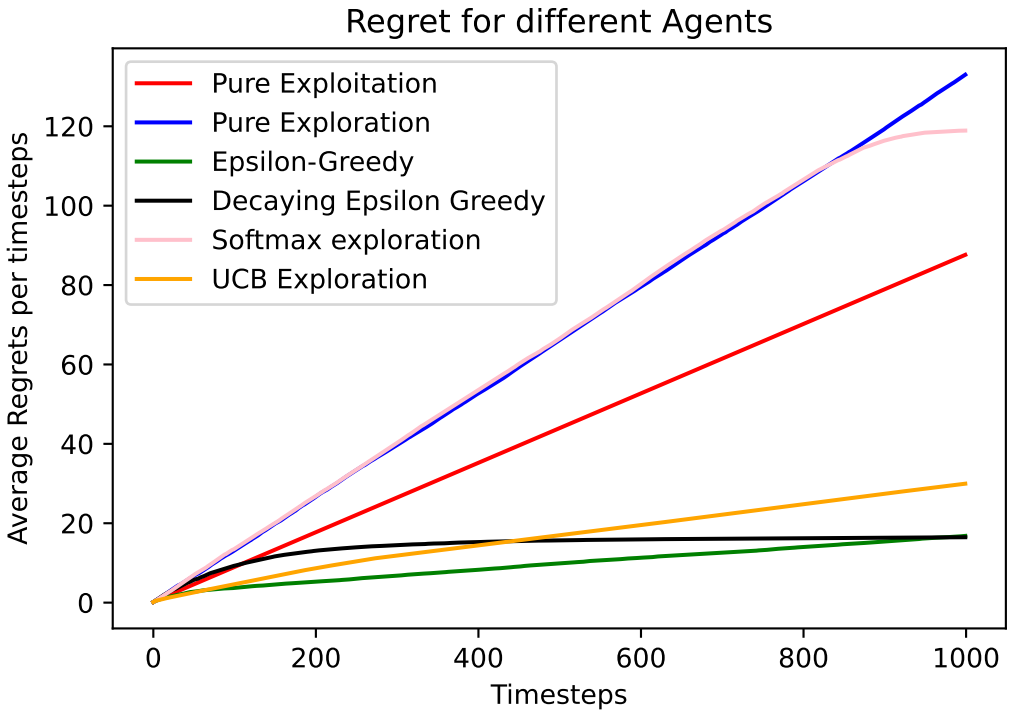

epsilonGreedy:Average rewards recieved vs timestep 10armedBandit
Average Rewards per timesteps
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
200
400
600
800
1000
Timesteps


cayingEpsilonGreedy:Average rewards recieved vs timestep 10armedBa
Average Rewards per timesteps
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
200
400
600
800
1000
Timesteps

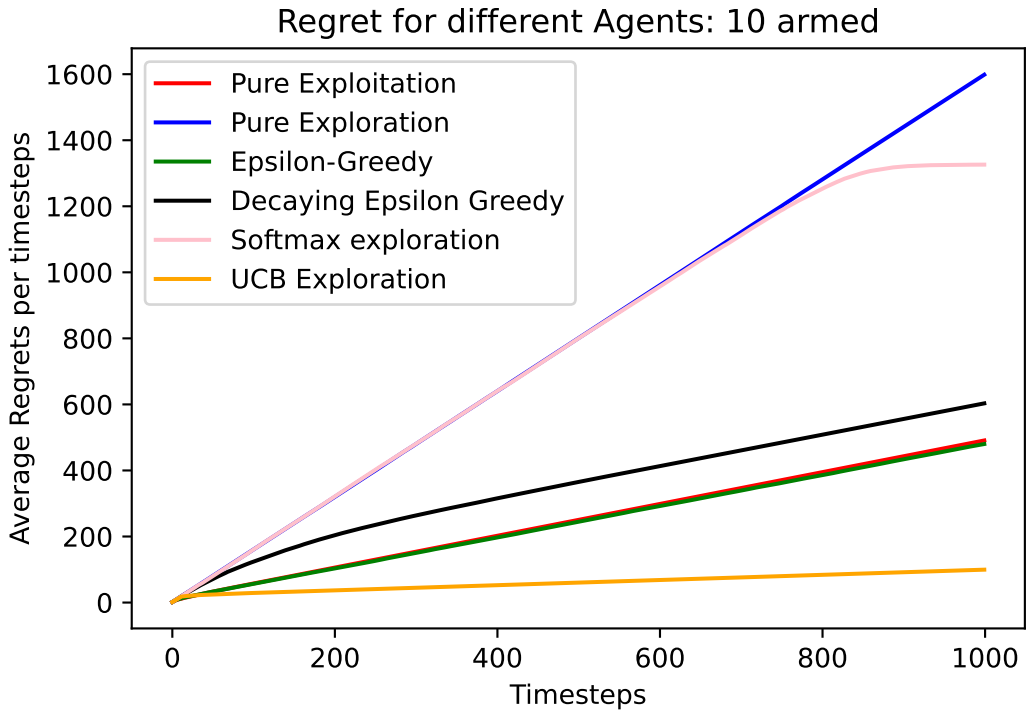

softMaxExploration:Average rewards recieved vs timestep 10armedBanc
Average Rewards per timesteps
2.0
1.5
1.0
0.5
0.0
−0.5
0
200
400
600
800
1000
Timesteps


UCBexploration:Average rewards recieved vs timestep 10armedBandit
Average Rewards per timesteps
2.0
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
Timesteps

6. Regret for Different Agents evolution with time 2 armed Bernoulli bandit environment.
   V* function can be found by taking maximum of probabilities of $\alpha$ and $\beta\dot{Q}^*$ for each action is just the probabilities of the corresponding action. Both of these can be found out using Bellman optimality equations. We see that UCB has the least regret beacuse of its high efficiency

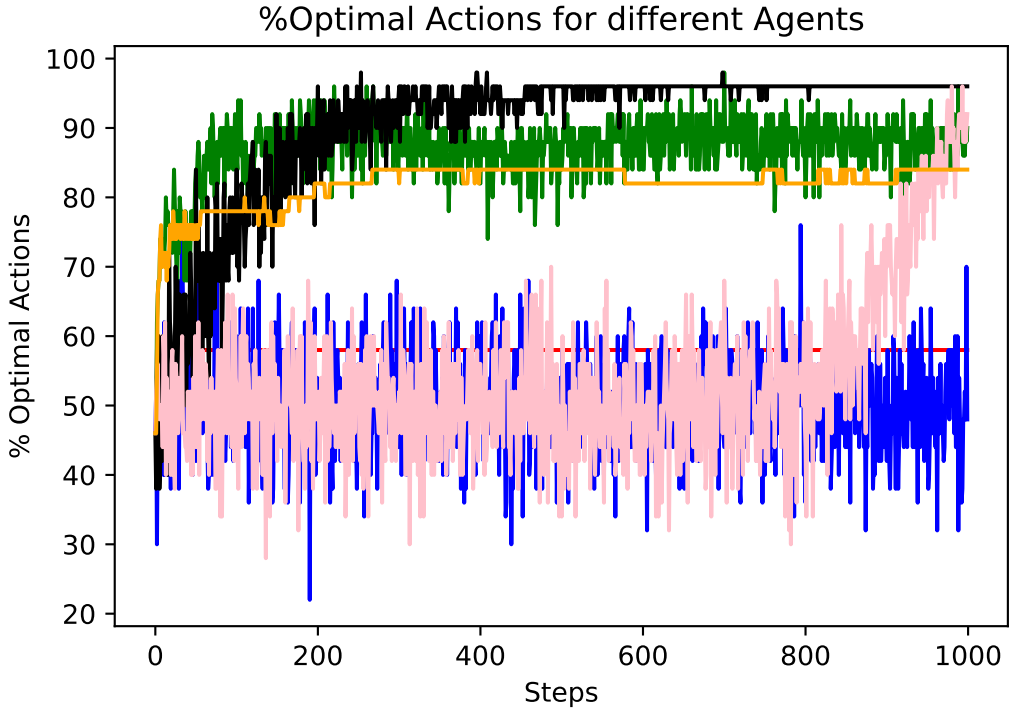


7. Regret for Different Agents evolution with time in 10 armed Gaussian Bandit environment.
   V* function can be found by taking maximum of probabilities of each Q*. Q* for each action is mean reward of each action. Both of these can be found out using Bellman optimality equations. We see that UCB has the least regret beacuse of its high efficiency



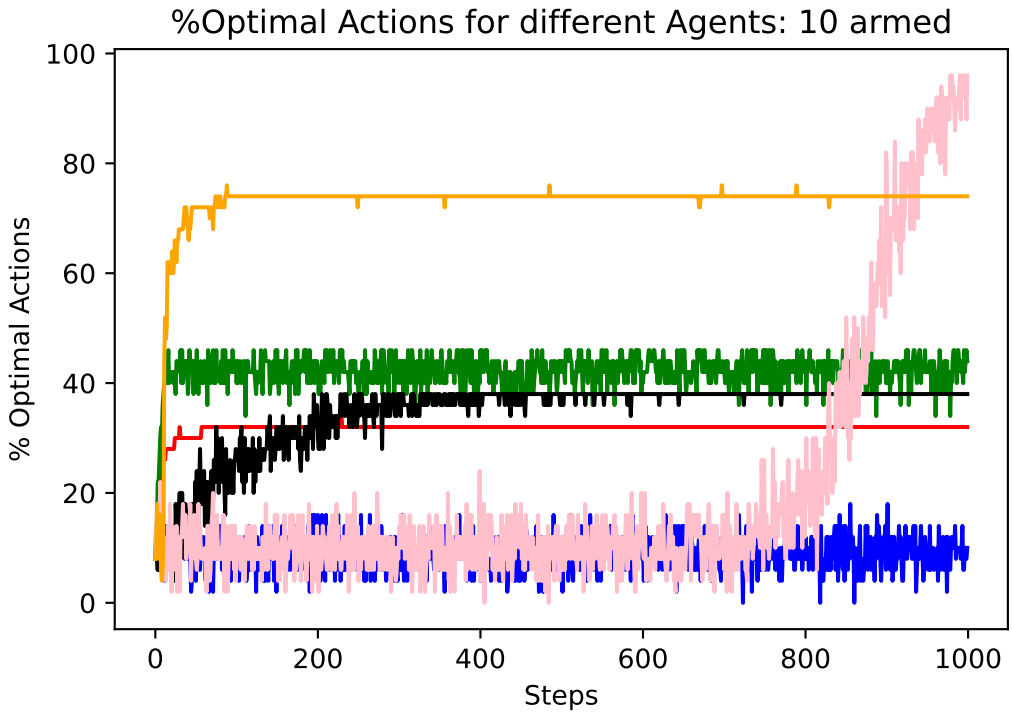
8. % of optimal Actions taken by different Agents in 2 armed Bernoulli bandit environment.

| | Color | Agent |
|---|---|---|
| | Red | Pure Exploitation |
| | Blue | Pure Exploration |
| **Legend** | Green | epsilon Greedy |
| | Black | Decaying Epsilon Greedy |
| | Pink | Soft Max Exploration |
| | Orange | UCB Exploration |



9. % of optimal Actions taken by different Agents in 10 armed Gaussian bandit environment.

| | Color | Agent |
|---|---|---|
| | Red | Pure Exploitation |
| | Blue | Pure Exploration |
| **Legend** | Green | epsilon Greedy |
| | Black | Decaying Epsilon Greedy |
| | Pink | Soft Max Exploration |
| | Orange | UCB Exploration |



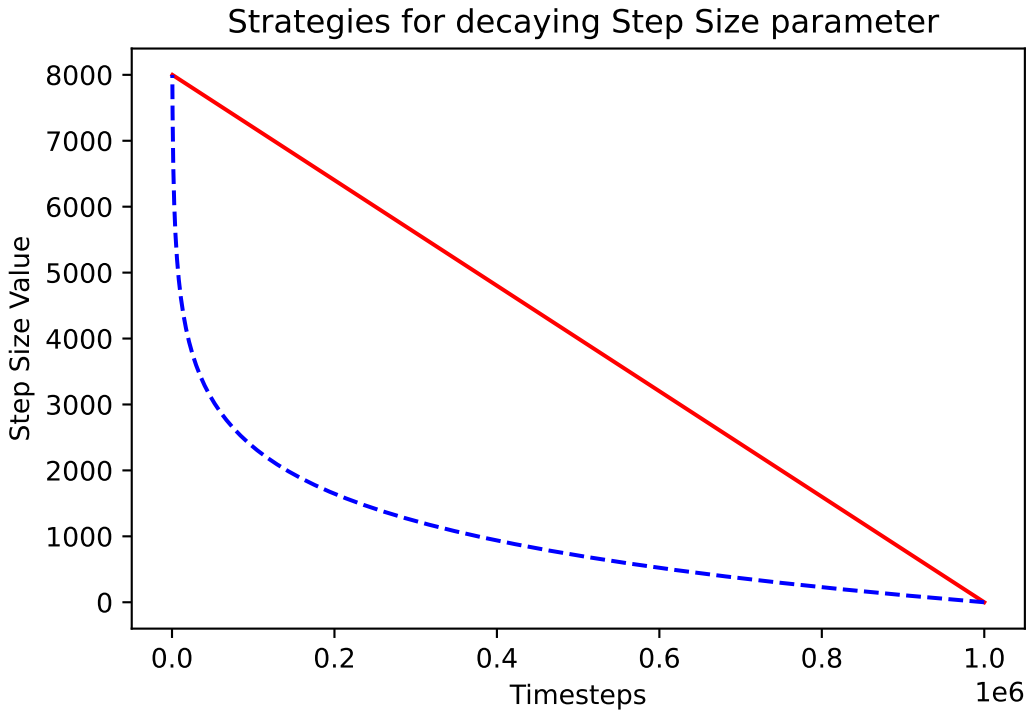
**Solution to Problem 2: MC Estimates and TD Learning**

1. 3 is globally used as a seed for all random number generation procedure. I set the goInGivenDirection probability to 1 and we see that it always goes in the direction of the action.

| Current State | Action 0 | End state | Rewards |
|---|---|---|---|
| 1 | 1 | 2 | 0 |
| 2 | 1 | 3 | 0 |
| 3 | 1 | 4 | 0 |
| 4 | 1 | 5 | 0 |
| 5 | 1 | 6 | 1 |

   The true estimate for each non terminal state can be found by using bellman equations. Also we can see that both probs are 0.5 so environment is symetric hence we can also assume directly that values of state are increasing in the order 0, 0.2, 0.4, 0.6, 0.8, 0

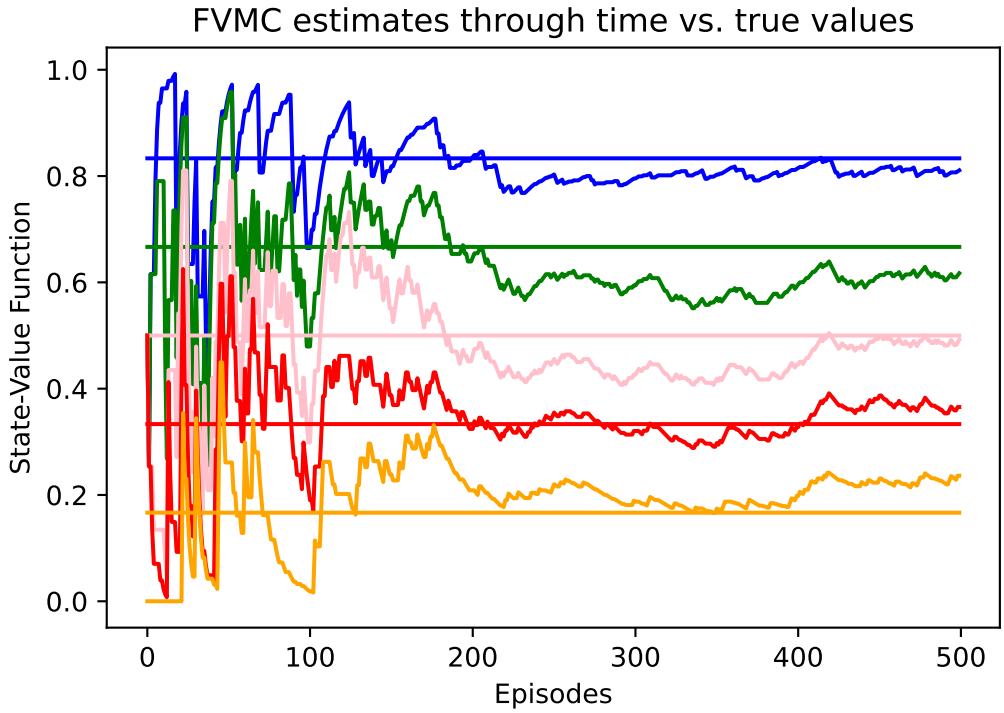
2. Linear and Exponential Decay of Step Size parameter



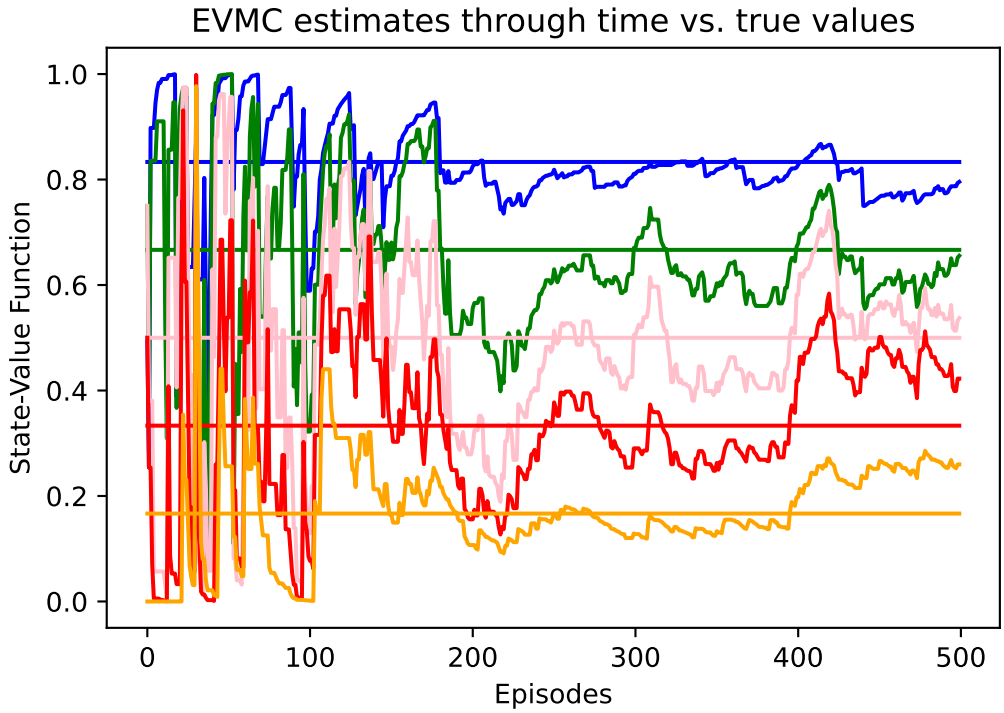

3. MC algorithm was implemented directly as mentioned in the slides. Its correctmness can be established from the below plots.

4. MC algorithm was implemented directly as mentioned in the slides. Its correctmness can be established from the below plots.

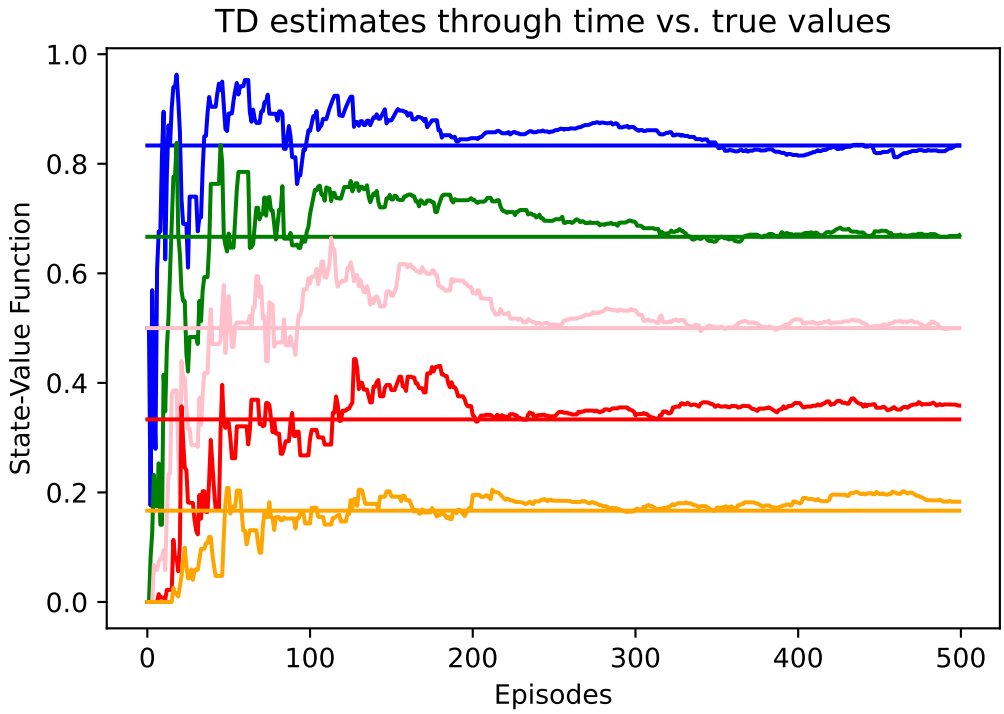
5. Monte Carlo-First Visit Monte Carlo estimate of each non-terminal state of Random Walk Environment as it progress through different episodes



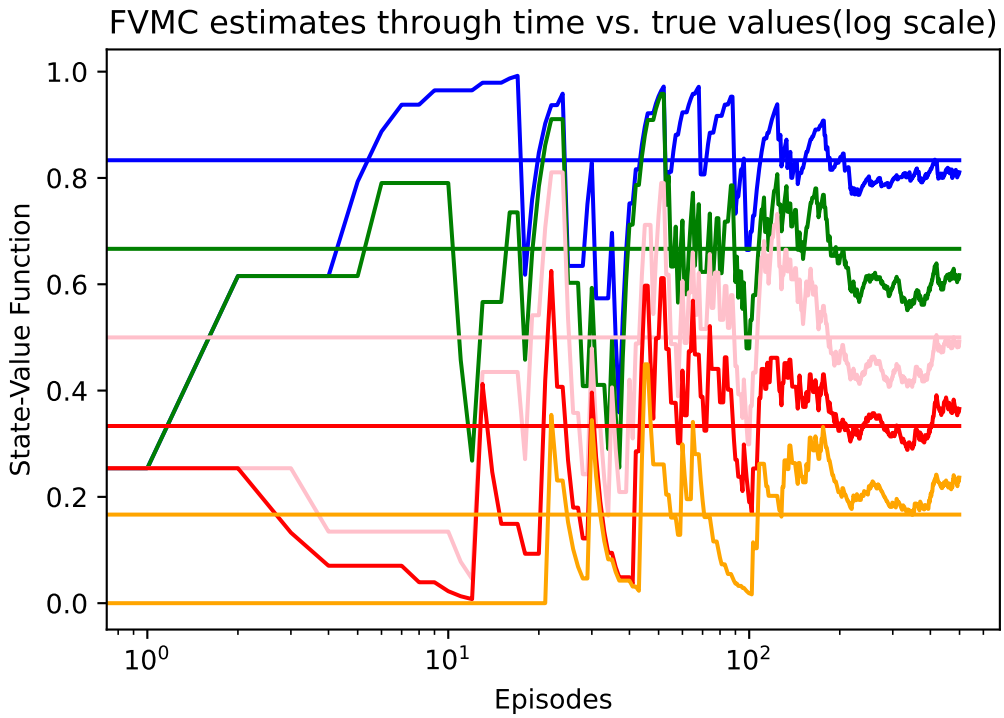

6. Monte Carlo-Every Visit Monte Carlo estimate of each non-terminal state of Random Walk Environment as it progress through different episodes



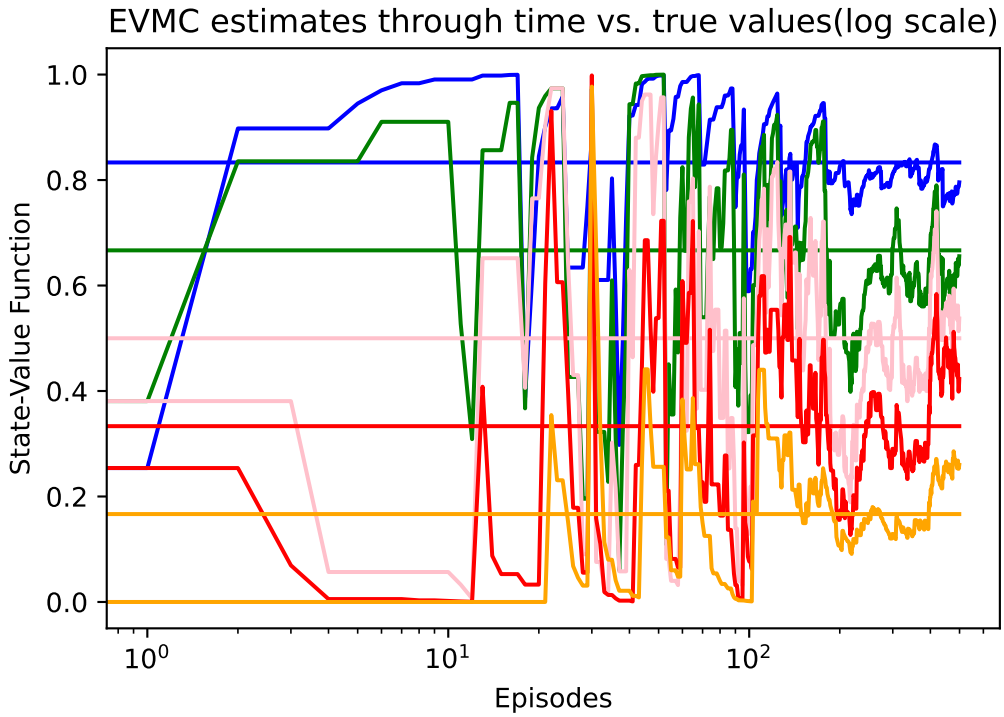

7. Temporal Difference estimate of each non-terminal state of Random Walk Environment as it progress through different episodes



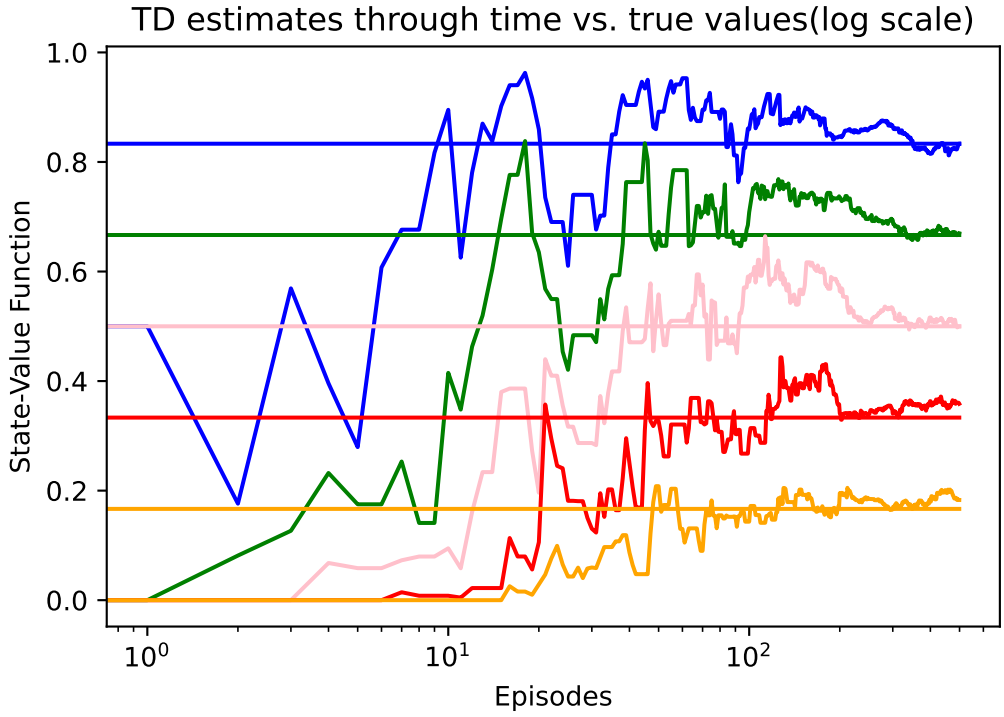

8. Monte Carlo-First Visit Monte Carlo estimate of each non-terminal state of Random Walk Environment as it progress through different episodes plotted on a log scale




9. Monte Carlo-Every Visit Monte Carlo estimate of each non-terminal state of Random Walk Environment as it progress through different episodes plotted on a log scale



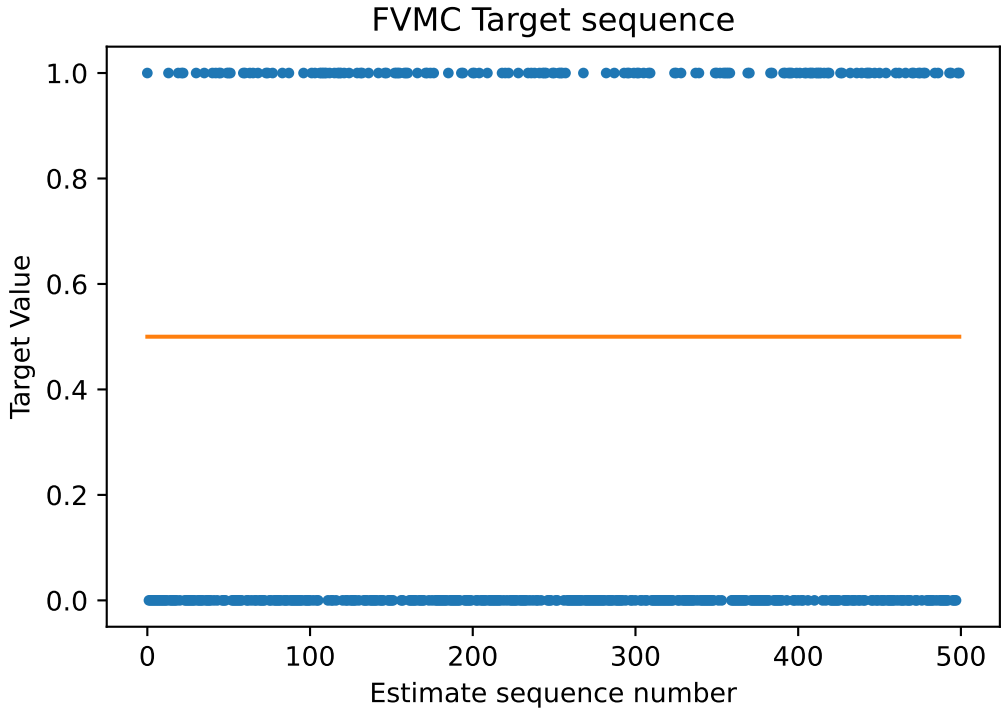

10. Temporal Difference estimate of each non-terminal state of Random Walk Environment as it progress through different episodes plotted on a log scale



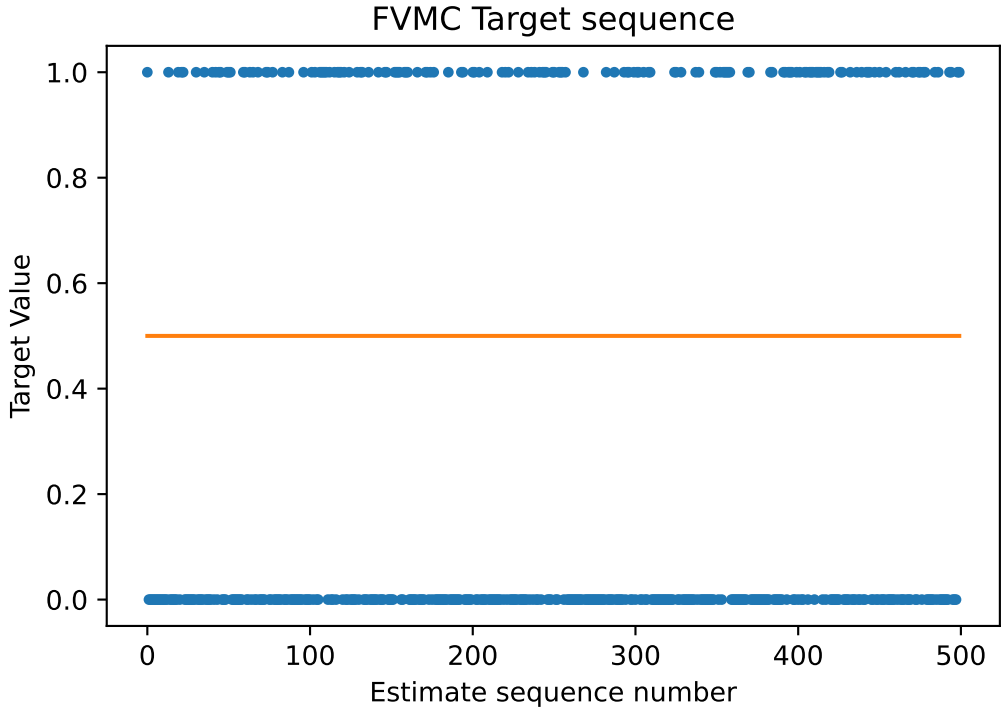

11. We see that initially FVMC and EVMC go near the true value quicker but keep oscillating for a longer time and have a higher variance. Whereas TD has a initial offset but converges to the true value faster (it has converged in the given episode length). It starts with a initial Bias but converges quickly with less variance compared to MC. In case of MC, EVMC converges faster than FVMC. We can see all of this asymptotic nature as well as Transient nature referring to noormal and log scale plotted above.

12. MC-FVMC Target Value of state = 3 of Random Walk Environment as it progress through different

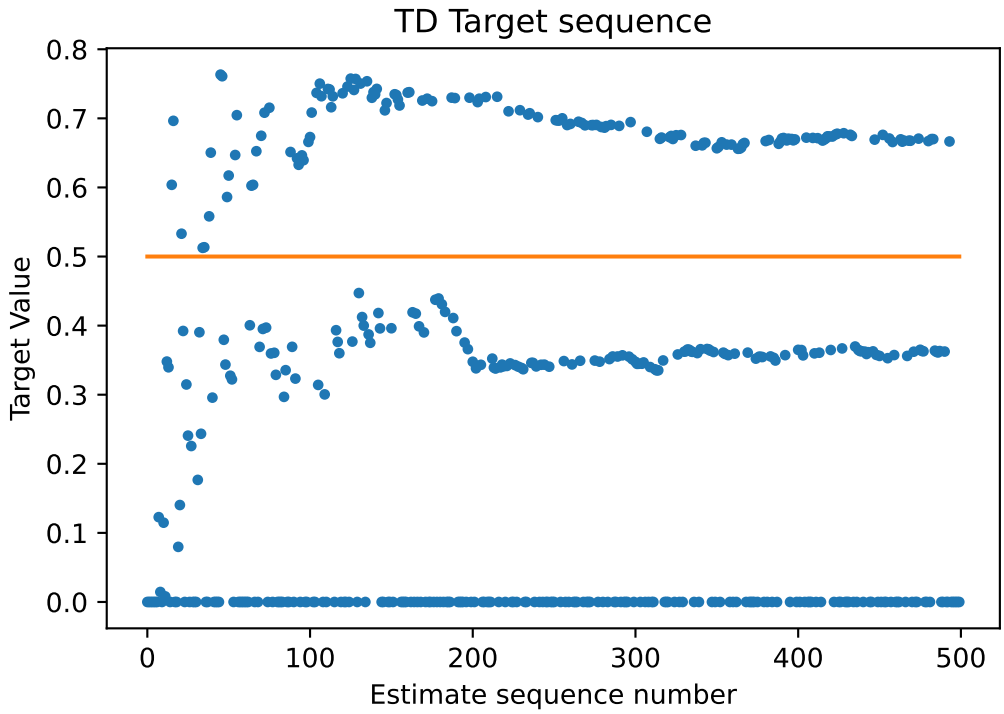
episodes




13. MC-EVMC Target Value of state = 3 of Random Walk Environment as it progress through different episodes




14. Temporal Difference Target Value of state = 3 of Random Walk Environment as it progress through different episodes




15. TD converges faster than MC. The target value that dictates incrementation of value of a function, varies from 0 to 1 which is much larger in MC compared to TD which slows its convergence rate than that of TD