Programming Assignment 1

MM16B023[1]

[a]*Indian Institute of Technology Madras*

Keyword: ϵ−-greedy, softmax, UCB, PAC Optimality, MEA

Abstract: This paper presents the solutions to first programming assignment of the Reinforcement Learning course (CS6700) at *IIT Madras*. All the notations used are as according with the textbook Reinforcement Learning by S. Sutton and G. Barto

Problem 1

First implement of the 10-arm bandit test. Try to reproduce the graphs in Figure 2.2 on page29 of the textbook. Specifically, implement the ϵ-greedy method on the 10-armed bandit testbed, for different values of ϵ

Use graphing software of your choice (for eg. Matlab or Gnunplot), to produce the graphs. Ensure that you have labelled the graphs correctly.

Solution:

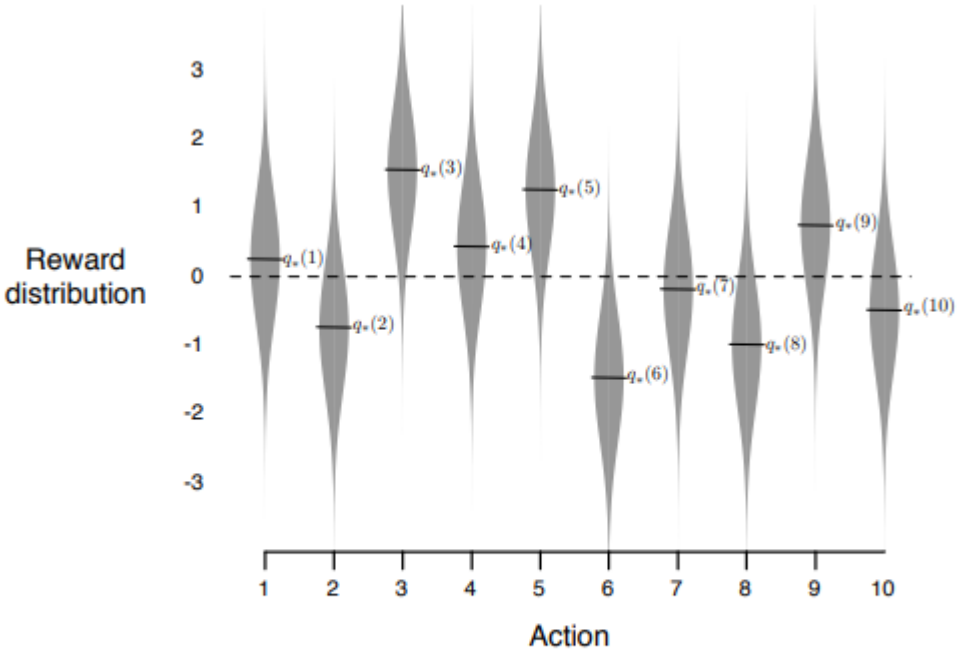

Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$ unit variance normal distribution, as suggested by these gray distributions.

$$Q_n = \frac{R_1 + R_2 + ... + R_{n-1}}{n-1} \tag{1}$$

$$\implies Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i = \frac{1}{n}(R_n + \sum_{i=1}^{n-1} R_i) = \frac{1}{n}(R_n + \sum_{i=1}^{n-1} R_i) = \frac{1}{n}(R_n + (n-1)Q_n) \tag{2}$$

$$\implies Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n] \tag{3}$$

Pseudo-Code

Initialize, for a = 1 to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
Loop through time steps:

$$A \leftarrow \begin{cases} argmax_a\ Q(a) & with\ probability\ 1-\epsilon \\ a\ random\ action & with\ probability\ \epsilon \end{cases}$$

$$R \leftarrow \text{bandit(A)}$$
$$N(A) \leftarrow \text{N(A)+1}$$
$$Q(A) \leftarrow \text{Q(A)+}\frac{[R - Q(A)]}{N(A)}$$

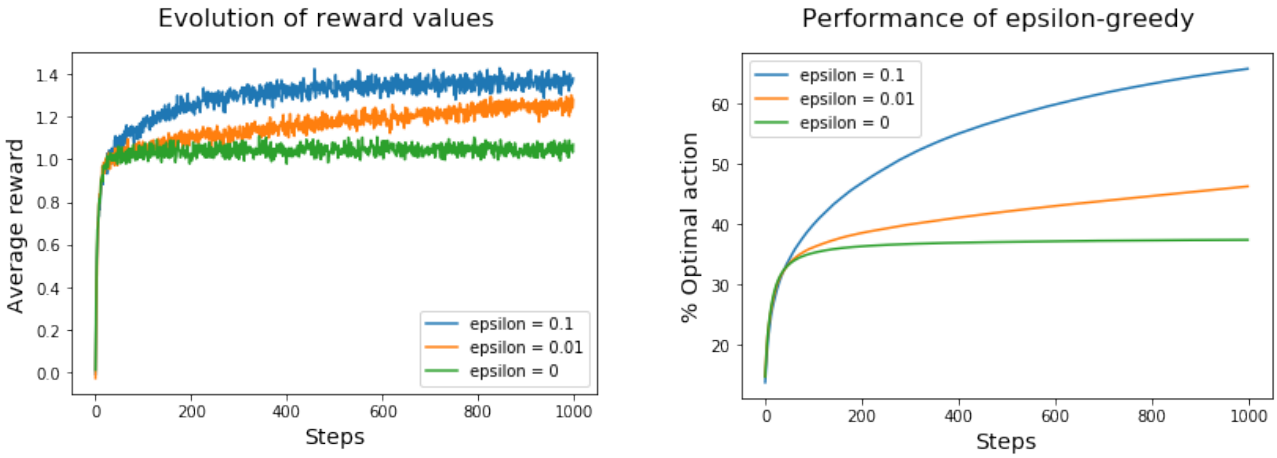

Fig 1: Performance of ϵ-greedy on the 10 armed bandit testbed averaged over 2000 instances

Inferences:

- The $\epsilon = 0$ method (greedy approach) never explored and ended up with low reward value and % Optimality.
- The $\epsilon = 0.1$ method explored more, and usually found the optimal action earlier, but it never selected that action more than 91% of the time.
- The $\epsilon = 0.01$ method improved more slowly, but would eventually perform better than the $\epsilon = 0.1$ method based on both the performance measures shown above.
- In practice, ϵ can be cooled down in steps to try to get the best of both high and low values.

Problem 2

How does the softmax action selection method using the Gibbs distribution fare on the 10-armed test? Implement the method and run it at several (at least 3) temperatures to produce graphs similar to the previous one. Note that now your are required to sample from the softmax distribution, and not take the action with the maximum probability!

Solution:

$$Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

Updates:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t)), \qquad \text{and}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \bar{R}_t)\pi_t(a), \qquad \text{for all a} \neq A_t$$

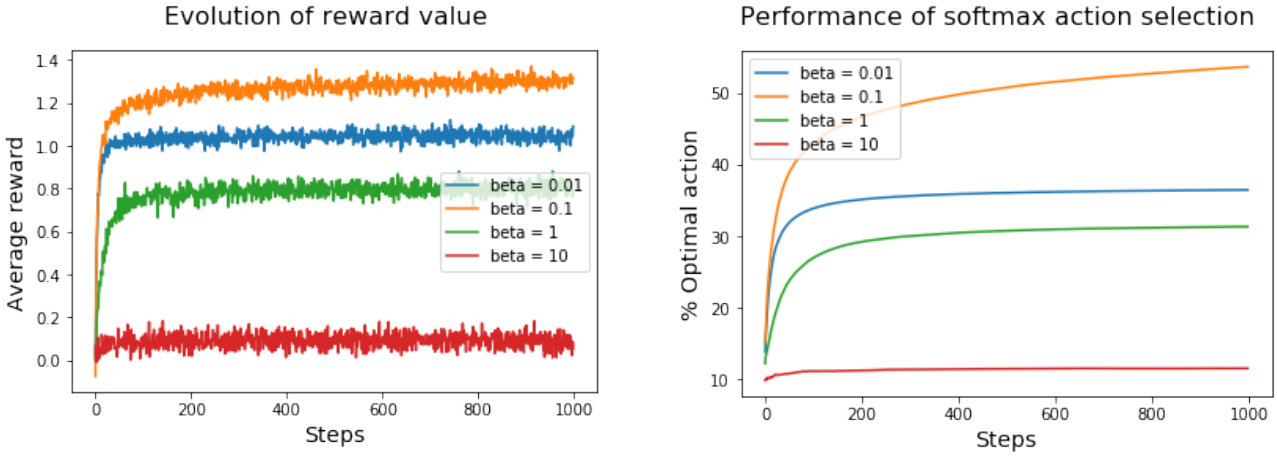

Fig 2: Performance of softmax action selection method at various temperatures

Inferences:

- The softmax action selection method does something cleverer than the ϵ-greedy algorithm by assigning probabilities based on the current Q value. This would facilitate throwing away horribly bad actions as they would have very less probability of being picked.
- As $\beta \uparrow$, the difference in the probabilities of the actions being picked decreases and hence, at a very high temperature value such as $\beta = 10$, actions are essentially being picked at random and therefore no learning occurs with very low % Optimality throughout the time steps.
- In the limit of $\beta \to 0$, the action with the maximum reward gets picked which corresponds to $\epsilon \to 0$ in the case of ϵ-greedy method.
- For this particular bandit problem, $\beta = 0.1$ seems to be the most optimal setting balancing explore-exploit.

Problem 3

Implement the UCB1 algorithm. Compare the performance with that of the epsilon greedy and softmax. What do you observe? Why do you think this is so?

Solution:

$$A_t \doteq \underset{a}{\text{argmax}} \left[Q_t(a) + c\sqrt{\frac{ln\ t}{N_t(a)}} \right]$$

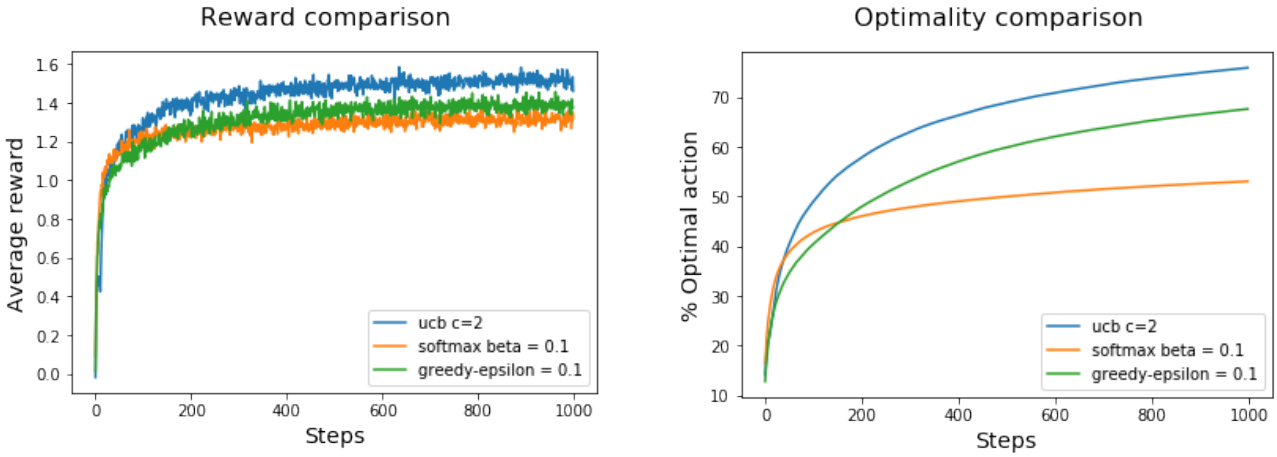

Fig 3: Performance comparison of UCB1 algorithm vs ϵ-greedy vs softmax

Inferences:

- Performance of UCB1 > ϵ-greedy > softmax for the 10 armed bandit problem averaged over 2000 instances.
- Softmax in some sense is greedier than ϵ-greedy as the arms of high expectation values have higher chances to be picked more often. This less exploration leads to reduced optimality in the longer run.
- UCB seems to be the best algorithm in this scenario to balance explore and exploit. Initially it tries all the arms (explores) but catches up and ends in a high total reward as it tries to minimize the regret.
- If by chance a better action is picked initially, its UCB value will be magnified resulting in that action being picked more often with large action values leading to some distinct spikes. Also, the spikes are more prominent at high c values as the magnification of UCB $= Q + c\sqrt{\frac{ln\ t}{N_t(a)}}$ will be larger for a larger c value.

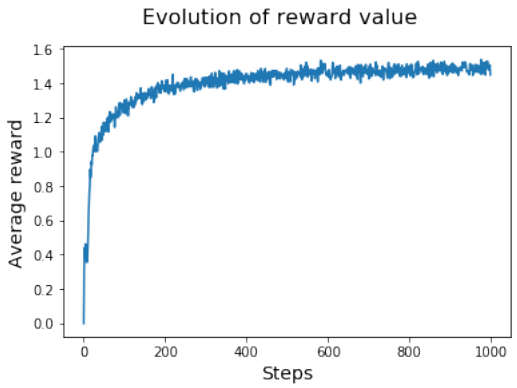

Fig 4: Spurious spikes in UCB1 Algorithm with c=2

Problem 4

Implement the Median Elimination Algorithm discussed in class.

- Compare the performance with the others. What do you observe? Why do you think this is so?
- What is the computational cost of computing median? Is it the rate determining step? Can you make it faster?

Solution:

Psuedo-Code

Input : $\epsilon > 0,\ \delta > 0$
Output : i^{th} arm
Set $S_1 = A,\ \epsilon_1 = \epsilon/4,\ \delta = \delta/2,\ l = 1$

`repeat`{
sample every arm a $\epsilon\ S_l$ for l_k times
where, $l_k = \frac{1}{(\epsilon_l/2)^2} \log \frac{3}{\delta_l}$
Let $Q_l(a)$ denote the i^{th} ivalue. Find m_l = median $\{Q_l(a)\}$
$S_{l+1} = S_l \setminus \{a : Q_l(a) < m_l\}$
$\epsilon = \frac{3}{4}\epsilon_l;\ \ \delta_{l+1} = \delta_/2,\ \ l = l + 1$
until $|S_l| = 1$ }

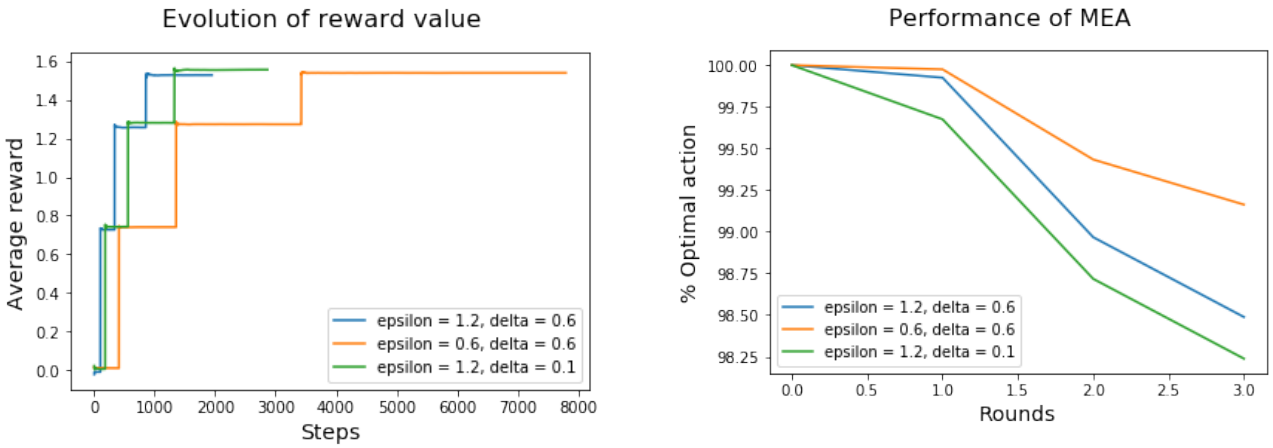

Fig 5: Performance of Median Elimination Algorithm for various ϵ, δ values

Inferences

- For each l_k set of iterations, no learning occurs (flat portion of the reward curve). This due the fact that the rewards obtained (means) are essentially averages of random samples drawn from the set S_l.
- Learning (vertical jump) occurs when half of the bad actions ($q_*(a) < m_l$) are thrown away. The Median Elimination Algorithm in this process guarantees the (ϵ, δ) PAC bounds.
- $\epsilon = 0.6,\ \delta = 0.6$ seems to be the most optimal setting in this case. In general, lower the values of ϵ, δ, the more optimal the algorithm gets due to stricter PAC bounds but the downside is that computational cost increases significantly.
- Optimality is defined quite differently in the case of MEA. The algorithm is optimal throughout if it has the best action in each of the set S_l. Due to this, % Optimal action of MEA can't be compared with other algorithms. However, evolution of reward can be compared to measure relative performance.

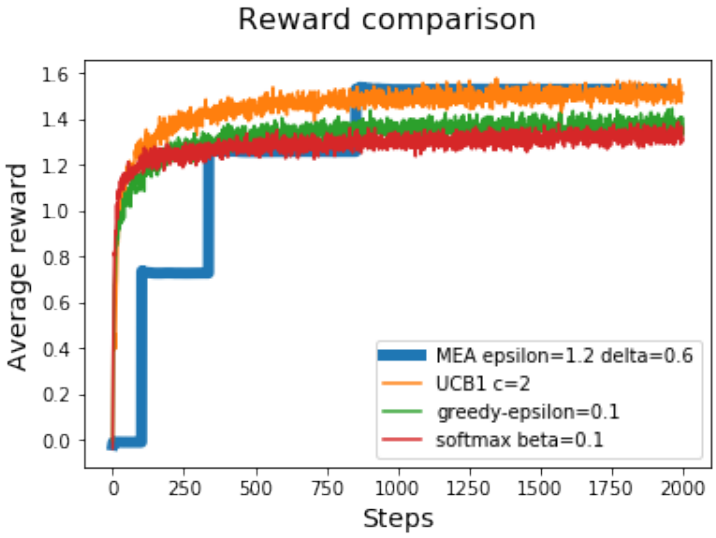

Fig 6: Performance comparison of MEA vs UCB1 vs ϵ-greedy vs sotmax on the 10 armed testbed

- Performance of MEA > UCB1 > ϵ-greedy > softmax. The order of the last 3 algorithms was explained in problem 3. With regards to MEA, the reason why it is able to outperform all the other 3 algorithms is that it is able to eliminate a large chunck of actions at one go as well as maintain the PAC bounds in a discretized as opposed to a continuous learning process.

Computational cost

The complexity of computing median after each set of l_k iterations is $O(n\ log\ n)$ where n is the number of bandit arms in the set S_l

Total median complexity $T_M = \sum_{k=0}^{M} \frac{N}{2^k}\ log\left(\frac{N}{2^k}\right)$ where M = total no. of rounds = $log_2\ N$

$$\implies T_M = \sum_{k=0}^{M} \frac{N}{2^k}\Big(log(N) - log(2^k)\Big) = Nlog(N)\sum_{k=0}^{M}\frac{1}{2^k} - \sum_{k=1}^{M}\frac{k}{2^k} = Nlog(N)\left(\frac{1-(\frac{1}{2})^M}{1-(\frac{1}{2})}\right) - \sum_{k=1}^{M}\frac{k}{2^k}$$

Sum of the Arithmetic Geometric progression (AGP) $\sum_{k=1}^{M}\frac{k}{2^k} = -\frac{log(N)}{N} + 2\Big(1-\frac{1}{N}\Big)$

$$\implies \quad T_M = 2Nlog(N)\left(1-(\tfrac{1}{2})^M\right) + \frac{log(N)}{N} - 2\left(1-\tfrac{1}{N}\right) \implies T_M = 2Nlog(N)\left(1-\tfrac{1}{N}\right) - \left(\frac{2N-1}{N}\right) + \frac{log(N)}{N}$$

Total median complexity $T_M = \left(\frac{log(N)\Big(2N^2-1\Big) - \Big(2N-1\Big)}{N}\right) \sim \boxed{O(Nlog(N))}$

Sample Complexity $T_S \sim O(n_1L_1 + n_2L_2 + ... + n_kL_k)$

$T_S = \sum_{k=0}^{M}\frac{n}{2^k}\frac{4}{{\epsilon_l}^2}log\left(\frac{3}{\delta_l}\right) \sim \frac{N}{\epsilon^2}log\left(\frac{1}{\delta}\right)$ where M = total no. of rounds = $log_2\ N$

Median Complexity T_M is independent of (ϵ, δ) PAC bounds. So, for a given bandit problem (fixed N), the time complexity cannot be reduced further.

If ϵ, δ are not set to be too big, sample complexity takes a much higher value than median complexity. Thus, sampling of the arms is usually the rate determining step in case of the median elimination algorithm and it can be made faster by increasing the values of ϵ, δ (loosening the bounds).

Problem 5

What happens as the number of arms grows? Run the above algorithms on a 1000 arm bandit setup and compare their performance.

Solution:

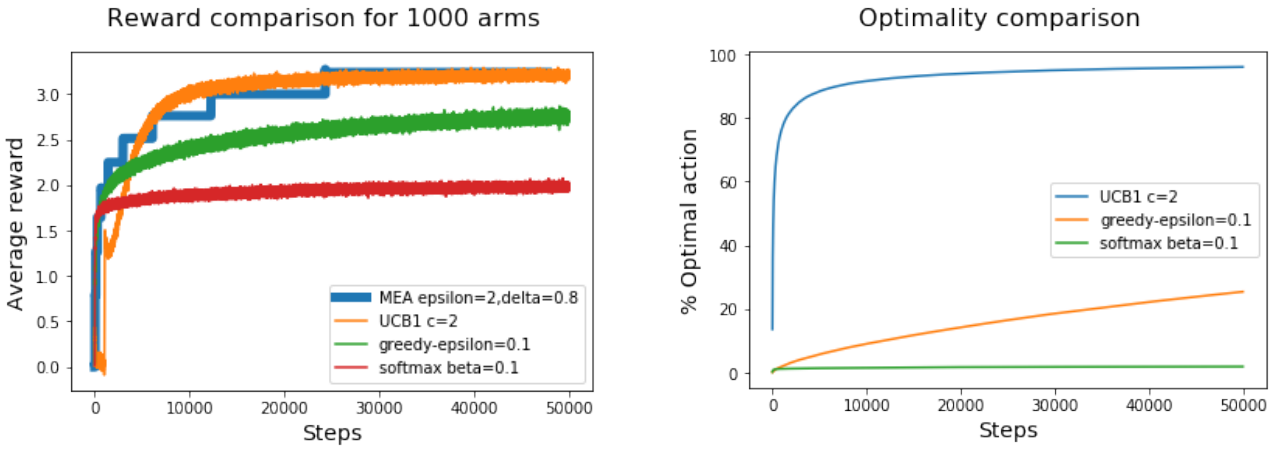

Fig 7: Performance comparison of MEA vs UCB1 vs ϵ-greedy vs softmax for the 1000 armed testbed

Inferences

- Performance of MEA $\simeq$ UCB1 $>$ ϵ-greedy $>$ softmax for the 1000 arm testbed.

- Performance of MEA and UCB1 increases as the number of arms grows and the increase is more significant is the case of the UCB1 algorithm.

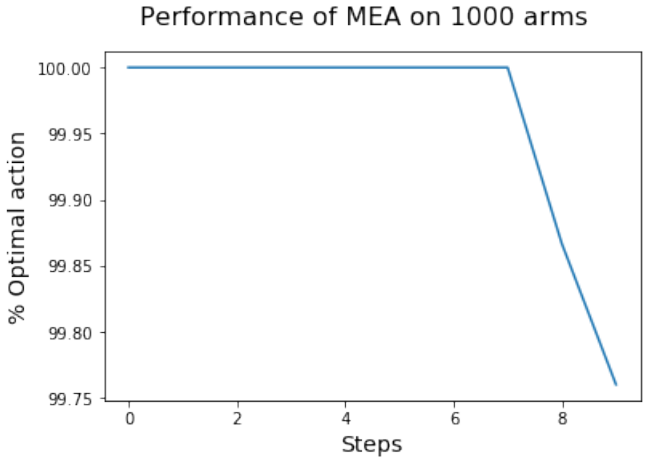

Fig 8: % Optimal action vs steps (rounds) for MEA(2, 0.8) in the case of 1000 armed testbed

- ϵ-greedy is not able to sample all the 1000 arms enough number of times and this reflects in its reduced performance which is not yet converged. One way to increase its % Optimal action is to sweep through more number of time steps and let the optimality curve saturate to a better value.
- Performance of softmax is really poor and increasing the number of time steps is not really going to help in this regard. One way to improve performance might be to drastically $\downarrow \beta$, the temperature parameter.
