CS6700 : Reinforcement Learning

Programming Assignment #1 Nirav Bhavsar (CS17S016)

Solution to Q1 (ϵ-greedy) :

- To implementation ϵ-greedy on 10-armed Testbed, we run it for 1000 steps, where each step value being the average of the performance on 2000 different bandit problems. These 2000 different bandit problems are generated from standard normal distribution, Where the true mean $q^\star(a)$ of each of the ten arms are selected according to a normal distribution with mean zero and unit variance, and then the actual rewards are selected according to a mean $q^\star(a)$ unit variance normal distribution.

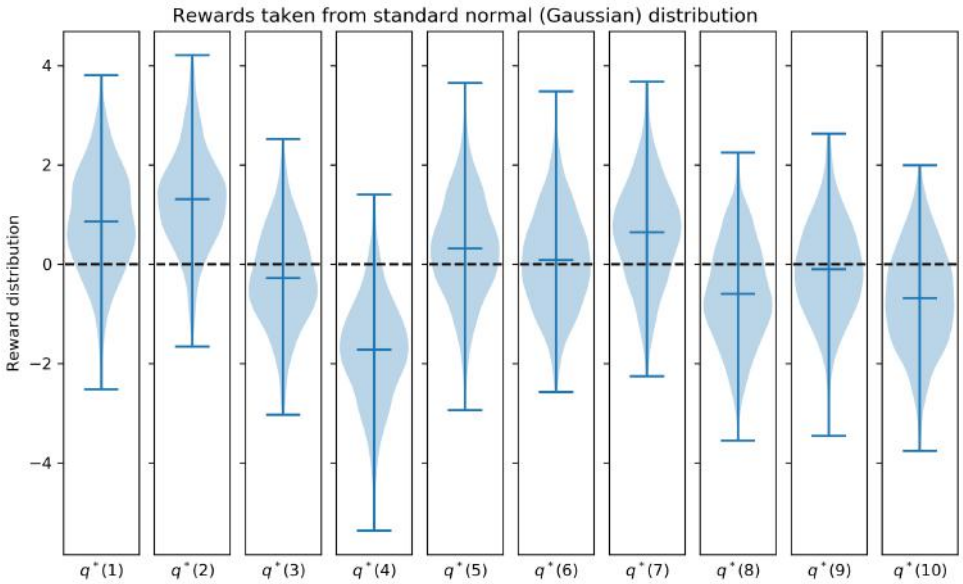
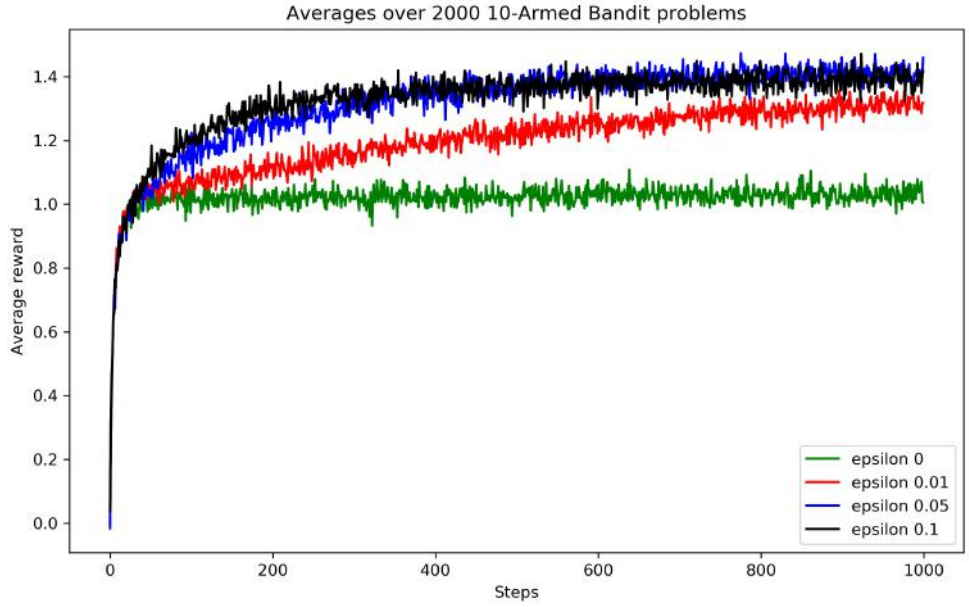


Figure 1: An example bandit problem from the 10-armed testbed, true mean $q^\star(a)$ for each arms taken from standard normal distribution.

- Now we compare average performance of ϵ-greedy algorithm for different values of ϵ ($\epsilon = 0$, 0.01, 0.05, 0.1), on the 10-armed testbed. Below plots are averages over 2000 runs with different bandit problems. Sample-average technique to estimate arm means. The first graph shows the increase in average reward with experience. The greedy method ($\epsilon = 0$) improved slightly faster than the other methods at the very beginning, but then leveled off at a lower level. It managed to achieved a reward-per-step of only about 1, compared with the best of about 1.50 on this testbed. The greedy method will perform even worse in the long run because it will often get stuck pulling suboptimal actions. While for other ϵ like 0.1& 0.05 performed almost equally better as they spent more time in exploring for the optimal arm, and we can see that $\epsilon = 0.05$ will perform better than all other in long run for this testbed.

- The second graph shows that the greedy method ($\epsilon = 0$) found the optimal action in only approximately one-third of the tasks. In the other two-thirds, its initial samples of the optimal arm were disappointing, so it never returned to it. The ϵ-greedy methods performed better

because they continued to explore and to improve their chances of recognizing the optimal arm. The $\epsilon = 0.1$ method explored more, and found the optimal action earlier, hence performed better in initial stages. The $\epsilon = 0.05$ method improved more slowly, but eventually it will perform better than the $\epsilon = 0.1$ method on both performance measures.



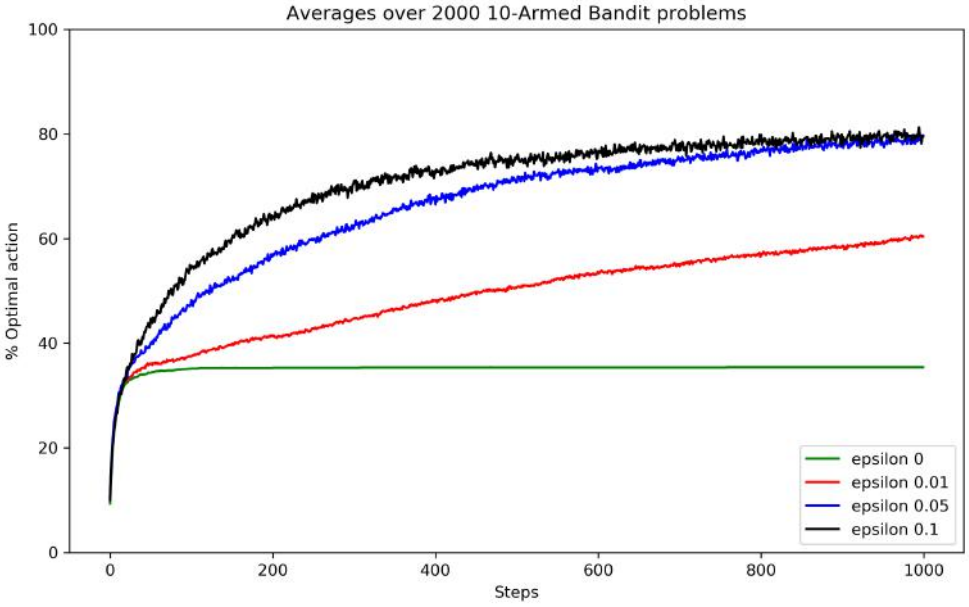


Figure 2: Average performance of ϵ-greedy algorithm over 2000 different 10-armed bandit problems.

Solution to Q2 (Softmax) :

- Now we implement softmax action selection algorithm using the Gibbs distribution on the similar 10-armed testbed, run it for 1000 steps & observe it's average performance over 2000 different bandit problem.
- We compare softmax action selection algorithm for different values of cooling parameter: temperature (0.01, 0.1, 0.3, 1). Given initial empirical means $\hat{\mu_1}(0), ..., \hat{\mu_k}(0)$

$p_i(t+1) = \frac{e^{\hat{\mu_i}(t)/t}}{\sum_{j=1}^{k} e^{\hat{\mu_i}(t)/t}}$, $i = 1, ..., k$

where t is a temperature parameter.

- Here softmax algorithm selects an arm using a Gibbs or Boltzmann distribution obtained from above formula for each arm at each time step.

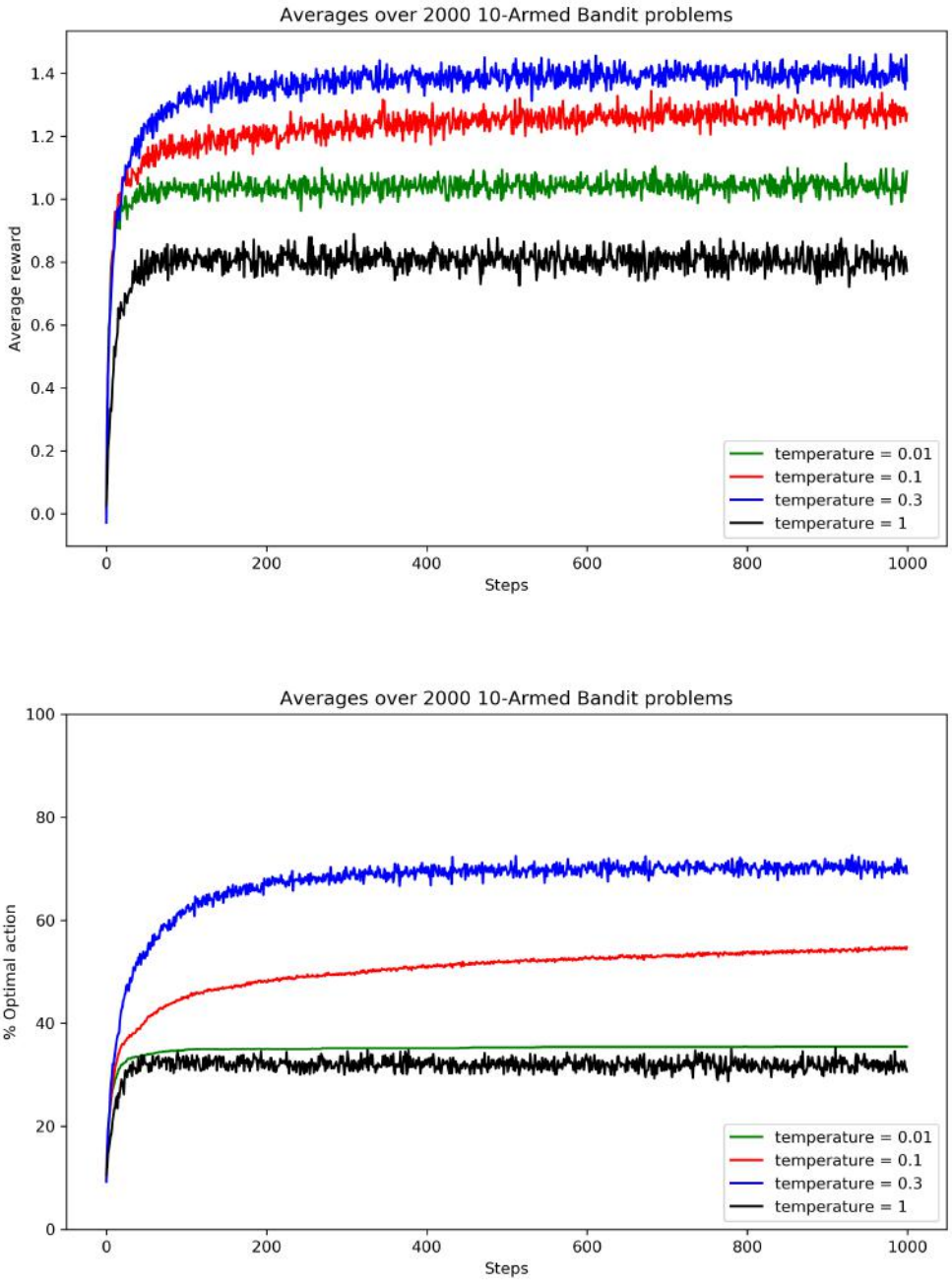


Figure 3: Average performance of softmax algorithm over 2000 different 10-armed bandit problems.

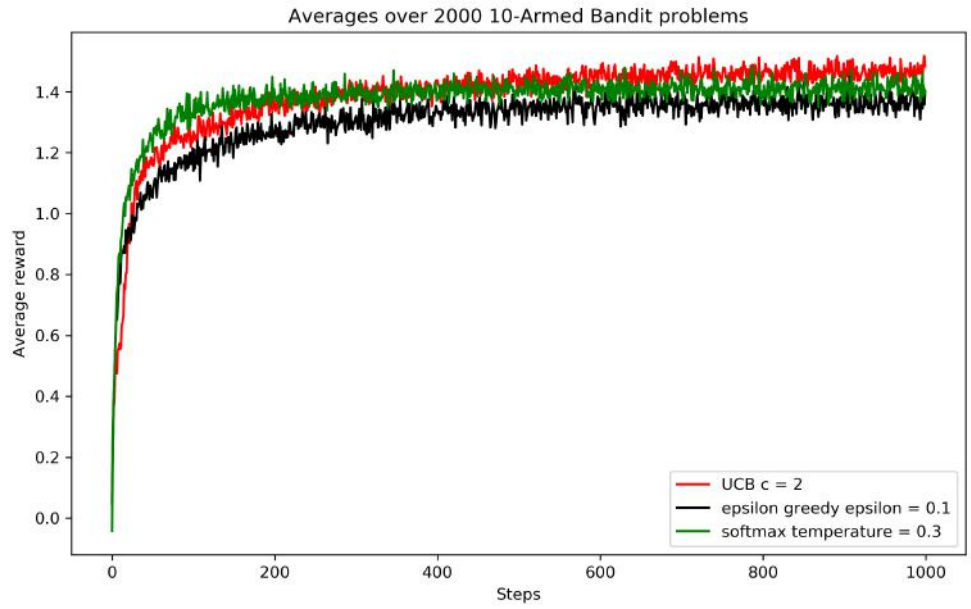
- In softmax algorithm, temperature parameter controlls the randomness of the choice. When t is close to 0 like 0.01 in our case, softmax algorithm will act like pure greedy; which ever arm mean estimate is highest will have very high probability and all other arms will have very small probability, which we can observe that plot for temperatue for $t = 0.01$ is similar to greedy $\epsilon = 0$.
- And as t tends to infinity, the algorithms picks arms uniformly at random; what ever mean estimate for each arm you have, all arms value will become close to zero. Since our rewards are

taken from standard normal distribution, so when $t = 1$, it will pick arms uniformly at random, hence will perform poorly as it will not be pulling optimal arm more times.

- Thus, at $t = 0$ it performs mostly exploitation & $t = 1$ it performs mostly exploration. Hence to perform better we need to take value of temperature t in middle or better less than $t = 0.5$ so that it will do more exploitation but will also do exploration enough number of times so that it has better chances of recognizing the optimal arm. Which we can observe from above graphs that at $t = 0.3$ it performs similar to $\epsilon = 0.1$. Hence softmax algorithm will perform similar to ϵ-greedy if we set the temperature parameter t correctly.
- We will compare Softmax algorithm at $t = 0.3$ with ϵ-greedy at $\epsilon = 0.1$ in next solution along with UCB1.

Solution to Q3 (UCB1) :

- Now we implement UCB1 algorithm on the similar 10-armed testbed, run it for 1000 steps & compare it's average performance over 2000 different bandit problem with ϵ-greedy & softmax.
- UCB1 algorithm is a simpler, more elegant implementation of the idea of optimism in the face of uncertainty. It uses the fact that there is always uncertainty about the accuracy of the action-value estimates, hence exploration is needed. Thus it would be to better to select non optimal arms according to their potential for actually being optimal, by taking into account both how close their mean estimates are to being maximal and the uncertainties in those estimates.
- UCB1 selects arm at time t : $A_t = \underset{a}{argmax}[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}]$
  where $N_t(a)$ denotes the number of times that action a has been selected prior to time t, $Q_t(a)$ is the mean estimate of arm a, & the number $c > 0$ controls the degree of exploration.
- Here, the square root term, determines the variance or uncertainty in the estimate of a's value. Since $N_t(a)$ appears in denominator the lower its value, the higher will be the uncertainty, which means if we have pulled arm a very less time, then it's current mean estimate is poor and there is very high uncertainity, thus we pull arm a which increases $N_t(a)$ and reduces it's uncertainity. While on the other hand, each time an arm other than a is selected, t increases but $N_t(a)$ does not; thus the uncertainty estimate increases for arm a.
- Thus UCB1 guarantees that all arms will eventually be selected, and arms with lower value estimates, or for which $N_t(a)$ is more, will be selected less frequenctly over time. And the arm with higher value estimate (optimal arm) will be pulled more frequently with time.
- From below graphs (on next page) we can observe that UCB1 performs better than ϵ-greedy & Softmax action selection, except in the intial steps, where it selects arms randomly which are not pulled enough time ($N_t(a)$ is low) and it's variance is high.
- Unlike ϵ-greedy & Softmax action selection algorithm, where we have to select right parameter like ϵ and temperature t, so that algorithm performs better. In UCB1 we do not have to select any parameter. In below graphs we have selected optimal value of parameter ϵ and t which performs best in this testbed from previous solution, but UCB1 with no parameter to select, performs better than both ϵ-greedy & Softmax algorithm. In case if we have selected other values for parameter ϵ and t it will perform even worse.
- UCB1 uses $N_t(a)$, $Q_t(a)$ & t to calculate the upper confidence bound of each arm, which will keep on changing with those values and thus it will do both exploration & exploitation from time to time without needing any external parameter, by adjusting the upper confidence bound for each arms. Thus in general UCB1 will perform better than ϵ-greedy & Softmax algorithm.



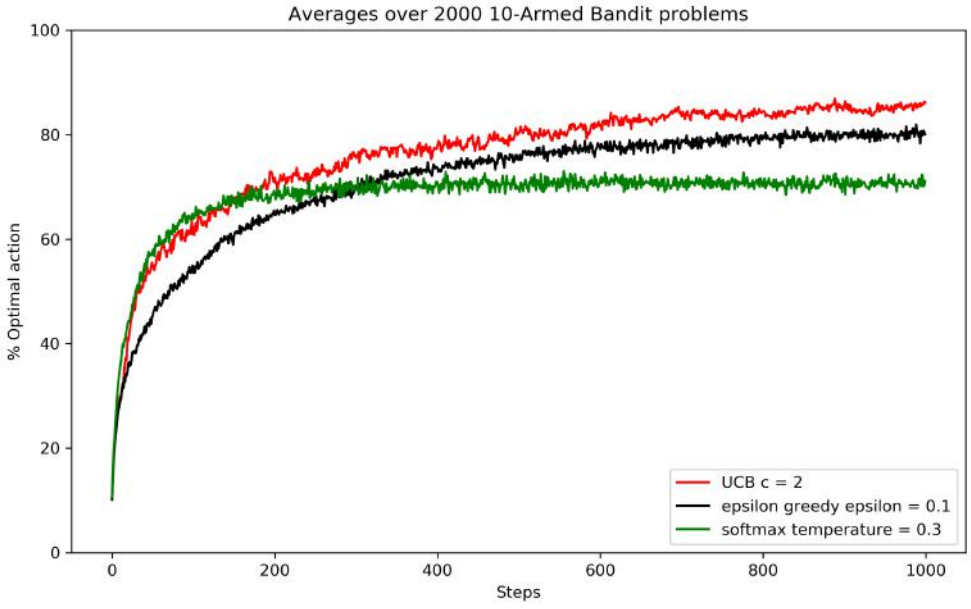


Figure 4: Comparison of average performance of UCB1, Softmax & ϵ-greedy algorithm over 2000 different 10-armed bandit problems.

Solution to Q4 (MEA) :

- The Median Elimination algorithm is an (ϵ, δ)-PAC algorithm. In every round it eliminates half of the bad arms, thus in $\log k$ steps it will return an arm within ϵ range of optimal arm with at least $(1 - \delta)$ probability.
- The sample complexity of the Median Elimination is $\boldsymbol{O(k \log(3/\delta)/\epsilon^2)}$, where k is the number of arms. Which is verified with the empirical results shown in below table for constant $c = 384$.

epsilon ϵ	delta δ	Total samples	% Optimal arm retained	#times optimal arm not returned	#arm returned outside epsilon range
0.9	0.90	4636	96.60	68	0
0.9	0.10	8082	97.65	47	0
0.5	0.90	15067	97.95	41	0
0.5	0.50	18057	99.25	15	0
0.5	0.10	26216	99.25	15	0
0.1	0.90	376990	99.95	1	0
0.1	0.10	655473	99.75	5	0
0.2	0.98	91540	99.35	13	0
0.2	0.02	214863	99.50	10	0

Table 1: Median Elimination algorithm average performance for different parameters averaged over 2000 different 10 armed-bandit problems for different values of ϵ & δ

- From the empirical results and graphs (on next page) we can observe that when ϵ is large i.e. close to 1 (remember our rewards come from gaussian distribution with means close to 0 & variance 1), then the number of samples required to achieve (ϵ, δ)-PAC gaurentee would be less as we don't need to be more confident while eliminating arm since ϵ is large. In other case, when ϵ is small i.e. close to 0, then the sample required for (ϵ, δ)-PAC would be large, because we need to be more confident that the arms we are eliminating will not be within ϵ range of optimal arm.

- Also, we can observe that % optimal arm retained is relatively little less when ϵ is large, because since we are taking less samples probability of eliminating optimal arm is more since ϵ is large. For example for $\epsilon = 0.9$ & $\delta = 0.9$, total samples taken is 4636 & number of times optimal arm is not returned is 68. Similarly when ϵ is less, we sample it more times and hence we would be retaining optimal arm with high probability. For example r $\epsilon = 0.1$ & $\delta = 0.9$, total samples taken is 376990 & number of times optimal arm is not returned is 1.

- For same value of ϵ and different value of δ, we can observe that when δ is large, means $(1-\delta)$ is small, hence we would need less samples for (ϵ, δ)-PAC and % optimal arm retained would also be less, number of times optimal arm is not returned would be more. Similarly when for same ϵ, δ is small, means $(1-\delta)$ is large, hence need more samples, better % optimal arm retained & less times eliminating optimal arm.

- Thus, with these empirical observation, we can conclude that Median elimination algorithm is indeed (ϵ, δ)-PAC algorithm with sample complexity of $O(k \log(3/\delta)/\epsilon^2)$.

- *Is finding median the rate-determining-step ?*
  Within Median elimination algorithm, finding out the median of estimated mean values is not the rate determining step, particularly in our case as the number of arms is small (i.e 10) and also the number of arms gets halved every round. But in case the number of arms are more like 1000 or 10000, then by using standard method of sorting and then finding middle element can be the rate determining step as that would take $O(k \log k)$ time, in that case we can use better algorithms which run in linear time $O(k)$.

- For example: Deterministic linear time median-of-medians algorithm or randomized Monte Carlo algorithm as described in [4]. It is a linear time algorithm that fails to produce a solution with probability at most $k^{-1/4}$. It is significantly simpler than the deterministic $O(k)$ algorithm and yields a smaller constant factor in the linear running time.

- *Compare MEA with other algorithm ?*
  To compare Median Elimination Algorithm with others, first of all MEA is an action elimination algorithm which gaurentees (ϵ, δ)-PAC, while others like ϵ-greedy, Softmax & UCB1 are

action selection algorithm, they never eliminated any arm. Thus in each time step action selection algorithm determines which arm to pull, while in each round action elimination algorithm determines which arm to eliminate.

- If the number of time steps is very large, then we can first use MEA to find out optimal arm with (ϵ, δ)-PAC and pull that arm for remaining time steps, but if time steps are less than the sample complexity of MEA for (ϵ, δ)-PAC, then we would do better using action selection algorithm like UCB1.
- For a fixed time steps, if the sample complexity of MEA is not less than 30-35% of total time steps, then it is better to choose action elimination UCB1 algorithm which is guaranteed to give lower regret without hassel of choosing any parameter.
- We could also use combination of action elimination & action selection algorithm, when the number of arms is very large like 1000 or 10000, in such case we could use MEA for first few rounds to eliminate the large number of bad arms as it will remove half of the arms in every round, then use action elimination UCB1 algorithm on the remaining arms for the remaining time steps.

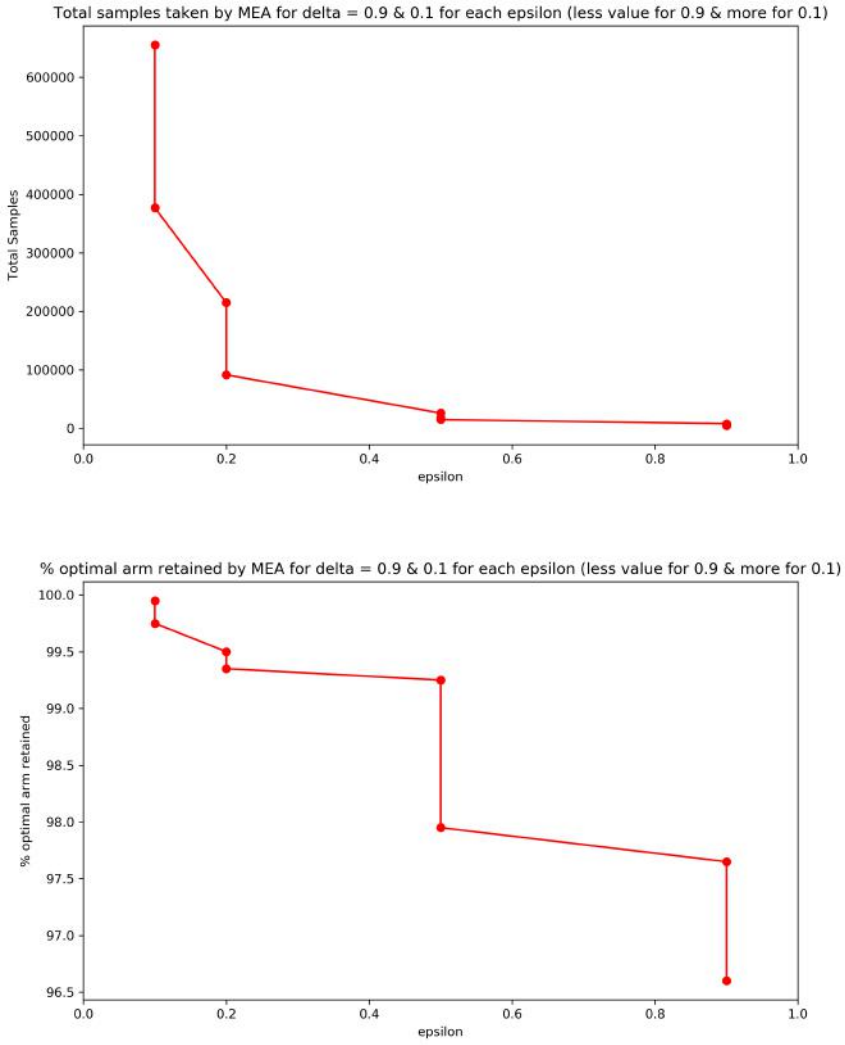


Figure 5: Comparing Average Total samples taken & % Optimal arm retained by MEA for $\delta = 0.9$ & 0.1 for each epsilon (less value for 0.9 & more for 0.1) for different values of ϵ over 2000 different 10-armed bandit problems.

Solution to Q5 (1000-armed bandit) :

- Here we compare average performance of UCB1, Softmax & ϵ-greedy algorithm on the 1000-armed testbed over 2000 different bandit problems for time steps 10000 & 100000.
- As the number of arms grows in our case to 1000, we need to increase time steps also so that each arm is pulled enought times, so that algorithm is able to identify which arm is optimal.
- Thus, we run all three algorithms for 10000 & 100000 time steps. We can observe that Softmax & ϵ-greedy algorithm performs better in initial steps, as it will try to pull arm which highest mean estimates with more probability and explore other arms with less probability. But UCB1 does not perform well in intial steps as it will do more exploration & selects arms randomly which are not pulled enough time ($N_t(a)$ is low) and who's variance is high.
- But eventually with time (for 100000 case and even in 10000 case after 5000 steps) we can observe that UCB1 performs far better than softmax and ϵ-greedy. As softmax and ϵ-greedy will not be able to do enough exploration when arms are more and it will eventually start pulling sub-optimal arm. Thus we need to select different parameters of ϵ and t for those algorithm to do better for more number of arms
- Hence, we can observe that selection of parameter (ϵ & t) is crucial for ϵ-greedy & softmax algorithm to work well, when number of arm increases/decreases. While UCB1 will adjust it's upper confidence bound for each arm based on $N_t(a)$, $Q_t(a)$ & t and will do enough exploration, when number of arms increases in initial steps, and later will lock on to pulling optimal arm more number of times.

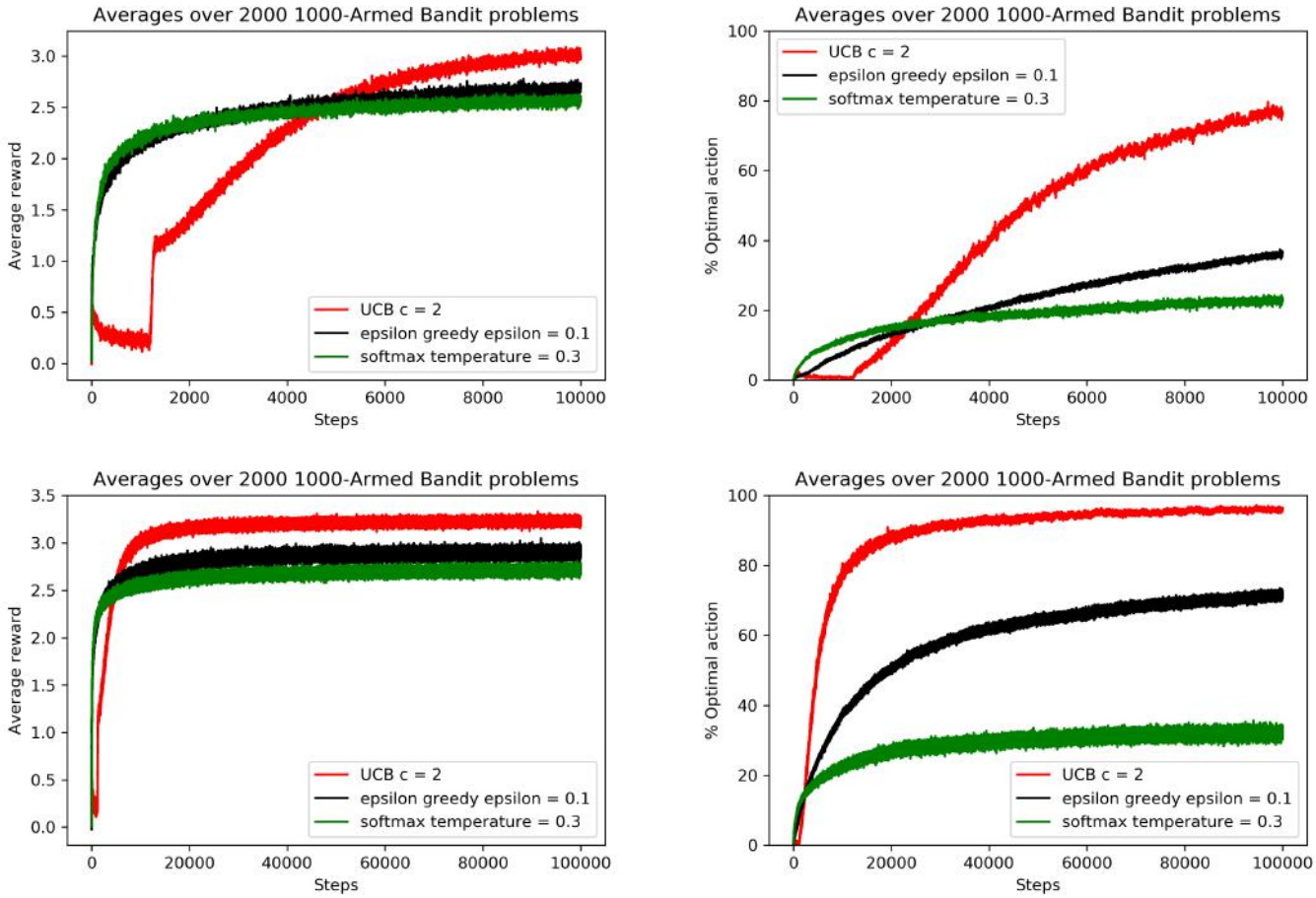


Figure 6: Comparison of average performance of UCB1, Softmax & ϵ-greedy algorithm over 2000 different 1000-armed bandit problems for time steps 10000 & 100000.

- Now, we run Median Elimination algorithm for 1000 armed-bandit case. And similar to earlier case when ϵ is large, total samples required would be comparitively less and when ϵ is small more total samples would be required to achieve (ϵ, δ)-PAC.
- But now as number of arms are 1000 it's sample complexity would also be increase by 100x compared to earlier case when there we 10 arms, which can be observed from below table and graphs.

epsilon ϵ	delta δ	Total samples	% Optimal arm retained	#times optimal arm not returned	#arm returned outside epsilon range
0.9	0.9	1085332	99.45	11	0
0.9	0.1	1631785	99.20	16	0
0.5	0.9	3521043	99.50	1	0
0.5	0.5	3995040	99.60	8	0
0.5	0.1	5290045	99.55	9	0

Table 2: Median Elimination algorithm average performance for different parameter averaged over 2000 different 1000 armed-bandit problems for different values of ϵ & δ

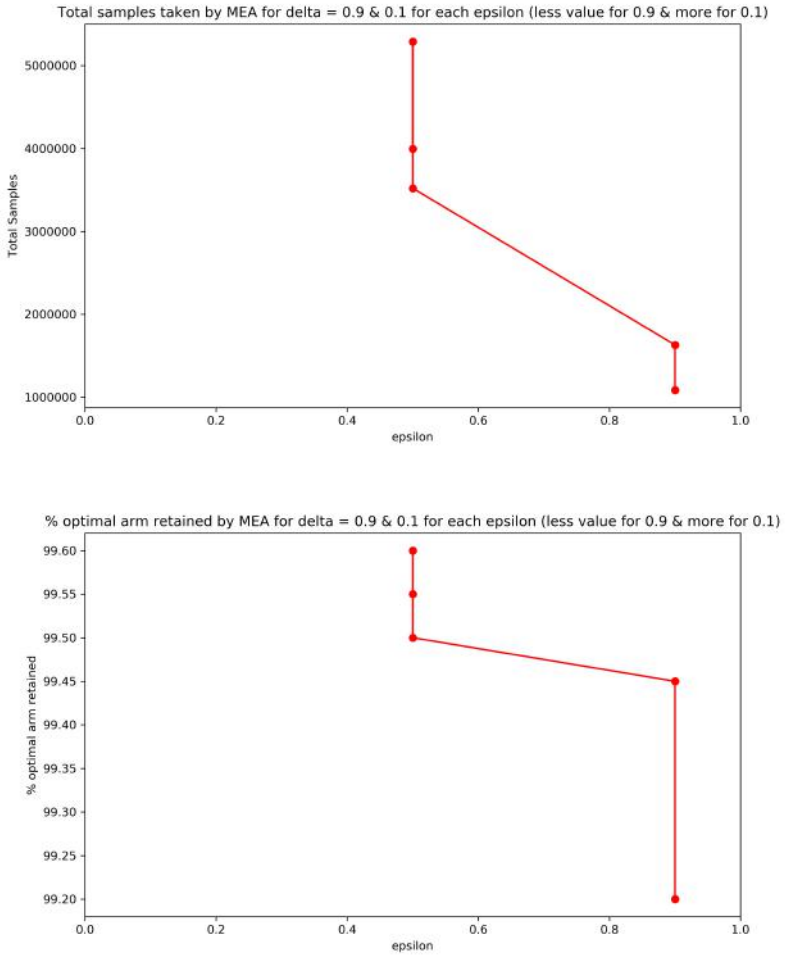


Figure 7: Comparing Average Total samples taken & % Optimal arm retained by MEA for $\delta = 0.9$ & 0.1 for each epsilon (less value for 0.9 & more for 0.1) for different values of ϵ over 2000 different 1000-armed bandit problems.

References:

1. Richard S. Sutton and Andrew G. Barto, Reinforcement Learning: An Introduction, Second edition

2. Even-Dar, Eyal, Shie Mannor, and Yishay Mansour. "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems."Journal of machine learning research 7.Jun (2006): 1079-1105.

3. Github Link: Shangtong Zhang, Python implementation of Reinforcement Learning: An Introduction.

4. [MU2017] Michael Mitzenmacher and Eli Upfal. "Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis," chapter 3, pages 57-62. Cambridge University Press, Second edition, 2017.