CS6770 Reinforcement Learning Programming Assignment-1

Aniruddha Roy ee18d031

February 2020

1 ϵ Greedy Algorithm

- We will have to conduct experiments on the 10-arm testbed. For conducting this experiment, there are 1000 pulls and 2000 bandits are given. Each bandit means one run. That is this we are given 1000 steps and 2000 runs to conduct this experiment.
- These bandits problems are sample from standard normal or Gaussian distribution(mean = 0 and standard deviation =1).We will get true mean ($q^*(a)$) of each arm from this standard normal distribution.
- To compare the performances of average reward for different values of ϵ,we have to run one bandit problem over 1000 steps. Then repeating this for 2000 runs and these runs are independent. Then we will average reward for each time steps over 2000 runs.

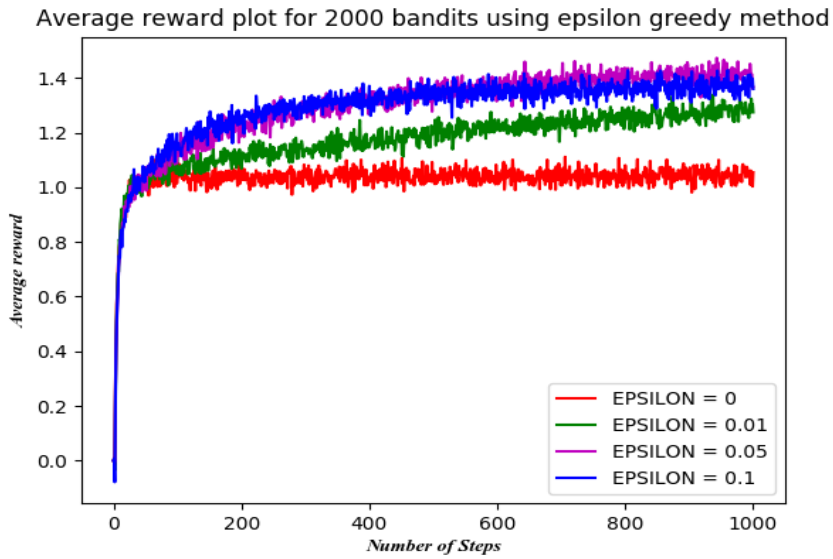

Figure 1: Average reward vs. steps(ϵ greedy algorithm)

- From the above figure.1, we are comparing the performances for different values of $\epsilon = 0, 0.1, 0.05, 0.1$. The greedy method ($\epsilon = 0$) improved slightly faster than other methods at the very beginning, but the leveled off at a lower level. This method performed significantly worse in the long run because it often get stuck performing sub-optimal actions.

- This figure.1 shows that the increase in average reward with learning. If we observe clearly, average reward for both $\epsilon = 0.1$ and $\epsilon = 0.05$ is almost equal because they spent more time to chose the optimal actions for exploring.

- But for long number of steps, we are getting slightly better average reward for $\epsilon = 0.05$ than $\epsilon = 0.1$.

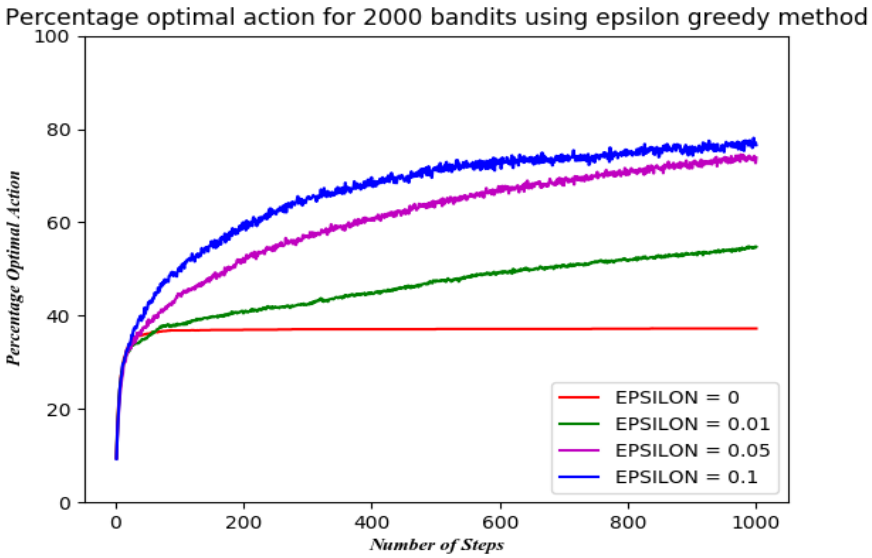

Figure 2: Percentage optimal action vs. steps(ϵ greedy algorithm)

- From the above figure.2, it shows that the greedy method got the optimal action only approximately 33% of the tasks. In the other 63% of the tasks, its initial sample of the optimal action were disappointing, and it never return it.

- It shows that ϵ greedy method performs better with compare to greedy method because it is exploring more to chose the optimal action. The $\epsilon = 0.05$ method improves more slowly with eventually performed better on performance measure $\epsilon = 0.1$.

2 Soft-max Algorithm

- Now we have to implement the Soft-max algorithm on 10 arm testbed over 2000 different bandits problems and 1000 time steps. We will take sample from the soft-max distribution or Gibbs distribution.

$$\text{pr}(A_t = a) = \frac{e^{\frac{Q_t(a)}{\beta}}}{\sum_{b=1}^{k} e^{\frac{Q_t(b)}{\beta}}}$$

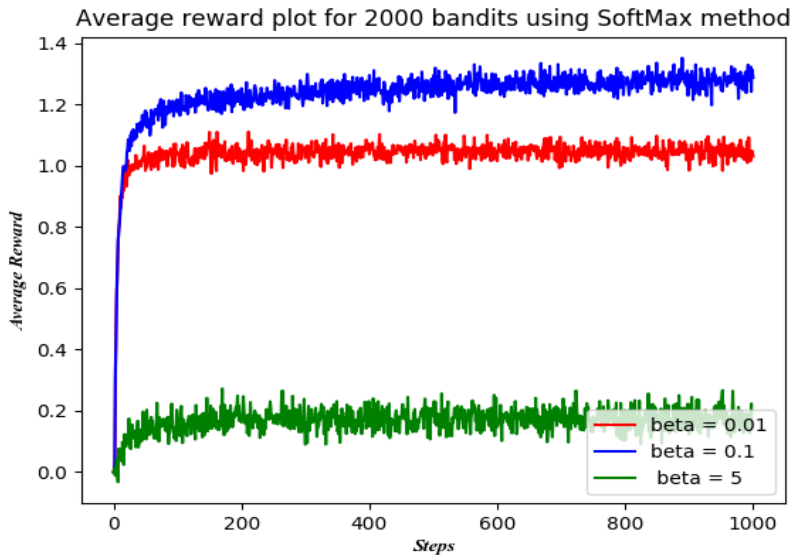

Figure 3: Average reward vs. steps (SoftMax)

- Here β is the temperature parameter. We compare the performance measure for different values of this parameter.

- From the figure.3, it shows average reward performances for the different parameter values ($\beta = 0.01$, $\beta = 0.1$, $\beta = 5$). There is always randomness among choice the arms. This parameter actually control this randomness choice.

- When $\beta = 0.01$, this method will perform like a greedy($\epsilon = 0$) method.Its is clearly observed from the figure.3.

- When $\beta = 1$ and $\beta = 0$ it is explored and exploited more respectively. When β tends to infinite, this method will pull the arms uniformly at random. That means it will not perform well as it will not picking optimal arms many times. So, the β value should be the middle of 0 and 1 to get the best performance.

- So, we took the β value greater than one (in our case, it is 5) and it is observed that it performs very bad. Here, average reward is very low (refer to the figure.3) and it is also not pulling optimal arms as many times (refer to the figure.4)

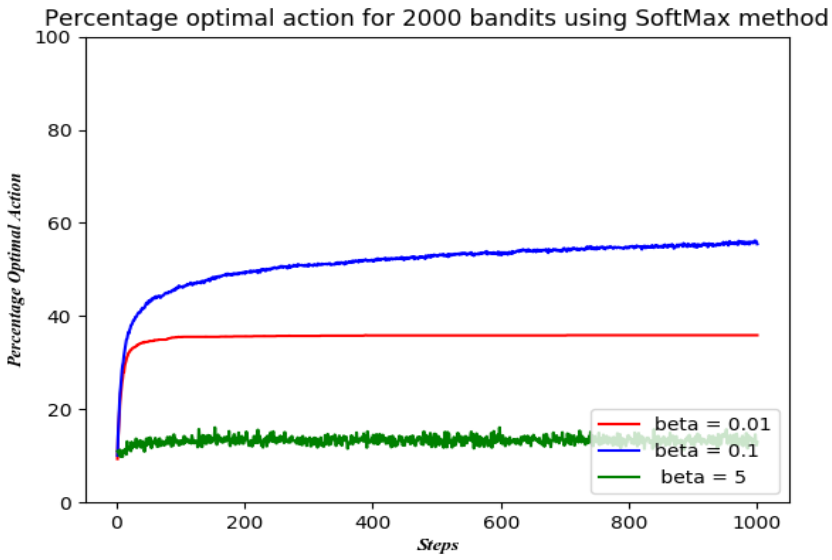

Figure 4: Percentage optimal action vs. steps (SoftMax)

3 UCB1 Algorithm

- Now we have to implement UCB1 algorithm on similar 10 arm testbed with 1000 steps and 2000 bandits. The greedy action always explore best action at present but they never explore other actions may actually better. So there is an uncertainty regarding the estimate of average value.

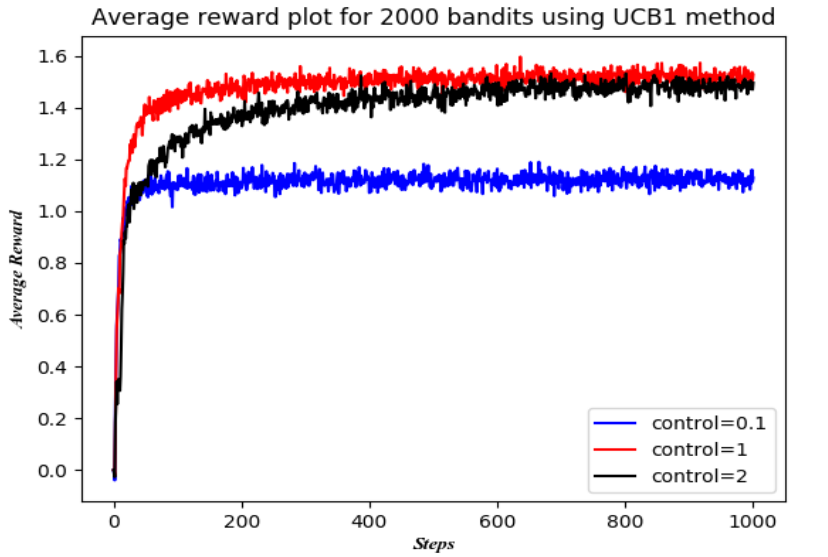

Figure 5: Average reward vs. steps (UCB1 algorithm)

- The main idea is to select among the non-greedy actions for those are actually being optimal by taking into account both how close their estimates are to being maximal and the uncertainties in those estimates.

- UCB1 selects an arm $A_t = argmax[Q_t + c\sqrt{\frac{lnt}{N_t(a)}}]$ Where lnt denotes natural logarithm of t, $N_t(a) =$ the number of times that action a has been selected prior to time t and here c is a non-negative number which is controls the degree of exploration. If $N_t(a) = 0$ then a is considered to be a maximizing action.

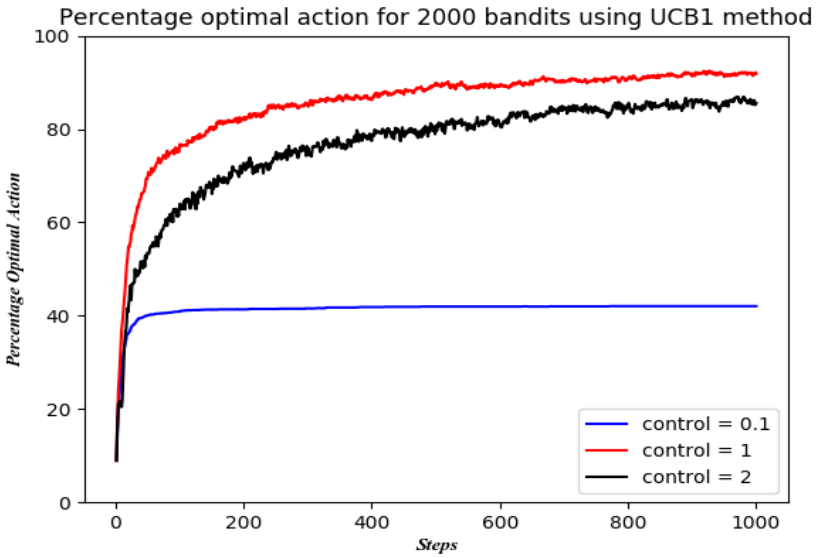

Figure 6: Percentage optimal action vs. steps (UCB1 algorithm)

- The figure.5 shows that average reward plot with respect to the steps for 2000 bandits problem for different values of $c = 0.1, 1, 2$. It is observed that for $c = 1$ shows a distinct spike and it is less prominent compare to other two c values

- The figure.6 shows that the percentage optimal arms selection. It is clear that with large number of time steps both $c = 1$ and $c = 2$ choose 80% of the optimal action of the tasks.

- This algorithm guarantees that all arms will eventually be selected. The optimal arm will be played more frequently with compare to arm with lower value estimates.

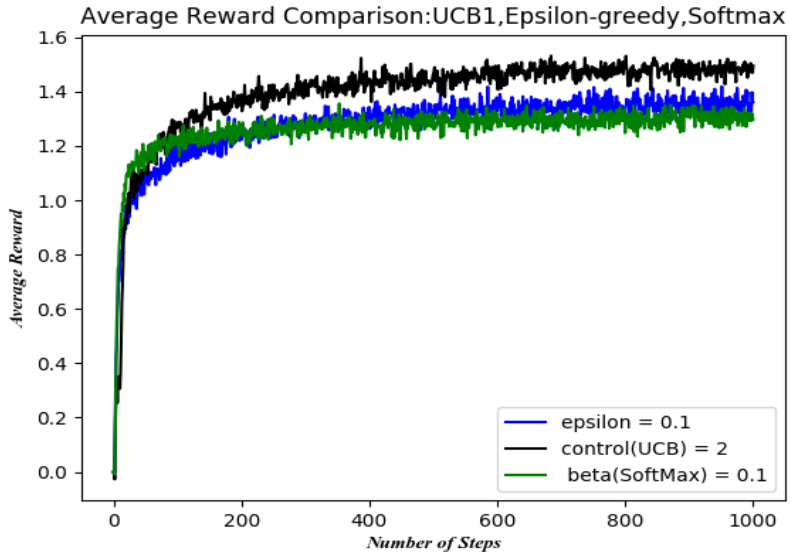

Figure 7: Average Reward comparison vs. steps (ϵ greedy, Soft-max, UCB1)

- To calculate the upper confidence bound of each arm,which is keep changing, UCB1 algorithm will do both exploration and exploitation from time to time.So, It will adjust the bounds for each arm and will do both exploration and exploitation without adjusting the any external parameters.
- So, UCB1 performs better than softmax and greedy method.
- From the figure.7, which shows the average reward comparison among ϵ greedy, UCB1 and soft-max algorithms. It shows that UCB1 method performance measure is better except the initial steps. At the initial steps soft-max performs better than UCB but for large number of steps UCB1 performed well than other two methods. We have to tune the parameters well for ϵ greedy and soft-max algorithms to get the desired performance. In UCB1, we do not have to select any parameters.

4 Median Elimination Algorithm

- The Median Elimination Algorithm (MEA) is a round based algorithm. After every rounds it will eliminate half of the bad arms.
- This algorithm is an (ϵ, δ) PAC algorithm. In $logk$ stages it will give an arm within ϵ range of optimal arm with at least $(1-\delta)$ probability.
- We will have to sample every arm $\frac{2}{\epsilon^2}\log\frac{3}{\delta}$ times. Our main target to get the best arm. So, we will stop this algorithm when length of arm vector will be one.
- The sample complexity of the MEA is $O(klog(\frac{3}{\delta}))$. Here k is the number of arms.

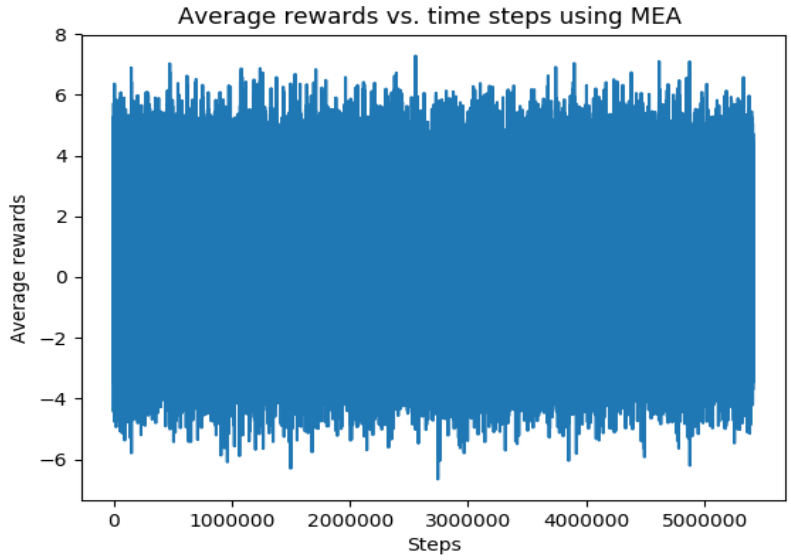

Figure 8: Average Reward vs. steps (MEA algorithm)

- **Is finding the median the rate-determining step ?** We took arm number is equal to 10. So, it is very less. To find out median of estimated mean values is not the rate determining step. Because we have 10 arms. After every round there will be a remaining of half of the bad arms. So, set will be like $\{5, 3, 2, 1\}$ after the first round.So,after every rounds the number of arms are decreases.
  But, if we increase the number of arms (say 10000), then using standard method sorting and then finding middle element can be rate determining step. In that case the time complexity will be $O(klogk)$.

5 1000 Arm Bandit Setup

- This problem is basically related to performance measure of different algorithms if the number of arms grows.
- We have compared the ϵ greedy (ϵ =0.1), Soft-max ($\beta = 0.1$) and UCB1 (c=2) methods on a 1000 arm bandit setup with time steps 60000. We have to increase the number of steps as the arms grows to get the optimal actions.
- For simulation perspective, it will take too much time to run above three algorithms with 2000 bandits and 1000 arms. We are taking large number of steps to get because of more number of arms.

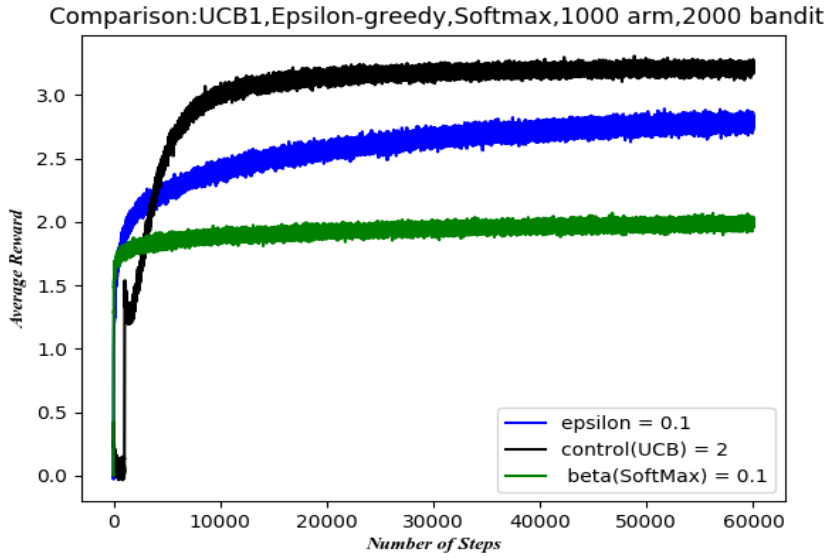

Figure 9: Average rewards comparison vs. steps (ϵ greedy, softmax and UCB1)

- From the figure.9, it shows that for initial number of steps ϵ greedy and soft-max performs better because it will try to pull the arm which highest mean estimates with more probability and explore other arms with less probability.
- If we observe the figure, it is clear that in initial states UCB1 does not perform well because of that it will explore more to pull the arm randomly which are not pulled within time and having high variance.
- UCB1 will adjust its upper bound based on $N_t(a), Q_t(a)$ and t and will do enough exploration when the number of arms increases in initial stages.
- When the number of arms increases or decreases, we observed that for both the greedy and soft-max methods, the selection of parameters are very important and based on the selection parameter both will perform well and it will pull more optimal arms with increasing time horizon.

References

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

2. GitHub Link, Shantong Zang, Python Implementation for Reinforcement Learning.

3. Eyal Even-Dar, Shie Mannora and Yishay Mansour,PAC Bounds for Multi-Armed Bandit and Markov Decision Processes.