CS6700 Assignment No. 1: Report

-Akash Reddy A, EE17B001

February 4, 2020

Overview

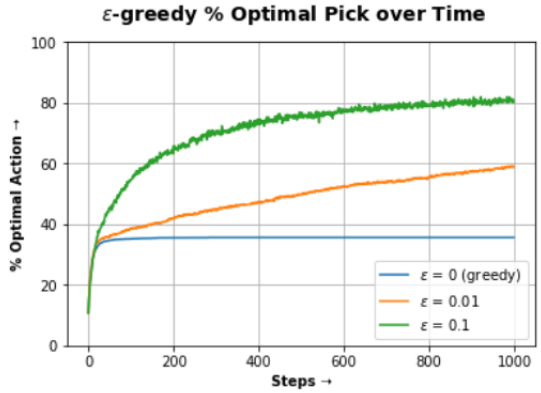
This assignment applies various methods to pick the optimal arm from a multi-armed bandit. It then compares the performance between these various algorithms, and also by changing the total number of arms.

`matplotlib` has been used to plot the graphs.

1 Question 1

The ϵ-greedy algorithm has been implemented for $\epsilon = 0, 0.01, 0.1$. We observe that an ϵ of 0 (greedy) policy does well initially but easily gets stuck on a sub-optimal action. $\epsilon = 0.1$ does best in the time frame we have given but saturates in performance, while $\epsilon = 0.01$ is still increasing and eventually beats $\epsilon = 0.1$ beyond 1000 time steps.

This is reflective of the exploration-exploitation trade-off. The right amount of exploration is the one that is compatible with the amount of variance in the data.




```
Time taken for eps-greedy (10 arms) = 0.40415477752685547
```

Figure 1: Plots for ϵ-greedy with 10 arms

2 Question 2

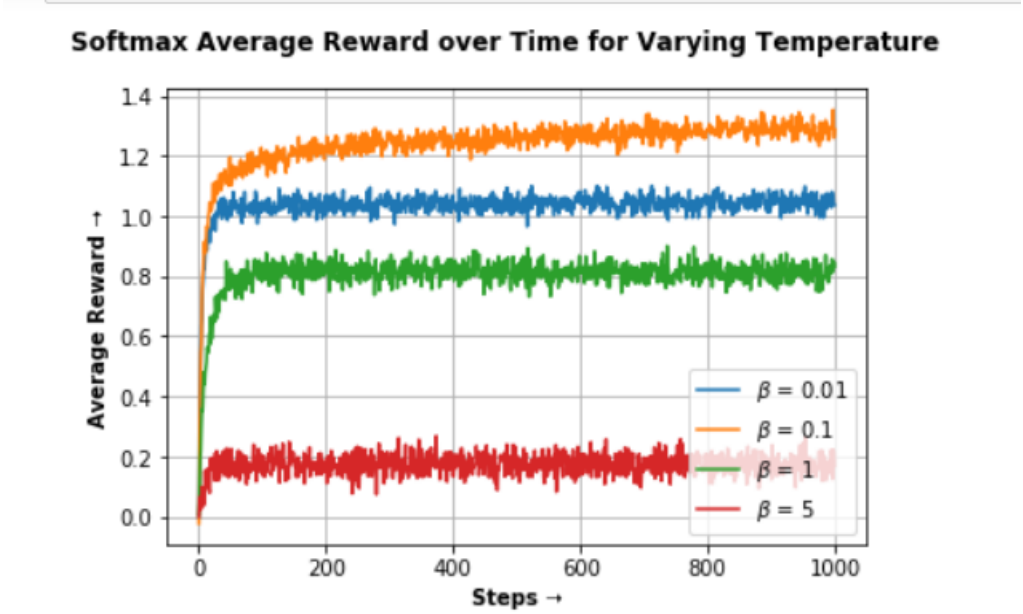
The softmax selection algorithm does best for a temperature (β) of around 0.1. This is because there has to be such an optimal temperature, as too low β can lead to the policy becoming greedy, and too high β can lead to the policy becoming random.

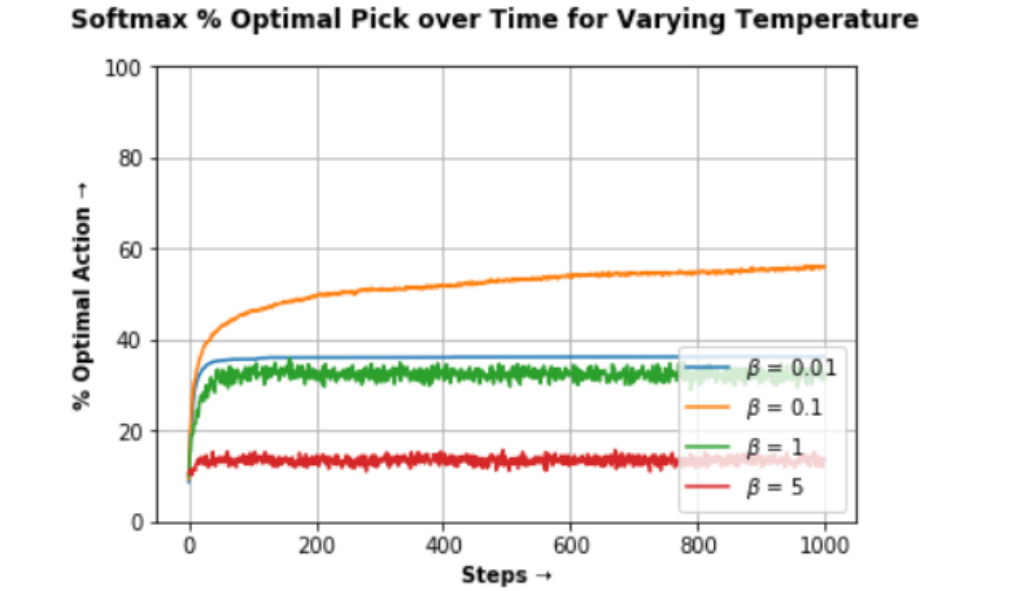
This is visible in the graphs, where high β leads to very low average rewards. Low β (0.01) behaves similarly to the greedy algorithm in the previous question.

Also, in general, softmax has a faster convergence than ϵ-greedy (but converges to nearly the same average reward). This is because the agent keeps picking good actions with good probability, even if they are not

necessarily the best ones, thereby reducing the possibility of picking outliers whose true average rewards are lower.

Softmax has lower percentage of optimal action pick compared to ϵ-greedy, because it also picks other good non-optimal actions very often leading to high reward as well.






```
Time taken for softmax (10 arms) = 0.5373361110687256
```

Figure 2: Plots for softmax with 10 arms

3 Question 3

The UCB1 algorithm is a regret-optimal algorithm, which is evident from the plot. The initial slope of the rewards is very high (higher than ϵ-greedy and softmax both) leading to low regret.

Also, the final reward reached at the 1000th time step is higher in UCB1 than in the previous two algorithms.

UCB1 has higher % optimal action picked at the end of 1000 time steps.

There is a spike on the 11th time step because after the average rewards of the 10 arms over 2000 runs in no particular order, the 11th selection is made by maximising $Q(j) + \sqrt{\frac{2 \ln n}{n_j}}$. This leads to sudden high average reward.

Beyond this, there is a drop in some subsequent rewards. This is because, after picking the 11th-timestep action, its n_j has increased, and the quantity to be maximised decreases for this action. Therefore we move to the next action, which has $n_j = 1$, but a smaller $Q(j)$, leading to smaller average reward for the subsequent timesteps.

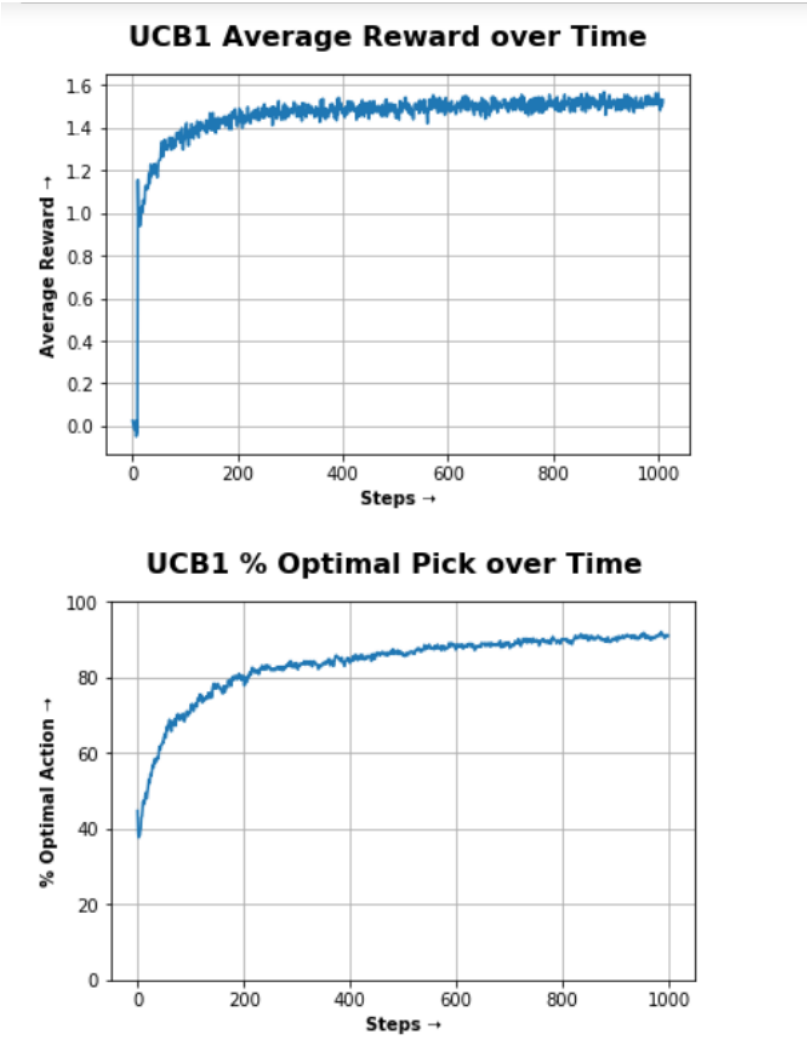


Figure 3: Plots for UCB1 with 10 arms

4 Question 4

The Median Elimination algorithm is a PAC-guarantee algorithm.

As ϵ and δ decrease (accurate action picked more confidently), the sample complexity which is empirically of the order of $\frac{k}{\epsilon^2} \ln \frac{k}{\delta}$ will increase rapidly. This means that the number of times non-optimal actions are picked during training will increase, which leads to higher total regret as well as seen in the plot below.

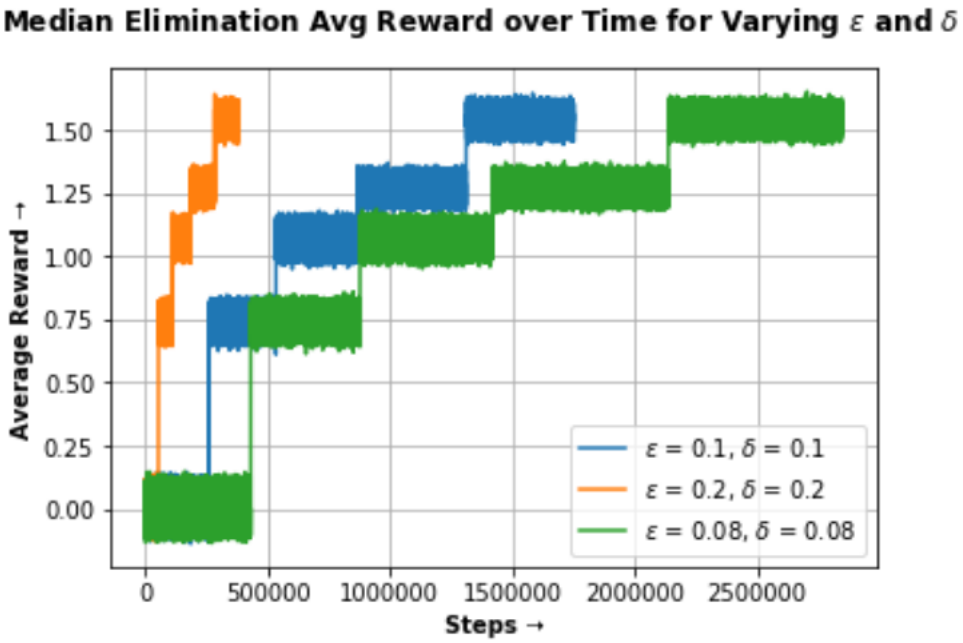


Figure 4: Plot for Median Elimination with 10 arms, varying values of ϵ and δ

For the comparison with other algorithms, $\epsilon = \delta = 0.1$ has been chosen.

Initially, when we sample all arms from all runs and average the rewards, they will be close to 0, but once we have dropped the lower arms after one round, the rewards will be close to the average of the true means of the remaining arms. This process repeats. Hence, we see a step-like evolution of average rewards, with a jump at the end of each round, when half the arms are dropped. This is drastically different from the gradually increasing plots in previous methods.

Also, Median Elimination converges to a high value of average reward. While plotting the average reward, each pull of 1 arm has been treated as a time step.

(The percentage optimal action has not been plotted because it is obvious: actions are picked one by one, multiple times, which means that this percentage will be positive when we are picking the optimal action in any of the runs, and be equal to 0 when we are not picking any optimal action anywhere.)

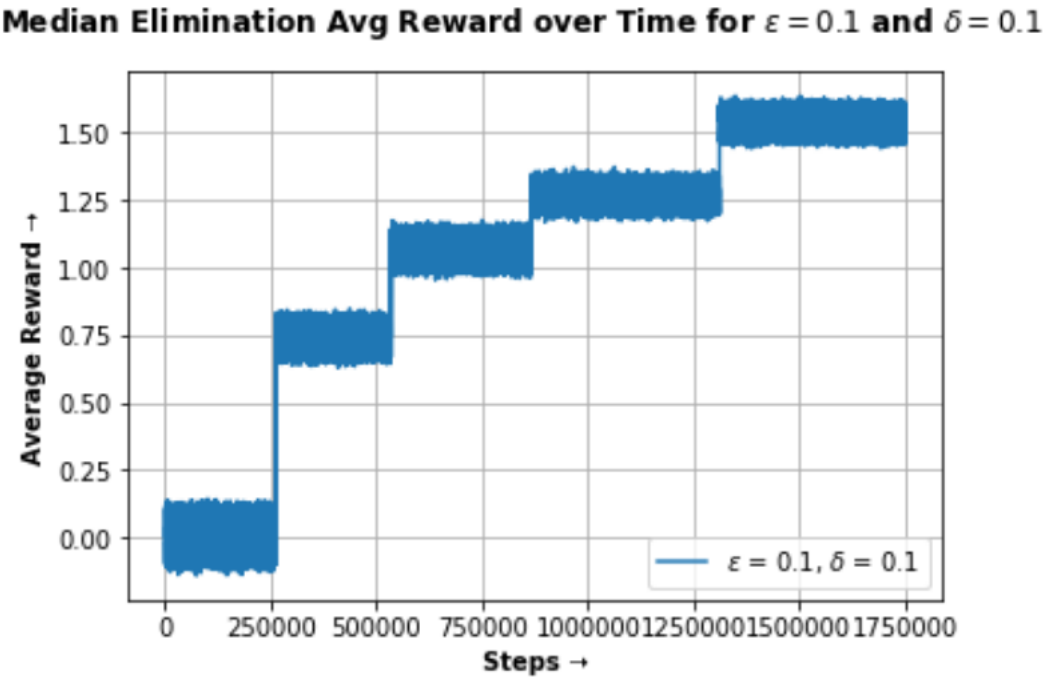


Figure 5: Plot for Median Elimination with 10 arms, $\epsilon = \delta = 0.1$

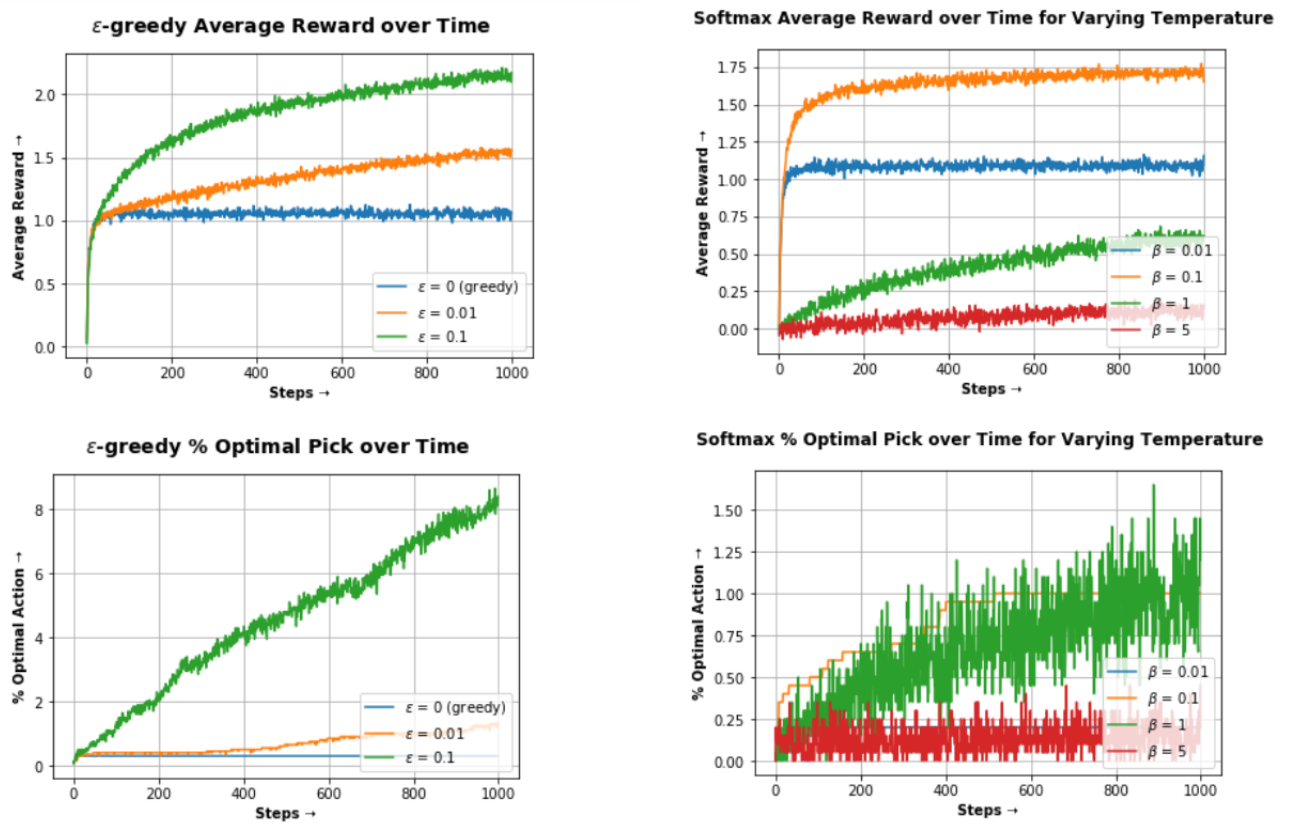
5 Question 5

Using a 1000-armed bandit shows us that the algorithms need more time steps in order to converge.

We can see that both ϵ-greedy and softmax have average reward plots that aren't saturated (continuing to grow). The final rewards seem to be higher than the 10-arm case. This is because among 1000 arms, the 1 optimal action has a higher chance of having a high outlier of the (0,1) standard normal distribution as its mean.

We also see that the % of optimal action picked is very small in 1000 time steps, and growing. Clearly, the algorithm needs more time steps to converge to a solution.

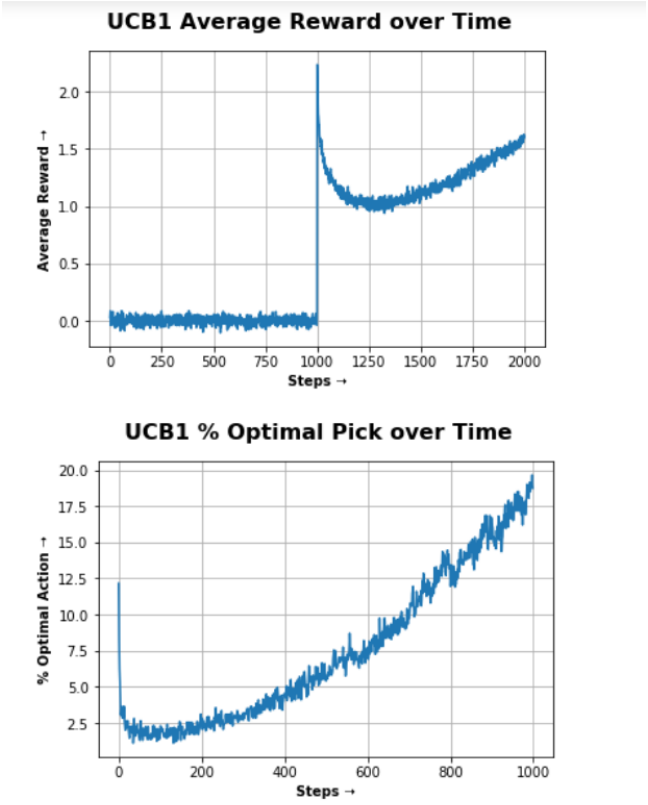
In the 1000-arm UCB1 plot, the rewards are nearly 0 for the first 1000 time steps (as opposed to the first 10 time steps). The 1001st average reward is a spike (instead of the 11th one in the 10-arm case), and the subsequent rewards are smaller, yet growing, similar to the 10-arm case. Once again, more time steps are needed for a solution. (The % optimal action plot is plotted from the 1001st step).




```
Time taken for eps-greedy (1000 arms) = 8.1108050346374!
```

```
Time taken for softmax (1000 arms) = 54.05052828788757
```

Figure 6: Plots for ϵ-greedy and softmax with 1000 arms

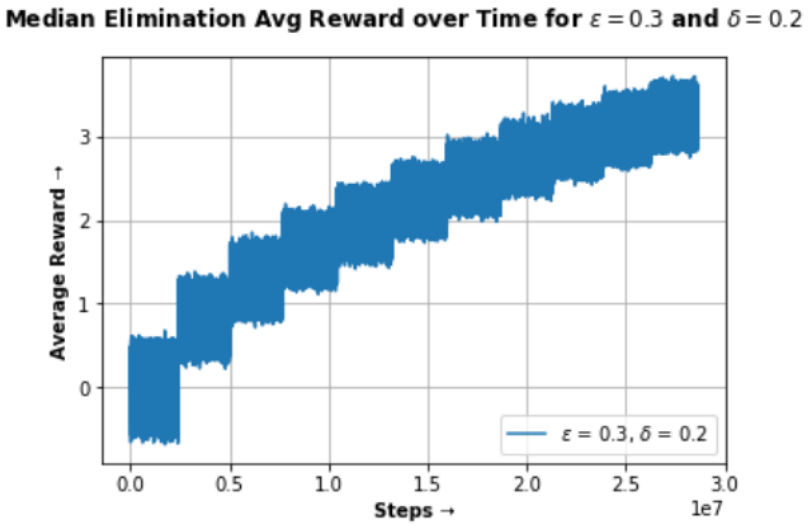



```
Time taken for UCB1 (1000 arms) = 29.198557138442993
```

Figure 7: Plot for UCB1 with 1000 arms

In Median Elimination for 1000 arms, number of runs is made 100, and $\epsilon = 0.3$ and $\delta = 0.2$ (both larger than 0.1) for faster execution of code (through smaller sample complexity). We observe that the final reward is higher than the 10-arm case, and also there are more jumps in average reward because of more number of rounds in the case of 1000 arms. Number of rounds = ceil($log_2 k$) where k is the number of arms. Therefore, there are 10 jumps in this plot.

Another obvious distinction is the significant increase in time taken to run all algorithms for 1000 arms.




```
Time taken for Median Elimination (1000 arms) = 243.9287977218628
```

Figure 8: Plot for Median Elimination with 1000 arms, $\epsilon = 0.3$, $\delta = 0.2s$

****