Reinforcement Learning Assignment-2
Multi-Armed Bandit Problem

Utkarsh Prakash
180030042

February 14, 2022

1 Problem Statement

A multi-armed Bandit is a set of distributions $\{\mathcal{R}_a | a \in \mathcal{A}\}$ where $\mathcal{A}$ is a known set of arms and $\mathcal{R}_a$ is the distribution of the rewards given the arm a. K represents the number of arms being considered in the problem. At each time step, we select an arm $A_t \in \mathcal{A}$ and get a reward $R_t \sim \mathcal{R}_{A_t}$. Let $q(a)$ denote the expected reward that can be obtained if the arm a is pulled, i.e. $q(a) = \mathrm{E}[R_t | A_t = a]$. Using, $q(a)$, we can define an optimal arm as the one having the highest value of $q(a)$. Let's denote the value $q(a)$ of this optimal arm as v_*.

The goal of the problem is to minimize the regret over time (T), i.e. the difference between the between the maximum reward that can be collected till time T and Expected total reward obtained by an algorithm.

2 Experiment Setting

We consider Bernoulli and Normal reward Distributions for our experiments. We assume that the variance (σ^2) of the Normal reward distribution is known and is equal to $\sigma^2 = 0.1^2$. The following means of the reward distribution were used for comparison:

- For Bernoulli Reward Distribution:
  - For K=2, [0.37, 0.95]
  - For K=5, [0.15, 0.37, 0.59, 0.73, 0.95]
  - For K=10, [0.05, 0.15, 0.16, 0.37, 0.59, 0.60, 0.70, 0.73, 0.86, 0.95]
- For Normal Reward Distribution:
  - For K=2, [0.37, 0.95]
  - For K=5, [0.15, 0.37, 0.59, 0.73, 0.95]
  - For K=10, [0.05, 0.15, 0.16, 0.37, 0.59, 0.60, 0.70, 0.73, 0.86, 0.95]

For each experiment we run our algorithm for 1000 time steps. This is considered to be a run for an algorithm with a given Bandit problem. In order to compare the performance of the algorithms fairly, we evaluate the performance of the algorithms on the same Bandit problem. We run our algorithms for 1000 runs with the same bandit problem. We use the following metrics for evaluating the performance of the algorithm:

- The total regret accumulated over time.
- The regret as a function of time.
- The percentage of plays in which the optimal arm is pulled.
- Average reward as a function of time.

Note that we average these metrics over 1000 runs of the algorithm. In this report we have included only a subset of the plots obtained from different experiment. This is done because most of the experiments showed similar patterns and therefore, including them would just be repetition. However, we include all the plots where we observe something interesting and draw meaningful conclusions from them. In order to see all the plots visit the `.ipynb` notebook attached along with this report.

Note that throughout the report wherever we don't mention the schedule to decrease a quantity over time, we refer to the following schedule:

$$x = \frac{\zeta}{t} \tag{1}$$

where x is the quantity that has to be decreased over time and ζ is a tunable hyperparameter.

3 Notations

Apart from the notations and concepts introduced above, we also use following notations in our report:

- $N_t(a)$ represents the number of times arm a has been pulled uptil time t (including time t).
- $Q_t(a)$ represents the estimate of $q(a)$ over time t, which is calculated as follows:

  $$Q_t(a) = \frac{\sum_{n=1}^{t} 1(A_t == a)R_t}{\sum_{n=1}^{t} 1(A_t == a)}$$

  where 1 represents an indicator function. We can also calculate $Q_t(a)$ incremently as:

  $$Q_t(a) = Q_{t-1}(a) + \alpha(R_t - Q_{t-1}(a))$$

  where $\alpha = \frac{1}{N_t(a)}$.
- $\pi_t(a)$ represents the probability of picking arm a at time t. This is also known as policy.

The way the rest of the report is organized is as follows: In section 4 we compare the performance of different algorithms on Bernoulli reward distribution. Firstly, for $K = 2$ arms, we show the performance of different variants of an algorithm and then with the best performing variant of the algorithm (algorithm with the best set of hyperparameters) we compare different algorithms. Thereafter, we also compare the performance of algorithms for different number of arms. In section 5 we compare the performance of different algorithms for Normal reward distribution. We do the comparison on the similar lines as we did for the Bernoulli reward distribution. Finally, in section 6, we show a case where reinforce algorithm with baseline outperforms the algorithm without baseline.

4 Bernoulli Reward Distribution

4.1 Algorithms

4.1.1 Greedy Algorithm

In this section, we compare the pure Greedy algorithm, ϵ-greedy with fixed $\epsilon = 0.3$ and variable ϵ with the following schedule:

$$\epsilon_t = min\{1, \frac{C}{t}\} \tag{2}$$

where $C = 10$ and t is the total number of plays.

4.1.2 Softmax Policy

In this section, we compare the Softmax Policy which is defined as follows:

$$\pi_t(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{i=1}^{K} e^{Q_t(a_i)/\tau}}$$

where τ is the temperature hyperparameter. We compare the performance of Softmax Policy for $\tau = 0.01$, $\tau = 100$ and variable τ with schedule as defined in (1) where $\zeta = 10$. We reduce the τ overtime to control exploration-exploitation tradeoff.

4.1.3 UCB Algorithm

In this section we compare the UCB algorithm where we pick the arm which has the highest value of

$$\arg max_{a \in \mathcal{A}} \left(q_t(a) + C\sqrt{\frac{2 \ln t}{n_t(a)}} \right)$$

where C is a hyperparameter which controls exploration-exploitation tradeoff. We used $C = 1$.

4.1.4 Thompson Sampling

In Thompson Sampling, we maintain a Beta distribution prior over $q(a)$. We sample a value $Q_t(a)$ from this Beta distribution for each arm, i.e. $Q_t(a) \sim q(a)$. Now, we select the arm which has the highest value of $Q_t(a)$.

4.1.5 Reinforce Algorithm

The reinforce algorithm follows the following policy:

$$\pi_t(a) = \frac{e^{\theta_t(a)}}{\sum_{i=1}^{K} e^{\theta_t(a_i)}}$$

where $\theta_t(a)$ are the learnable parameters of the algorithm. These parameters can be learned using Stochastic Gradient Ascent with the following update rule:

$$\theta_t(a) = \begin{cases} \theta_t(A_t) + \alpha(R_t - b)(1 - \pi_t(A_t)) & if a = A_t \\ \theta_t(a) + \alpha(R_t - b)\pi_t(a) & if a \neq A_t \end{cases}$$

where b is a baseline and is generally chosen to be the average of all the rewards up through and including time t. The following graphs were obtained with and without baselines:

4.2 Comparison of all Algorithms for K=2 arms

In this section we compare the performance of all the algorithms for the 2 arm problem. The following algorithms were used for comparison:

- ϵ-Greedy with $\epsilon = 0.3$.
- Variable ϵ-Greedy where ϵ decreases according to the schedule defined in (2) with $C = 10$.
- Softmax with $\tau = 0.01$ and $\tau = 100$.
- Variable τ-Softmax where τ decreases according to the schedule defined in (1) with $\zeta = 10$.
- UCB with $C = 10$.
- Thompson Sampling Algorithm
- Reinforce Algorithm with and without baseline.

The results obtained are as follows:

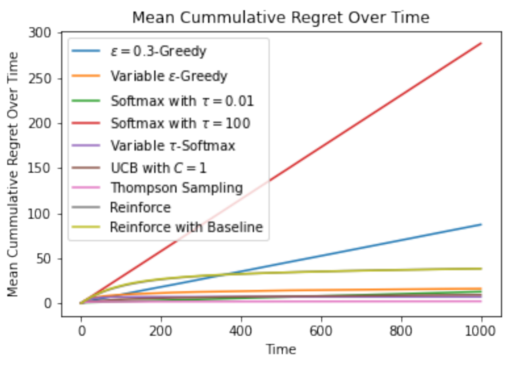

Figure 1: Mean Cummulative Regret Over Time

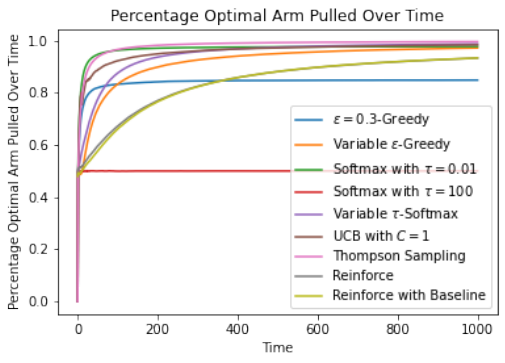

Figure 2: Percentage Optimal Arm Pulled Over Time

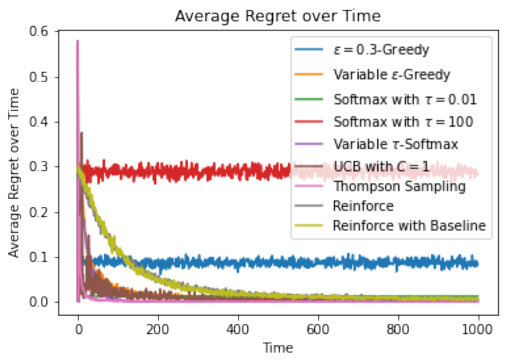

Figure 3: Average Regret over Time

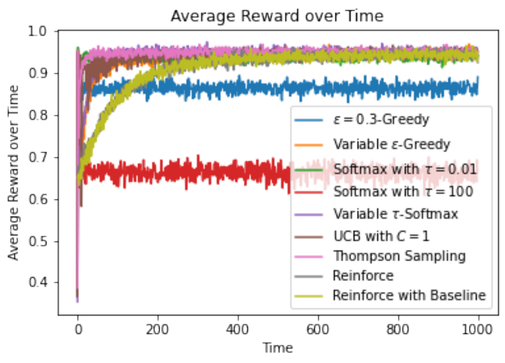

Figure 4: Average Reward over Time

Observations:

- The variable ϵ-Greedy tends to perform better than all of its Greedy counter-parts. The fixed ϵ-Greedy tends to have linear regret whereas variable ϵ-Greedy has logarithmic regret.
- For small values of $\tau = 0.01$, the algorithm performs poorly because in the limit when $\tau \to 0$, then the algorithm performs greedily.
- For large values of $\tau = 100$, the algorithm performs poorly because in the limit when $\tau \to \infty$, then the algorithm picks an arm uniformly at random. Therefore, for such large values of τ the algorithm explores at lot.
- The variable τ-Softmax tends to perform better than all of its counter-parts. This is because earlier when the value of τ is high, we tend to explore more, whereas as time progresses and we gather knowledge of different arms, we reduce the value of τ so as to exploit the knowledge that we have gathered (i.e., pull the arm which has highest estimated average reward with high probability).
- UCB algorithm tends to have logarithmic regret.
- We really don't see any difference in performance between the algorithm with and without baselines.
- Variable ϵ-Greedy, Variable τ-Softmax, Variable C-UCB and Thompson Sampling algorithm seems to perform equally well. This means that any algorithm which optimally trades-off between exploration and exploitation, should perform well i.e. have logarithmic cummalative regret over time. This is an interesting observation as simple algorithms with simple heuristics tend to perform at par with other algorithms with more sophisticated heuristics.
- Reinforce algorithm doesn't tend to perform that well when compared to its counter-parts. If we look at the plot for the average regret over time for Reinforce algorithm, we find that it intially decreases very slowly. However, later it reaches the same level as that of other algorithms. This nature can be attributed to the fact that the parameters of the algorithm are learned slowly. Thereby, increasing the learning rate for the stochastic gradient ascent may fix the problem and the performance can become at par with other algorithms.

4.3 Comparison of all Algorithms for K=5 arms

We repeated similar experiments for each of the algorithm as we did for $K = 2$ arm case. The results obtained are as follows:

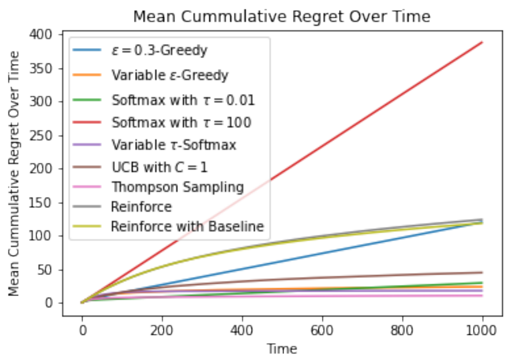

Figure 5: Mean Cummulative Regret Over Time

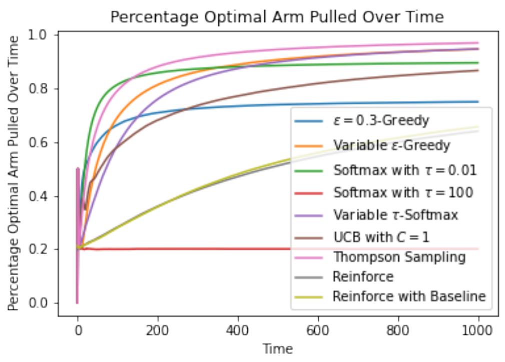

Figure 6: Percentage Optimal Arm Pulled Over Time

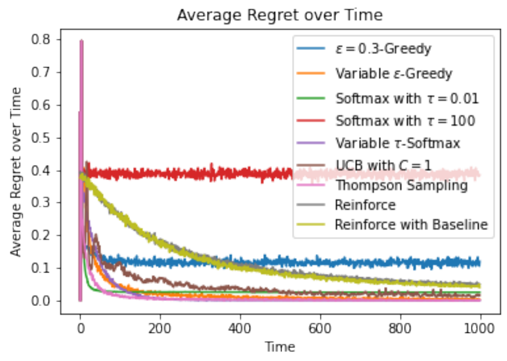

Figure 7: Average Regret over Time

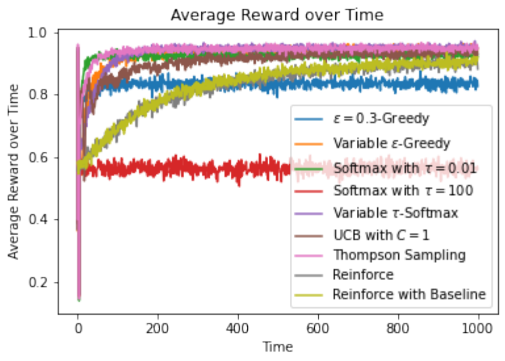

Figure 8: Average Reward over Time

Since, the trends in the graphs are similar to $K = 2$ case, we defer the further discussion on the results to the section 4.5 where we compare the relative performance of the algorithm for different number of arms.

4.4 Comparison of all Algorithms for K=10 arms

We repeated similar experiments for each of the algorithm as we did for $K = 2$ arm case. The results obtained are as follows:

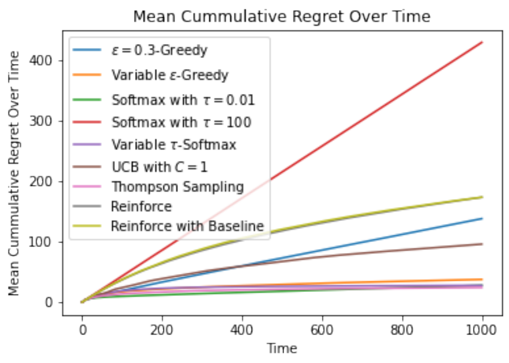

Figure 9: Mean Cummulative Regret Over Time

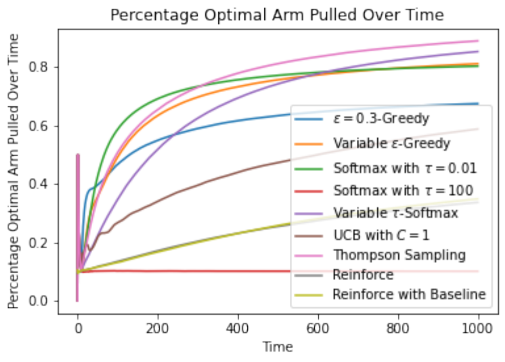

Figure 10: Percentage Optimal Arm Pulled Over Time

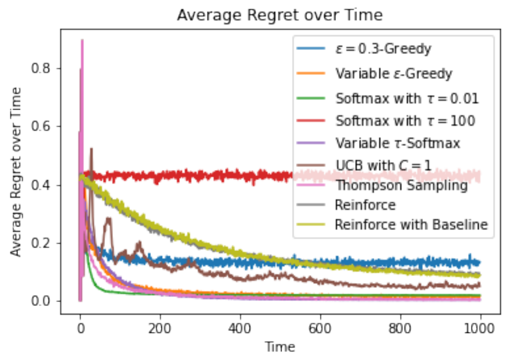

Figure 11: Average Regret over Time

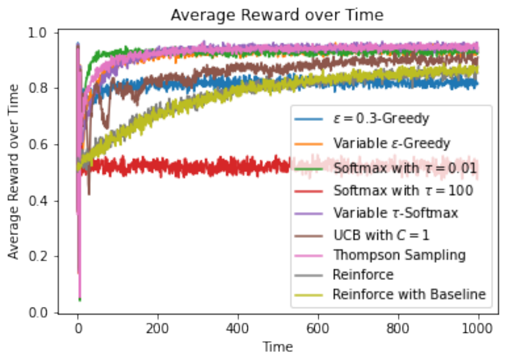

Figure 12: Average Reward over Time

Since, the trends in the graphs are similar to $K = 2$ case, we defer the further discussion on the results to the section 4.5 where we compare the relative performance of the algorithm for different number of arms.

4.5 Comparison of algorithms for different number of arms

The following table lists the total regret at the end of 1000 plays averaged over 1000 runs:

Algorithms	**K=2 Arms**	**K=5 Arms**	**K=10 Arms**
ϵ-Greedy ($\epsilon = 0.3$)	87.33	120.03	138.31
Variable ϵ-Greedy	16.37	23.99	37.98
Softmax ($\tau = 0.01$)	13.04	29.71	28.15
Softmax ($\tau = 100$)	288.08	387.85	429.20
Variable τ-Softmax	7.32	18.19	28.18
UCB ($C = 1$)	9.53	45.01	96.24
Thompson Sampling	2.42	10.79	24.66
Reinforce Algorithm (without baseline)	38.66	123.91	173.50
Reinforce Algorithm (with baseline)	38.49	118.54	173.69

Table 1: Comparison of cummalative regret of different algorithms for different number of arms for 1000 plays averaged over 1000 runs.

One thing to note is that we used the same hyperparameters for different number of arms in order to facilitate fair comparison between the performance of the algorithms. The above table and the plots of section 4.2, 4.3 and 4.4 highlights that the performance of all the algorithms decreases considerably as the number of arms are increased. However, the relative order of performance between algorithms remain as it is for a given number of arms. We observe similar trends for other metrics as well.

5 Normal Reward Distribution

Most of the algorithms remain exactly the same for the normal reward distribution as well. However, UCB and Thompson Sampling change slightly as described below.

- **UCB1-Normal**

Algorithm 1 UCB1-Normal

```
for t=1,2,.., do
    1. If there is an arm which has been played less than $\lceil 8logt \rceil$ times then play this arm.
    2. Otherwise play arm a that maximizes
        $$Q_t(a) + C\sqrt{16 \cdot \frac{S_t(a) - N_t(a)Q_t(a)^2}{N_t(a) - 1} \cdot \frac{ln(t-1)}{N_t(a)}}$$
    where $S_t(a)$ is the sum of square rewards obtained from arm $a$ and C is a tunable hyperparameter which controls the exploration-exploitation tradeoff.
    3. Update $Q_t(a)$ and $S_t(a)$ with the obtained reward $R_t(a)$.
end for
```

- **Thompson Sampling**

  In case of Normal reward distribution we place a normal prior $\mathcal{N}(\mu_0, \sigma_0^2)$ over the mean of the reward distribution i.e. $q(a) \sim \mathcal{N}(\mu_0, \sigma_0^2)$. We assume that the variance of the reward distribution is known to us and hence, we don't put a prior over the distribution. Let's assume this variance to be σ_1^2. The posterior update after time step t will be given by

$$q(a) \sim \mathcal{N}(\mu, \sigma^2)$$

  where

$$\sigma^2 = \left(\frac{1}{\sigma_0^2} + \frac{N_t(a)}{\sigma_1^2}\right)^{-1}$$
$$\mu = \sigma^2 \left(\frac{\mu_0}{\sigma_0^2} + \frac{N_t(a)Q_t(a)}{\sigma_1^2}\right)$$

  Now, we sample a value from this posterior distribution for each of the arm and play the arm which has the highest sampled value.

5.1 K=2 arm Problem

We repeat all the experiments as we did for the K=2 Bernoulli reward distribution. The following algorithms were used for comparison:

- ϵ-Greedy with $\epsilon = 0.3$.
- Variable ϵ-Greedy where ϵ decreases according to the schedule defined in (2) with $C = 10$.
- Softmax with $\tau = 0.01$ and $\tau = 100$.
- Variable τ-Softmax where τ decreases according to the schedule defined in (1) with $\zeta = 10$.
- UCB1-Normal with $C = 10$.
- Thompson Sampling Algorithm with Normal Prior Distribution
- Reinforce Algorithm with and without baseline.

The results obtained are as follows:

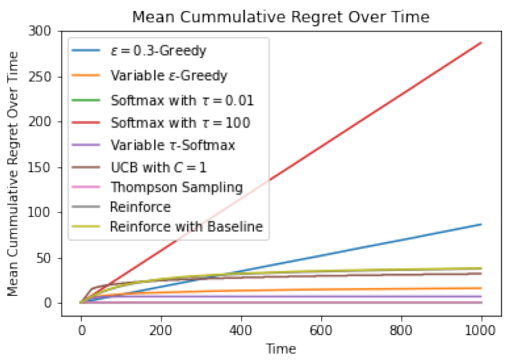

Figure 13: Mean Cummulative Regret Over Time

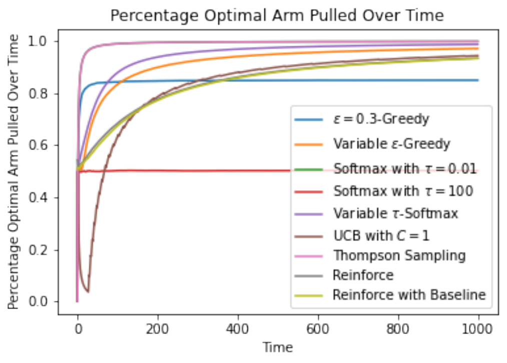

Figure 14: Percentage Optimal Arm Pulled Over Time

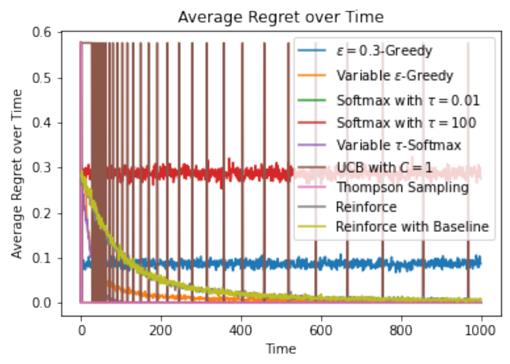

Figure 15: Average Regret over Time

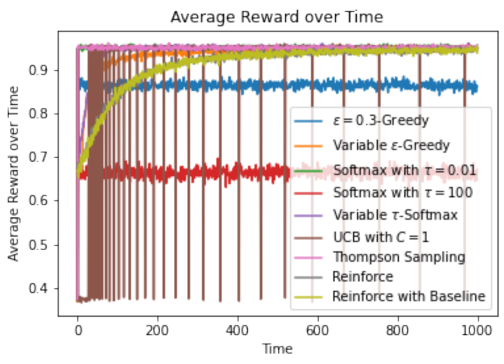

Figure 16: Average Reward over Time

Observations:

- Most of the patterns are the same as Bernoulli reward distributions. However, if we look at the plots for average regret over time and average reward over time, we find the graph for UCB1-Normal is initially flat and then has spikes. This can be attributed to the way the UCB1-Normal algorithm is designed. In UCB1-Normal algorithm, we give preference to the arm which has been played less than $\lceil 8logt \rceil$ times. Therefore, initially in the beginning, we can always find such an arm and we have to play that until that condition is satisfied. This makes the graphs flat in the beginning. However, one thing to note is that after this point also we are not guaranteed that this condition will never become satisfied again. For example, we may reach to a point where we may not have played the sub-optimal arm for a long time and hence, due to this under-representation of that arm we may have to play that arm again. Due to this reason, we observe spikes in the later parts of our graph.

5.2 K=5 arm Problem

We repeat all the experiments as we did for the K=2 Bernoulli reward distribution. The following plots compare the performance of each of these algorithms:

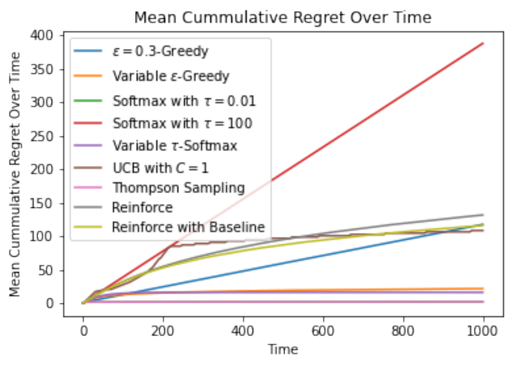

Figure 17: Mean Cummulative Regret Over Time

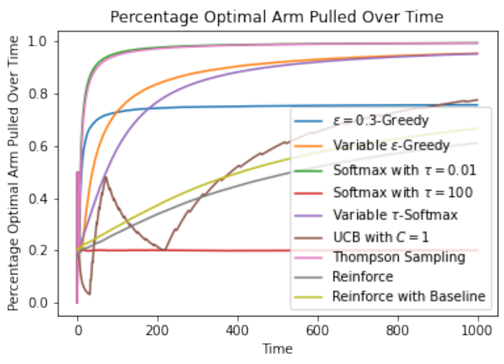

Figure 18: Percentage Optimal Arm Pulled Over Time

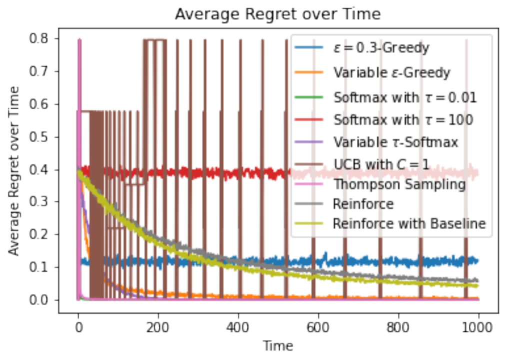

Figure 19: Average Regret over Time

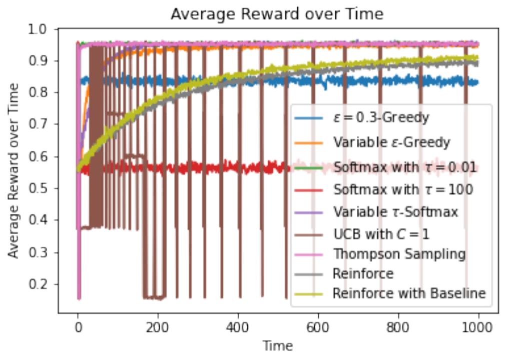

Figure 20: Average Reward over Time

Since the graphs follow the same pattern as $K = 2$ case we defer our discussion to section 5.4 where we compare the performance of the algorithms for different number of arms.

5.3 K=10 arm Problem

We repeat all the experiments as we did for the K=2 Bernoulli reward distribution. The following plots compare the performance of each of these algorithms:

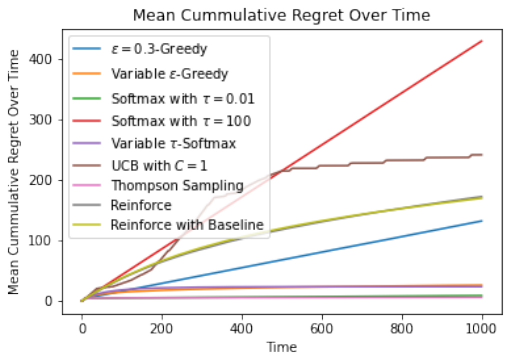

Figure 21: Mean Cummulative Regret Over Time

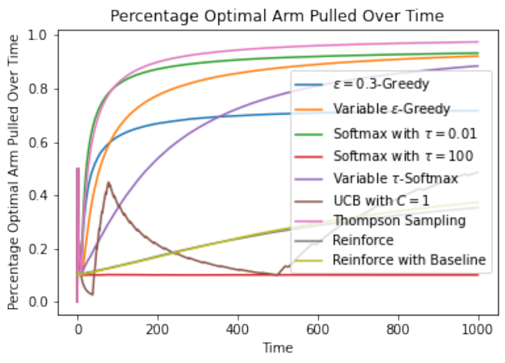

Figure 22: Percentage Optimal Arm Pulled Over Time

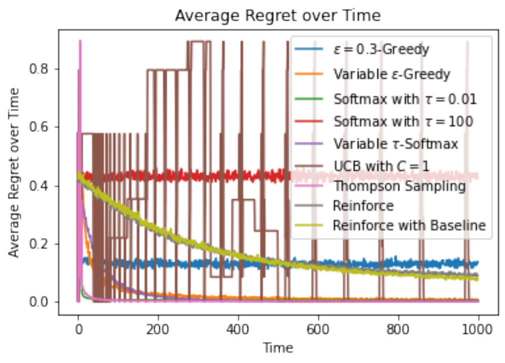

Figure 23: Average Regret over Time

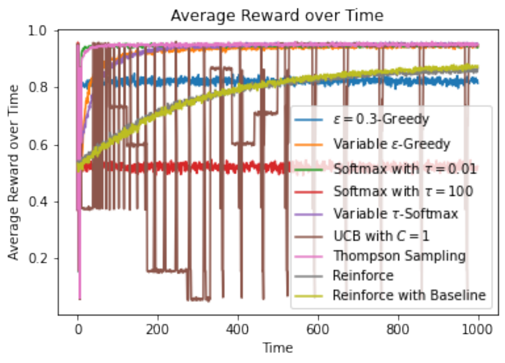

Figure 24: Average Reward over Time

Since the graphs follow the same pattern as $K = 2$ case we defer our discussion to section 5.4 where we compare the performance of the algorithms for different number of arms.

5.4 Comparison of algorithms for different number of arms

The following table lists the cummalative regret at the end of 1000 plays averaged over 1000 runs:

Algorithms	**K=2 Arms**	**K=5 Arms**	**K=10 Arms**
ϵ-Greedy ($\epsilon = 0.3$)	86.55	117.83	132.52
Variable ϵ-Greedy	16.25	22.07	26.60
Softmax ($\tau = 0.01$)	0.57	2.27	9.22
Softmax ($\tau = 100$)	286.79	387.75	429.78
Variable τ-Softmax	7.07	16.58	24.02
UCB ($C = 1$)	32.26	108.73	241.97
Thompson Sampling	0.58	2.46	6.31
Reinforce Algorithm (without baseline)	37.72	131.88	172.72
Reinforce Algorithm (with baseline)	38.30	116.30	170.23

Table 2: Comparison of cummalative regret of different algorithms for different number of arms for 1000 plays averaged over 1000 runs.

As we observed with the Bernoulli reward distribution that as we increase the number of arms in the Bandit, the performance of all the algorithm decreases considerably. Similarly, the relative ordering between the performance of the algorithms for a given number of arms remains the same. Note that all the experiments with different numbers of arms was done using the same hyperparameters which allows for fair comparison between algorithm.

6 An Interesting Case: Reinforce Algorithm with Baselines

In all the experiments performed above, we didn't find any considerable difference between reinforce algorithm with and without baselines. The reason was that our reward distribution had mean close to 0 and variance was also close to 0.01. We ran an experiment with $K = 10$ arm case with normal reward distribution with expected reward picked uniformly at random between [4, 5] and variance was fixed at 1.0. The results obtained for the experiment is shown in Figure 25, 26, 27 and 28.

This experiment clearly shows the advantage offered by the reinforce with baseline algorithm over the algorithm without baseline. This boost-up in performance can be attributed to the fact that when we subtract the baseline, we intuitively re-centering our data (rewards) at 0 and since stochastic gradient ascent algorithms tend to perform better when the data is centered at 0, hence, the algorithm with baselines is able to adapt to the new reward levels whereas the one without baseline suffers.

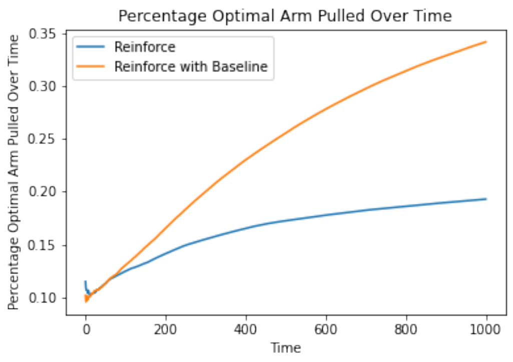

Figure 25: Percentage Optimal Arm Pulled Over Time

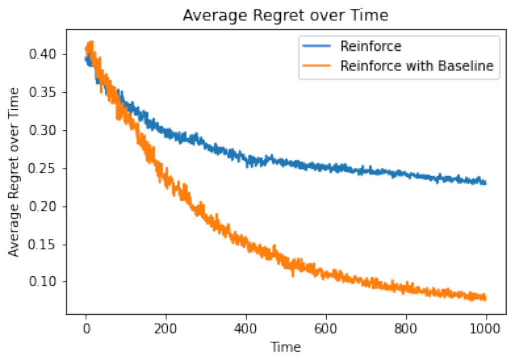

Figure 26: Average Regret over Time

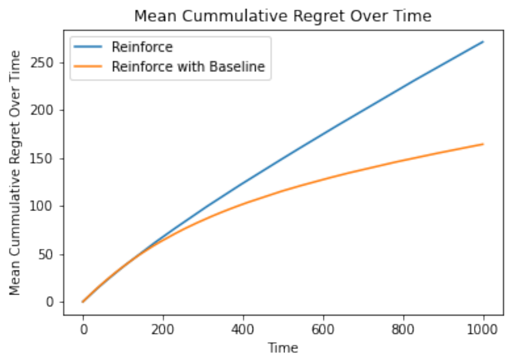

Figure 27: Mean Cummulative Regret Over Time

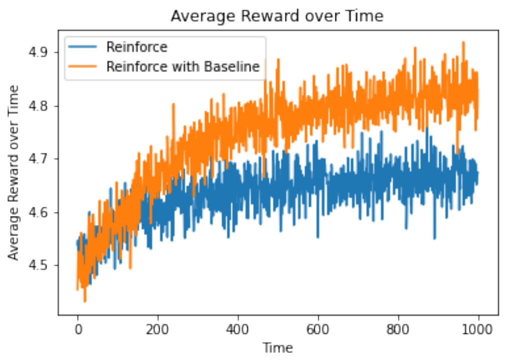

Figure 28: Average Reward over Time