# RL Lab 2

Harsh Raj (180010017)

Akhilesh Bharadwaj (180010009)

**Note**: Code for all questions (including simulations etc) is attached in the submitted zip. Artifacts including the images generated are also attached in the zip. Please make sure to install dependencies mentioned in the requirements.txt before running any submitted code.

```
# Install dependencies
pip3 install -r requirements.txt

# run codes corresponding to each question
python3 runner.py

# change hyperparameters:
# either change manually from conf/configs.yaml for each ques
# or change dynamically from command line
python3 runner.py seed=5
# config management is done via hydra. Read more
https://github.com/facebookresearch/hydra
```

All the charts can be accessed online on this wandb repo.

## 1. Bernoulli Distribution of underlying rewards for each arm:

Each Algorithm is given 5_000 chances per episode. In the case of Bernoulli distribution, each arm has an underlying distribution defined by p=1/(arm_index+1), index starts from 1.

- Epsilon Greedy:
    - Probability of randomly selecting an action: 0.3

- Softmax:
    - Initial Temperature: 1_000
    - Decay factor: 0.9, ie. after each time step, `temp = temp * decay_factor`
    - `mean / temp` ratio is clipped to be in range: 0.001 - 700. This is to ensure that the softmax does not shoot up to infinity

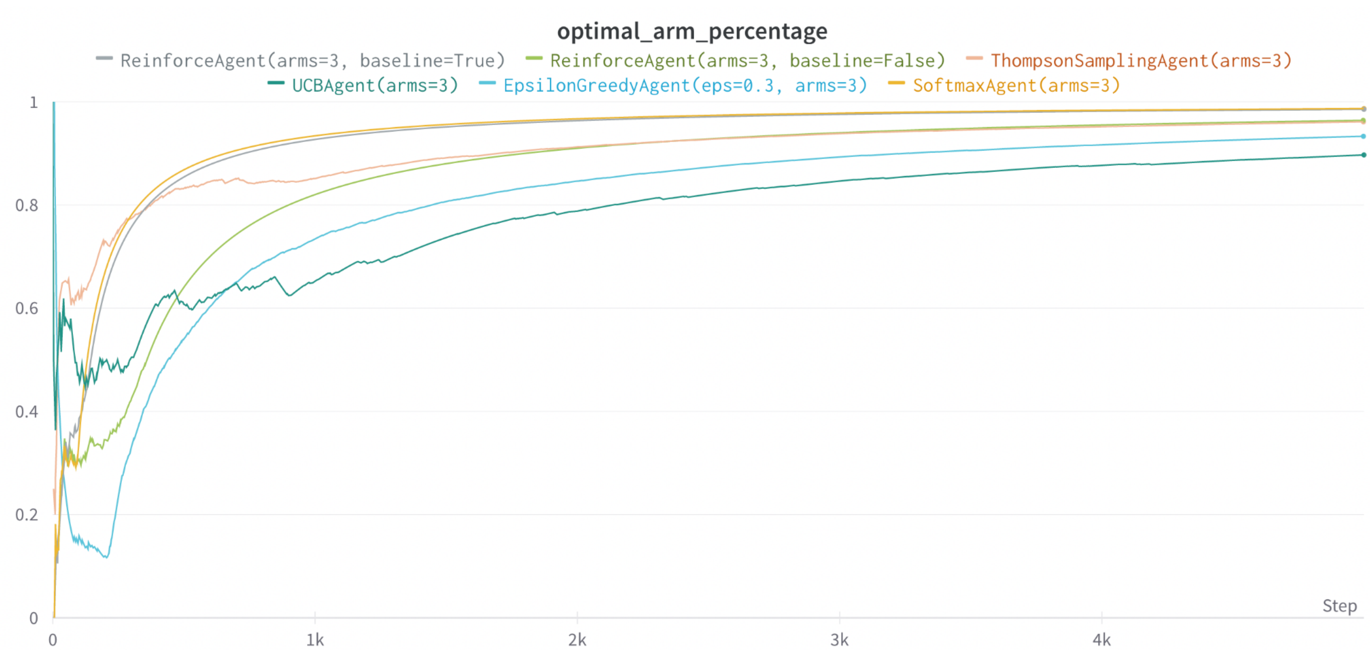
- UCB:
    - All default arguments.

- Thompson Sampling:
    - All default arguments.

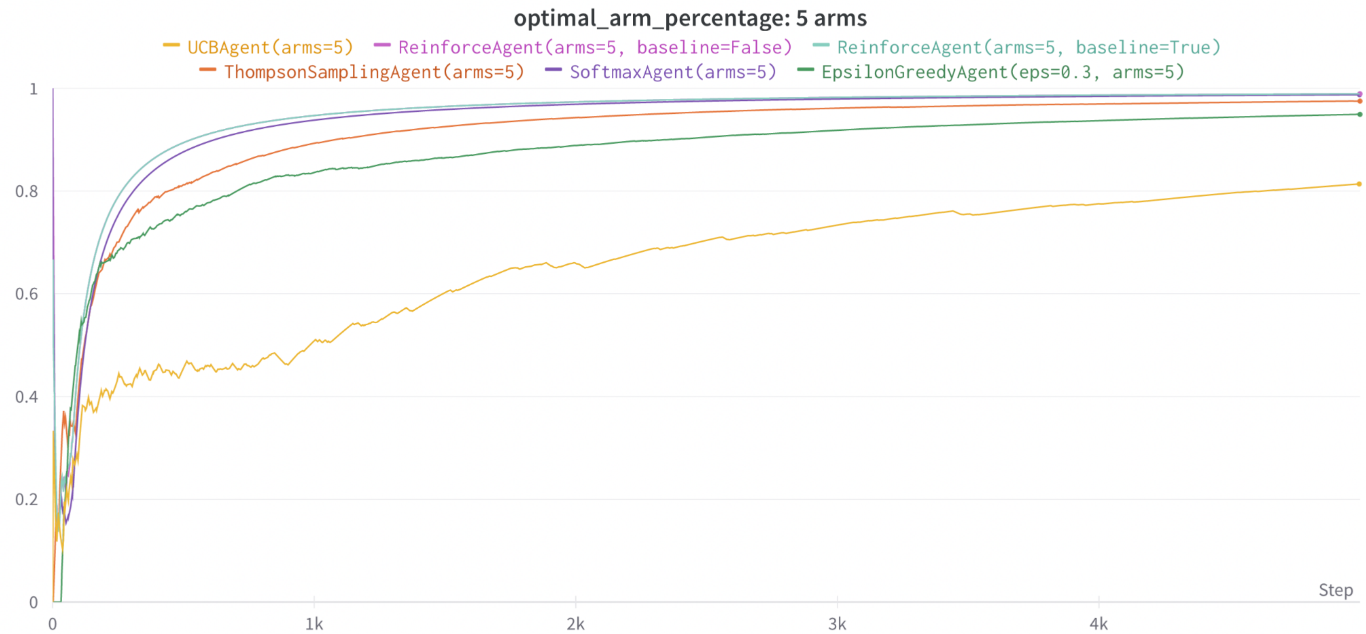
- Reinforce:
    - We experiment with alpha=0.8 and beta=0.3,

```
running_mean_reward = (1-alpha) * running_mean_reward + alpha * reward
preference = preference + beta * (reward - (average_reward if using_baseline else 0))
```

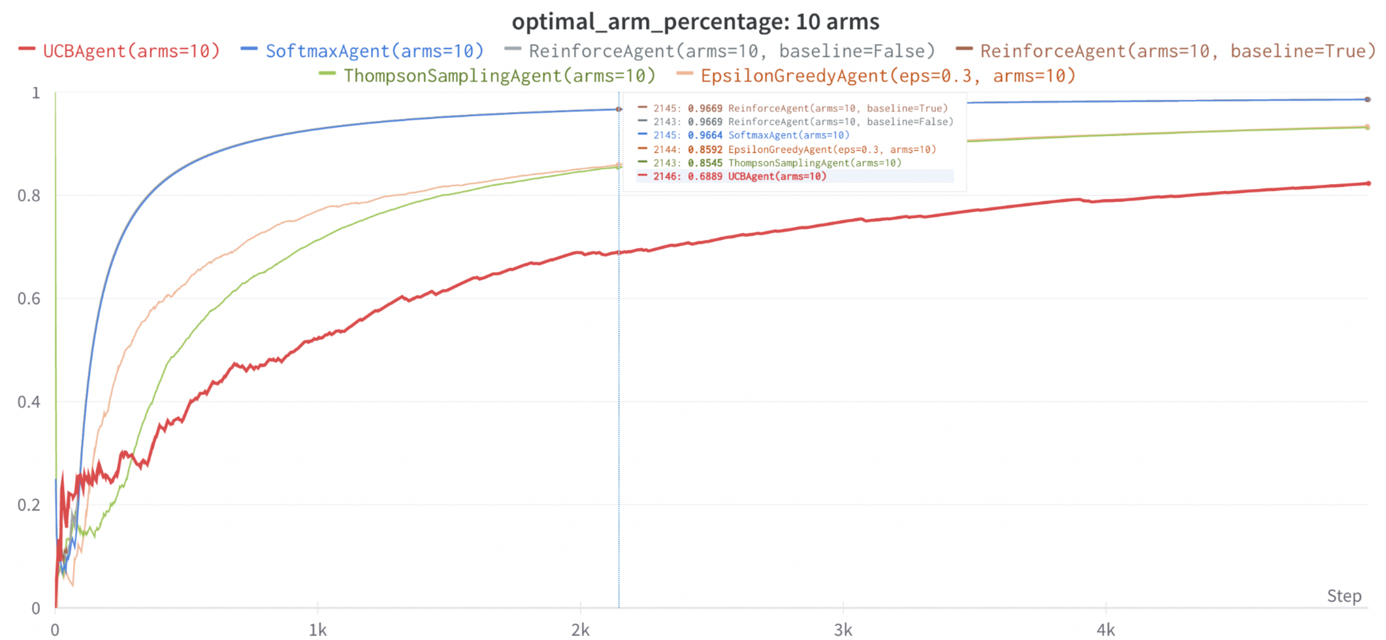
    - The preference of each arm is clipped in range: 0 - 200. This is to ensure that the softmax does not shoot up to infinity.
    - For baseline, we use the running mean reward.



Ratio of optimal arm hits for different algorithms, for 3-arm Bernoulli Bandit




Ratio of optimal arm hits for different algorithms, for 5-arm Bernoulli Bandit



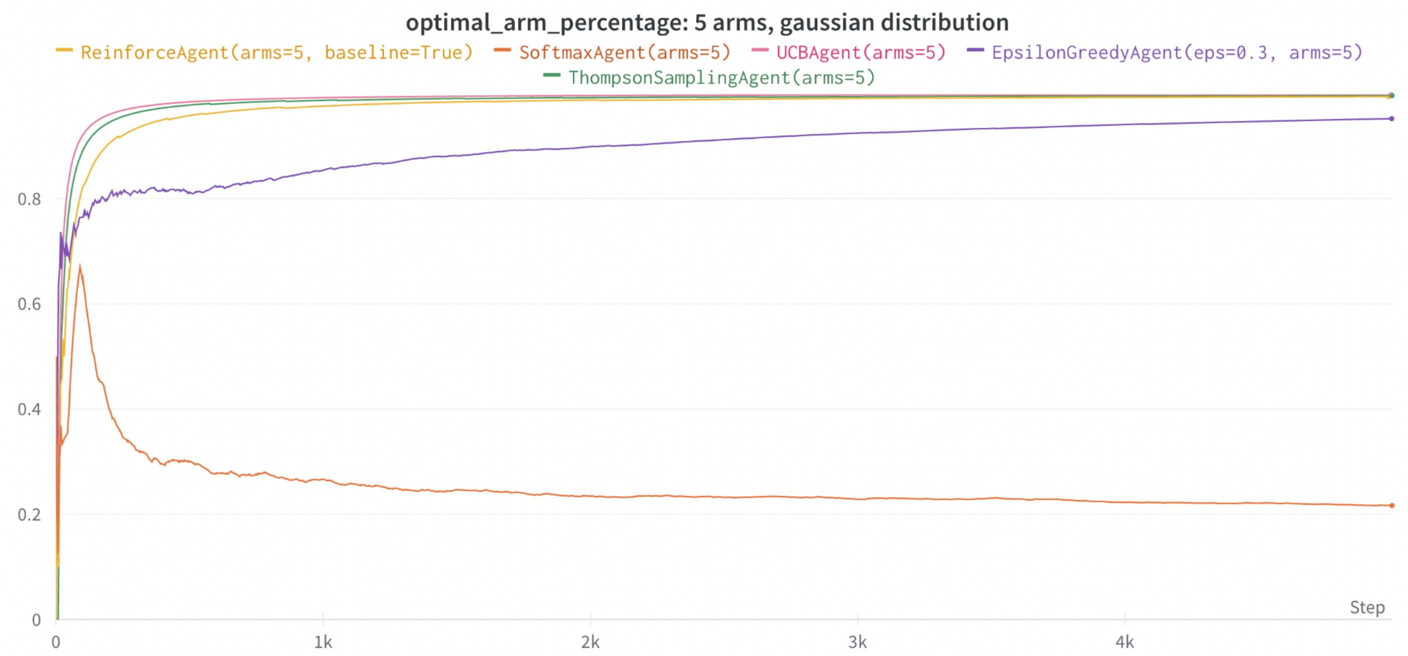

Ratio of optimal arm hits for different algorithms, for 10-arm Bernoulli Bandit
Note that Reinforce and Softmax perform equally well in this case

## 2. Gaussian Distribution of underlying rewards for each arm:

Each Algorithm is given 5_000 chances per episode. The underlying Gaussian reward distributions were initialized with mean=10 * arm_index, and std=arm_index * 2, with arm_index starting from 1.

To do:

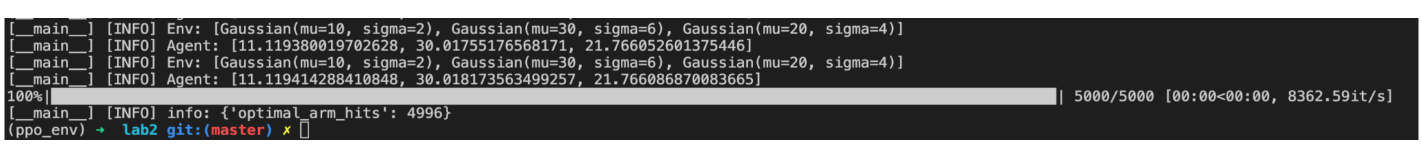
- Fix Reinforce without baseline for Gaussian




### Thompson Sampling:

The initial statistics about the means are taken as mean=0 and std=10000. This gives a very spread gaussian distribution, stating that we are not very sure about the underlying distribution and so we assume it to be similar to a uniform distribution by taking a high value of std.
We use the 2nd update rule mentioned in the bayesNormal.pdf to update the empirical mean and standard deviation of the arms.

### UCB:



```
[__main__] [INFO] Env: [Gaussian(mu=10, sigma=2), Gaussian(mu=30, sigma=6), Gaussian(mu=20, sigma=4)]
[__main__] [INFO] Agent: [11.119380019702628, 30.01755176568171, 21.766052601375446]
[__main__] [INFO] Env: [Gaussian(mu=10, sigma=2), Gaussian(mu=30, sigma=6), Gaussian(mu=20, sigma=4)]
[__main__] [INFO] Agent: [11.119414288410848, 30.018173563499257, 21.766086870083665]
100%|████████████████████████████████████████| 5000/5000 [00:00<00:00, 8362.59it/s]
[__main__] [INFO] info: {'optimal_arm_hits': 4996}
(ppo_env) → lab2 git:(master) ✗
```

UCB has been performing quite well. Identifying the underlying mean pretty well, and hitting optimal arms quite high.

## Epsilon Greedy

```
[__main__] [INFO] Env: [Gaussian(mu=10, sigma=2), Gaussian(mu=30, sigma=6), Gaussian(mu=20, sigma=4)]
[__main__] [INFO] Agent: [9.921916630594954, 30.109302180858304, 20.15952771983299]
[__main__] [INFO] Env: [Gaussian(mu=10, sigma=2), Gaussian(mu=30, sigma=6), Gaussian(mu=20, sigma=4)]
[__main__] [INFO] Agent: [9.921916630594954, 30.10982391075628, 20.15952771983299]
100%|████████████████████████████████████████| 5000/5000 [00:00<00:00, 5395.58it/s]
[__main__] [INFO] info: {'optimal_arm_hits': 4783}
(ppo_env) → lab2 git:(master) ✗
```

Epsilon Greedy Agent also performs reasonably well over here.

## Softmax

```
[__main__] [INFO] Env: [Gaussian(mu=10, sigma=2), Gaussian(mu=30, sigma=6), Gaussian(mu=20, sigma=4)]
[__main__] [INFO] Agent: [10.059498169155235, 29.842792037728923, 20.07546047975044]
[__main__] [INFO] Env: [Gaussian(mu=10, sigma=2), Gaussian(mu=30, sigma=6), Gaussian(mu=20, sigma=4)]
[__main__] [INFO] Agent: [10.059465323025005, 29.842792037728923, 20.07546047975044]
100%|████████████████████████████████████████| 5000/5000 [00:01<00:00, 4624.21it/s]
[__main__] [INFO] info: {'optimal_arm_hits': 1757}
(ppo_env) → lab2 git:(master) ✗
```

Though softmax is able to capture the mean of underlying reward distribution, it has quite high regret in this case.

## Reinforce:

- Reinforce with baseline, seems to be quite sensitive to hyperparameters for Gaussian distribution. We use beta=0.2, alpha=0.9 for the 5 arm Gaussian Multi Arm bandit case.