CS780: Deep Reinforcement Learning

Assignment #1

Name: Divyaksh Shukla
Roll NO.: 231110603

Solution to Problem 1: Multi-armed Bandits

1. Created the environment using Gymnasium library. The environment is a 2-armed bernoulli bandit with some stochasticity. The environment characteristics are taken from a config file which specify the α and β values. I tested out with different values of α and β $[(0,0),(0,1),(1,0),(1,1),(0.5,0.5)]$ and the environment is working as expected. The agent recieves a reward only if it takes an action which takes it correctly in the direction of movement. That is if the the agent moves 'left' and lands in state 1 or moves 'right' and lands in state 2, only then it gets a positive reward.

2. I created a similar environment with 10-arms just like the above. But in this case the environment is not stochastic. The action and state-transitions are deterministic. The expected reward for each action $q_*(s,a)$ is sampled from a standard normal distribution ($\mathcal{N}(0,1)$). The agent then receives a reward from a normal distribution with mean $q_*(s,a)$ and variance 1. The agent has to then learn the optimal action to take in each state, by taking actions and observing the rewards.

3. I created 6 types of bandit agents following different strategies to solve the bandit problem. The agents are:

   (a) Greedy Agent: This agent always takes the action with the highest estimated value. It does not explore the environment.

   (b) Epsilon-Greedy Agent: This agent takes the action with the highest estimated value with probability $1-\epsilon$ and takes a random action with probability ϵ.

   (c) Decaying Epsilon-Greedy Agent: This agent is similar to the epsilon-greedy agent, but the value of ϵ decays "linearly" $\epsilon = max(0, \epsilon_0 - decay\_rate * episode)$ or "exponentially" $\epsilon = \epsilon_0 e^{-decay\_rate * episode}$ with time.

   (d) Softmax Agent: This agent takes actions with probability proportional to the exponential of the estimated value of the action. The agent explores the environment by taking actions by choosing from the below distribution

   $$\pi(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_b e^{Q(s,b)/\tau}} \tag{1}$$

   (e) UCB Agent: This agent takes actions by choosing the action with the highest upper confidence bound. The upper confidence bound is calculated as $Q(s,a) + c\sqrt{\frac{\ln t}{N(s,a)}}$ where c is a constant and $N(s,a)$ is the number of times the action a has been taken in state s. The action is taken by taking the argmax of the upper confidence bound.

4. Created 50 different bandit problems for 2-armed Bernoulli Bandit with α and β values chosen from uniform distribution $\mathcal{U}(0,1)$. The agents were then tested on these bandit problems. The agents were tested for 1000 episodes and the average reward, average regret and optimal action percentage was calculated. The results are shown in the plots below (Figure 1, 3, 5).

5. Created 50 different bandit problems for 10-armed Gaussain Bandit with $q_*(s,a)$ values chosen from standard normal distribution $\mathcal{N}(0,1)$ and then the agent receives a reward from $\mathcal{N}(q_*(s,a),1) \forall a \in A$. The agents were then tested on these bandit problems for 1000 episodes and the average reward, average regret and optimal action percentage was calculated. The results are shown in the plots below (Figure 2, 4, 6).

6. Created a plot of the average regret of the agents for the 2-armed Bernoulli Bandit.

7. Created a plot of the average regret of the agents for the 10-armed Gaussian Bandit.

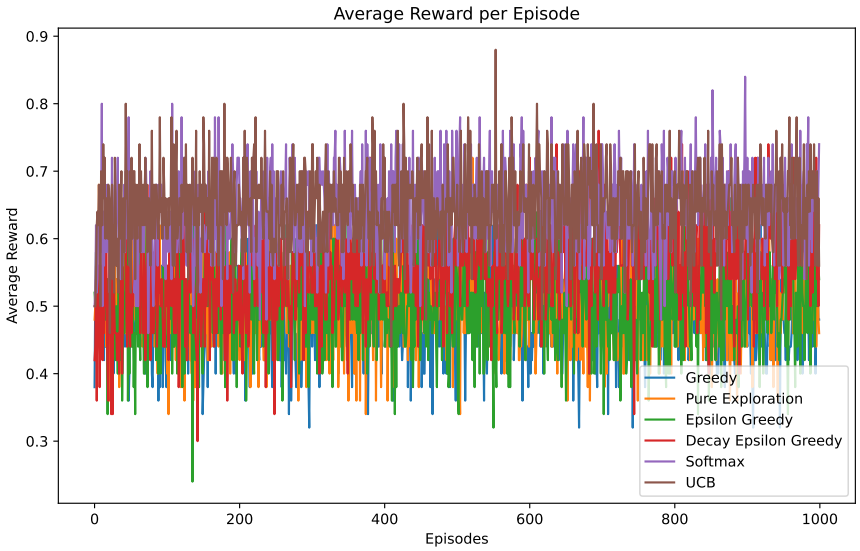

Figure 1: Average reward of 2-armed Bernoulli Bandit

8. This is the same as question 4.
9. Plotting the optimal action percentage of each agent for the 2-armed Bernoulli Bandit.
10. Plotting the optimal action percentage of each agent for the 10-armed Gaussian Bandit.

Solution to Problem 2: MC Estimates and TD Learning

I started by implementing the `RandomWalk-v0` environment by subclassing the `gym.Env` class. The environment is a simple random walk with 7 states. The agent starts at the middle state and only moves left following the "go-left" policy. The agent terminates actions and receives a reward of +1 if it reaches the rightmost state and a reward of 0 if it reaches the leftmost state. The agent receives a reward of 0 on non-terminal states. The environment is completely stochastic and the agent has a probability of 0.5 of moving left and right. The environment is working as expected, based on pygame observations and trajectory anaylsis.

In all the below test cases and scenarios a seed of 21 is used. I have also added `CONFIG` dictionary at any code block which is producing an output to be able to change values and observe outputs.

1. I created a function `generate_episode_trajectory(environment, config=None)` which takes in the `environment` and a `config` dictionary and returns a list of tuples of the form (s, a, r, s'). I tested the function with the `RandomWalk-v0` environment and the trajectory is as expected. The trajectory is shown in the code block below.

```
Episode Trajectory: [(3, 0, 0.0, 2), (2, 0, 0.0, 3), (3, 0, 0.0, 4),
(4, 0, 0.0, 3), (3, 0, 0.0, 2), (2, 0, 0.0, 3), (3, 0, 0.0, 4),
(4, 0, 0.0, 5), (5, 0, 1.0, 6)]
```

2. I wrote a routine to decay the step size at every step/episode: `decay_step_size(initial_value, final_value, episode, max_episode, decay_type, decay_stop=None)`. This routine returns $\alpha(e)$ which starts at `initial_value` and ends at `final_value` based on the `decay_type`. I tested the function and the results are present in Figure 7 and Figure 8. The step size is decaying as expected.
3. I implemented a routine to run Monte Carlo prediction for first-visit and every-visit scenarios. This is again controlled by `CONFIG` parameter. Below is an output from the routine for first-visit Monte Carlo prediction. The output consists of the state values, state-values for all episodes and the trajectory of the agent.

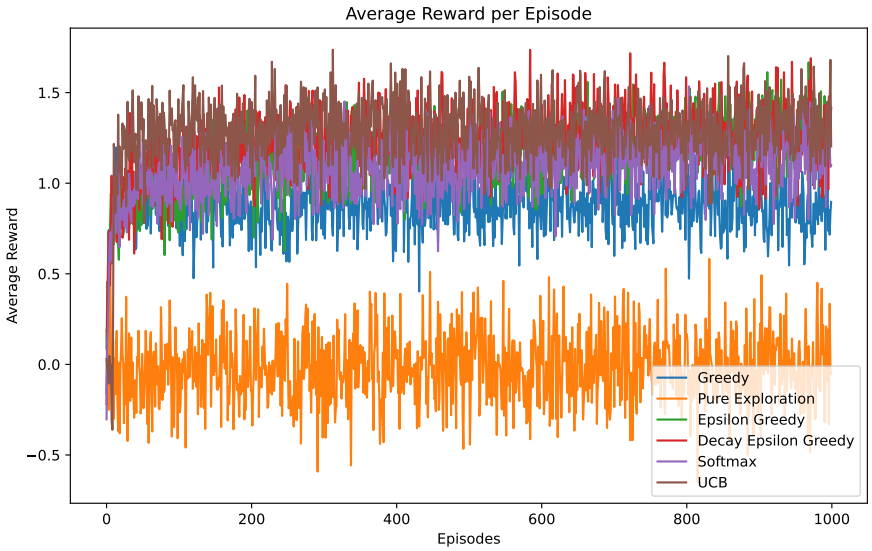

Figure 2: Average reward of 10-armed Gaussian Bandit

```
(array([0., 0., 0., 0., 0., 0., 0.]),
array([[0., 0., 0., 0., 0., 0., 0.]]),
[(3, 0, 0.0, 2), (2, 0, 0.0, 1), (1, 0, 0.0, 0)])
```

4. I implemented a routine to run Temporal difference prediction. This routine like the above not only does prediction but also computes the state-values. This is again controlled by `CONFIG` parameter. Below is an output from the routine for TD prediction routine. The output consists of the state values, state-values for all episodes and the trajectory of the agent.

   ```
   (array([0. , 0. , 0. , 0. , 0. , 0.1, 0. ]),
   array([[0. , 0. , 0. , 0. , 0. , 0.1, 0. ]]),
   [(3, 0, 0.0, 2), (2, 0, 0.0, 1), (1, 0, 0.0, 0)])
   ```

5. I wrote a routine to plot the state-values across episodes and *log*(episodes) to answer multiple questions below. `plot_episode_values(V_r, image_name, config=None)` takes the state-values across episodes along with an `image_name` to save the image and a `config` for additional information.

   I plotted the state-values for each state for First-Visit MC in Figure 9 and 10. We can see from the from the figures that there is initially very high variance in the state-values but as the number of episodes increase the variance decreases and the state-values converge to the true values. The variance is higher in the default case as the step size is too high and the decay is too slow. The variance is lower in the tuned case as the step size is lower and the final value is also very low. On the log-scale plot, we can see that the state-values slowly rise towards the true values.

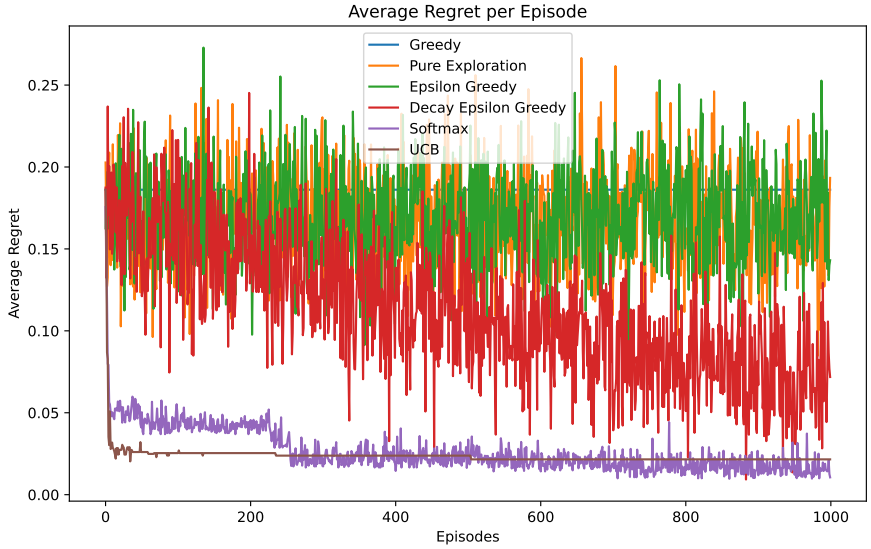

Figure 3: Average regret of 2-armed Bernoulli Bandit

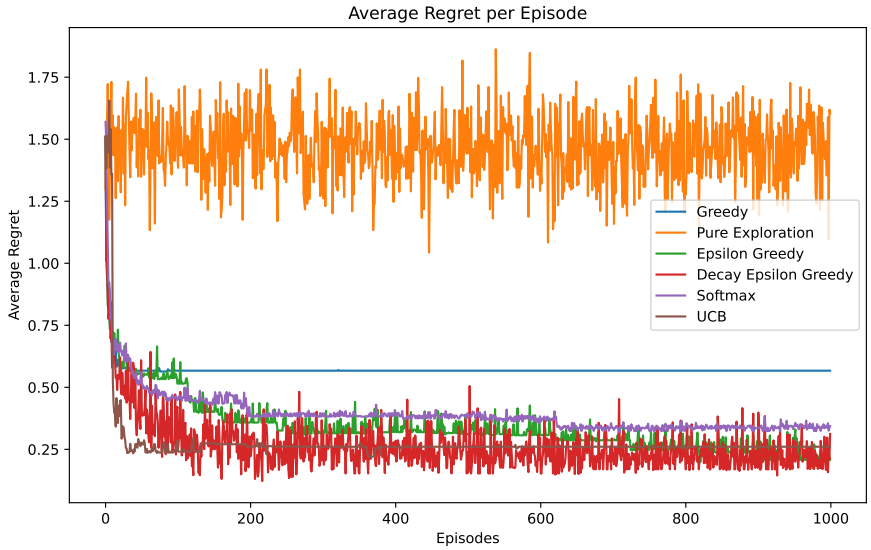

Figure 4: Average regret of 10-armed Gaussian Bandit

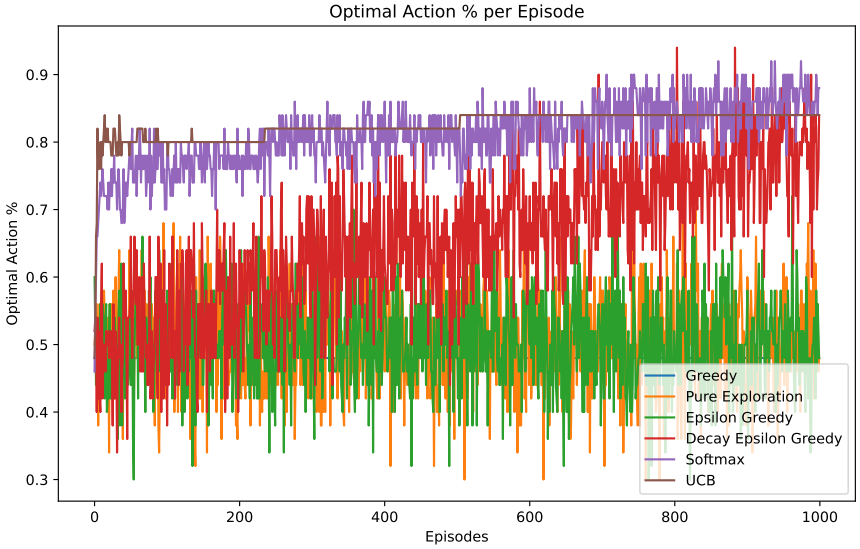

Figure 5: Optimal Action % of 2-armed Bernoulli Bandit

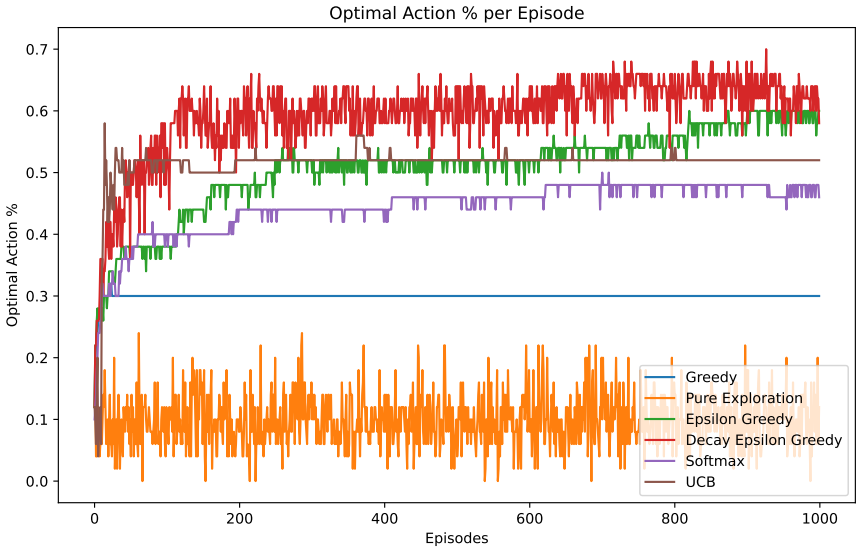

Figure 6: Optimal Action % of 10-armed Gaussian Bandit

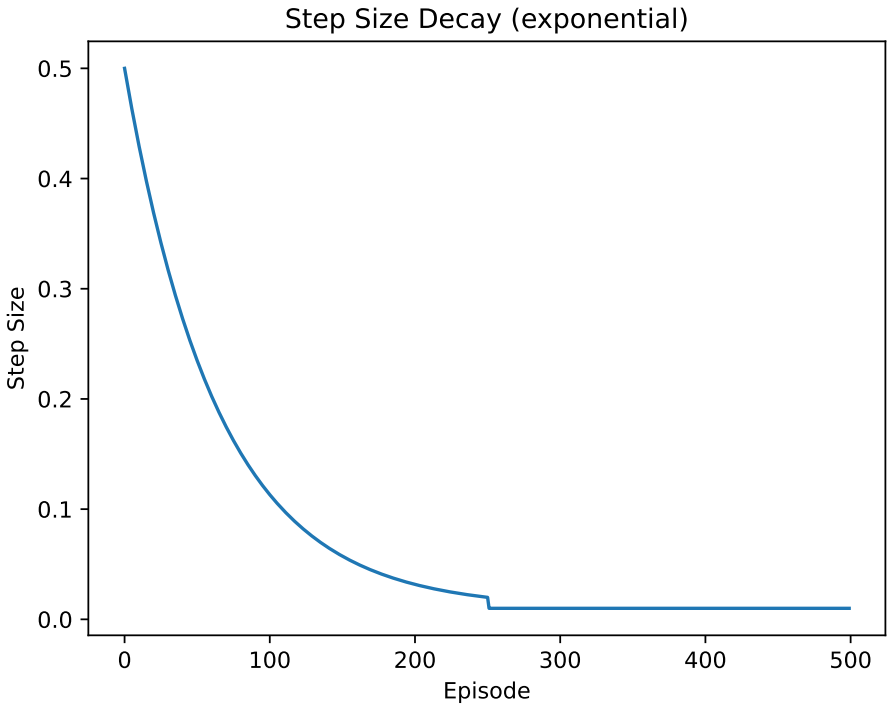

Figure 7: Exponential decay of step size

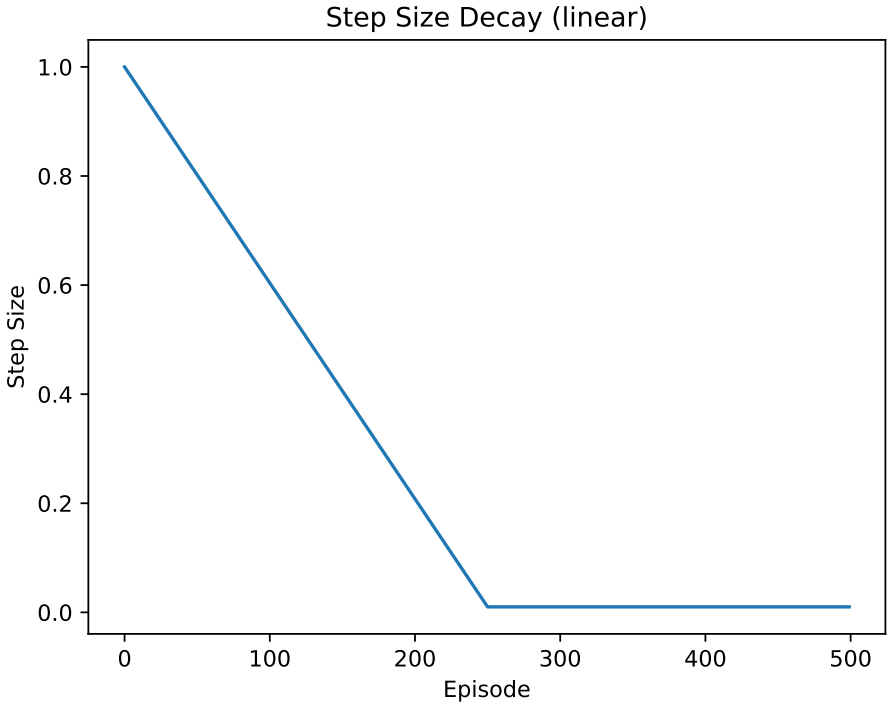

Figure 8: Exponential decay of step size

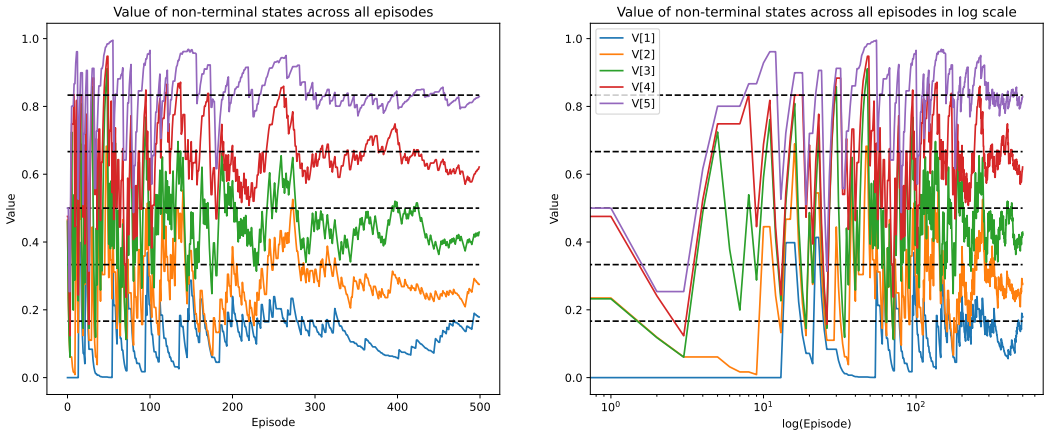

Figure 9: FVMC with exponential decay of step size

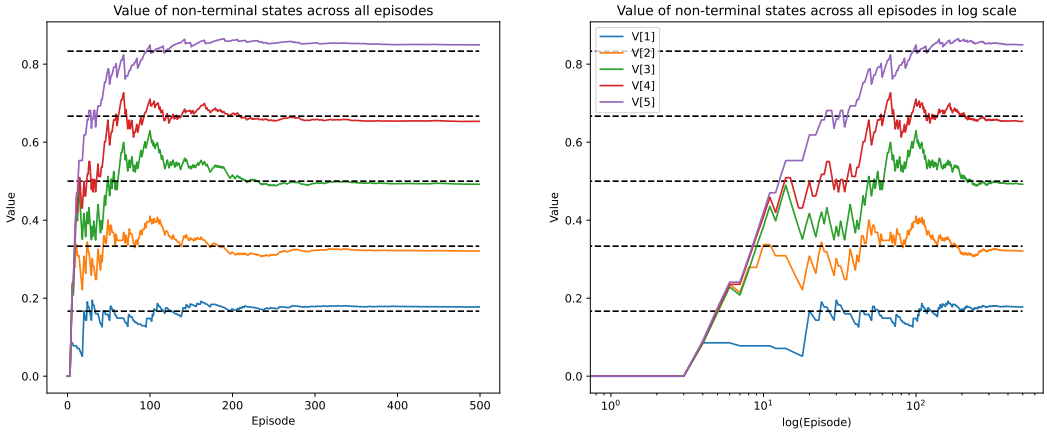

Figure 10: FVMC with exponential decay and tuned hyperparameters