Amrita Vishwa Vidyapeetham
Amrita School of Computing, Bangalore
Department of Computer Science and Engineering
22AIE401 – Reinforcement Learning
Lab Worksheet - 2
Multi-Armed Bandits

Note: All the excercise problems are in the GitHub repository: https://github.com/NippunKumaar/22AIE401-Reinforcement-Learning.git

1. ε-Greedy Bandit Implementation

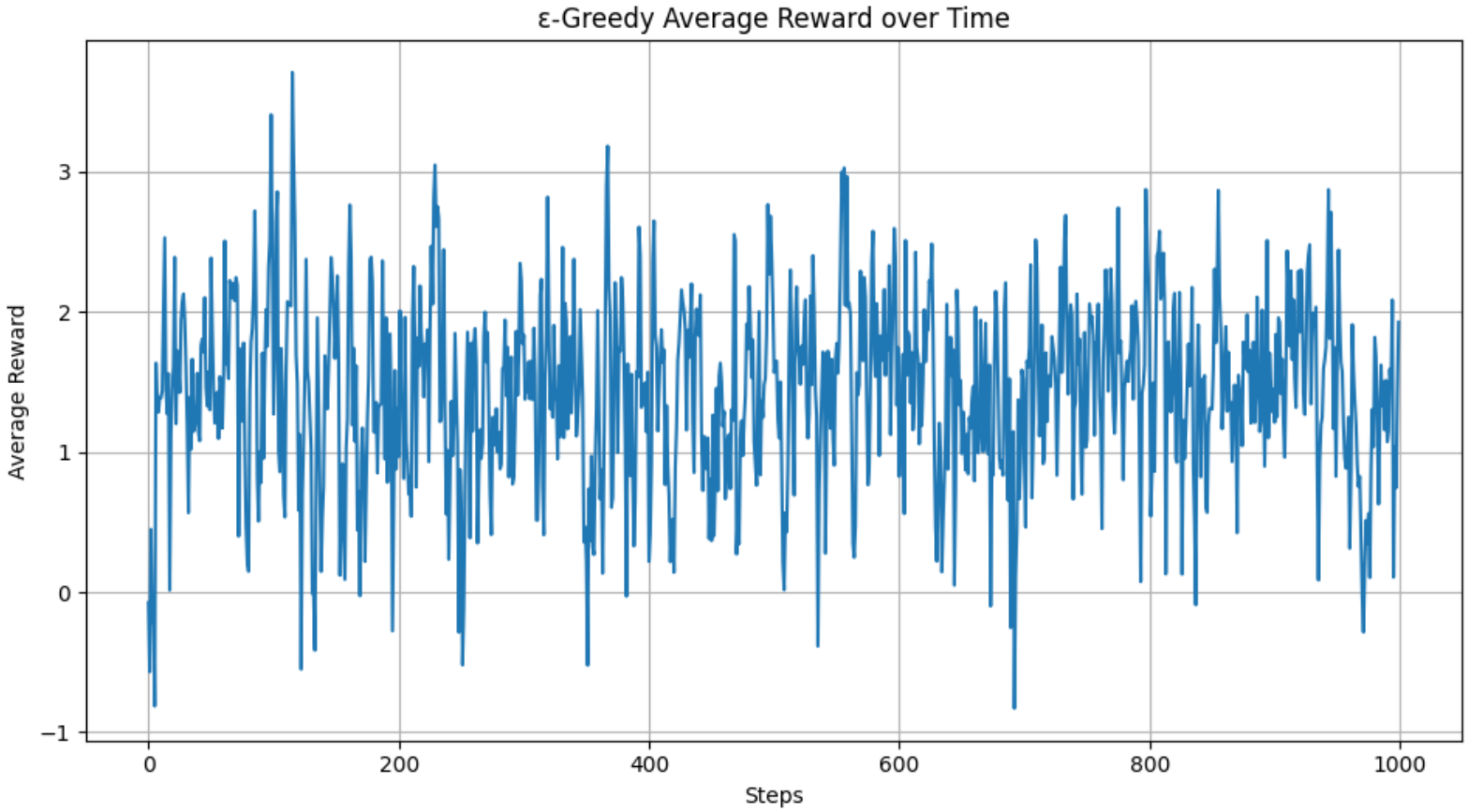
You are given a bandit environment with 10 arms. Each arm provides a reward sampled from a stationary Gaussian distribution with unknown mean and unit variance. Implement an agent using the ε-Greedy strategy (ε = 0.1) to interact with the bandit over 1000 time steps and plot the average reward.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)  # reproducibility

n_arms = 10
steps = 1000
epsilon = 0.1

true_values = np.random.normal(0, 1, n_arms)
Q = np.zeros(n_arms)
action_counts = np.zeros(n_arms)
avg_rewards = []

for t in range(1, steps + 1):
    if np.random.rand() < epsilon:
        action = np.random.randint(n_arms)
    else:
        action = np.argmax(Q)

    reward = np.random.normal(true_values[action], 1)
    action_counts[action] += 1

    # Incremental Q update
    Q[action] += (1 / action_counts[action]) * (reward - Q[action])
    avg_rewards.append(np.mean(avg_rewards[-1:] + [reward]) if avg_rewards else reward)

# Convert counts to percentage
action_percentages = 100 * action_counts / steps

print("True Mean Rewards of Each Arm:\n", np.round(true_values, 3))
print("\nEstimated Q-values:\n", np.round(Q, 3))
print("\nAction Selection Percentage:\n", np.round(action_percentages, 1))
# Plotting
plt.plot(avg_rewards)
plt.xlabel("Steps")
plt.ylabel("Average Reward")
plt.title("ε-Greedy Average Reward over Time")
plt.grid()
plt.show()
```

Results:

```
True Mean Rewards of Each Arm:
 [ 0.497 -0.138  0.648  1.523 -0.234 -0.234  1.579  0.767 -0.469  0.543]

Estimated Q-values:
 [ 0.284 -0.195  0.263  1.584 -0.262 -0.137  1.544  1.066 -1.043  0.501]

Action Selection Percentage:
 [ 2.2  1.5  1.1 88.1  1.2  0.8  2.2  1.   1.3  0.6]
```



2. Softmax Bandit Implementation

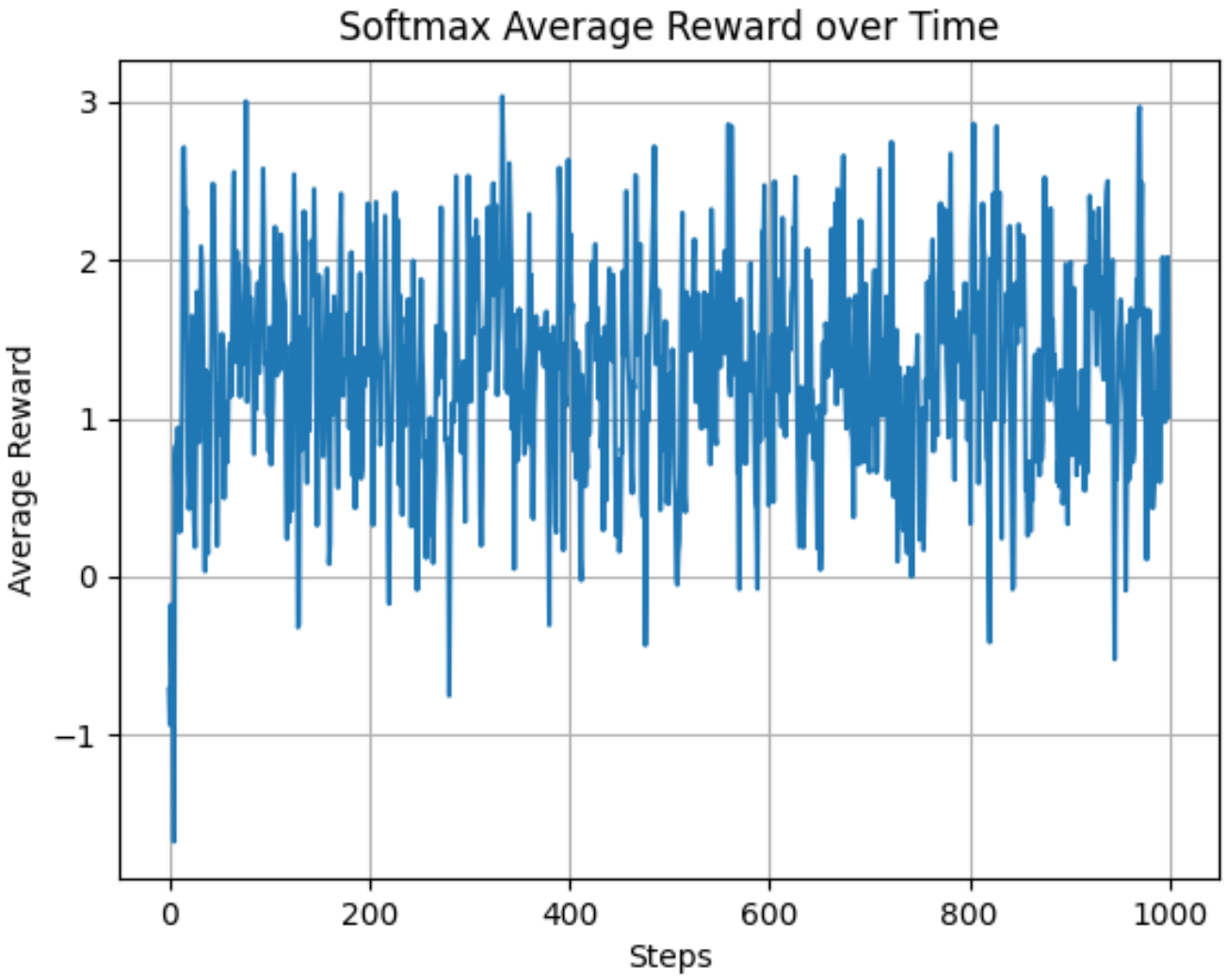
Now use the same 10-arm bandit environment but implement the Softmax strategy with a temperature parameter $\tau = 0.5$. Run the agent for 1000 steps, compare action selection probabilities, and plot the average reward vs. time.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)  # reproducibility

n_arms = 10
steps = 1000
epsilon = 0.1

true_values = np.random.normal(0, 1, n_arms)


def softmax(q_values, tau=0.5):
    exp_q = np.exp(q_values / tau)
    return exp_q / np.sum(exp_q)

Q = np.zeros(n_arms)
action_counts = np.zeros(n_arms)
avg_rewards = []

for t in range(1, steps + 1):
    probs = softmax(Q, tau=0.5)
    action = np.random.choice(np.arange(n_arms), p=probs)
    reward = np.random.normal(true_values[action], 1)
    action_counts[action] += 1

    Q[action] += (1 / action_counts[action]) * (reward - Q[action])
    avg_rewards.append(np.mean(avg_rewards[-1:] + [reward]) if
avg_rewards else reward)

action_percentages = 100 * action_counts / steps

print("True Mean Rewards of Each Arm:\n", np.round(true_values, 3))
print("\nEstimated Q-values:\n", np.round(Q, 3))
print("\nAction Selection Percentage:\n",
np.round(action_percentages, 1))

plt.plot(avg_rewards)
plt.xlabel("Steps")
plt.ylabel("Average Reward")
plt.title("Softmax Average Reward over Time")
plt.grid()
plt.show()
```

Results:

```
True Mean Rewards of Each Arm:
 [ 0.497 -0.138  0.648  1.523 -0.234 -0.234  1.579  0.767 -0.469  0.543]

Estimated Q-values:
 [ 0.615 -0.408  0.568  1.533 -0.404 -1.158  1.586  0.886 -2.609  0.488]

Action Selection Percentage:
 [ 6.   0.9  7.1 34.3  0.5  0.1 37.9 11.3  0.1  1.8]
```



Assignment

1. Personalized News Recommendation

Model a news website that shows one of 5 article categories (Tech, Sports, Business, Health, Travel). The reward is 1 if the user clicks, 0 otherwise.

- Implement ε-Greedy and Softmax strategies.
- Simulate user preferences with predefined means.
- Plot and compare action distributions and average rewards.

2. Ad Placement on a Website

You have 4 ads with different click-through rates (CTRs).

- Use ε-Greedy and Softmax to identify the best ad.
- Simulate for 2000 steps.
- Analyze which method converges faster and more accurately.

3. Smart Vacuum Cleaner – Spot Cleaning Strategy (Anchor Example)

Your smart vacuum cleaner operates in a room with **5 predefined dirty zones**. Each time it starts cleaning, it must choose **one spot** to begin with. The likelihood of dirt (and hence reward) in each zone varies.

- Model this as a **5-armed bandit** problem where:
    - Each arm represents a dirty zone (Zone A to Zone E).
    - The reward is +1 if the zone is actually dirty and cleaned successfully, else 0.
    - Each zone has a different true dirt probability (you can simulate these using a normal or Bernoulli distribution).

Your Tasks:

- Implement both **ε-Greedy** and **Softmax** strategies for 1000 episodes.
- Track and compare:
    - Estimated action-values,
    - Percentage of times each zone is selected,
    - Average reward over time.
- Which strategy learns to target the dirtiest zones more effectively?

4. LLM Integration Task

Use ChatGPT or any LLM to:

- Ask the LLM to critique their code (e.g., "How can I make my ε-Greedy implementation more efficient?")
- Ask the LLM to summarize differences between ε-Greedy and Softmax using their own simulation outputs
- Prompt examples:
- "Explain why my ε-Greedy agent chose suboptimal arms more often."
- "What could be the reason Softmax worked better in the pricing experiment?"