REPORT - REINFORCEMENT LEARNING ASSIGNMENT - 1

Team members -3

Contributions-

1. Anya Kalluri –SE22UCSE033

UCB Algorithm , question 7

2. Stuti Garg –SE22UCSE263

Policy Gradient, questions 1&2.

3. Lakshmi –SE22UCSE148

epsilon- greedy algorithm,outputs

Q1) In every contextual bandits, there must be randomness in the reward associated with an action. Based on your reading of what is discussed above, what are the sources of this noise?

In the ServerAllocationEnv setting, potential sources of noise in the reward signal include:

★ Fluctuations in Job Arrival Patterns: The number and type of incoming jobs can change unpredictably, affecting how well a given action (server allocation) matches the system's actual workload.

★ Task Processing Variability: Even with the same number of servers allocated, the time to complete jobs might vary due to differences in job complexity or unexpected delays.

★ External Load Spikes: Sudden surges in traffic (system anomalies) could temporarily change the reward landscape, causing instability in reward observations.

★ Priority Drift: The priority of jobs may shift dynamically, meaning a high-priority job might be downgraded or its urgency might change, impacting the reward signal for the same context-action pair.

★ Server Performance Fluctuations: Servers themselves might have variability in performance (like hardware degradation, memory leaks), adding noise to the environment's response.

★ Exploration-Induced Variability: When using strategies like ε-greedy or UCB, the agent deliberately tries suboptimal actions to explore, which naturally introduces noise in the observed rewards.

Q2) Based on your reading of sections 4 and 5, find features from the context that will be useful for training RL agent in the subsequent parts. Your answer should address:

- Features that can be considered are:

1. Job Priority: Higher-priority jobs might need more servers, so capturing priority is crucial.

2. Estimated Processing Time: Helps predict how many servers are required to meet deadlines.

3. Number of Incoming Jobs: More jobs likely require more servers, so this is a vital feature.

4. Current Server Load: Tracks how busy the servers are, which influences allocation decisions.

5. Historical Performance Metrics: Past reward signals or latency measurements can guide the agent's decisions

• How to deal with the fact that the context size is varying? This is important because most ML/DL models can't deal with varying input size.

- Since job queues might change in size dynamically, handling variable-length contexts is critical. To deal with the with the varying context size we can use the methods:

1. Padding/Truncation: Pad shorter contexts with zeros or truncate longer ones to a fixed length. This keeps the input size consistent for ML models.

2. Statistical Aggregation: Compress variable-length inputs using statistics like mean, variance, max, and min of features like job processing times or priorities.

3. Sliding Window Representation: Instead of considering all jobs, use a fixed-size sliding window over the most recent jobs to capture temporal information.

4. Learned Embeddings: Use an embedding layer in a neural network to project varying-length context vectors into a fixed-dimensional space.

5. Attention Mechanisms: If using a more advanced architecture, attention can help the model focus on the most important elements of a variable-length context dynamically.

By carefully selecting meaningful features and transforming variable-length contexts into fixed-size representations, the RL agent can learn more effectively while maintaining model compatibility.

• Can we reduce the number of features? This will eventually help you in training agents in the subsequent parts.

- Yes, we can reduce the features to improve training efficiency and decision-making. We can use the methods as follows:

1. Domain Knowledge Filtering: Keep only essential features like priority, processing time, and job count.

2. Feature Importance Scoring: Use simple models to rank and select the most impactful features.

3. Dimensionality Reduction Techniques: Apply PCA or autoencoders to compress feature space.

4. Clustering or Binning: Group similar jobs or bin continuous features to simplify inputs.

5. Incremental Feature Selection: Iteratively test feature subsets to find the optimal combination.

Q3) ε-greedy algorithm

```
Sequence(Tuple(Discrete(3, start=1), Discrete(3), Box(0.0, 1.0, (1,), float32), Box(0.0, 30.0, (1,), float32)), stack=False)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [127], in <cell line: 61>()
     58 observation_space = env.observation_space
     59 print(observation_space)
---> 61 LinGreedyPolicy(env)

Input In [127], in LinGreedyPolicy(env)
     32     action = np.random.randint(Nactions)   #Exploration
     33 else:
---> 34     action = np.argmax(np.matmul(theta,z))  #Exploitation
     35     #Taking the action
     36     next_obsv, reward, _, truncated, _ = env.step(action)

ValueError: matmul: Input operand 1 has a mismatch in its core dimension 0, with gufunc signature (n?,k),(k,m?)->(n?,m?) (si
ze 5 is different from 8)
```

Q6)

Not able to answer as codes did not get executed completely.

Q7(a): Python Code for Sampling an Action from a Gaussian Distribution

We need to sample an action ata_tat from a Gaussian policy with:

- Mean: θ1Tx\theta_1^T xθ1Tx
- Standard deviation: exp⁡(θ2Tx)\exp(\theta_2^T x)exp(θ2Tx)

```
import numpy as np


def sample_action(theta1, theta2, x):
    mean = np.dot(theta1.T, x)
    std_dev = np.exp(np.dot(theta2.T, x))
    action = np.random.normal(mean, std_dev)  # Sample from Gaussian
    return action
```

7(b)

Step 2

$\log \pi_\theta(a|s)$

Eq (8.5) $\nabla_\theta J(\theta) = E\left[\sum_{t=0}^{T} R_t \nabla_\theta \log \pi(a_t|x_t,\theta)\right]$

GD $\pi(a|x,\theta) = \frac{1}{\sqrt{2\pi\sigma_0^2(x)}} \exp\left(\frac{-(a-\mu_0(x))^2}{2\sigma_\theta^2(x)}\right)$

Take log

$\log \pi(a|x,\theta) = \frac{-(a-\theta_1^T x)^2}{2\exp(2\theta_2^T x)} - \theta_2^T x - \log\sqrt{2\pi}$

Step 3

$\nabla_\theta J(\theta) = E[R_t \nabla_\theta \log \pi(a|x,\theta)]$

wrt θ_1

$\nabla_{\theta_1} \log \pi(a|x,\theta) = \frac{(a-\theta_1^T x)}{\exp(2\theta_2^T x)} x$

gradient update for θ_1:

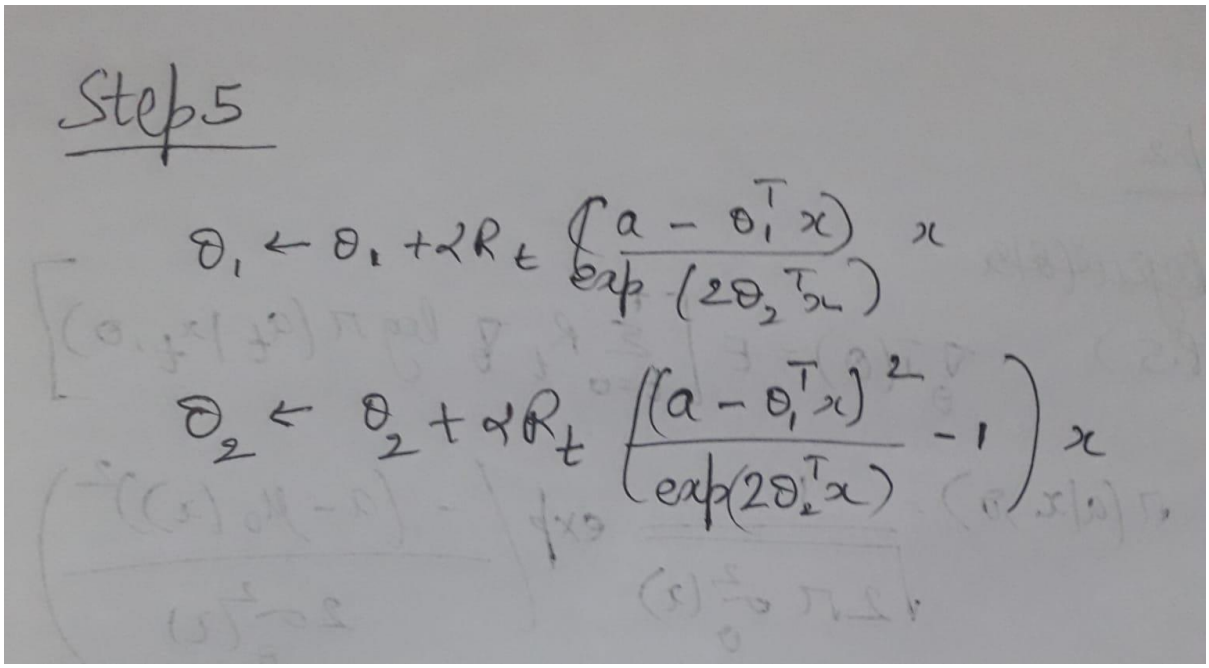
$\theta_1 \leftarrow \theta_1 + \alpha R_t \frac{(a-\theta_1^T x)}{\exp(2\theta_2^T x)} x$

Step 4

wrt θ_2 $\nabla_{\theta_2} \log \pi(a|x,\theta) = \left(\frac{(a-\theta_1^T x)^2}{\exp(2\theta_2^T x)} - 1\right) x$

gradient update for θ_2:

$\theta_2 \leftarrow \theta_1 + \alpha R_t \left(\frac{(a-\theta_1^T x)^2}{\exp(2\theta_2^T x)} - 1\right) x$



7(c) Pseudocode for Policy Gradient Algorithm

Initialize θ1, θ2 randomly

for each episode:

  Observe initial state s0

  for each time step t:

    Compute mean: μt = θ1^T xt

    Compute std deviation: σt = exp(θ2^T xt)

    Sample action at ~ N(μt, σt^2)

    Execute at, observe rt and next state st+1

    Store (xt, at, rt)

  Compute returns Rt = ∑ γ^k rt+k

  for each step t:

    Compute policy gradient updates:

      θ1 ← θ1 + α Rt * ((at - μt) / σt^2) * xt

$\theta 2 \leftarrow \theta 2 + \alpha\ Rt * (((at - \mu t)^2 / \sigma t^2) - 1) * xt$

Update θ1, θ2 using gradient ascent