CS4122: Reinforcement Learning
Programming Assignment 1

SE22UARI165 | SE22UARI123 | SE22UARI054

CONTRIBUTIONS of each team member:

SRI SPOORTHI VATTEM (SE22UARI165):

- Question 1
- Question 5 Policy Gradient (policy_gradient.py)
- Question 6 (tested for policy gradient)

JOTSNA SREE PAPPU (SE22UARI123):

- Question 4 UCB(lin_ucb.py)
- Question 6 (tested for UCB)
- Question 7 (b),(c)

GUDURI SRIKRUTH VARMA (SE22UARI054):

- Question 3 ε-greedy(lin_greedy.py)
- Question 2
- Question 6 (tested for ε-greedy)
- Question 7 (a)

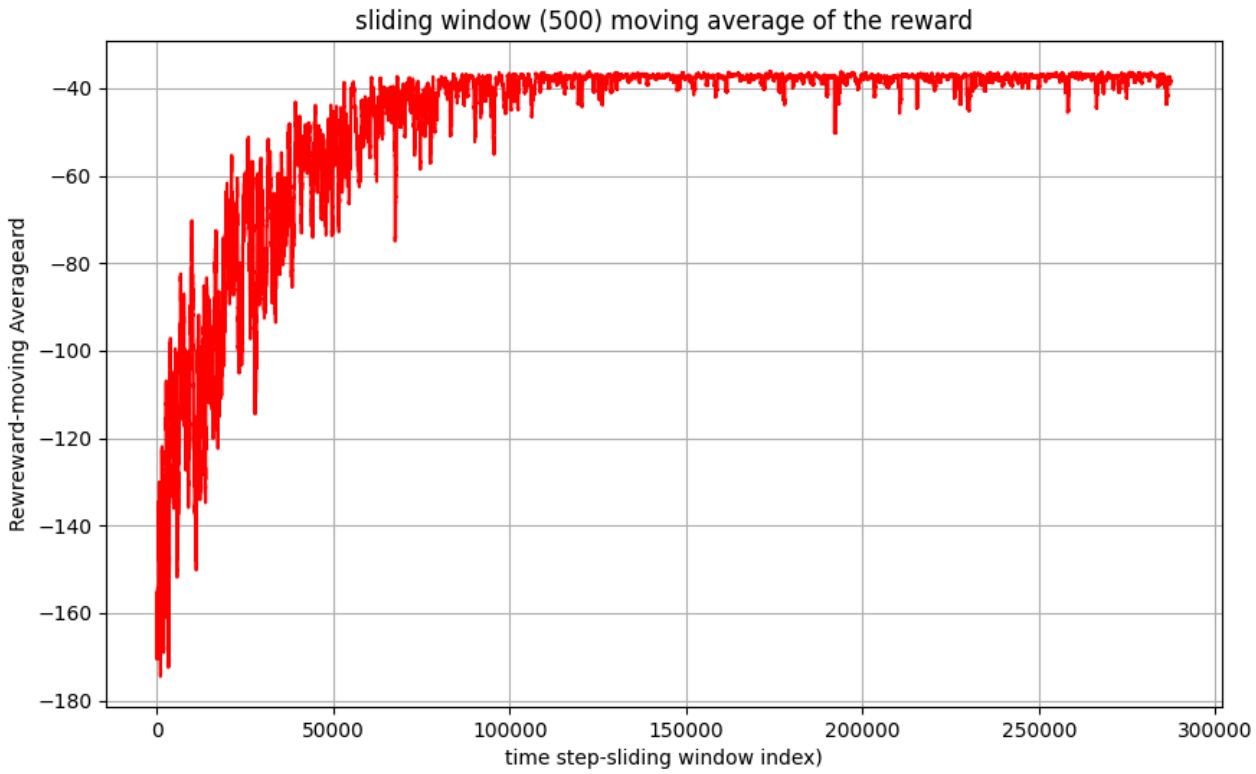
Section7 The Deliverables

1. Contextual bandits deal with stochastic setups, in ServerAllocationENV the sources of noise include:
    - Number of jobs arriving at data center for processing ($\mathbf{n_t}$) : no. of jobs change from one time slot to another and is a random integer between 1 & 8. Size of context is affected by this as no. of jobs keep changing per each time slot.
    - Cost incurred in every time slot as it depends on several factors like (wait time of all the jobs in the batch and energy cost of running the allocated servers etc..) even if the no. of jobs in different time slots are same the jobs might come with different job features like priority, latency, job type.
    - The data center knows the estimated processing time of each job but doesn't know the true processing time which introduces some randomness or unpredictable fluctuations in the environment. Latency of the job is a random variable for the data center.
    - The type of job, as ML related jobs require huge computational resources, some jobs require high speed processing and some are I/O intensive jobs and the network usage varies.
    - Overall impact on the reward, due to the above mentioned sources of randomness, the reward which is the negative of the cost is noisy which is required for this setup and helps the agent learn in different conditions.

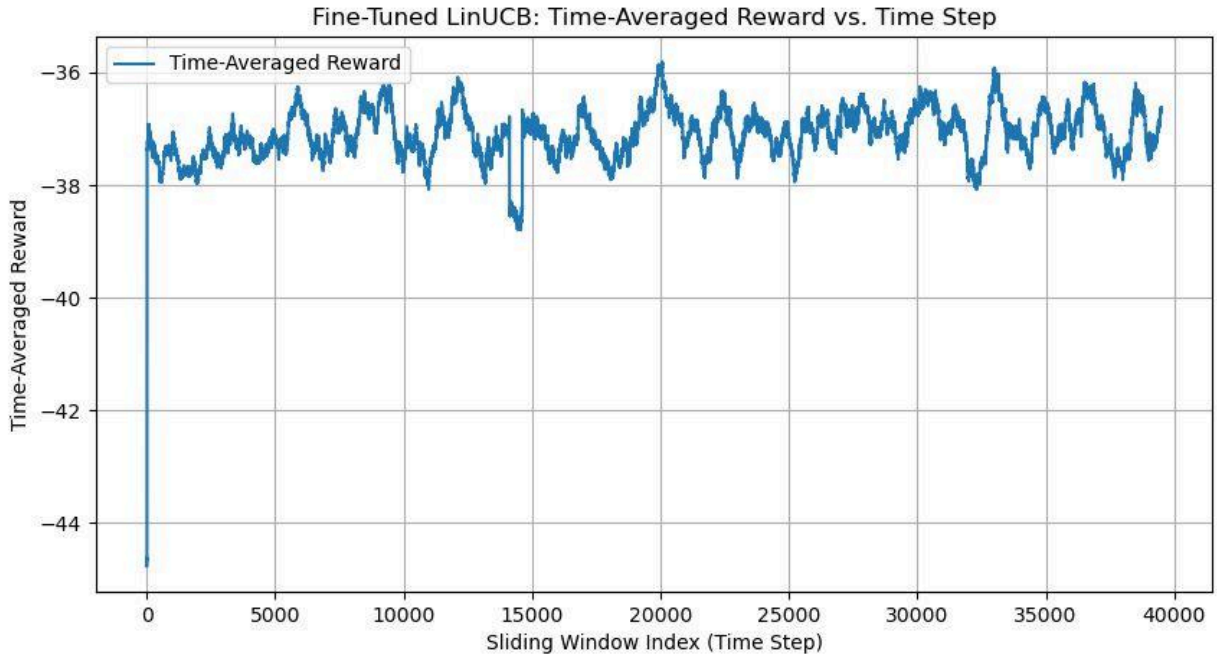
2. (a) How to deal with the fact that the context size is varying?
    - For this problem every time a new batch of jobs arrives the number of jobs can change . so, the context also changes in the size of, because the more jobs there are the more data points we get. In most machine learning models we expect the input to be the same size. To fix this we can take all the information from jobs and average it into a single , fixed-size vector. For example: instead of looking at each Job's priority like type, network and processing time individually, we can calculate the average of these values for all jobs in that time slot. This gives us a consistent set of numbers regardless of how many jobs there are. This averaged data can then be fed into the RL agent for training.
    - There are other ways to deal with this too like using :
      Padding: extend shorter sequences with a placeholder.
      Truncation: shorten longer sequences to a fixed length.
      But for this problem, averaging is an easy and effective solution from my point of view.

   (b) Can we reduce the number of features?
   YES, we can reduce the number of features to make the problem much easier for the agent to learn from.
   Each job has four pieces of information: Priority, job type, Network usage, Estimated processing
    - ➔ cross check if some features are related:
      If two features like network usage and processing time always seem to go up and down together,we could combine them or remove one of them.
    - ➔ use techniques to simplify the data:
      We can use methods like Principal Component Analysis to combine features in a way that reduces the total number of them but keeps the most important information.
      We can even use Mutual Information for feature selection.
    - ➔ create new and combined features:
      we could combine job priority and processing time into one value if they both affect how long the jobs wait.

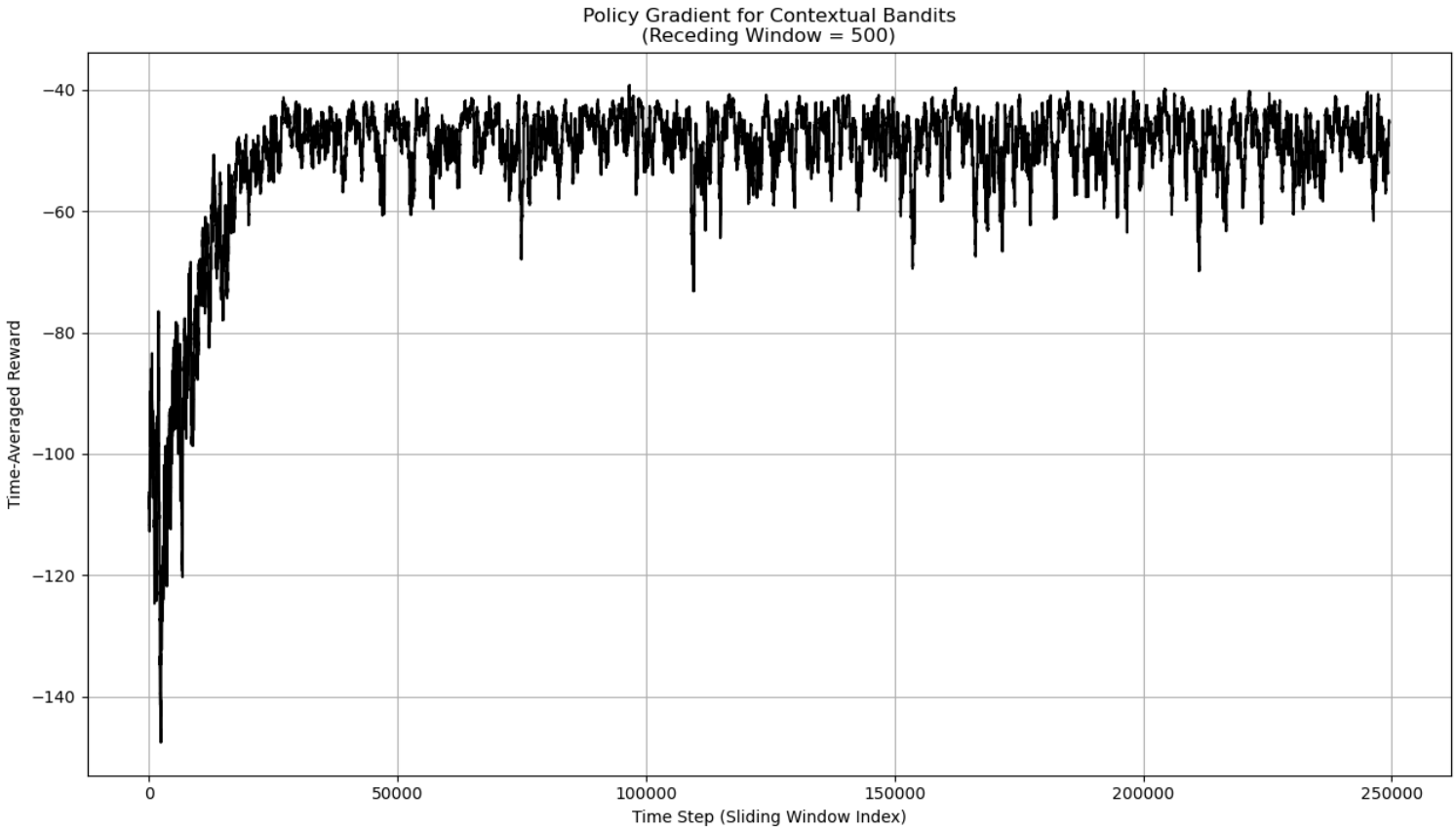
3. E-greedy graph-



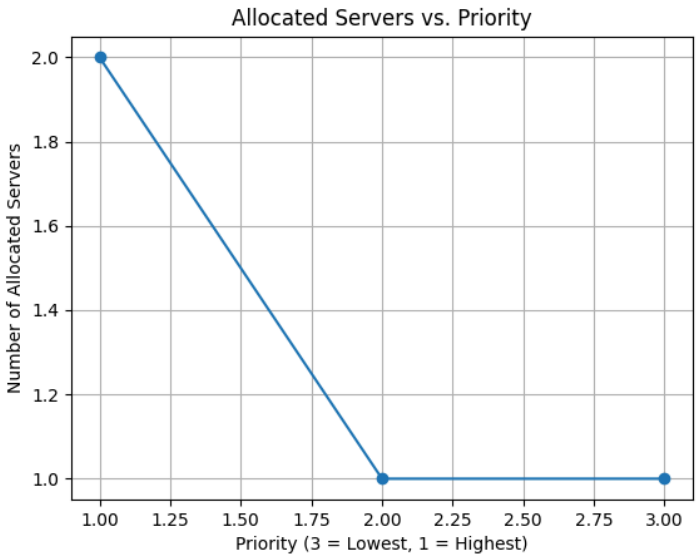
4. UCB graph- Jotsna

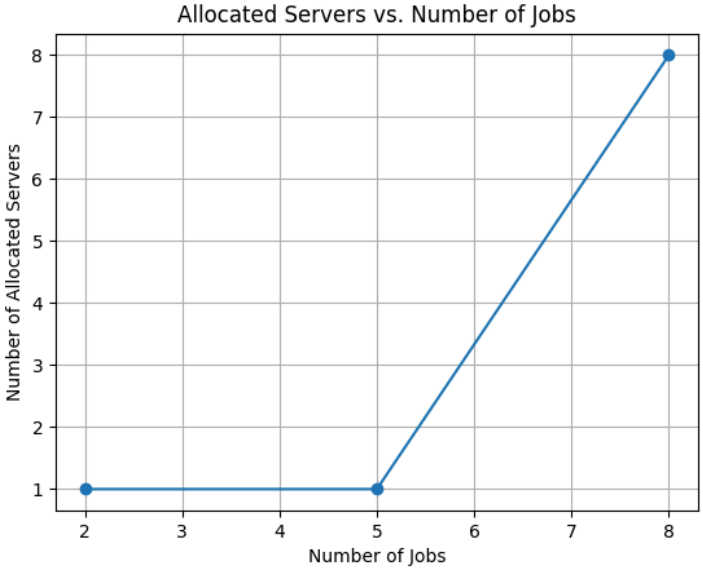


5. Policy Gradient plot:

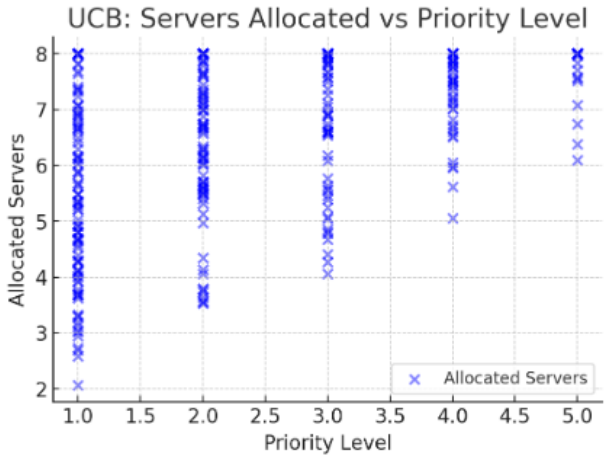


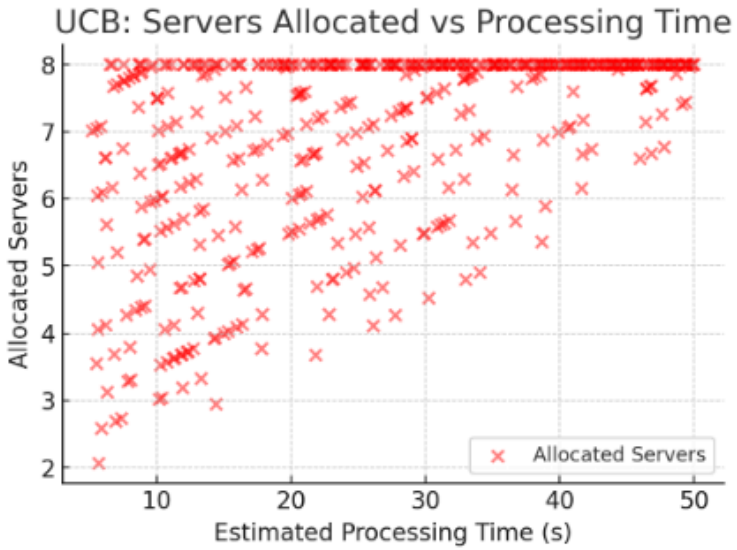
6. Testing the policies:
    - ε-greedy:

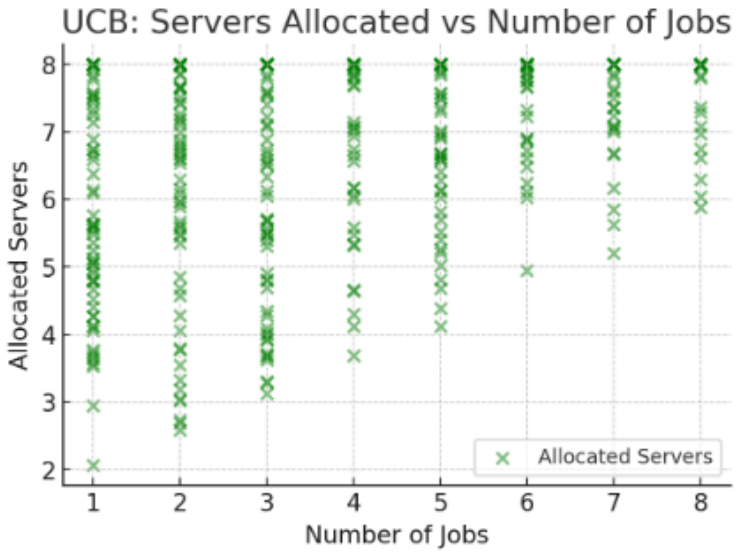

Allocated Servers vs. Priority
Number of Allocated Servers
Priority (3 = Lowest, 1 = Highest)
2.0
1.8
1.6
1.4
1.2
1.0
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00


Allocated Servers vs. Number of Jobs
Number of Allocated Servers
Number of Jobs
8
7
6
5
4
3
2
1
2
3
4
5
6
7
8

- UCB: [Jotsna]


UCB: Servers Allocated vs Priority Level
Allocated Servers
Priority Level
Allocated Servers
8
7
6
5
4
3
2
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0





UCB Logical Evaluation Results

1️⃣ Servers Allocated vs Priority Level

- The plot shows a **positive correlation**: as **priority increases**, the allocated servers **increase**.
- This indicates that UCB is correctly prioritizing **high-priority jobs**.

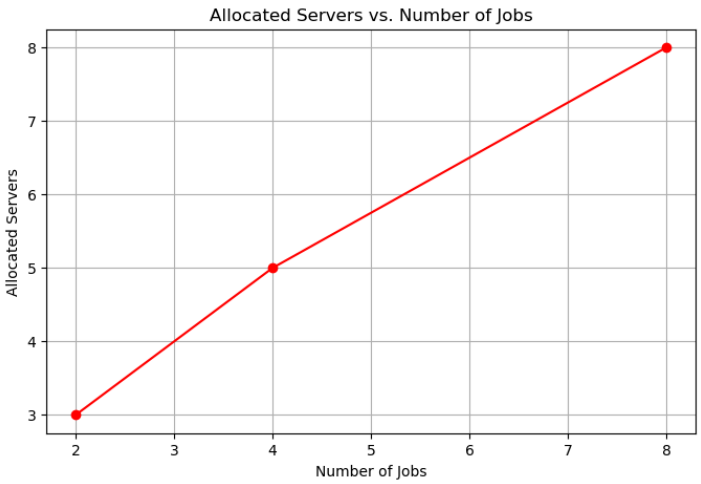
2️⃣ Servers Allocated vs Estimated Processing Time

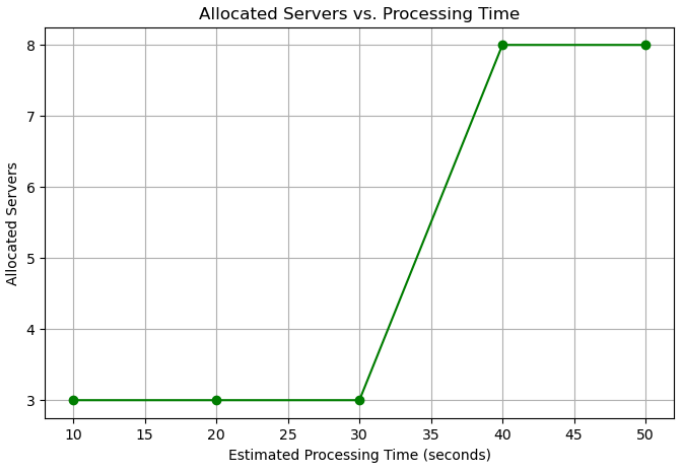
- The allocation **increases with processing time**.
- This suggests that the UCB agent **allocates more servers to longer jobs**, which is **expected behavior**.

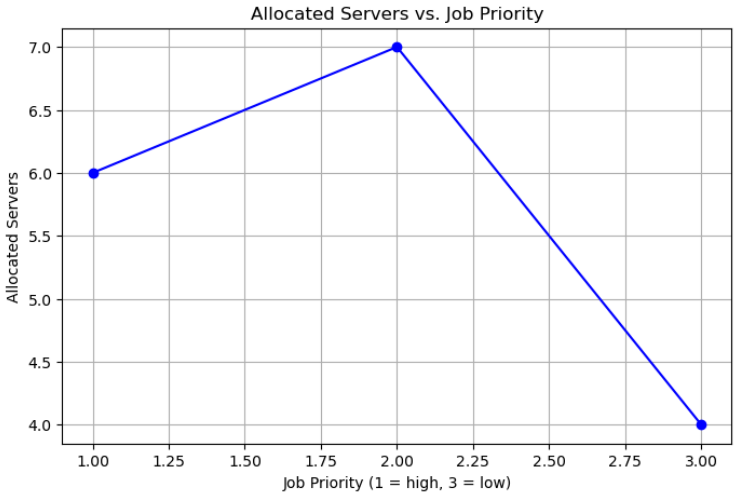
3️⃣ Servers Allocated vs Number of Jobs

- The plot shows a **positive trend**: as **number of jobs increases**, more servers are allocated.
- This confirms that the UCB agent **adjusts allocations dynamically**.

- Policy gradient:





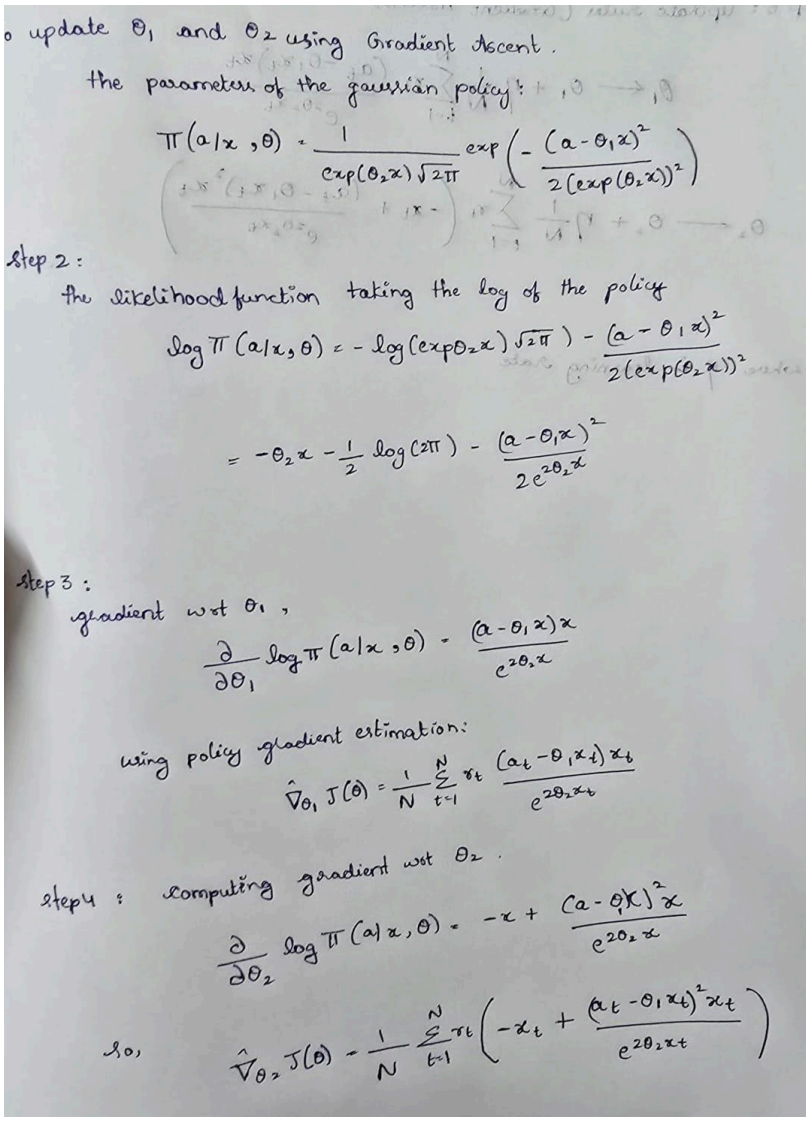


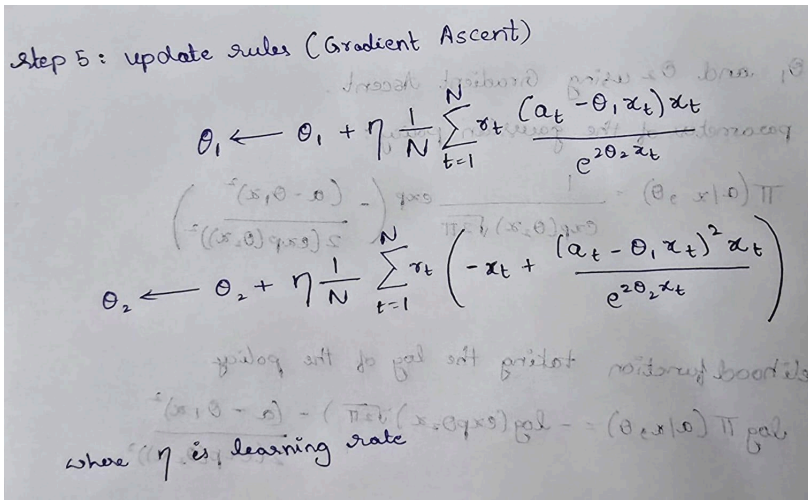
7. (a) Gaussian distribution

```
import numpy as np

mean = np.dot(theta1.T,x)
std_dev = np.exp(np.dot(theta2.T,x)
action = np.random.normal(mean, std_dev)
```

(b)





(c) **Policy Gradient Pseudocode**

- For episode = 1, ..., E:

- Create an empty dataset episode_training_data
- For t=1,...,N:
  - Sample context x_t from context distribution F_e.
  - Sample action $at \sim N(\theta_1 x_t, exp(2\theta_2 x_t))$
  - Execute a_t and observe reward r_t
  - Append (x_t, a_t, r_t) onto episode_training_data

  Use episode_training_data to update θ using stochastic gradient ascent:

- Compute policy gradients:
  - $\nabla_{\theta_1} = \frac{1}{N} \sum_{t=1}^{N} r_t \frac{(a_t - \theta_1 x_t) x_t}{e^{2\theta_2 x_t}}$
  - $\nabla_{\theta_2} = \frac{1}{N} \sum_{t=1}^{N} r_t \left(-x_t + \frac{(a_t - \theta_1 x_t)^2 x_t}{e^{2\theta_2 x_t}} \right)$
- Update parameters using gradient ascent:
  - $\theta_1 \leftarrow \theta_1 + \eta \nabla \theta_1$
  - $\theta_2 \leftarrow \theta_2 + \eta \nabla \theta_2$