CS 4122: Reinforcement Learning and Autonomous Systems
Programming Assignment #1 (for Module 1)

Team 1
Kuhu Gupta - SE21UARI070
Nandika Soni - SE21UARI090
Ananya Hazarika - SE21UCAM002

Q1) Sources of Noise in Contextual Bandits

In contextual bandits, the rewards associated with actions exhibit randomness due to several inherent factors.

Stochastic Nature of the Environment: The environment is dynamic and uncertain. External factors (e.g., traffic conditions, user behavior) lead to fluctuations in rewards. For example, mmWave communication can experience rapid changes in channel conditions due to obstacles or interference.

Exploration-Exploitation Trade-off: Algorithms like ϵ-greedy or Thompson Sampling introduce randomness through exploration. The agent encounters varying rewards when trying different actions to gather information.

Measurement Errors: Inaccuracies in measuring and recording rewards can lead to discrepancies. Examples include sensor noise in vehicular applications or errors in data collection.

User-Specific Preferences: Individual user preferences introduce variability in reward outcomes. Different users may react differently to the same action, resulting in a range of possible rewards.

Randomized Decision Processes: Some algorithms intentionally incorporate randomness in decision-making. This randomness can lead to variations in received rewards based on the agent's actions.

Understanding these sources of noise is crucial for developing effective contextual bandit algorithms and improving performance in uncertain environments.

Q2) Addressing Varying Context Vector Sizes in mmWave Vehicular Communication

To address the issue of varying context vector sizes in your mmWave vehicular communication setup, where each vector comprises coordinates (x, y) and velocity, consider the following approaches:

Fixed-Size Representation

- **Padding:** Use zero-padding to extend smaller context vectors to a fixed size (e.g., the maximum size of 12). This allows all inputs to have the same dimensionality, making them compatible with most ML/DL models.
- **Truncation:** If the context vector exceeds the maximum size, truncate the vector to fit the desired length. This approach risks losing potentially valuable information but can simplify processing.

Dynamic Pooling Techniques

- **Max Pooling and Average Pooling:** For varying-sized inputs, one can apply pooling techniques to reduce dimensions. For instance, if you have multiple vehicles, you can take the maximum or average values of the coordinates and velocities to create a single fixed-size context vector.

- **Min Pooling:** Similar to max and average pooling, but captures the minimum value, which can provide insights into the worst-case scenarios in vehicle behavior.

Graph Neural Networks (GNNs)

GNNs are particularly suited for handling variable-sized input data, such as varying numbers of vehicles in your scenario. They can model relationships and interactions between nodes (vehicles) dynamically. By treating each vehicle as a node in a graph, you can learn from the interactions between them, regardless of the number of vehicles.

Attention Mechanisms

Implement attention mechanisms to dynamically adjust focus on different parts of the context vector. This allows the model to weigh the importance of each vehicle's information based on the current context, thus effectively handling variable input sizes by prioritizing relevant data.

Recurrent Neural Networks (RNNs)

RNNs and their variants (like LSTMs and GRUs) can process sequences of variable lengths. You can represent each vehicle's state over time, effectively capturing the dynamics of communication among vehicles without being limited by fixed input sizes.

Q6)

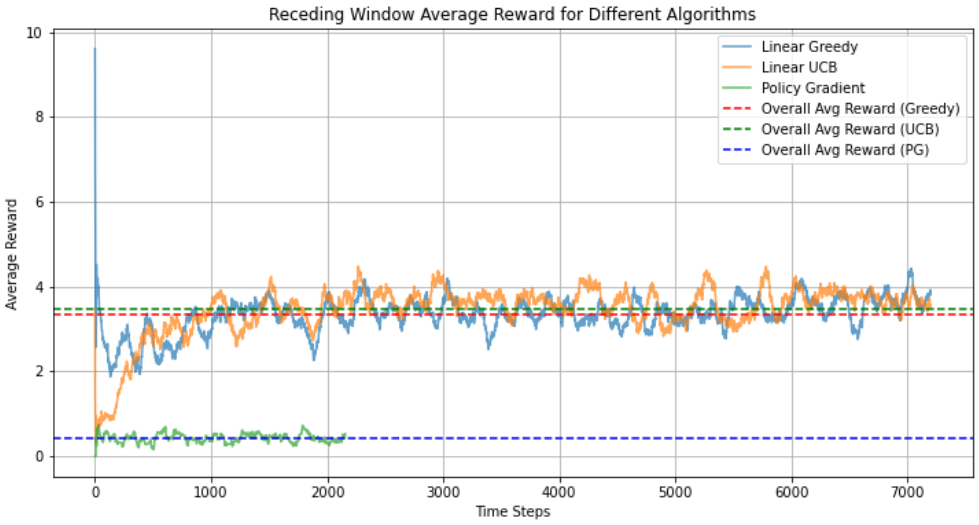

Figure 1: This is a sample image.

Used 10 episodes for epsilon greedy and UCB , but for policy gradient , only 3 episodes.

Q7- a)

The neural network used in the context of mmWave vehicular communication with a value function-based policy for contextual bandits has the following specifications:

Inputs

- **Input**: The input to the neural network is the **context-action pair**. This means that the number of inputs will be:
  - **Context Features**: Features that describe the current environment/context (e.g., vehicle positions, speed,, etc.).

– **Action Features**: Features that describe the action being taken (e.g., communication strategy- which beam, which station selection).

Outputs

- **Output**: The output of the neural network is the **estimated mean reward** corresponding to the context-action pair. This is a single value that represents the expected reward from taking a specific action in a specific context.

Summary

- **Number of Inputs**: $N_{context} + N_{action}$ (total features representing both the context and the action).
- **Number of Outputs**: 1 (the estimated mean reward).

Thus, the neural network will have $N_{context} + N_{action}$ inputs and 1 output.

For example, if there are $N_{context} = 12$ context features and $N_{action} = 20$ action features, the total number of inputs would be $12 + 20 = 32$, while the output remains 1.

Q7-b) Regression Problem in Contextual Bandits

Training a neural network to predict the context-action value in contextual bandits is fundamentally a **regression problem** for several reasons:

Reasons

1. **Output Nature**: - The neural network aims to estimate the expected reward for a specific context-action pair (x, a). The output $f(x, a; \theta)$ is a continuous value representing this expected reward, typical of regression tasks.

2. **Loss Function**: - Regression tasks often use loss functions like Mean Squared Error (MSE) to measure prediction accuracy. Here, the loss function is:

$$L(\theta) = \frac{1}{2} \sum_{k=0}^{t} (r_k - f(x_k, a_k; \theta))^2$$

- This minimizes the squared error between predicted and actual rewards, a hallmark of regression.

3. **Output Interpretation**: - The output is a continuous value rather than discrete class labels, reinforcing that this is a regression problem.

In contrast, a **classification problem** would involve predicting discrete labels, which is not applicable here.

Q7-c) Gradient of the Loss Function and Parameter Update

Step 1: Define the Loss Function

The loss function for a single time step t is given by:

$$\hat{L}(\theta) = \frac{1}{2} (r_t - f(x_t, a_t; \theta))^2$$

where:

- r_t is the reward received at time t.
- $f(x_t, a_t; \theta)$ is the predicted reward based on the context x_t and action a_t parameterized by θ.

Step 2: Compute the Gradient

To find the gradient of the loss function with respect to θ, we can use the chain rule. The gradient $\nabla_\theta \hat{L}(\theta)$ can be computed as follows:

$$\nabla_\theta \hat{L}(\theta) = \nabla_\theta \left(\frac{1}{2}(r_t - f(x_t, a_t; \theta))^2 \right)$$

Applying the chain rule:

$$\nabla_\theta \hat{L}(\theta) = (r_t - f(x_t, a_t; \theta)) \cdot \nabla_\theta (r_t - f(x_t, a_t; \theta))$$

Since $\nabla_\theta (r_t - f(x_t, a_t; \theta)) = -\nabla_\theta f(x_t, a_t; \theta)$, we can rewrite this as:

$$\nabla_\theta \hat{L}(\theta) = -(r_t - f(x_t, a_t; \theta)) \cdot \nabla_\theta f(x_t, a_t; \theta)$$

Step 3: Update Equation Using Gradient Descent

In gradient descent, the update rule for the parameters θ is given by:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \hat{L}(\theta)$$

where:

- α is the learning rate.

Substituting the gradient we found:

$$\theta \leftarrow \theta + \alpha (r_t - f(x_t, a_t; \theta)) \cdot \nabla_\theta f(x_t, a_t; \theta)$$

In total

1. **Gradient of the Loss Function**:

$$\nabla_\theta \hat{L}(\theta) = -(r_t - f(x_t, a_t; \theta)) \cdot \nabla_\theta f(x_t, a_t; \theta)$$

2. **Update Equation**:

$$\theta \leftarrow \theta + \alpha (r_t - f(x_t, a_t; \theta)) \cdot \nabla_\theta f(x_t, a_t; \theta)$$

This equation shows how to update the parameters θ using the observed reward, the predicted reward, and the gradient of the function f with respect to the parameters. This iterative process allows the model to learn from the rewards received in the context of the actions taken.

Q7-d) Pseudocode for ϵ-greedy Algorithm

```
# Initialize parameters
epsilon = 0.1  # Exploration probability
alpha = 0.01   # Learning rate
num_actions = 20  # Total number of actions
num_context_features = 12  # Number of context features
num_episodes = 1000  # Number of episodes for training
neural_network = initialize_neural_network(input_shape=(num_context_features + 1,))

# Main Loop
for episode in range(num_episodes):
    # Reset environment and get initial context
    context, _ = env.reset()  # Replace with your environment's reset method

    # Reshape context for neural network input
    z = np.concatenate((context, np.array([1])))  # Adding bias term
```

```
z = z.reshape(1, -1)  # Reshape for the neural network

# Choose action using epsilon-greedy strategy
if np.random.uniform(0, 1) < epsilon:
    action = np.random.randint(num_actions)  # Exploration
else:
    predicted_rewards = neural_network.predict(z)  # Exploitation
    action = np.argmax(predicted_rewards)  # Best action based on prediction

# Take action in the environment
next_context, reward, _, truncated, _ = env.step(action)  # Replace with your environment's step method

# Prepare next context for the neural network
next_z = np.concatenate((next_context, np.array([1])))  # Adding bias term
next_z = next_z.reshape(1, -1)  # Reshape for the neural network

# Compute loss for the current time step
predicted_reward = neural_network.predict(z)
loss = (reward - predicted_reward) ** 2  # Squared error

# Update neural network using Keras
neural_network.fit(z, reward, epochs=1, verbose=0)  # Update the model with current observation

# Update context for next iteration
context = next_context
```