Siddhesh Maheshwari MDS202347
Narendra C MDS202336
Gauranga Kumar Baishya MDS202325

Advanced Machine Learning Assignment 03

Project Report: Training and Evaluation of a Reinforcement Learning Agent

1. Introduction

Reinforcement learning (RL) is a machine learning paradigm where an agent learns to make decisions by interacting with an environment. In this project, we implemented, trained, and evaluated a Deep Q-Learning (DQN) agent to solve a task, comparing its performance against a random agent. The task involved interacting with an environment to maximize cumulative rewards by choosing optimal actions. The trained agent was evaluated for its ability to improve over time in terms of rewards, efficiency, and decision-making accuracy.

2. Objectives

- Implement a Deep Q-Learning model for an agent.
- Train the agent to optimize its performance on the task.
- Compare the trained agent's performance with that of a random agent.
- Visualize metrics such as episode durations, rewards, and percentage of optimal actions to highlight the differences between the two agents.

3. Methodology

3.1 Environment Setup

We used a simulation environment that provides:

State Space: The environment's state space is represented as a 6-dimensional vector that captures the angles and angular velocities of the two links in the system.

Action Space: The agent interacts with the environment through a discrete action space consisting of three possible actions:

- **0**: Apply torque to the left.
- **1**: Apply no torque.
- **2**: Apply torque to the right.

Rewards: A reward of **-1** is given at every time step until the target height is achieved, incentivizing the agent to minimize the number of steps taken.

The environment was reset at the start of each episode, and the agent interacted with it by choosing actions based on its policy.

3.2 Model Architecture and Training Process

Network Architecture

The Q-network maps the input state vector to Q-values for each possible action. The architecture is described as follows:

- **Input Layer**: The network accepts a 6-dimensional state vector representing the environment's current state.
- **Hidden Layers**: Two fully connected layers, each with 128 neurons.
    - **First Hidden Layer**:
        - Linear transformation (input_size → 128)
        - Batch Normalization to stabilize learning
        - ReLU activation for non-linearity
        - Dropout (p=0.2) for regularization
    - **Second Hidden Layer**:
        - Linear transformation (128 → 128)
        - Batch Normalization
        - ReLU activation
        - Dropout (p=0.2)
- **Output Layer**: Produces Q-values for all actions (128 → output_size, where output_size = 3 for the Acrobot environment).

This design balances model complexity and regularization, ensuring the network can effectively approximate the Q-value function while avoiding overfitting.

Training Details

1. **Replay Memory**: Stores up to 10,000 experiences, sampled in mini-batches for training.
2. **Optimization**: The Mean Squared Error (MSE) loss between predicted and target Q-values is minimized.
3. **Hyperparameters**:
    - Learning Rate: 0.001
    - Discount Factor: 0.99
    - Batch Size: 64
    - Epsilon: Decays from 1.0 to 0.01 over episodes

3.3 Random Agent

A baseline random agent was implemented to select actions uniformly at random, serving as a control for evaluating the performance of the trained agent.

4. Results and Analysis

4.1 Performance Comparison

- **Reward**: The trained agent achieved significantly higher average rewards compared to the random agent. The reward graph showed steady improvement during training, whereas the random agent's performance remained flat and poor.
- **Optimal Actions**: The trained agent consistently selected optimal actions, achieving a near 100% success rate after sufficient training episodes. The random agent hovered around chance-level performance (~10-20%).
- **Episode Duration**: The trained agent completed episodes more efficiently, with durations decreasing steadily during training. The random agent often reached the maximum allowable steps, indicating inefficient exploration. (In this scenario, episode duration is the negation of the rewards gained)

4.2 Visualisation to compare performance metrics

Graphs were plotted to compare the agents:

- **Rewards and Optimal Actions**: The trained agent's on an average gained rewards of -125(approximately) and the random agent the reward gained was approximately -499.
- **Episode Durations**: Smoothed durations for the trained agent displayed a steady decline, reflecting learning, whereas the random agent showed no discernible pattern.

5. Conclusion

The project successfully implemented and evaluated a Deep Q-Learning agent. The trained agent significantly outperformed the random agent across all metrics, demonstrating the efficacy of reinforcement learning in solving complex decision-making tasks. By leveraging experience replay, a target network, and epsilon-greedy exploration, the agent was able to learn an optimal policy efficiently.