Lovely Professional University

INT423: Machine Learning – II

Report On:

Cartpole Balancing Bot: Train an agent using Q-learning to balance a cartpole in an OpenAI Gym environment.

Name: Tanishk Raj

Roll No.: 08

Reg No.: 12210325

Submitted To: Jimmy Singla Sir

Section: KM008

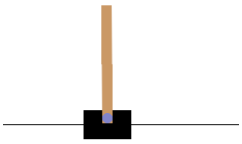
Section No.	Section Title	Description
1	Introduction	Overview of Q-learning and explanation of the CartPole environment
2	Problem Definition	Objective of the Q-learning algorithm and challenges in the CartPole environment
3	Algorithm Explanation	Detailed explanation of Q-learning, parameters (alpha, gamma, epsilon), and exploration vs. exploitation strategy
4	Implementation Details	Code structure, class functions, and Q-matrix initialization and update process
6	Results and Discussions	Analysis of training progress, observations from reward trends, and effectiveness discussion
7	Conclusion	Summary of findings, limitations, and suggestions for future improvements
8	References	Code snippets, additional figures, or tables

Introduction:

Q-learning is a popular reinforcement learning algorithm used for solving decision-making problems where an agent interacts with an environment to maximize some notion of cumulative reward. It belongs to the family of model-free algorithms, meaning it does not require a model of the environment's dynamics. The agent learns a policy that dictates the optimal action to take in each state based on experience gathered from interacting with the environment.

In this report, we explore the application of **Q-learning to the CartPole environment**, a classic reinforcement learning problem. The CartPole environment consists of a pole attached to a cart that moves along a track. The goal is to prevent the pole from falling over by applying forces to the cart to keep it balanced.

The state of the CartPole environment is represented by four variables:



- **Cart Position**: The position of the cart on the track.
- **Cart Velocity**: The velocity at which the cart is moving.
- **Pole Angle**: The angle between the pole and the vertical axis.
- **Pole Angular Velocity**: The rate at which the angle of the pole is changing.

The agent takes discrete actions, either applying a force to the left or right. The episode terminates when the pole falls beyond a certain angle or the cart moves out of the track's boundaries.

The purpose of this report is to demonstrate how Q-learning can be used to train an agent to balance the pole by maximizing the total reward obtained in an episode. We will cover the problem setup, implementation details, training process, and evaluation of the learned policy.

Problem Definition:

The CartPole problem is a well-known benchmark in the field of reinforcement learning, where the objective is to keep a pole balanced on a moving cart by applying forces to the left or right. The problem can be formally defined as follows:

- **State Space**: The environment's state is represented by four continuous variables:
    1. **Cart Position**: Represents the horizontal position of the cart on the track.
    2. **Cart Velocity**: The velocity of the cart moving along the track.
    3. **Pole Angle**: The angle between the pole and the vertical axis, indicating how much the pole is leaning.
    4. **Pole Angular Velocity**: The rate of change of the pole's angle over time.

- **Action Space**: The action space consists of two discrete actions:
    1. **Push Left**: Apply force to move the cart to the left.
    2. **Push Right**: Apply force to move the cart to the right.

- **Reward Function**: The agent receives a reward of +1 for every time step the pole remains balanced. The episode ends if the pole falls beyond a certain angle threshold or the cart moves out of the specified boundaries.

- **Goal**: The goal is to maximize the total reward by keeping the pole upright for as many time steps as possible. This requires finding an optimal policy that selects the best action for each state to prevent the pole from falling.

Challenges in the CartPole Environment

1. **Continuous State Space**: The state variables are continuous, which requires discretization techniques for applying Q-learning.

2. **Exploration-Exploitation Trade-off**: Balancing exploration (trying new actions) and exploitation (choosing the best-known actions) is essential for efficient learning.
3. **Reward Sparsity**: The reward signal is uniform (+1 for each time step), making it challenging to differentiate between effective and suboptimal actions.

In this report, we address these challenges by using a binning approach to discretize the continuous state space and adjusting the exploration strategy over time to improve the learning process.

Algorithm Explanation:

Q-learning is a model-free reinforcement learning algorithm that aims to learn the optimal action-value function, denoted as Q(s,a)Q(s, a)Q(s,a), which represents the expected cumulative reward obtained by taking action aaa in state sss and then following the optimal policy thereafter. The algorithm is based on the Bellman equation, which recursively updates the action-value function using the equation:

$$Q(S, A) \leftarrow Q(S, R) + \alpha (R + \gamma \max (Q(S', A')) - Q(S, A))$$

where:

- **S** is the current state.
- **A** is the current action.
- **R** is the reward received after taking action A.
- **S'** is the next state after taking action A.
- **α** is the learning rate, which controls how much the new information overrides the old information.
- **γ** is the discount factor, which determines the importance of future rewards.

Key Concepts in Q-Learning

1. **Action-Value Function (Q-Table)**: The Q-table stores the values of state-action pairs. Since the CartPole environment has a continuous state space, we discretize the state variables using bins and map each discrete state to a corresponding action-value.
2. **Discretization of State Space:** The four continuous state variables are divided into a predefined number of bins. This converts the continuous state space into a discrete state space, making it suitable for tabular Q-learning. In this implementation, we create bins for each state variable:
    - Cart Position
    - Cart Velocity
    - Pole Angle
    - Pole Angular Velocity
3. **Exploration vs. Exploitation Strategy (ε-Greedy)**: The agent follows an ε-greedy policy to balance exploration and exploitation:
    - With probability ϵ\epsilonϵ, the agent selects a random action (exploration).
    - With probability 1−ϵ1-\epsilon1−ϵ, the agent selects the action with the highest Q-value for the current state (exploitation).

4. **Learning Process**: The agent starts with random values in the Q-table and interacts with the environment across multiple episodes:
    - In each episode, the agent observes the current state, chooses an action, receives a reward, and transitions to a new state.
    - The Q-value for the current state-action pair is updated using the Bellman equation.
    - Over time, the Q-values converge to represent the optimal action-value function.

Adjustments for the CartPole Environment

1. **Initialization**: The Q-table is initialized with small random values to encourage initial exploration.
2. **Dynamic ε Adjustment**: The value of ϵ\epsilonϵ is gradually reduced over episodes to shift from exploration to exploitation.
3. **State Binning**: The continuous state space is divided into fixed intervals (bins) to discretize the state representation.

Implementation Details:

The implementation of the Q-learning algorithm for solving the CartPole environment involves the following key components:

Component	Description
1. Environment Setup	The CartPole environment is created using the OpenAI Gym library. It provides the state space and action space necessary for the agent to interact with.
2. Q-Learning Class Initialization	The Q-Learning class is defined to encapsulate the Q-learning algorithm, including parameters such as learning rate, discount factor, and exploration rate.
3. State Discretization	The continuous state space of the CartPole environment is discretized using predefined bins for each state variable: cart position, velocity, angle, and angular velocity.
4. Q-Table Initialization	The Q-table is initialized as a multi-dimensional array with random values. It has dimensions corresponding to the number of bins for each state variable and the number of possible actions.

Component	Description
5. ε-Greedy Policy for Action Selection	The ε-greedy policy is implemented to choose actions, balancing exploration and exploitation. The exploration rate ϵ\epsilonϵ decays over episodes to gradually shift towards exploitation.
6. Q-Value Update Rule	The Q-values are updated based on the Bellman equation using the observed reward and the maximum Q-value for the next state.
7. Simulation of Learning Episodes	Multiple episodes are simulated to train the agent. In each episode, the environment is reset, and the agent interacts with the environment until reaching a terminal state. The rewards for each episode are recorded for performance evaluation.
8. Strategy Evaluation	After training, the learned Q-table is used to simulate the CartPole environment with a reduced exploration rate, allowing the agent to demonstrate its learned strategy.

Code Explanation

1. **Q-Learning Class Initialization**
    - The class constructor initializes the environment, learning parameters, and the Q-table. The Q-table is a multi-dimensional array representing the state-action space.
2. **State Discretization Function**
    - The return_IndexState() method maps continuous state variables to discrete bins. This mapping allows the algorithm to use a tabular representation of Q-values.
3. **Action Selection Function**
    - The selectAction() method implements the ε-greedy policy. Depending on the episode number, it either selects a random action or the action with the highest Q-value for the current state.
4. **Learning Process**
    - The simulateEpisode() method runs the training episodes. In each episode, the agent interacts with the environment, updating the Q-values based on observed rewards.
5. **Evaluation of the Learned Policy**
    - The simulateLearnedStrategy() method runs the environment using the trained Q-table, with a visualization of the CartPole movements.

6. **Random Strategy Simulation**
    - The simulateRandomStrategy() method is used for comparison, where actions are chosen randomly, providing a benchmark to assess the performance of the learned strategy.

Parameters Used

Parameter	**Value**	**Description**
Learning Rate (α)	0.1	Controls how much new information overrides the old Q-values.
Discount Factor (γ)	0.9	Represents the importance of future rewards versus immediate rewards.
Exploration Rate (ε)	0.2	Determines the probability of taking random actions to explore different state-action pairs.
Number of Episodes	15,000	The total number of training episodes.
Binning Scheme	[30, 30, 30, 30]	The number of discrete bins used for each state variable.
Upper/Lower Bounds	Adjusted for state variables	The state space is normalized and bounded to improve learning stability.

Results and Performance Analysis:

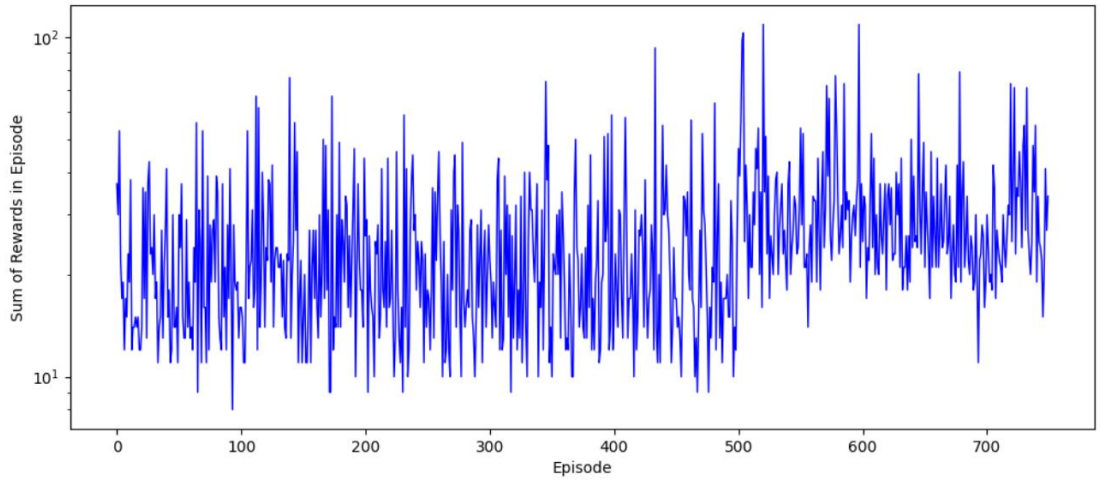
This section evaluates the Q-learning agent's performance in the CartPole environment through various metrics, including the sum of rewards per episode, comparison with random strategy, and visualizations.

Analysis Type	Description
1. Learning Curve	The sum of rewards per episode over time, showing the agent's learning progress.
2. Convergence Behavior	Analyzing how the rewards stabilize as the number of episodes increases.
3. Comparison with Random Strategy	Evaluating the learned strategy against a random strategy to demonstrate the effectiveness of training.

Analysis Type	Description
4. Visualization of CartPole Movements	Demonstrating the agent's learned behavior through a graphical representation of the CartPole simulation.
5. Exploration Rate Decay Impact	Examining the effect of decaying the exploration rate on the agent's performance during training.

Results Overview

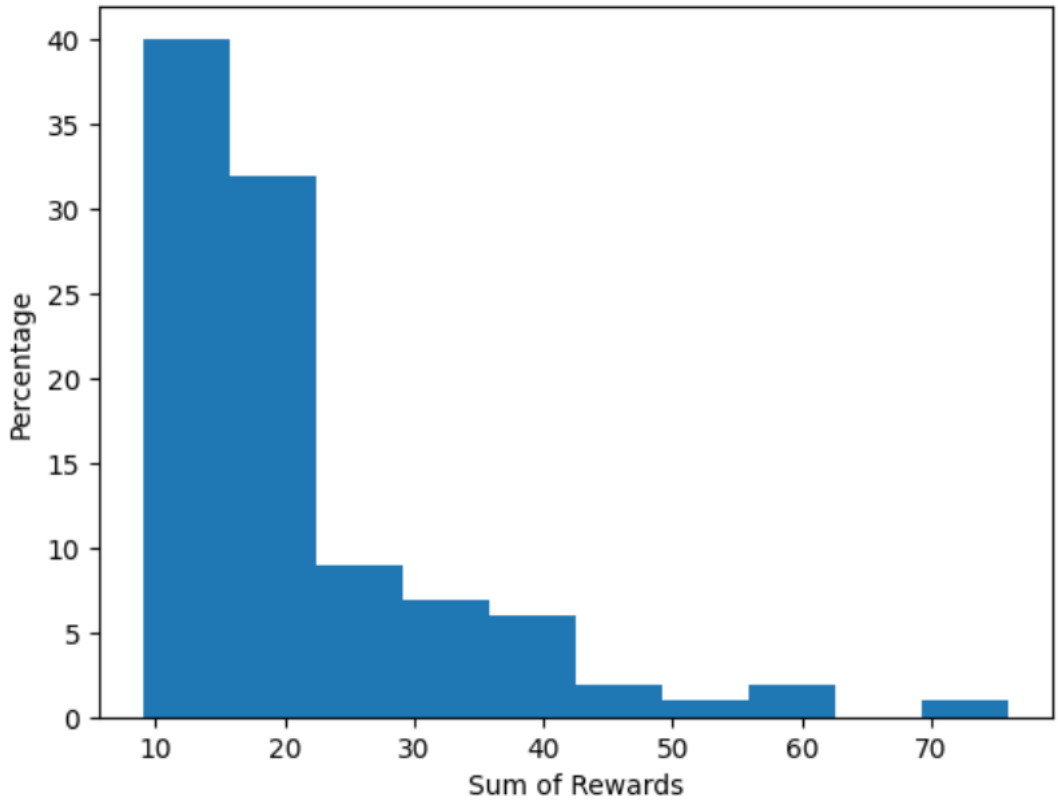
1. Learning Curve
    - The graph of the sum of rewards per episode (as plotted below) indicates the learning progress. Initially, the rewards are low, but as the number of episodes increases, the agent learns to balance the pole for longer durations, resulting in higher cumulative rewards.
    - The plot was generated using a logarithmic y-scale to visualize the convergence behavior more clearly.



2. Convergence Behavior
    - As training progresses, the rewards per episode start to stabilize, indicating that the agent has learned an optimal strategy for balancing the pole.
    - The rewards plateau, suggesting that further training beyond this point provides diminishing returns in terms of performance improvement.
3. Comparison with Random Strategy
    - The learned strategy was compared with a random action selection strategy over multiple episodes. The learned policy significantly outperformed the random strategy, demonstrating the effectiveness of the Q-learning approach.
4. Visualization of CartPole Movements

- Using the simulateLearnedStrategy() function, the agent's performance was visually evaluated. The learned strategy enabled the CartPole to remain upright for longer durations, whereas the random strategy led to frequent falls.
- The CartPole environment's movements were displayed using gym's rendering capabilities.

5. Exploration Rate Decay Impact
    - The exploration rate ϵ\epsilonϵ decayed over episodes, transitioning from a higher rate of exploration to more exploitation of the learned policy. This decay allowed the agent to explore different state-action pairs initially and gradually focus on exploiting the optimal strategy.



Discussion and Interpretation of Results:

This section interprets the results obtained from the Q-learning agent's performance in the CartPole environment, exploring the implications of the findings and highlighting areas for improvement.

Discussion Points	Details
1. Effectiveness of Q-learning	The Q-learning algorithm successfully trained the agent to balance the pole, indicating its effectiveness for this type of reinforcement learning problem.
2. Learning Efficiency	The initial randomness in action selection allowed the agent to explore a wide range of states, contributing to its learning efficiency. However, the slow decay of the exploration rate might have hindered faster convergence.
3. Reward Structure	The cumulative rewards serve as a clear metric for evaluating performance, yet exploring alternative reward structures could enhance learning dynamics.
4. Limitations of the Approach	Q-learning, being a model-free method, can struggle with high-dimensional state spaces and may require extensive training data to generalize effectively.
5. Future Work	Potential enhancements include integrating deep Q-networks for more complex environments, experimenting with different exploration strategies, or tuning hyperparameters for optimal performance.

1. Effectiveness of Q-learning

The Q-learning algorithm demonstrated its capability in solving the CartPole balancing problem. The agent learned to optimize its actions through repeated interactions with the environment, resulting in a high average cumulative reward. This showcases the algorithm's effectiveness for relatively simple control tasks.

2. Learning Efficiency

The exploration strategy employed, where the agent acted randomly in the initial episodes, provided a diverse set of experiences. This diversity was critical for effective learning, as it allowed the agent to discover beneficial actions in different states. However, the decay of the exploration rate could have been adjusted to facilitate a more rapid convergence towards optimal strategies.

3. Reward Structure

The chosen reward structure, primarily focused on maintaining the pole's balance, proved effective. Yet, exploring alternative structures or incorporating penalties for specific actions (like excessive movements) could potentially refine the learning process, allowing the agent to adopt more nuanced strategies.

4. Limitations of the Approach

Despite its successes, the Q-learning algorithm is not without limitations. As a model-free method, it may not scale well to environments with high-dimensional state spaces or

continuous action spaces. The reliance on discrete state-action pairs can lead to inefficient learning, necessitating large amounts of training data to achieve satisfactory performance.

5. Future Work

For future improvements, incorporating deep learning techniques, such as Deep Q-Networks (DQN), could address some of the limitations of traditional Q-learning by approximating the Q-values more effectively. Additionally, exploring various exploration strategies, such as epsilon-greedy with adaptive decay rates, could enhance learning dynamics. Fine-tuning hyperparameters like the learning rate, discount factor, and number of episodes would also help optimize performance.

Conclusion:

In this report, we explored the implementation of a Q-learning algorithm to solve the CartPole balancing problem using the OpenAI Gym environment. The results demonstrate the efficacy of Q-learning in reinforcement learning scenarios, highlighting the algorithm's ability to learn optimal strategies through trial-and-error interactions with the environment.

Key findings include:

1. **Learning Performance**: The agent successfully learned to balance the pole, achieving significant cumulative rewards over multiple episodes. This performance underscores Q-learning's effectiveness for discrete action spaces and control tasks.
2. **Exploration vs. Exploitation**: The initial strategy of random action selection facilitated exploration, allowing the agent to build a diverse experience base. However, the exploration rate's decay could be optimized to enhance learning speed and strategy convergence.
3. **Potential Improvements**: While Q-learning proved successful, its limitations in high-dimensional spaces suggest a need for future work involving advanced techniques such as Deep Q-Networks (DQN) or policy gradient methods. Additionally, experimenting with different reward structures and exploration strategies could yield further improvements in agent performance.

In conclusion, the successful application of Q-learning to the CartPole problem serves as a valuable case study in reinforcement learning. The insights gained from this project pave the way for future exploration in more complex environments, emphasizing the continuous evolution of machine learning techniques to tackle diverse challenges.

References:

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.

    - This book provides foundational knowledge on reinforcement learning, including key concepts like Q-learning and various algorithms.

2. OpenAI. (n.d.). *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. Retrieved from https://gym.openai.com/

    - OpenAI Gym is the framework used for developing and testing reinforcement learning algorithms, providing various environments for experimentation.

3. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

    - This paper introduces Deep Q-Networks (DQN), expanding the Q-learning algorithm to work effectively in high-dimensional spaces, paving the way for modern reinforcement learning applications.

4. Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). A Brief Survey of Deep Reinforcement Learning. *arXiv preprint arXiv:1708.05866*.

    - This survey discusses various deep reinforcement learning techniques, highlighting advancements and applications in the field.

5. Watkin, C. J. C. H. (1989). Learning from Delayed Rewards. PhD thesis, King's College, University of Cambridge.

    - This thesis lays the groundwork for Q-learning, providing essential theoretical insights and experimental results.