# Epsilon-Greedy Exploration Algorithm for CartPole-v1

---

## Project Overview

This project demonstrates the application of the **Epsilon-Greedy Exploration Algorithm** in conjunction with **Q-Learning** to solve the **CartPole-v1** environment from OpenAI's Gym. The goal is to train an agent to balance a pole on a moving cart by choosing appropriate actions to prevent the pole from falling.

---

## Why Epsilon-Greedy Exploration?

The Epsilon-Greedy strategy is a simple yet effective method to balance exploration and exploitation in reinforcement learning. With a probability of ? (epsilon), the agent explores by selecting a random action, and with a probability of 1 - ?, it exploits its current knowledge by choosing the action with the highest estimated reward. This approach ensures that the agent continues to explore new actions while leveraging learned behaviors to maximize rewards.

---

## Prerequisites

### Required Libraries

- `numpy` : For numerical computations.
- `gym` : For the CartPole-v1 environment.

### Installation

Install the necessary libraries using pip:

```
pip install numpy gym
```

---

## Code Description

The implementation consists of the following key components:

### 1. Importing Libraries

```
import numpy as np
import gym
```

### 2. Environment Setup

Initialize the CartPole-v1 environment:

```
env = gym.make('CartPole-v1')
```

### 3. Hyperparameters

Define the hyperparameters for the Q-Learning algorithm:

```
epsilon = 0.1  # Exploration rate
alpha = 0.1    # Learning rate
gamma = 0.99   # Discount factor
```

### 4. Q-Table Initialization

Initialize the Q-table to store the estimated values of state-action pairs:

```
Q = np.zeros((env.observation_space.shape[0], env.action_space.n))
```

**Note**: The CartPole-v1 environment has a continuous state space, which poses a challenge for Q-Learning that traditionally operates on discrete state spaces. To address this, state discretization techniques can be applied, such as dividing the continuous state space into discrete bins. This process is essential for effectively applying Q-Learning to environments like CartPole-v1. For more details on state discretization, refer to this tutorial on Q-Learning in Python with CartPole.

### 5. Epsilon-Greedy Policy Function

Define the policy for selecting actions based on the epsilon-greedy strategy:

```
def epsilon_greedy_policy(state):
    if np.random.rand() < epsilon:
        return np.random.choice(env.action_space.n)  # Explore
    else:
        return np.argmax(Q[state])  # Exploit
```

### 6. Q-Learning Algorithm

Implement the Q-Learning algorithm to update the Q-values based on the agent's interactions with the environment:

```
def q_learning(env, n_episodes=1000):
    for episode in range(n_episodes):
        state = env.reset()
        done = False
        while not done:
            action = epsilon_greedy_policy(state)  # Select action based on epsilon-gree
            next_state, reward, done, _ = env.step(action)

            # Q-value update
            Q[state, action] += alpha * (reward + gamma * np.max(Q[next_state]) - Q[sta

            state = next_state
```

### 7. Running the Algorithm

Execute the training function:

```
q_learning(env)
```

---

# Expected Outcomes

- **Policy Development**: The agent learns a policy to balance the pole by selecting actions that maximize the expected reward.

- **Exploration and Exploitation Balance**: The epsilon-greedy strategy ensures that the agent explores new actions while exploiting learned behaviors to achieve optimal performance.

---

## Use Cases

- **Reinforcement Learning**: Applying Q-Learning with epsilon-greedy exploration in environments where balancing exploration and exploitation is crucial.
- **Control Problems**: Solving control tasks like CartPole, where an agent must learn to maintain stability through appropriate actions.

---

## Future Enhancements

- **State Discretization**: Implement state discretization techniques to handle the continuous state space of CartPole-v1 effectively.
- **Dynamic Epsilon**: Adjust the epsilon value dynamically to reduce exploration as the agent becomes more confident in its policy.
- **Reward Shaping**: Modify the reward structure to provide more informative feedback to the agent, potentially accelerating learning.

---

## References


- Q-Learning in Python with Tests in Cart-Pole OpenAI Gym Environment
- Balancing CartPole-v1 from OpenAI Gym by employing Epsilon-Greedy strategy for Q-Learning


---