ML Lab-10

Name: Kresha Shah Sapid:60009220080 Batch: D22

Colab link:
https://colab.research.google.com/drive/11fP7GuqE71hJomoLusApIkNVvArj2YAy?usp=sharing

Code:

Q-Learing:

```
[8] import gym
    import numpy as np

    # Initialize the environment
    env = gym.make('CartPole-v1')

    # Q-learning parameters
    alpha = 0.1  # Learning rate
    gamma = 0.99  # Discount factor
    epsilon = 0.1  # Exploration rate
    num_episodes = 1000

    # Discretization parameters
    num_bins = 10
    state_bins = [np.linspace(env.observation_space.low[i], env.observation_space.high[i], num_bins + 1)[1:-1] for i in range(env.observation_space.shape[0])]
    action_space_size = env.action_space.n

    # Initialize Q-table
    q_table_shape = tuple([num_bins] * env.observation_space.shape[0]) + (action_space_size,)
    q_table = np.zeros(q_table_shape)

    # Function to discretize the continuous state
    def discretize_state(state):
        discrete_state = tuple(np.digitize(state[i], state_bins[i]) for i in range(env.observation_space.shape[0]))
        return discrete_state

    # Q-learning algorithm
    for episode in range(num_episodes):
        state = env.reset()
        discrete_state = discretize_state(state)
        done = False
```

```
        while not done:
            # Exploration-exploitation trade-off
            if np.random.uniform(0, 1) < epsilon:
                action = env.action_space.sample()  # Explore action space
            else:
                action = np.argmax(q_table[discrete_state])  # Exploit learned values

            # Take action and observe next state and reward
            next_state, reward, done, _ = env.step(action)
            discrete_next_state = discretize_state(next_state)

            # Update Q-table
            q_table[discrete_state + (action,)] += alpha * (reward + gamma * np.max(q_table[discrete_next_state]) - q_table[discrete_state + (action,)])

            # Transition to next state
            discrete_state = discrete_next_state

        # Print progress
        if (episode + 1) % 100 == 0:
            print("Episode {}/{}".format(episode + 1, num_episodes))

    # Evaluate the agent
    total_reward = 0
    num_episodes_eval = 100
    for _ in range(num_episodes_eval):
        state = env.reset()
        discrete_state = discretize_state(state)
        done = False
        while not done:
            action = np.argmax(q_table[discrete_state])
            next_state, reward, done, _ = env.step(action)
            total_reward += reward
            discrete_state = discretize_state(next_state)
```

```
# Calculate average reward
average_reward = total_reward / num_episodes_eval
print("Average reward over {} episodes: {}".format(num_episodes_eval, average_reward))

# Close the environment
env.close()
```

```
Episode 100/1000
Episode 200/1000
Episode 300/1000
Episode 400/1000
Episode 500/1000
Episode 600/1000
Episode 700/1000
Episode 800/1000
Episode 900/1000
Episode 1000/1000
Average reward over 100 episodes: 96.76
```

Sarsa:

```
import gym
import numpy as np

# Initialize the environment
env = gym.make('CartPole-v1')

# SARSA parameters
alpha = 0.1  # Learning rate
gamma = 0.99  # Discount factor
epsilon = 0.1  # Exploration rate
num_episodes = 1000

# Discretization parameters
num_bins = 10
state_bins = [np.linspace(env.observation_space.low[i], env.observation_space.high[i], num_bins + 1)[1:-1] for i in range(env.observation_space.shape[0])]
action_space_size = env.action_space.n

# Initialize Q-table
q_table_shape = tuple([num_bins] * env.observation_space.shape[0]) + (action_space_size,)
q_table = np.zeros(q_table_shape)

# Function to discretize the continuous state
def discretize_state(state):
    discrete_state = tuple(np.digitize(state[i], state_bins[i]) for i in range(env.observation_space.shape[0]))
    return discrete_state

# SARSA algorithm
for episode in range(num_episodes):
    state = env.reset()
    discrete_state = discretize_state(state)
    done = False
```

```
    # Choose action epsilon-greedily
    if np.random.uniform(0, 1) < epsilon:
        action = env.action_space.sample()  # Explore action space
    else:
        action = np.argmax(q_table[discrete_state])  # Exploit learned values

    while not done:
        # Take action and observe next state and reward
        next_state, reward, done, _ = env.step(action)
        discrete_next_state = discretize_state(next_state)

        # Choose next action epsilon-greedily
        if np.random.uniform(0, 1) < epsilon:
            next_action = env.action_space.sample()  # Explore action space
        else:
            next_action = np.argmax(q_table[discrete_next_state])  # Exploit learned values

        # Update Q-table
        q_table[discrete_state + (action,)] += alpha * (reward + gamma * q_table[discrete_next_state + (next_action,)] - q_table[discrete_state + (action,)])

        # Transition to next state-action pair
        discrete_state = discrete_next_state
        action = next_action

    # Print progress
    if (episode + 1) % 100 == 0:
        print("Episode {}/{}".format(episode + 1, num_episodes))
```

+ Code + Text

```
# Evaluate the agent
total_reward = 0
num_episodes_eval = 100
for _ in range(num_episodes_eval):
    state = env.reset()
    discrete_state = discretize_state(state)
    done = False
    while not done:
        action = np.argmax(q_table[discrete_state])
        next_state, reward, done, _ = env.step(action)
        total_reward += reward
        discrete_state = discretize_state(next_state)

# Calculate average reward
average_reward = total_reward / num_episodes_eval
print("Average reward over {} episodes: {}".format(num_episodes_eval, average_reward))

# Close the environment
env.close()
```

```
Episode 100/1000
Episode 200/1000
Episode 300/1000
Episode 400/1000
Episode 500/1000
Episode 600/1000
Episode 700/1000
Episode 800/1000
Episode 900/1000
Episode 1000/1000
Average reward over 100 episodes: 21.51
```