cs104-denis-practical-8-q-learning

October 3, 2024

[4]:
```python
import numpy as np
import random

# Environment: Simple Grid
class GridEnvironment:
    def __init__(self, grid_size=(4, 4), start=(0, 0), goal=(3, 3)):
        self.grid_size = grid_size
        self.start = start
        self.goal = goal
        self.state = start
        self.done = False

    def reset(self):
        self.state = self.start
        self.done = False
        return self.state

    def step(self, action):
        if self.done:
            raise Exception("Episode is done. Please reset the environment.")

        next_state = list(self.state)

        if action == 0:   # Up
            next_state[0] = max(0, self.state[0] - 1)
        elif action == 1: # Down
            next_state[0] = min(self.grid_size[0] - 1, self.state[0] + 1)
        elif action == 2: # Left
            next_state[1] = max(0, self.state[1] - 1)
        elif action == 3: # Right
            next_state[1] = min(self.grid_size[1] - 1, self.state[1] + 1)

        self.state = tuple(next_state)

        if self.state == self.goal:
            self.done = True
            reward = 1  # Reward for reaching the goal
```

```
        else:
            reward = -0.1  # Small penalty for each step

        return self.state, reward, self.done

    def render(self):
        for i in range(self.grid_size[0]):
            for j in range(self.grid_size[1]):
                if (i, j) == self.state:
                    print("A", end=" ")  # Agent position
                elif (i, j) == self.goal:
                    print("G", end=" ")  # Goal position
                else:
                    print(".", end=" ")  # Empty space
            print()  # New line after each row
        print()  # Extra line for better readability
```

[5]:
```
# Q-Learning Algorithm
class QLearningAgent:
    def __init__(self, actions, learning_rate=0.1, discount_factor=0.9,␣
 ↪exploration_rate=1.0, exploration_decay=0.99, min_exploration_rate=0.01):
        self.q_table = {}
        self.actions = actions
        self.learning_rate = learning_rate
        self.discount_factor = discount_factor
        self.exploration_rate = exploration_rate
        self.exploration_decay = exploration_decay
        self.min_exploration_rate = min_exploration_rate

    def get_q_value(self, state, action):
        return self.q_table.get((state, action), 0.0)

    def update_q_value(self, state, action, reward, next_state):
        best_next_q = max(self.get_q_value(next_state, a) for a in range(self.
 ↪actions))
        current_q = self.get_q_value(state, action)
        new_q = current_q + self.learning_rate * (reward + self.discount_factor␣
 ↪* best_next_q - current_q)
        self.q_table[(state, action)] = new_q

    def choose_action(self, state):
        if random.uniform(0, 1) < self.exploration_rate:
            return random.choice(range(self.actions))  # Explore
        else:
            return np.argmax([self.get_q_value(state, a) for a in range(self.
 ↪actions)])  # Exploit
```

```
    def decay_exploration(self):
        self.exploration_rate = max(self.min_exploration_rate, self.
 ↪exploration_rate * self.exploration_decay)
```

[6]:
```
# Main loop to run Q-learning
def train_agent(episodes=1000):
    env = GridEnvironment()
    agent = QLearningAgent(actions=4)

    for episode in range(episodes):
        state = env.reset()

        while True:
            action = agent.choose_action(state)
            next_state, reward, done = env.step(action)
            agent.update_q_value(state, action, reward, next_state)
            state = next_state

            if done:
                break

        agent.decay_exploration()

    return agent
```

[7]:
```
if __name__ == "__main__":
    trained_agent = train_agent(episodes=5000)

    # Test the trained agent
    env = GridEnvironment()
    state = env.reset()
    total_reward = 0

    while True:
        action = trained_agent.choose_action(state)
        state, reward, done = env.step(action)
        total_reward += reward
        env.render()  # Implement a render function to visualize the␣
 ↪environment if needed

        if done:
            print(f"Total Reward: {total_reward}")
            break
```

```
. A . .
. . . .
. . . .
```

```
. . . G

. . . .
. A . .
. . . .
. . . G

. . . .
. . . .
. A . .
. . . G

. . . .
. . . .
. . A .
. . . G

. . . .
. . . .
. . . .
. . A G

. . . .
. . . .
. . . .
. . . A

Total Reward: 0.5
```