machine learning lab homework 10

March 13, 2024

1 Machine Learning Lab HW 10

2 Connor O'Keefe

3 03/13/2024

4 1.) Define Enviornment

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import clear_output
import time

grid_size = 5
actions = ['up', 'down', 'left', 'right']
num_actions = len(actions)
agent_position = [0, 0]
goal_position = [4, 4]

# Rewards
rewards = {'goal': 1, 'other': -0.01}  # Minor negative reward to encourage␣
 ↪exploration

# Initialize Q-table
Q_table = np.zeros((grid_size, grid_size, num_actions))

# Learning parameters
learning_rate = 0.1
discount_factor = 0.95
episodes = 1000
epsilon = 0.1  # Exploration rate
```

putting an "agent" on a grid that has to figure out how to get to the end (G)

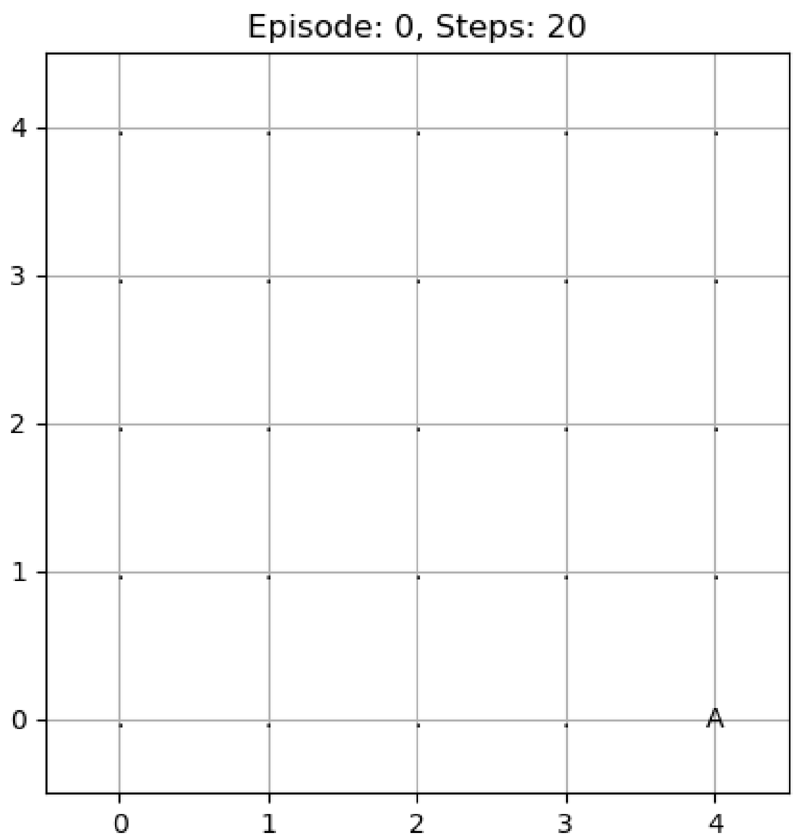
5 2.) Define Action Rewards

[2]:
```python
# Visualization setup
def plot_episode(steps, episode):
    clear_output(wait=True)
    plt.figure(figsize=(5, 5))
    plt.title(f"Episode: {episode}, Steps: {steps}")
    plt.xlim(-0.5, grid_size-0.5)
    plt.ylim(-0.5, grid_size-0.5)
    plt.grid()

    for i in range(grid_size):
        for j in range(grid_size):
            if [i, j] == agent_position:
                plt.text(j, grid_size-1-i, 'A', ha='center', va='center')
            elif [i, j] == goal_position:
                plt.text(j, grid_size-1-i, 'G', ha='center', va='center')
            else:
                plt.text(j, grid_size-1-i, '.', ha='center', va='center')

    plt.show()

def move_agent(agent_position, action_index):
    if actions[action_index] == 'up' and agent_position[0] > 0:
        agent_position[0] -= 1
    elif actions[action_index] == 'down' and agent_position[0] < grid_size - 1:
        agent_position[0] += 1
    elif actions[action_index] == 'left' and agent_position[1] > 0:
        agent_position[1] -= 1
    elif actions[action_index] == 'right' and agent_position[1] < grid_size - 1:
        agent_position[1] += 1
    return agent_position

def get_reward(agent_position):
    if agent_position == goal_position:
        return rewards['goal']
    else:
        return rewards['other']
```

6 3.) Implement Basic Q learning

[4]:
```python
for episode in range(episodes):
    agent_position = [0, 0]  # Reset position at start of each episode
    steps = 0

    while agent_position != goal_position:
```

```
        steps += 1
        if np.random.rand() < epsilon:  # Explore
            action = np.random.randint(num_actions)
        else:  # Exploit
            action = np.argmax(Q_table[agent_position[0], agent_position[1], :])

        old_position = list(agent_position)
        new_position = move_agent(list(agent_position), action)
        reward = get_reward(new_position)

        # Update Q-table
        old_q_value = Q_table[old_position[0], old_position[1], action]
        future_q_value = np.max(Q_table[new_position[0], new_position[1], :])
        Q_table[old_position[0], old_position[1], action] = old_q_value + learning_rate * (reward + discount_factor * future_q_value - old_q_value)

        agent_position = new_position

        # Visualization every 100 episodes
        if episode % 100 == 0:
            plot_episode(steps, episode)
            time.sleep(0.1)  # Slow down the visualization

    if steps <= grid_size * 2:  # Early stop if it finds a reasonably good path
        break
```



The algorithm originally took 116 steps to reach the end, but it re-uses the Q-table, so after running it again, it only took 20 steps to reach the end.