Московский государственный технический университет им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

Лабораторная работа №4

по дисциплине

«Методы машинного обучения»

Выполнила:

студентка группы ИУ5И-23М

Цзян Юхуэй

Москва — 2024 г.

Задание

На основе рассмотренного на лекции примера реализуйте алгоритм Policy Iteration для любой среды обучения с подкреплением (кроме рассмотренной на лекции среды Toy Text / Frozen Lake) из библиотеки Gym (или аналогичной библиотеки).

Основной раздел кода

Загрузка библиотек функций и определений функций:

```
import numpy as np

import gym

import matplotlib.pyplot as plt


def policy_evaluation(env, policy, gamma=0.99, theta=1e-6):

    V = np.zeros(env.nS)

    V_history = [np.copy(V)]  # 初始化值函数历史

    while True:

        delta = 0

        for s in range(env.nS):

            v = 0

            for a, action_prob in enumerate(policy[s]):

                for prob, next_state, reward, done in env.P[s][a]:

                    v += action_prob * prob * (reward + gamma * V[next_state])

            delta = max(delta, np.abs(v - V[s]))

            V[s] = v

        V_history.append(np.copy(V))

        if delta < theta:

            break

    return V, V_history

def policy_improvement(env, V, gamma=0.99):

    policy = np.zeros([env.nS, env.nA]) / env.nA

    for s in range(env.nS):

        q_values = np.zeros(env.nA)

        for a in range(env.nA):

            for prob, next_state, reward, done in env.P[s][a]:

                q_values[a] += prob * (reward + gamma * V[next_state])

        best_a = np.argmax(q_values)
```

```
        policy[s][best_a] = 1.0
    return policy
def policy_iteration(env, gamma=0.99):
    policy = np.ones([env.nS, env.nA]) / env.nA  # 初始化随机策略
    V_history_total = []
    while True:
        V, V_history = policy_evaluation(env, policy, gamma)
        V_history_total.extend(V_history)
        new_policy = policy_improvement(env, V, gamma)
        if np.array_equal(new_policy, policy):
            break
        policy = new_policy
    return policy, V, V_history_total
def discretize_state(state, bins):
    return tuple(np.digitize(s, b) for s, b in zip(state, bins))
def run_policy(env, policy, state_bins, num_bins):
    state = env.reset()
    state = discretize_state(state, state_bins)
    total_reward = 0
    frames = []
    while True:
        action = np.argmax(policy[sum([state[i] * (num_bins ** i) for i in range(len(state))])])
        next_state, reward, done, _ = env.step(action)
        frames.append(env.render(mode='rgb_array'))
        total_reward += reward
        state = discretize_state(next_state, state_bins)
        if done:
            break
    return frames, total_reward
```

Настройте среду и создайте оптимальную политику:

```
# 创建环境
env_name = 'CartPole-v1'
env = gym.make(env_name)
# CartPole-specific 参数
num_bins = 10
state_bins = [np.linspace(-4.8, 4.8, num_bins), np.linspace(-4, 4, num_bins), np.linspace(-.418, .418, num_bins), np.linspace(-4, 4, num_bins)]
# 离散化环境
env.nS = num_bins ** env.observation_space.shape[0]
env.nA = env.action_space.n
env.P = {s: {a: [] for a in range(env.nA)} for s in range(env.nS)}
# 填充环境 P
for s in range(env.nS):
    state = np.array([s % num_bins, (s // num_bins) % num_bins, (s // (num_bins ** 2)) % num_bins, (s // (num_bins ** 3)) % num_bins])
    state = [state_bins[i][j] for i, j in enumerate(state)]
    for a in range(env.nA):
        env.reset()
        env.env.state = state
        next_state, reward, done, _ = env.step(a)
        next_state = discretize_state(next_state, state_bins)
        next_s = sum([next_state[i] * (num_bins ** i) for i in range(len(next_state))])
        env.P[s][a].append((1.0, next_s, reward, done))
# 运行策略迭代算法
optimal_policy, optimal_value_function, V_history_total = policy_iteration(env)
# 打印最优策略和值函数
print("最优策略：")
print(optimal_policy)
```

```
print("最优值函数：")

print(optimal_value_function)

# 运行最优策略并生成图像

frames, total_reward = run_policy(env, optimal_policy, state_bins, num_bins)

# 生成折线图，限制显示的状态数量

plt.figure(figsize=(10, 6))

for i in range(min(env.nS, 10)):  # 这里只显示前 10 个状态

    plt.plot([V[i] for V in V_history_total], label=f'State {i}')

plt.xlabel('Iterations')

plt.ylabel('Value')

plt.title('Value Function Convergence')

plt.legend(loc='best')

plt.show()

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

Результат

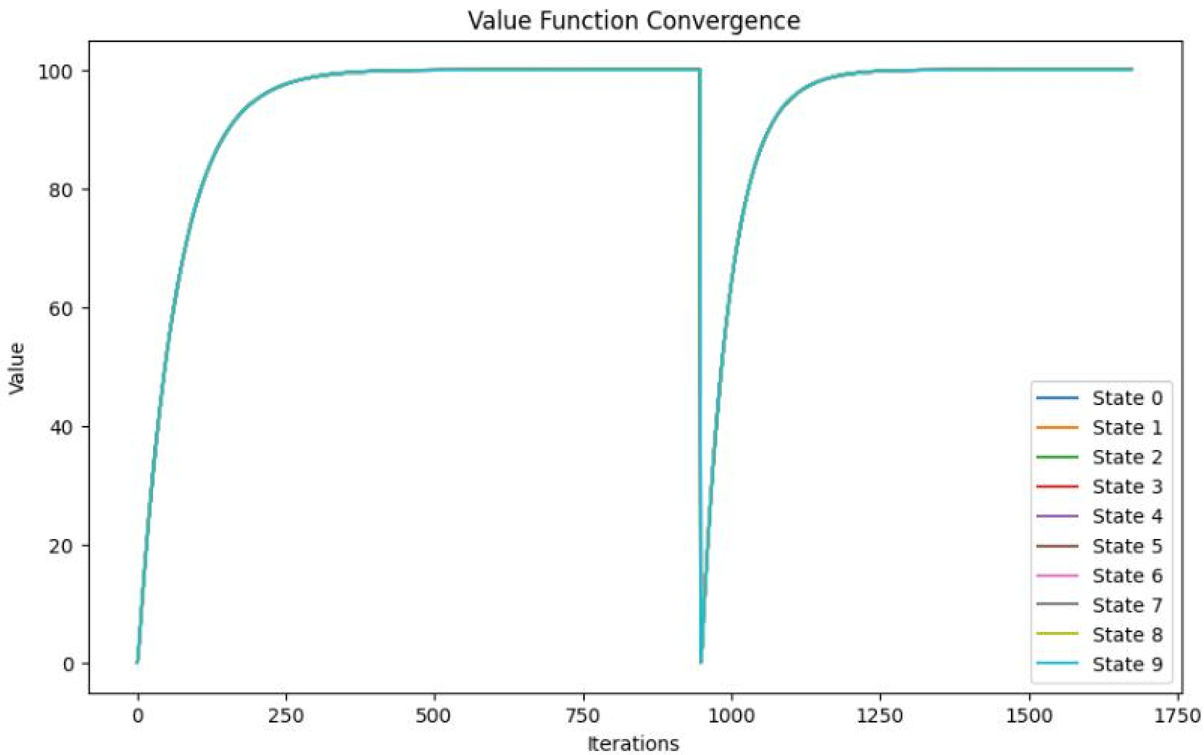

Рис 1. Сходимость функции стоимости.