**Gridworld 强化学习实践报告**

**1. 项目概述**

本项目旨在实现一个基于Gridworld环境的强化学习agent。通过这个实践，我们可以加深对强化学习基本概念的理解，并掌握如何将理论应用到实际问题中。

**2. 环境设计**

Gridworld被设计为一个5x5的网格，共25个状态。agent可以在每个状态执行上、下、左、右四种动作。大多数状态的奖励为0，但有两个特殊状态A和B，分别提供+10和+5的奖励。如果agent试图移动到网格外，会得到-1的惩罚并保持原位。

**3. 算法设计与实现**

**3.1 算法选择**

我们选择使用Q-learning算法来训练agent。Q-learning是一种无模型（model-free）的强化学习算法，它通过学习动作价值函数（Q函数）来优化策略。

**3.2 算法原理**

Q-learning的核心思想是迭代更新Q值表。Q值表存储了在每个状态下采取每个动作的预期累积奖励。更新公式如下：

Q(s,a) ← Q(s,a) + α[r + γ \* max(Q(s',a')) - Q(s,a)]

其中：

* s, a 是当前的状态和动作
* s', a' 是下一个状态和可能的动作
* α 是学习率
* γ 是折扣因子
* r 是即时奖励

**3.3 实现过程**

1. 初始化Q值表为零矩阵。
2. 对于每个episode：  
   a. 随机选择初始状态  
   b. 当未达到终止条件时：
   * 使用ε-greedy策略选择动作
   * 执行动作，观察奖励和下一个状态
   * 更新Q值
   * 移动到下一个状态
3. 重复步骤2直到完成指定的episode数量。

源代码：

import numpy as np

from collections import defaultdict, deque

from tqdm import tqdm

**class** QLearning:

**def** \_\_init\_\_(self, actions, learning\_rate=0.1, discount\_factor=0.99, epsilon=1.0, epsilon\_decay=0.9995, epsilon\_min=0.01):

        self.actions = actions

        self.lr = learning\_rate

        self.gamma = discount\_factor

        self.epsilon = epsilon

        self.epsilon\_decay = epsilon\_decay

        self.epsilon\_min = epsilon\_min

        self.q\_table = defaultdict(**lambda**: np.zeros(len(actions)))

        self.experience\_replay = deque(maxlen=1000)

**def** get\_action(self, state):

        if np.random.uniform() < self.epsilon:

            return np.random.choice(self.actions)

        else:

            return self.actions[np.argmax(self.q\_table[tuple(state)])]

**def** update(self, state, action, reward, next\_state):

        self.experience\_replay.append((state, action, reward, next\_state))

        if len(self.experience\_replay) >= 32:

            self.batch\_update()

        self.epsilon = max(self.epsilon\_min, self.epsilon \* self.epsilon\_decay)

**def** batch\_update(self):

        mini\_batch = np.random.choice(len(self.experience\_replay), 32, replace=False)

        for idx in mini\_batch:

            state, action, reward, next\_state = self.experience\_replay[idx]

            current\_q = self.q\_table[tuple(state)][self.actions.index(action)]

            next\_max\_q = np.max(self.q\_table[tuple(next\_state)])

            new\_q = current\_q + self.lr \* (reward + self.gamma \* next\_max\_q - current\_q)

            self.q\_table[tuple(state)][self.actions.index(action)] = new\_q

**class** Gridworld:

**def** \_\_init\_\_(self):

        self.height = 5

        self.width = 5

        self.state = [0, 0]

        self.actions = ['up', 'down', 'left', 'right']

        self.special\_states = {

            'A': {'position': [1, 1], 'reward': 20},

            'B': {'position': [3, 3], 'reward': 10}

        }

        self.max\_steps = 50

        self.visited\_special = set()

**def** reset(self):

        self.state = [np.random.randint(0, self.height), np.random.randint(0, self.width)]

        self.steps = 0

        self.visited\_special = set()

        return self.state

**def** step(self, action):

        self.steps += 1

        next\_state = self.state.copy()

        if action == 'up':

            next\_state[0] = max(0, next\_state[0] - 1)

        elif action == 'down':

            next\_state[0] = min(self.height - 1, next\_state[0] + 1)

        elif action == 'left':

            next\_state[1] = max(0, next\_state[1] - 1)

        elif action == 'right':

            next\_state[1] = min(self.width - 1, next\_state[1] + 1)

        reward = -0.1

        for special\_name, special in self.special\_states.items():

            if tuple(next\_state) == tuple(special['position']) and special\_name not in self.visited\_special:

                reward = special['reward']

                self.visited\_special.add(special\_name)

                break

        self.state = next\_state

        done = self.steps >= self.max\_steps or len(self.visited\_special) == len(self.special\_states)

        return self.state, reward, done

**def** train\_agent(env, agent, episodes=50000):

    for episode in tqdm(range(episodes), desc="Training"):

        state = env.reset()

        done = False

        total\_reward = 0

        while not done:

            action = agent.get\_action(state)

            next\_state, reward, done = env.step(action)

            agent.update(state, action, reward, next\_state)

            state = next\_state

            total\_reward += reward

        if (episode + 1) % 1000 == 0:

            print(**f**"\nEpisode {episode + 1}, Total Reward: {total\_reward**:.2f**}")

**def** test\_agent(env, agent, episodes=100):

    total\_rewards = 0

    for episode in tqdm(range(episodes), desc="Testing"):

        state = env.reset()

        done = False

        episode\_reward = 0

        while not done:

            action = agent.get\_action(state)

            next\_state, reward, done = env.step(action)

            state = next\_state

            episode\_reward += reward

        total\_rewards += episode\_reward

    average\_reward = total\_rewards / episodes

    print(**f**"\nAverage Reward over {episodes} episodes: {average\_reward**:.2f**}")

*# 使用示例*

env = Gridworld()

agent = QLearning(env.actions)

*# 训练智能体*

print("Training agent...")

train\_agent(env, agent)

*# 测试智能体*

print("\nTesting agent...")

test\_agent(env, agent)

**4. 实验结果与分析**

我们训练了50,000个episode，每1000个episode记录一次总奖励。以下是部分训练输出：

Training: 2%|███▊ | 993/50000 [00:08<08:24, 97.11it/s]

Episode 1000, Total Reward: 15.10

...

Training: 100%|█████████████████████| 50000/50000 [06:02<00:00, 137.78it/s]

Testing agent...

Testing: 100%|█████████████████████| 100/100 [00:00<00:00, 3839.60it/s]

Average Reward over 100 episodes: 16.98

实验结果截图：



分析：

1. 训练过程中，奖励在5.10到29.70之间波动，大部分时间维持在15.10左右。
2. 训练速度相对较快，50,000个episode用时约6分钟。
3. 测试阶段的平均奖励为16.98，表明agent学到了一定的策略，但可能还未达到最优。

**5. 遇到的困难与解决方案**

1. 奖励波动大：我们发现奖励在训练过程中波动较大。为了解决这个问题，我们尝试调整了学习率和探索率，以平衡探索和利用。
2. 性能瓶颈：初始实现较慢。我们通过使用NumPy进行矩阵运算来优化代码，显著提高了训练速度。

**6. 改进方向**

1. 参数调优：可以进一步调整学习率、折扣因子和探索率，以获得更好的性能。
2. 算法升级：考虑使用更先进的算法，如DQN或Actor-Critic方法。
3. 环境复杂化：增加障碍物或动态元素，使环境更具挑战性。

**7. 结论**

通过这个项目，我们成功实现了一个基于Q-learning的Gridworld agent。虽然agent的表现还有提升空间，但这次实践帮助我们深入理解了强化学习的核心概念和实现挑战。未来，我们将继续优化算法，探索更复杂的环境和任务。