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：

$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * \max(Q(s',a')) - Q(s,a)]$

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.inde
x(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):
```

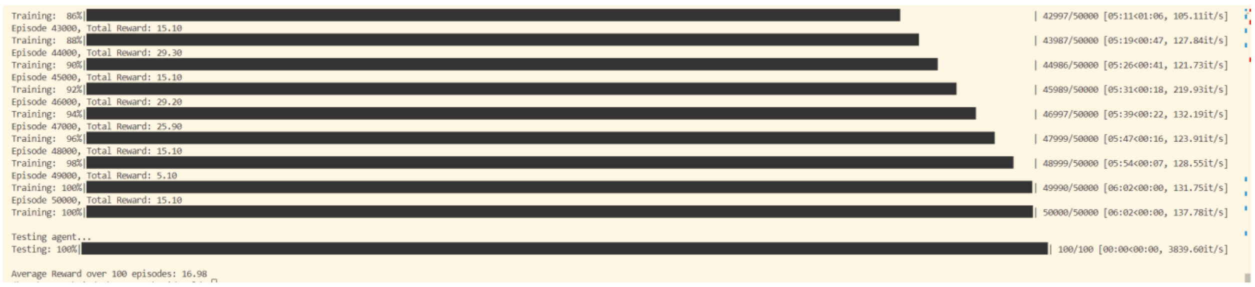
```python
    total_rewards = 0
    for episode in tqdm(range(episodes),
desc="Testing"):
        state = env.reset()
        done = False
        episode_reward = 0

        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：



```
Training:  86%|█████████████████                | 42997/50000 [05:11<01:06, 105.11it/s]
Episode 43000, Total Reward: 15.10
Training:  88%|█████████████████▌               | 43987/50000 [05:19<00:47, 127.84it/s]
Episode 44000, Total Reward: 29.30
Training:  90%|██████████████████               | 44986/50000 [05:26<00:41, 121.73it/s]
Episode 45000, Total Reward: 15.10
Training:  92%|██████████████████▍              | 45989/50000 [05:31<00:18, 219.93it/s]
Episode 46000, Total Reward: 29.20
Training:  94%|██████████████████▊              | 46997/50000 [05:39<00:22, 132.19it/s]
Episode 47000, Total Reward: 25.90
Training:  96%|███████████████████▏             | 47999/50000 [05:47<00:16, 123.91it/s]
Episode 48000, Total Reward: 15.10
Training:  98%|███████████████████▌             | 48999/50000 [05:54<00:07, 128.55it/s]
Episode 49000, Total Reward: 5.10
Training: 100%|████████████████████| 49990/50000 [06:02<00:00, 131.75it/s]
Episode 50000, 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
```

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。