时序差分算法实验报告

姓名 周雨扬 学号 2000013061

## 一、实验要求

（1）阅读《基于时序差分的路径规划-实验指导书》，尝试运行并理解时序差分算法在冰湖路径规划问题上的示例代码。

（2）在示例代码的基础上，尝试实现n-step TD算法，比较并分析n的不同取值对策略收敛速度的影响（由于随机性建议多次实验）。

（3）在示例代码的基础上，尝试实现TD(λ)算法，比较并分析λ的不同取值对策略收敛速度的影响（由于随机性建议多次实验）。

## 二、代码修改思路

### Nstep-TD

|  |
| --- |
| state = env.reset()  epsilon = eps0 / (1 + decay \* i\_episode)  episode = []  # 采样  while True:  action = epsilon\_greedy(Q, state, env.action\_space.n, epsilon)  next\_state, reward, done, \_ = env.step(action)  episode.append((state, action, reward))  if done:  episode.append((next\_state, None, reward))  break  state = next\_state  G = 0  episode\_len = len(episode)  td\_target = [0 for \_ in range(episode\_len)]  # 计算 Q 函数应当靠近的目标 td\_target  # G 存储了考虑至多 n 步之后，不考虑最后一步后补足的 Q 函数的贡献，动作序列的权值  for j in reversed(range(episode\_len)):  state, action, reward = episode[j]  G = gamma \* G + reward  td\_target[j] = G  if action == None:  continue  if j + n < episode\_len:  past\_state, past\_action, past\_reward = episode[j + n]  G = G - (gamma \*\* n) \* past\_reward  # 长度溢出，移除末尾元素  td\_target[j] = G  if past\_action != None:  # Not Done  td\_target[j] = G + (gamma \*\* n) \* Q[past\_state, past\_action]  # 更新  for j in reversed(range(episode\_len)):  state, action, reward = episode[j]  if action != None:  Q[state][action] += alpha \* (td\_target[j] - Q[state][action]) |

### TD(λ)

|  |
| --- |
| state = env.reset()  epsilon = eps0 / (1 + decay \* i\_episode)  if i\_episode % 1000 == 999:  print(f"iteration: {i\_episode}, epsilon: {epsilon}")  episode = []  # 采样  while True:  action = epsilon\_greedy(Q, state, env.action\_space.n, epsilon)  next\_state, reward, done, \_ = env.step(action)  episode.append((state, action, reward))  if done:  episode.append((next\_state, None, reward))  break  state = next\_state  episode\_len = len(episode)  for i in range(0, episode\_len):  state, action, reward = episode[i]  if action == None:  continue  target\_td = 0  G = reward  # G 存储了不考虑最后一步补足的Q function 的权重，i:i+n 的动作权值。  for n in range(1, episode\_len - i):  past\_state, past\_action, past\_reward = episode[i + n]  target\_td\_n = G  if past\_action != None:  target\_td\_n += (gamma \*\* n) \* Q[past\_state, past\_action]  # 根据 lmd 加权计算路径权值和，作为 Q 函数的目标  target\_td += (lmd \*\* (n - 1)) \* target\_td\_n  G = G + (gamma \*\* n) \* past\_reward  target\_td \*= 1 - lmd  target\_td += (lmd \*\* (episode\_len - i - 1)) \* reward  delta = target\_td - Q[state, action]  Q[state, action] += alpha \* delta |

## 三、实验结果和分析

### n-step TD实验

测试采用我自己编写的 n-step TD 算法，运行轮数 10000 轮，更新率 alpha = 0.1, 取 5 轮实验的平均值。

观察发现 n-step TD 算法在 n=1 的时候即为 sarsa 算法，两者实验结果也相同。随着 n 的增大，最终优化的结果也会越来越差。根据网上查询的资料，n-stepTD 算法成功率也与更新率 alpha 相关。n 较大的时候较小的 alpha 容易取得较好的结果。这也与我们的实验结果吻合。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 算法 | n | Alpha | 运行时间 | 成功率 |
| Sarsa | / | 0.1 | 3.67 | 0.571 |
| n-step TD | 1 | 0.1 | 3.67 | 0.714 |
| n-step TD | 2 | 0.1 | 3.42 | 0.476 |
| n-step TD | 3 | 0.1 | 3.05 | 0.524 |
| n-step TD | 4 | 0.1 | 2.72 | 0.351 |
| n-step TD | 4 | 0.01 | 3.06 | 0.654 |

### TD-λ 实验

测试采用我自己编写的 TD-λ 算法，运行轮数 10000 轮，更新率 alpha = 0.1, 衰减速率γ = 0.95，取 5 轮实验的平均值。

测试结果表示，其余参数固定的情况下，λ 越大，预测的 Q 函数方差也就越大，结果很可能会变得更差。值得注意的是，即使 λ 取到 0.7，其仍然有能力优化出正确率超过 0.7 的最优解，但是由于很多情况下其找到的都是一些非常之差的解，因此平均正确率还是被拉低了。

因此，在 λ 较大的时候我们需要通过调整其余参数，例如调低 alpha，调低 γ 的方式降低预测的误差，从而提升学习的效果。

|  |  |  |
| --- | --- | --- |
| λ | 运行时间 | 成功率 |
| 0 | 5.52 | 0.725 |
| 1 | 5.14 | 0.707 |
| 2 | 5.13 | 0.648 |
| 3 | 4.91 | 0.709 |
| 0.5 | 4.75 | 0.514 |
| 0.7 | 3.86 | 0.363 |