编程作业3实验报告

人工智能学院 181220076 周韧哲

用Python或者C++实现Q学习、Sarsa算法。复现这两个算法在悬崖行走问题上的实验结果。实现n步Sarsa和 Sarsa(λ)算法，在悬崖行走问题上比较n步Sarsa和Sarsa(λ) 的实验结果，其中，n=1,3,5, λ =0,0.5,1。

- **代码架构**：
  - 悬崖行走环境调用了`gym`库的`CliffWalking-v0`。在`Sarsa_QLearning.py`中有四个class：`Sarsa`，`QLearning`，`NStep_Sarsa`，`Sarsa_Lambda`，初始化后调用`train`函数就可以实现Agent的训练。`test1`，`test2`函数分别比较了`Sarsa`和`QLearning`算法与`NStep_Sarsa`和`Sarsa_Lambda`算法的结果，并绘制了rewards和episode图。
- **算法实现**：(具体实现详见`Sarsa_QLearning.py`文件)

  每个算法的初始化中都有`alpha,epsilon,gamma`，还有`action_num`和`state_num`分别表示动作数和状态数，其中`NStep_Sarsa`多了一个参数`N`，`Sarsa_Lambda`多了一个参数`Lambda`。并且用了一个数组来保存Q矩阵，每一行代表一个state的各个动作的Q值。四个class都有一个函数`policy(state)`，它会根据$\epsilon-greedy$策略返回一个动作。
  - `Sarsa`
    - 训练过程如下：

```
for episode in range(iter_num):
  cur_state = self.env.reset()
  cur_action = self.policy(cur_state)
  done = False
  while not done:
    obs,reward,done,info = self.env.step(cur_action)
    next_action = self.policy(obs)
    '''使用下一个状态-动作对的Q值更新当前Q值'''
    self.Q[cur_state,cur_action] += self.alpha*(reward+self.gamma *
                                    self.Q[obs,next_action] -
                                    self.Q[cur_state,cur_action])
    cur_state, cur_action = obs, next_action
```

  - `QLearning`
    - 训练与`Sarsa`类似

```
for episode in range(iter_num):
  state = self.env.reset()
  done = False
  while not done:
    action = self.policy(state)
    obs,reward,done,info = self.env.step(action)
    '''使用最大化下一个状态Q值的行动来更新当前Q值'''
    self.Q[state,action] +=self.alpha * (reward + self.gamma *
                           np.max(self.Q[obs])-self.Q[state,action]
    state = obs
```

  - `NStep_Sarsa`

- 训练过程如下：

```
for episode in range(iter_num):
  state_list,action_list,reward_list = [],[],[0] #用来存储每一步的结果
  state = self.env.reset()
  action = self.policy(state)
  state_list.append(state)
  action_list.append(action)
  T = np.Infinity
  t = 0
  while True:
    if t<T:
      obs,reward,done,info = self.env.step(action_list[-1])
      state_list.append(obs)
      reward_list.append(reward)
      if done:
        T = t+1
      else:
        action_list.append(self.policy(state_list[-1]))
    tau = t-self.N+1
    if tau>=0:
      G = 0
      '''计算n步Sarsa更新的目标值'''
      for i in range(tau+1,min(tau+self.N,T)+1):
        G += self.gamma**(i-tau-1)*reward_list[i]
        if tau+self.N < T:
          s, a = state_list[tau+self.N], action_list[tau+self.N]
          G += self.gamma**self.N * self.Q[s,a]
        s, a = state_list[tau], action_list[tau]
        '''n步Sarsa更新'''
        self.Q[s,a] += self.alpha * (G - self.Q[s,a])
    if tau==T-1:
      break
    t += 1
```

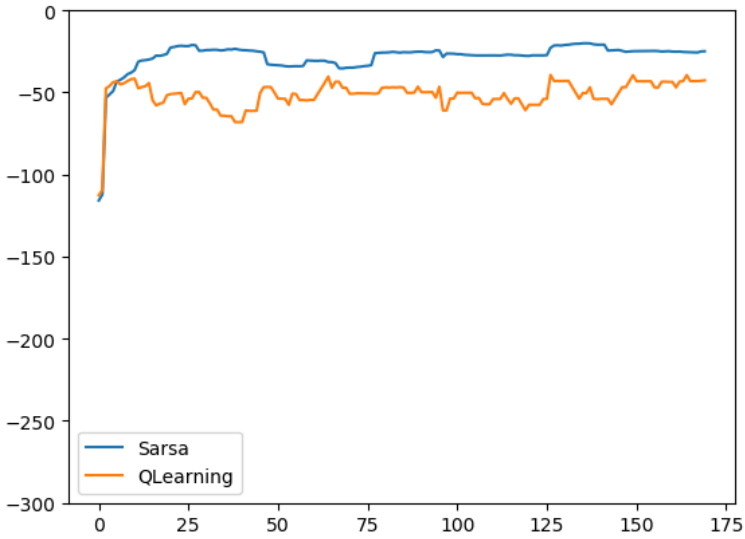
- `Sarsa_Lambda`
  - 训练过程如下：

```
for episode in range(iter_num):
  Z = np.zeros((self.state_num,self.action_num))   #初始化资格迹
  cur_state, cur_action = self.env.reset(),
self.env.action_space.sample()
  done = False
  while not done:
    obs,reward,done,info = self.env.step(cur_action)
    next_action = self.policy(obs)
    '''计算TD误差'''
    TD_error = reward + self.gamma * self.Q[obs,next_action] -
               self.Q[cur_state,cur_action]
    '''更新资格迹'''
    Z[cur_state,cur_action] += 1
    for s in range(self.state_num):
      for a in range(self.action_num):
        self.Q[s,a] += self.alpha*TD_error*Z[s,a] #更新值函数
        Z[s,a] *= self.gamma*self.Lambda  #更新资格迹
    cur_state, cur_action = obs, next_action
```

- **实验结果**:

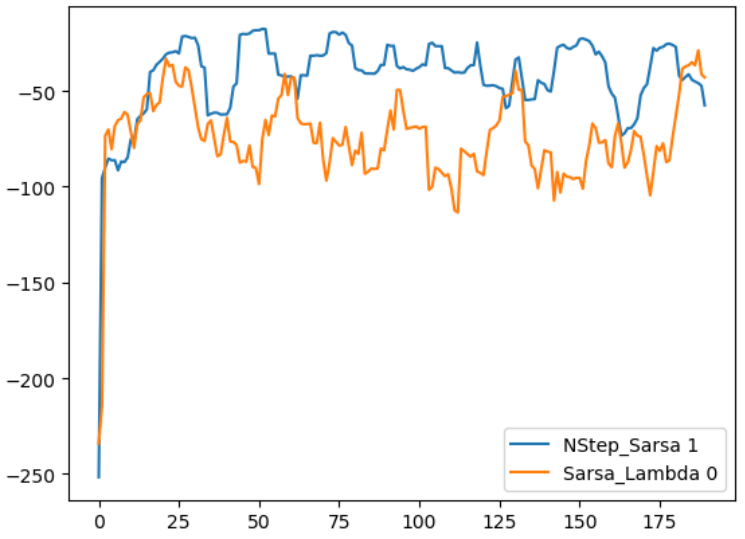
  运行 `test1` 函数比较 `Sarsa` 和 `QLearning` 算法，运行 `test2` 函数比较 `NStep_Sarsa` 和 `Sarsa_Lambda` 算法。

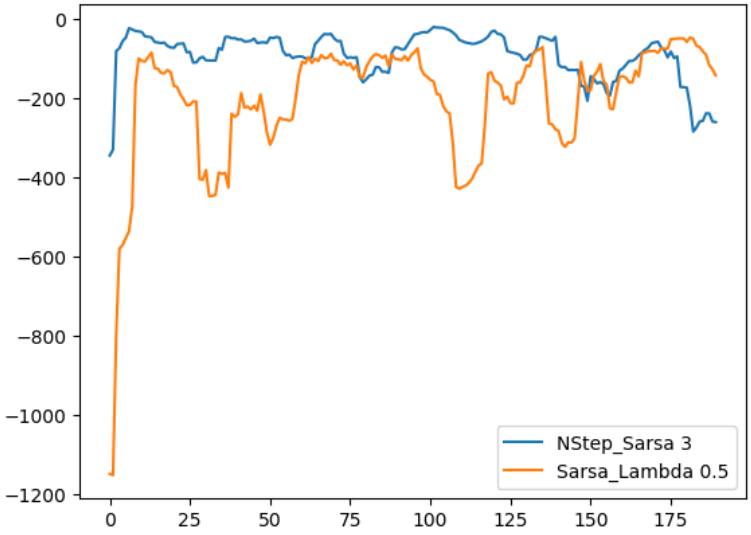
  - 在Sarsa和QLearning算法的比较中，设置参数 `alpha=0.9,epsilon=0.1,gamma=0.8`，得到

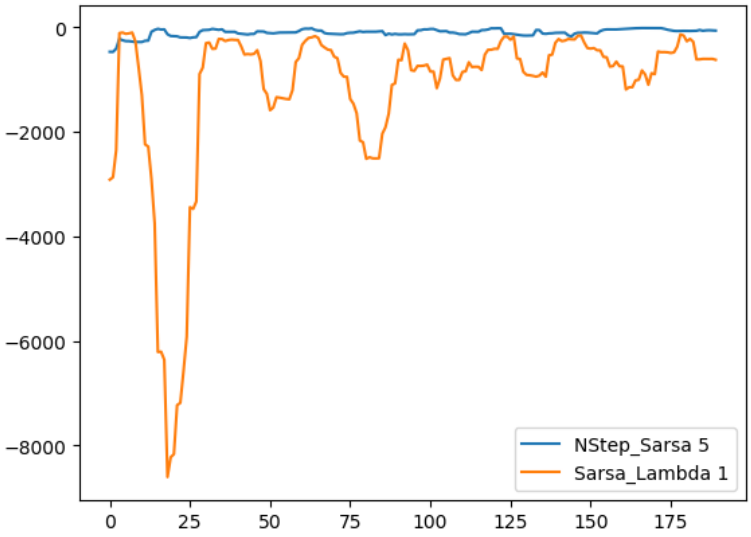


    可以看出Sarsa的10情节平均无折扣回报之和比QLearning的高一些，前者稳定在-25左右，后者稳定在-50左右，复现了书中的结果。在可视化观察中，我也观察到Sarsa算法得出的是一条安全路径，它会走到远离悬崖那侧然后走到终点，而QLearning算法得出的是一条最短路径，它会贴着悬崖到达终点。

  - 在NStep_Sarsa和Sarsa_Lambda算法的比较中，设置参数 `alpha=1,epsilon=0.1,gamma=0.9`，得到







可以看出，随着N的增大，NStep_Sarsa学习的速度增加，很快就能收敛；NStep_Sarsa基本上都比Sarsa_Lambda的结果要好一些。$N=1$，$Lambda=0$时其实就是最初的Sarsa算法，所以两者的结果是十分相近的。$N=3$，$Lambda=0.5$时两者差别也不大。$N=5$，$Lambda=1$时Sarsa_Lambda变为了回合更新，Sarsa_Lambda的结果较为不稳定，但最后也会趋近于一个值附近。