QLearning算法实现悬崖寻路的问题

作者：LDJ

$QLearning$介绍

在开始之前我们首先需要介绍一下什么是$QLearning$,$QLearning$算法其实本质上是叫做异策略时序差分控制。

那么什么是异策略时序差分控制呢？

异策略在学习的过程中，有两种不同的策略：目标策略（target policy）和行为策略（behavior policy）。第一个策略是我们需要去学习的策略，即目标策略，一般用 π 来表示。目标策略就像是在后方指挥战术的一个军师，它可以根据自己的经验来学习最优的策略，不需要去和环境交互。

另外一个策略是探索环境的策略，即行为策略，一般用 μ 来表示。μ 可以大胆地去探索到所有可能的轨迹，采集轨迹，采集数据，然后把采集到的数据喂给目标策略去学习。而且喂给目标策略的数据中并不需要A_{t+1}。行为策略像是一个战士，可以在环境里面探索所有的动作、轨迹和经验，然后把这些经验交给目标策略去学习。比如目标策略优化的时候，$QLearing$不会管你下一步去往哪里探索，它就只选收益最大的策略。

再举个例子。比如环境是一个波涛汹涌的大海，但学习策略（learning policy）太胆小了，没法直接跟环境去学习，所以我们有了探索策略（exploratory policy），探索策略是一个不畏风浪的海盗，他非常激进，可以在环境中探索。他有很多经验，可以把这些经验写成稿子，然后喂给学习策略。 学习策略可以通过稿子来进行学习。

Q 学习有两种策略：行为策略和目标策略。目标策略 π 直接在 Q 表格上使用贪心策略，就取它下一 步能得到的所有状态.

$$\pi(S_{t+1}) = \underset{a`}{arg\,max}\ Q(S_{t+1}, a`) \tag{1}$$

行为策略 μ 可以是一个随机的策略，但我们采取 ε-贪心，让行为策略不至于是完全随机的，它是基于 Q 表格逐渐改进的。

这里呈现一下Q_table的形状，因为基本上看一眼就解决了的：

Q表格指导每一步的动作

状态	上	下	左	右
坐标（1，1）	0	0	0	0
坐标（1，2）	0	0	0	0
坐标（1，3）	0	0	0	0
坐标（1，4）	0	0	0	0
坐标（1，5）	0	0	0	0
坐标（1，6）	0	0	0	0
…	…	…	…	…

每一行代表一个状态，每一列代表行动。

我们可以构造 Q-learning target，Q-learning 的 next action 都是通过 arg max 操作来选出来的，于是我们可以代入 arg max 操作，可以得到下式：

$$\begin{aligned} R_{t+1}+\gamma Q(S_{t+1},A^{`}) &= R_{t+1}+\gamma Q(S_{t+1},arg\ max\ Q(S_{t+1},a^{`})) \\ &= R_{t+1}+\gamma \underset{a^{`}}{max}\ Q(S_{t+1},a^{`}) \end{aligned} \tag{2}$$

接着我们可以把 Q 学习更新写成增量学习的形式，时序差分目标就变成 max 的值，如下式所示。

$$Q(S_t,A_t)\ \leftarrow\ Q(S_t,A_t)+\alpha[R_{t+1}+\gamma \underset{a}{max}\ Q(S_{t+1,a}-Q(S_t,A_t))] \tag{3}$$

悬崖寻路正题导入

前提：使用OpenAI Gym开发的$CliffWalking-v0$环境，实现$QLearning.$

我们首先简单介绍一下这个环境，该环境中文名叫悬崖寻路（$CliffWalking$），是一个迷宫类问题。如 图 3.37 所示，在一个 4×12 的网格中，智能体以网格的左下角位置为起点，以网格的下角位置为终点，目 标是移动智能体到达终点位置，智能体每次可以在上、下、左、右这 4 个方向中移动一步，每移动一步会 得到-1 单位的奖励。

起终点之间是一段悬崖，即编号为 37 到 46 的网格，智能体移动过程中会有如下的限制：（1）智能体不能移出网格，如果智能体想执行某个动作移出网格，那么这一步智能体不会移动，但是 这个操作依然会得到-1 单位的奖励；（2）如果智能体“掉入悬崖”，会立即回到起点位置，并得到-100 单位的奖励；（3）当智能体移动到终点时，该回合结束，该回合总奖励为各步奖励之和。 我们的目标是以最少的步数到达终点，容易看出最少需要 13 步智能体才能从起点到终点，因此最佳 算法收敛的情况下，每回合的总奖励应该是-13，这样人工分析出期望的奖励也便于我们判断算法的收敛 情况作出相应调整。现在我们可以在代码中定义环境，如下：

```
import gym
from gridworld_env import CliffwalkingWapper #导入自定义装饰器
env=gym.make("Cliffwalking-v0") #定义环境
env=CliffwalkingWapper(env) #装饰环境
```

这里我们在程序中使用了一个装饰器重新定义环境，具体的代码如下所示：

```
#定义我们的装饰器环境，首先是从gym中的wrapper中得到继承
class CliffwalkingWapper(gym.Wrapper):
    def __init__(self,env):
        gym.Wrapper.__init__(self,env)
        self.t=None #表示的是海龟画图的指针
        self.unit=50
        self.max_x=12
        self.max_y=4
'''
上面是定义类变量，主要有整个表格的长宽也就是最大值max_x,max_y;
'''
```

```
'''
下面的方法都是以海龟画图进行定义的
'''
    def draw_x_line(self,y,x0,x1,color="gray"):
        assert x1>x0 #assert的用法其实你可以当做是if的用法，主要作用是不满足条件便触发异常机制
        self.t.color(color) #将x_line设置为gray的颜色
        self.t.setheading(0)#设置海龟的朝向，0代表朝东
        self.t.up() #拿出笔或者说启动笔
        self.t.goto(x0,y)#定位到(x0,y)这个位置，进行一个划线
        self.t.down()#画笔落下
        self.t.forward(x1-x0)#朝着东方向进行一个划线，划线距离是x1-x0

    def draw_y_line(self,x,y0,y1,color="gray"):
        assert y1>y0
        self.t.color(color)
        self.t.setheading(90)#朝北
        self.t.up()
        self.t.goto(x,y0)
        self.t.down()
        self.t.forward(y1-y0)

    def draw_box(self,x,y,fillcolor="",line_color="gray"):
        self.t.up()
        self.t.goto(x*self.unit,y*self.unit)
        self.t.color(line_color)
        self.t.fillcolor(fillcolor)
        self.t.setheading(90)
        self.t.down()
        self.t.begin_fill()
        for i in range(4):
            self.t.forward(self.unit)
            self.t.right(90)
        self.t.end_fill()

    def move_player(self,x,y):
        self.t.up()
        self.t.setheading(90)
        self.t.fillcolor("red")
        self.t.goto((x+0.5)*self.unit,(y+0.5)*self.unit)
```

```
    def render(self):
        if self.t==None:
            self.t=turtle.Turtle()
            self.wn=turtle.Screen()
            self.wn.setup(self.unit*self.max_x+100,
                          self.unit*self.max_y+100)


 self.wn.setworldcoordinates(0,0,self.unit*self.max_x,self.unit*self.m
ax_y)
            self.t.shape("circle")
            self.t.width(2)
            self.t.speed(0)
            self.t.color("gray")
            for _ in range(2):
                self.t.forward(self.max_x*self.unit)
                self.t.left(90)
                self.t.forward(self.max_y*self.unit)
                self.t.left(90)

            for i in range(1,self.max_y):

 self.draw_x_line(y=i*self.unit,x0=0,x1=self.max_x*self.unit)
            for i in range(1,self.max_x):

 self.draw_y_line(x=i*self.unit,y0=0,y1=self.max_y*self.unit)

            for i in range(1,self.max_x-1):
                self.draw_box(i,0,"black")
            self.draw_box(self.max_x-1,0,"yellow")
            self.t.shape("turtle")

        x_pos=self.s % self.max_x
        y_pos=self.max_y-1-int(self.s/self.max_x)
        self.move_player(x_pos,y_pos)
```

一般强化学习的训练模式，也是大多数算法伪代码遵循的套路，步骤如下：

- 初始化环境和智能体；
- 对于每个回合，智能体选择动作；

- 环境接收动作反馈下一个状态和奖励；
- 智能体进行策略更新（学习）；
- 多个回合算法收敛之后保存模型以及做后续的分析画图等。

```
import os
import sys
import gym
import torch
import datetime
from gridworld_env import CliffWalkingWapper #环境装饰器
from agent import QLearning #算法
from plot import plot_rewards,plot_rewards_cn #用于画图
from utils import save_results,make_dir #存储路径


# In[3]:


curr_time=datetime.datetime.now().strftime("%Y%m%d-%H%M%S") #获取当前时间
import os
curr_path=os.path.dirname(os.path.abspath(__file__)) #当前路径 #一般只能在终端进行使用，在编译器很有可能报错。
class QlearningConfig:
    '''训练相关参数'''

    def __init__(self):
        self.algo="Q-learning" #算法名称
        self.env="CliffWalking-v0" #环境名称

 self.result_path=curr_path+"/outputs/"+self.env+"/"+curr_time+"/results/" #保存结果的路径

 self.model_path=curr_path+"/outputs/"+self.env+"/"+curr_time+"/models/" #保存模型的路径
        self.train_eps=400 #训练的回合数
        self.eval_eps=30 #测试的回合数
        self.gamma=0.9 # reward的衰减率
        self.epsilon_start=0.95 #e-greedy策略中初始epsilon
        self.epsilon_end=0.01 #e-greedy策略中的终止epsilon
        self.epsilon_decay=300 #e-greedy策略中epsilon的衰减率
        self.lr=0.1 #学习率
```

```
        self.device=torch.device("cuda")

def env_agent_config(cfg,seed=1):
    env=gym.make(cfg.env)
    env=CliffWalkingWapper(env)
    env.seed(seed) #设置随机种子
    n_states=env.observation_space.n #状态维度
    n_actions=env.action_space.n #动作维度
    agent=QLearning(n_states,n_actions,cfg)
    return env,agent
#以上表示环境创建完毕!

def train(cfg,env,agent):
    print("开始训练! ")
    print(f"环境：{cfg.env}，算法：{cfg.algo}，设备：{cfg.device}")
    rewards=[] #记录奖励
    ma_rewards=[] #记录滑动平均奖励
    for i_ep in range(cfg.train_eps):
        ep_reward=0 #记录每个回合的奖励
        state=env.reset() #重置环境，即开始新的回合
        while True:
            action=agent.choose_action(state) #根据算法选择一个动作
            next_state,reward,done,_=env.step(action) #与环境进行一次动作
交互
            print(reward)
            agent.update(state,action,reward,next_state,done) #Q学习算法
更新
            state=next_state #更新状态
            ep_reward+=reward
            if done:
                break
        rewards.append(ep_reward)
        if ma_rewards:
            ma_rewards.append(ma_rewards[-1]*0.9+ep_reward*0.1)

        else:
            ma_rewards.append(ep_reward)

        print("回合数：{}/{}，奖励
{:.1f}".format(i_ep+1,cfg.train_eps,ep_reward))
    print("完成训练!")
```

```
    return rewards,ma_rewards
```

通常我们会记录并分析奖励的变化，所以在接口基础上加一些变量记录每回合的奖励，此外由于强化学习学习过程得到的奖励可能会产生振荡，因此我们也适用一个滑动平均的量来反映奖励变化的趋势，这就是以上trian中代码的含义。

现在我们看看 Q 学习算法具体是怎么实现的，前面讲到智能体其实在整个训 练中就做两件事，一个是选择动作，一个是更新策略，所以我们可以定义一个$Qlearning$类，里面主要包含两个函数，即 choose_action 和 update 。我们先看看 choose_action 函数是怎么定义的，如下：

```
def choose_action(self,state):
    self.sample_count+=1
    self.epsilon=self.epsilon_end+(self.epsilon_start-
self.epsilon_end)*math.exp(-1.*self.sample_count/self.epsilon_decay)
    #epsilon是会递减的，这里选择指数递减
    # e-greedy策略
    if np.random.uniform(0,1)>self.epsilon:
        action=np.argmax(self.Q_table[str(state)]) #选择Q(s,a)最大对应的
动作
    else:
        action=np.random.choice(self.action_dim) #随机选择动作
    return action
```

一般我们使用 ε-贪心策略选择动作，我们的输入就是当前的状态，随机选取一个值，当这个值大于我们设置的 epsilion 时，我们选取 Q 值最大对应的动作，否则随机选择动作，这样就能在训练中让智能体保持一定的探索率，这也是平衡探索与利用的技巧之一。

```
def update(self,state,action,reward,next_state,done):
        Q_predict=self.Q_table[str(state)][action]
        if done:#终止状态
            Q_target=reward
        else:

 Q_target=reward+self.gamma*np.max(self.Q_table[str(next_state)])
        self.Q_table[str(state)][action]+=self.lr*(Q_target-Q_predict)
```

这里面实现的逻辑就是伪代码中的更新公式，如下式所示。

$$Q(S,A) \leftarrow Q(S,A)+\alpha[R+\gamma \underset{a}{max}\, Q(S^{\backprime}-Q(S,A))] \tag{4}$$

注意终止状态下，我们是获取不到下一个动作的，我们直接将 Q_target 更新为对应的奖励即可。

到这里关键的代码已经都实现了，接下来我们看看画图的代码*plot.py*

```
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.font_manager import FontProperties


def chinese_font():
    return
FontProperties(fname=r"C:\Windows\Fonts\STKAITI.TTF",size=15)

def plot_rewards(rewards,ma_rewards,tag="train",env="CartPole-
v0",algo="DQN",save=True,path="./"):
    sns.set()
    plt.title("average learning curve of {} for {}".format(algo,env))
    plt.xlabel("epsiodes")
    plt.plot(rewards,label="rewards")
    plt.plot(ma_rewards,label="ma rewards")
    plt.legend()
    if save:
        plt.savefig(path+"{}_rewards_curve".format(tag))
    plt.show()

def plot_rewards_cn(rewards,ma_rewards,tag="train",env="CartPole-
v0",algo="DQN",save=True,path="./"):
    '''中文画图'''

    sns.set()
    plt.figure()
    plt.title(u"{}环境下{}算法的{}学习曲
线".format(env,algo,tag),fontproperties=chinese_font())
    plt.xlabel(u"回合数",fontproperties=chinese_font())
    plt.plot(rewards)
    plt.plot(ma_rewards)
    plt.legend((u"奖励",u"滑动平均奖励"),loc="best",prop=chinese_font())
    if save:
        plt.savefig(path+f"{tag}_rewards_curve_cn")
    plt.show()
```

这里附上$QLearning$的完整算法

```
import numpy as np
import math
import torch
from collections import defaultdict


# In[2]:


class QLearning(object):

    def __init__(self,state_dim,action_dim,cfg):
        self.action_dim=action_dim #行动的维度
        self.lr=cfg.lr #学习率
        self.gamma=cfg.gamma
        self.epsilon=0
        self.sample_count=0
        self.epsilon_start=cfg.epsilon_start
        self.epsilon_end=cfg.epsilon_end
        self.epsilon_decay=cfg.epsilon_decay
        self.Q_table=defaultdict(lambda:np.zeros(action_dim)) #映射状态
的嵌套字典

    def choose_action(self,state):
        self.sample_count+=1
        self.epsilon=self.epsilon_end+(self.epsilon_start-
self.epsilon_end)*math.exp(-1.*self.sample_count/self.epsilon_decay)
        #epsilon是会递减的，这里选择指数递减
        # e-greedy策略
        if np.random.uniform(0,1)>self.epsilon:
            action=np.argmax(self.Q_table[str(state)]) #选择Q(s,a)最大对
应的动作
        else:
            action=np.random.choice(self.action_dim) #随机选择动作
        return action
```

```
    def predict(self,state):
        action=np.argmax(self.Q_table[str(state)])
        return action

    def update(self,state,action,reward,next_state,done):
        Q_predict=self.Q_table[str(state)][action]
        if done:#终止状态
            Q_target=reward
        else:

 Q_target=reward+self.gamma*np.max(self.Q_table[str(next_state)])
        self.Q_table[str(state)][action]+=self.lr*(Q_target-Q_predict)

    def save(self,path):
        import dill

 torch.save(obj=self.Q_table,f=path+"Q_learning_model.pkl",pickle_modu
le=dill)
        print("保存模型成功！")

    def load(self,path):
        import dill

 self.Q_table=torch.load(f=path+"Q_learning_model.pkl",pickle_module=d
ill)
        print("加载模型成功！")
```

最后说一下，完整版代码就点击github这里即可查看。