人工智能实验报告 DQN

姓名:丁晓琪 学号:22336057

一.实验题目

在`CartPole-v0`环境中实现DQN算法。最终算法性能的评判标准：以算法收敛的reward大小、收敛所需的样本数量给分。 reward越高（至少是180，最大是200）、收敛所需样本数量越少，分数越高。

二.实验内容

1.算法原理

1.背景

- Cart Pole：车杆游戏，小车需要左右移动保持杆竖直
- `env.step(self,action)`：
  - 参数：`action`:只有0(左移)，1(右移)两个离散值
  - 返回值：
    - `Observation`: 执行新动作后的环境观测也就是小车的状态变量
      - x：小车在轨道上的位置（*position of the cart on the track*）
      - θ：杆子与竖直方向的夹角（*angle of the pole with the vertical*）
      - $\dot{x}$：小车速度（*cart velocity*）
      - $\dot{\theta}$：角度变化率（*rate of change of the angle*）
    - `Reward`:奖励，只有0，1两个离散值。杆坚持不倒，reward=1；否则，reward=0
    - `Done`: 完成，是否需要将环境重置(env.reset()), Done=1需要重置，否则不需要
    - `info`：调试过程的诊断信息
- `env.reset()`：返回值和`env.step()`类型一致

2.DQN

- DQN原理（Q_learning的神经网络版）：
  - Q值记录方式：Q_learning中是用表格记录Q(s,a),；在DQN中是用神经网络，输入为s状态值，输出为不同动作a的Q值。
  - Q值的学习迭代：
    - 在q_learning中是通过查询表格，通过$Q(s,a)=Q(s,a)+\alpha[r+\gamma max_{a'}Q(s',a')-Q(s,a)]$更新迭代；
    - 在DQN中（无目标网络时），输入状态s，得到动作的所有的Q值，根据ϵ-贪心策略得到执行动作a，与环境交互得到s‘。
    - 将s’输入网络中得到a‘的Q值，选最大Q值对应的a’，得到TDtarget和Target之间的均方loss，梯度下降更新神经网络参数（TDtarget和Q_learnning一样）

$$TDtarget = r + \gamma max_{a'} Q(s', a'; w)$$
$$\Theta_t = Q(s, a; w) - TDtarget$$
$$L(w) = \frac{1}{T} \sum_{t=1}^{T} \frac{\Theta_t^2}{2}$$
$$g_t = \frac{d\Theta_t^2/2}{dw} \quad \text{梯度}$$
$$w = w - \alpha \cdot g_t \quad \text{梯度更新}$$

- DQN的升级：经验回放池，目标网络

3.经验回放池

- 内容：存放之前得到的经验 Q(s,a;w)
- 操作：
  - `push`: 将每次与环境交互和执行新的动作后得到的Q(s,a;w)放入回放池，当池满后，替换里面的旧数据
  - `sample`：取样，从回放池中随机取出指定大小的一批样本
- 好处：
  - 打破训练的关联性：相邻的Q(s,a;w)存在强关联性，训练效果差。但是从回放池中随机抽取的一批四元数是独立的，消除了训练的相关性
  - 可重复使用经验：经验可以被重复抽取利用：能够用更少的样本得到同样的表现

4.目标网络

- 内容：和训练中使用的DQN网络是完全相同的网络结构
- 操作：
  - 每隔一定的训练周期，更新目标网络中的参数和训练的DQN一致
  - 计算TDtarget ,TDtarget的Q值不再通过训练网络得到，而是通过目标网络 $TDtarget = r + \gamma max_{a'} Q(s', a'; w_{target})$
- 好处：如果没有目标网络，TDtarget和当前Q值都依赖训练网络参数，而训练网络再不断更新，则TDtarget就会不稳定，导致训练DQN难以收敛。加入目标网络后，目标值变化更加平滑稳定，训练DQN容易收敛

5.DQN拓展升级

- `reward`的修改：在 Cart Pole中`reward`只有0，1两个值，当小杆掉落的时候游戏结束且`reward=0`，其他时候`reward=1`。为了增大小杆掉落使游戏结束的惩罚，这里将其训练时得到的`reward`改为-10
- DDQN：
  - DQN中确定TDtarget是将下一个状态`s‘`输入到目标网络中，根据目标网络得到使`Q’`最大的动作`a'`和对应的`Q'_max`
  - DDQN确定TDtarget是将下一个状态`s'`输入到目标网络，得到`Q‘`最大的`a'`；但是将`s'`输入回训练网络，在训练网络中根据`a'`选择`Q'`

    $Q' = r + Q_{train}(s', argmax_{a'} Q_{target}(s', a'; w_{target}); w_{train})$
  - 优点：改良目标网络对训练网络的过估计

6.总

这里TDtarget在实验中换成了DDQN的算法

Input: 经验回放池 B，批样本大小 B，目标网络更新间隔 d
Output: 策略 π

随机初始化 Q 值网络参数 ϕ，初始化学习率 α，最大回合数 N，一个回合的最大时间步 T，梯度更新次数 G
初始化目标值网络参数 $\phi' \leftarrow \phi$
for 每个回合 $ep = 1, 2, .., N$ **do**
  重置环境并获取环境的初始状态
  for 每个时间步 $t = 1, ..., T$ **do**
    在时刻 t，状态为 s 时，智能体根据 ϵ 贪心策略和动作的值函数选择对应的动作 a
    根据状态转移函数，环境转移到下一时间步状态 s' 并返回对应的奖励值 r
    将四元组 (s, a, r, s') 存入经验回放池 B 中
    for 每个梯度更新步 $g = 1, ..., G$ **do**
      从经验回放池 B 中采样一批大小为 B 数据样本
      使用公式 (2-17)更新神经网络参数 ϕ
    end for
    if t 除以 $d == 0$ **then**
      使用公式 (2-18)更新目标网络参数 ϕ'
    end if
  end for
end for

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s,a)\left(Q_\phi(s,a) - r(s,a) - \gamma \max_{a'} Q_{\phi'}(s',a')\right),$$

$$\phi' \leftarrow \phi, \textit{ every } d \textit{ time steps.}$$

2.关键代码展示

1.DQN网络结构

一层输入层，一层隐藏层，一层输出层，层与层之间用Relu激活函数连接

输入是四个元素的输入，对应算法原理中提及的状态向量`s`有四个元素

输出是两个元素的输出，对应算法原理中提及的只有0，1两个动作

```
class QNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(QNetwork, self).__init__()  #先设置一个输入为4，输出为60的隐藏层，两层神经网络
        # YOUR CODE HERE #
        self.hide=nn.Linear(input_size,hidden_size) #有四个状态值
        self.relu=nn.ReLU()  #这个是必要的
        self.hide2=nn.Linear(hidden_size,hidden_size)
        self.relu2=nn.ReLU()
        self.out=nn.Linear(hidden_size,output_size)  #有两个动作（0，1）


    def forward(self, inputs):
        # YOUR CODE HERE #
        x=self.relu(self.hide(inputs))
        #x=self.hide(inputs)
        x=self.relu2(x)
        output=self.out(x)
        return output
```

2.经验回放池构建

`push`的替换原则是FIFO

`sample`是随机抽样

```
class ReplayBuffer:
    def __init__(self, buffer_size):
        # YOUR CODE HERE #
        self.buffer_size=buffer_size
        self.displace=0 #经验回放池满了后从哪里置换新的经验
        self.buffer=[]

    def __len__(self):
        return len(self.buffer)

    def push(self, transition):  #经验样本的格式（s_t,a_t,r_t,s_(t+1)）
        # YOUR CODE HERE #
        #trasition是可变参数，可以直接push，但是push内容是元组形式
        if(len(self.buffer)<self.buffer_size):
            self.buffer.append(transition)
            self.displace+=1
        else:
            self.displace=self.displace%(self.buffer_size)
            self.buffer[self.displace]=transition
            self.displace+=1

    def sample(self, batch_size):#随机取样
        # YOUR CODE HERE #
        sample=random.sample(self.buffer,batch_size)
        state,action,reward,next_state,done=zip(*sample)
        return state,action,reward,next_state,done

    def clean(self):
        # YOUR CODE HERE #
        self.buffer.clear()
        self.displace=0
        self.buffer_size=0
```

3. `AgentDQN`实现

- 参数设置：
  - 注意目标网络和训练网络参数设置一致，这里设置为输入4个因子，隐藏250个因子，输出2个因子
  - 网络训练的损失函数是MSE均方loss，后面虽然用了不同的表达方式但是还是mse均方loss
  - 随机种子设置：要给环境env，网络torch，随机数random，numpy……具有不确定性的因素都设置随机种子，便于复现
  - 回放池大小设置为10000

```
    def __init__(self, env, args):
        """
        Initialize every things you need here.
        For example: building your model
```

```
        """
        self.env=env
        self.args=args##这里有学习率和损失因子 还有批次的大小
        seed_everything(self.args.seed)
        self.seed = self.args.seed
        self.np_random, seed = seeding.np_random(self.seed)
        self.env.seed(seed) #环境也要设置随机种子
        super(AgentDQN, self).__init__(env

        # YOUR CODE HERE #
        #设置好网络，一个训练，一个target
        self.DQN=QNetwork(4,250,2)
        self.TargetDQN=QNetwork(4,250,2)
        #self.TargetDQN.load_state_dict(self.DQN.state_dict()) #网络结构和参数一致
        ##网络训练所需
        self.loss_fn=nn.MSELoss(reduce=True, size_average=False) #平方和误差#要是标量
        self.optimizer=optim.Adam(self.DQN.parameters(),lr=0.0005)
        #建立经验回放池
        self.ReplayBuffer=ReplayBuffer(10000)
        #训练回合数和训练回合中最大步数
        self.train_round=120
        self.max_step=200
        #损失因子
        self.gamma=self.args.gamma
        self.e_greedy=0.15
```

- 训练`train`：

  这里有四层循环嵌套：

  - 第一层嵌套：对多个随机种子做重复训练，后续对不同随机种子的同一训练回合的数据做均值和方差的统计，用于分析

  - ```
        for m in range(0,5):
            self.seed+=66 #随机种子改变
            seed_everything(self.args.seed)
            self.np_random, seed = seeding.np_random(self.seed)
            self.env.seed(seed) #环境也要设置随机种子
            self.DQN=QNetwork(4,250,2)
            self.TargetDQN=QNetwork(4,250,2)
            #self.TargetDQN.load_state_dict(self.DQN.state_dict()) #网络结构和参数一致
            ##网络训练所需
            self.loss_fn=nn.MSELoss(reduce=True, size_average=False) #平方和误差#要是标量
            self.optimizer=optim.Adam(self.DQN.parameters(),lr=0.0005)
            self.ReplayBuffer=ReplayBuffer(10000)
            mean=[]
            std=[]
            ......
            print("mean:",mean)
            print("std:",std)
            self.ReplayBuffer.clean()
            means.append(mean)
            stds.append(std)
    ```
    ```

- 第二层循环：训练回合的迭代，实验中设置一次训练一个随机种子要训练120个回合，回合的含义就是进行游戏的次数（游戏结束：超过限制的最大步数，游戏失败）

```
    for i in range(0,self.train_round): #训练回合数
        state=self.env.reset() #初始状态
        t=0
        done=False
        count=0
        roll_reward=[]
        ......
        roll_reward.append(count) #记录下回报
        mean.append(np.mean(roll_reward))
        std.append(np.std(roll_reward))
        print(i,count)
```

- 第三层循环：每个回合中每步的迭代,当到达特定的周期，更新目标网络

```
                while t<self.max_step and done==False: #每个回合最大步数
                    #print(t)
                    #将状态输入到DQN，得到最大的a
                    t+=1
                    state_=torch.Tensor(state)
                    output=self.DQN(state_)
                    #贪心策略选动作
                    random_e = np.random.random()
                    if(random_e>self.e_greedy):
                        action= torch.argmax(output).item() #Q值最大的索引为动作
                    else:
                        action=np.random.randint(2)

 next_state,reward,done,info=self.env.step(action)

                    count+=reward
                    if(done==True):

                        reward=-10 #修改reward，达到优化
                    sample=[state,action,reward,next_state,done]
                    self.ReplayBuffer.push(sample) #加入采样池
                    state=next_state
                    ......
                    if(t%4==0):

self.TargetDQN.load_state_dict(self.DQN.state_dict())
```

- 第四层循环：每步中对训练网络的训练

```
                        for j in range(0,5): ##抽样

 train_state,train_action,train_reward,train_next_state,train_done=self.ReplayBuffer.sample(64)
                            states = torch.tensor(train_state, dtype=torch.float)
```

```
                            actions =
torch.tensor(train_action).view(-1,1)
                            rewards = torch.tensor(train_reward,
dtype=torch.float).view(-1,1)
                            next_states =
torch.tensor(train_next_state, dtype=torch.float)
                            dones = torch.tensor(train_done,
dtype=torch.float).view(-1,1)
                            q_values = self.DQN(states).gather(1,
actions)  # [b,1]
                            ## 1.改用DDQN 求TDtarget
                            next_q_values=self.DQN(next_states)
                            #print(next_q_values.size())
                            next_action =
torch.argmax(next_q_values,dim=1)#主网络找动作
                            next_action =
torch.tensor(next_action).view(-1,1)
                            #print(next_action.size())

                            #max_next_q_values =
self.TargetDQN(next_states).max(1)[0].view(-1,1)
                            max_next_q_values =
self.TargetDQN(next_states).gather(1,next_action)
                            q_targets = rewards + self.gamma *
max_next_q_values * (1 - dones)

                            # 2 目标网络和训练网络之间的均方误差损失
                            dqn_loss =
torch.mean(torch.nn.functional.mse_loss(q_values, q_targets))
                            # if(t%20==0 and j%5==0):
                            #     print(i,dqn_loss)
                            # PyTorch中默认梯度会累积,这里需要显式将梯度置
为0
                            # 3.梯度更新
                            self.optimizer.zero_grad()
                            dqn_loss.backward()
                            self.optimizer.step()
```

- 画图：

  前面已经统计了每个随机种子下每个训练回合的累计`reward`均值

  将所有随机种子的结果整合成一条带阴影的直线：

  - 横坐标x：训练回合的索引
  - 纵坐标y： 所有随机种子在第x个训练回合累计reward的均值
  - 阴影：阴影上界：均值+均方差；阴影的下界：均值-均方差
  - 均方差：所有随机种子的累计在第x回合累计reward的均方差

```
    def print_train(self,d):
        means = d["mean"]
        stds = d["std"]

        # 计算迭代次数
        iterations = len(means[0])
        # 计算平均值和标准差
```

```
        mean_values = np.mean(means, axis=0)
        std_values = np.std(means, axis=0)

    # 绘制曲线和阴影区域
        x = range(iterations)
        print("mean_values:",mean_values)
        plt.plot(x, mean_values, label="Mean", color="blue")
        print("std_values:",std_values)
        plt.fill_between(x, mean_values - std_values, mean_values +
std_values, alpha=0.3, color="blue")

        plt.xlabel("Iterations")
        plt.ylabel("Mean Return")
        plt.legend()
        plt.title("Training Curve with Shadow")
        plt.show()
```

- 训练完后的运行时的动作选择`make_action`：

  由于不是在训练了，而是测试运行，所以要选择效果最好的动作，也就是Q值最大的动作

```
    def make_action(self, observation, test=True): #在训练完后不再用
e_greedy策略
        """
        Return predicted action of your agent
        Input:observation
        Return:action
        """
        # YOUR CODE HERE #
        observation=torch.Tensor(observation)
        output=self.DQN(observation)
        action= torch.argmax(output).item()
        return action
```

3.创新点&优化

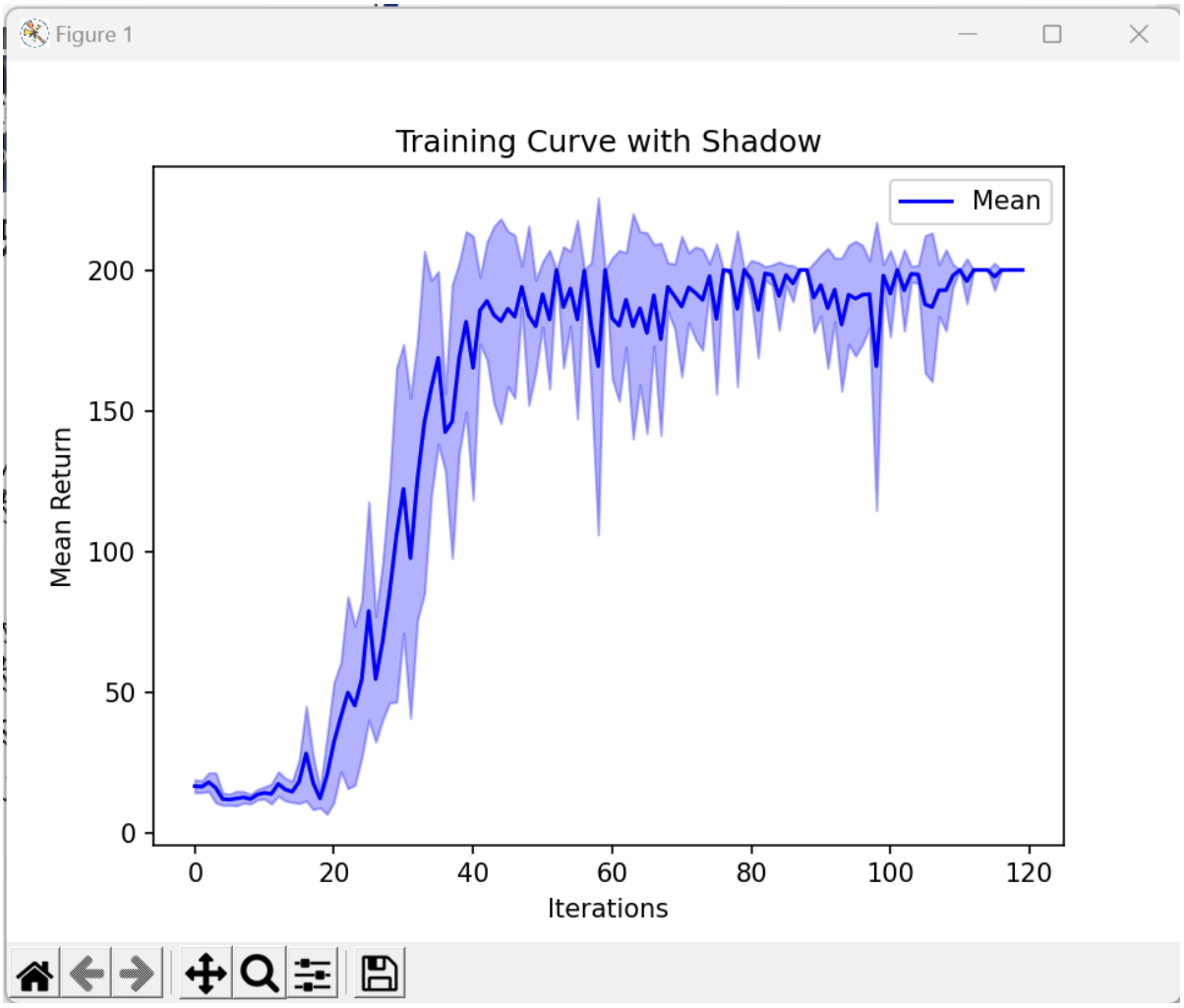
使用DDQN（具体可见算法原理），更改了小杆训练时掉落的惩罚（具体可见算法原理）

三.实验结果及分析

训练5个随机种子，每次训练迭代120个回合，每个回合最大步数为200

1.实验结果展示

- 5个随机种子的120个回合的训练曲线图



- 五个随机种子每个回合累计reward的均值

```
mean_values: [ 16.6  16.4  18.   16.   12.   11.8  12.2  12.6  12.   13.6  14.2  13.8
  17.4  15.4  14.6  18.2  28.2  17.8  12.2  20.4  32.2  41.2  49.8  45.2
  54.6  78.8  54.6  68.   85.4 105.8 122.2  97.6 125.4 145.8 158.2 168.8
 142.4 146.2 168.6 181.6 165.2 185.6 189.  184.  181.8 186.2 183.4 194.
 183.8 180.  191.4 182.4 200.  186.8 193.4 182.4 199.8 180.  165.8 200.
 182.8 180.2 189.4 180.  186.4 177.6 191.  175.4 194.  190.6 187.  193.8
 191.8 189.4 197.8 182.6 200.  199.6 186.2 200.  196.6 185.8 198.8 198.2
 190.8 198.2 195.2 200.  200.  190.2 194.6 186.4 193.  180.6 191.2 189.8
 191.2 191.4 165.8 198.  191.6 200.  192.8 198.6 198.4 187.8 186.8 192.8
 192.8 198.  200.  196.  200.  200.  200.  197.6 200.  200.  200.  200. ]
```

- 五个随机种子每个回合累计reward的方差

```
std_values: [ 2.33238076  2.05912603  3.34664011  5.32916504  2.19089023  1.93907194
  2.56124969  2.05912603  1.67332005  1.8547237   2.13541565  3.54400903
  4.31740663  4.02988834  3.72021505  7.78203058 16.80952111  9.53729521
  3.31058907 13.80724448 21.26405418 19.37420966 34.06699282 28.25172561
 27.82516846 38.75770891 22.27644496 27.87830698 39.15405471 59.36463594
 51.32406843 56.86686206 49.77388874 60.88809407 38.20157065 30.61633551
 13.6762568  48.59794234 33.87388375 32.03498088 46.86320518 11.87602627
 21.01428086 31.50238086 36.4        27.6        28.9108976   7.37563557
 31.90235101 16.43167673 11.55162326 24.67873579  0.         21.51650529
 13.2        35.2         0.4        22.59203399 59.80769181  0.
 21.40467239 26.76116589 16.82379268 40.         27.2        35.65164793
 18.         34.17367408  8.57904424 11.55162326 25.01199712 12.4
 16.4        17.90642343  4.4        26.72526894  0.          0.8
 27.6         0.          6.8        16.92808318  2.4         3.6
 12.10619676  3.6         6.4         0.          0.         12.33531516
 10.8        21.30352084 11.24277546 23.6101673  17.6        20.4
 17.6        11.8253964  51.17968347  4.         15.34405422  0.
 14.4         2.8         3.2        24.4        26.4         9.17387595
 14.4         4.          0.          8.          0.          0.
  0.          4.8         0.          0.          0.          0.        ]
```

2.评测指标展示及分析

- 训练曲线图分析：
  - 随着训练回合数的增加，平均累计reward不断上升，最终收敛到200
  - 阴影面积有所波动，中期均方差大，阴影面积大，但是最后四次阴影面积都为0，方差为0
- 训练样本：

  一次训练迭代120个回合，每个回合的每步从经验回放池中抽取5个64个数据大小的样本训练网络

四.参考资料

DDQN参考：https://blog.csdn.net/MR_kdcon/article/details/111245496