中山大学计算机学院人工智能本科生实验报告（2022学年春季学期）

课程名称：Artificial Intelligence

教学班级	专业（方向）	学号	姓名
2班	计算机科学与技术	21307174	刘俊杰

一、实验题目

Week 14 Deep Q-learning Network

二、实验内容

实验内容

- 用Deep Q-learning Network(DQN)玩CartPole-v1游戏，框架代码已经给出，只需要补充核心代码片段（'TODO'标记）
- QNet 补充一个线性层
- Choose_action 补充?-greedy策略代码
- Learn 补充Q值的计算，损失值计算，网络反向传播代码

结果要求与展示

- 500局游戏内，达到近10局reward均值为500
- 500局游戏的近10局reward均值曲线图

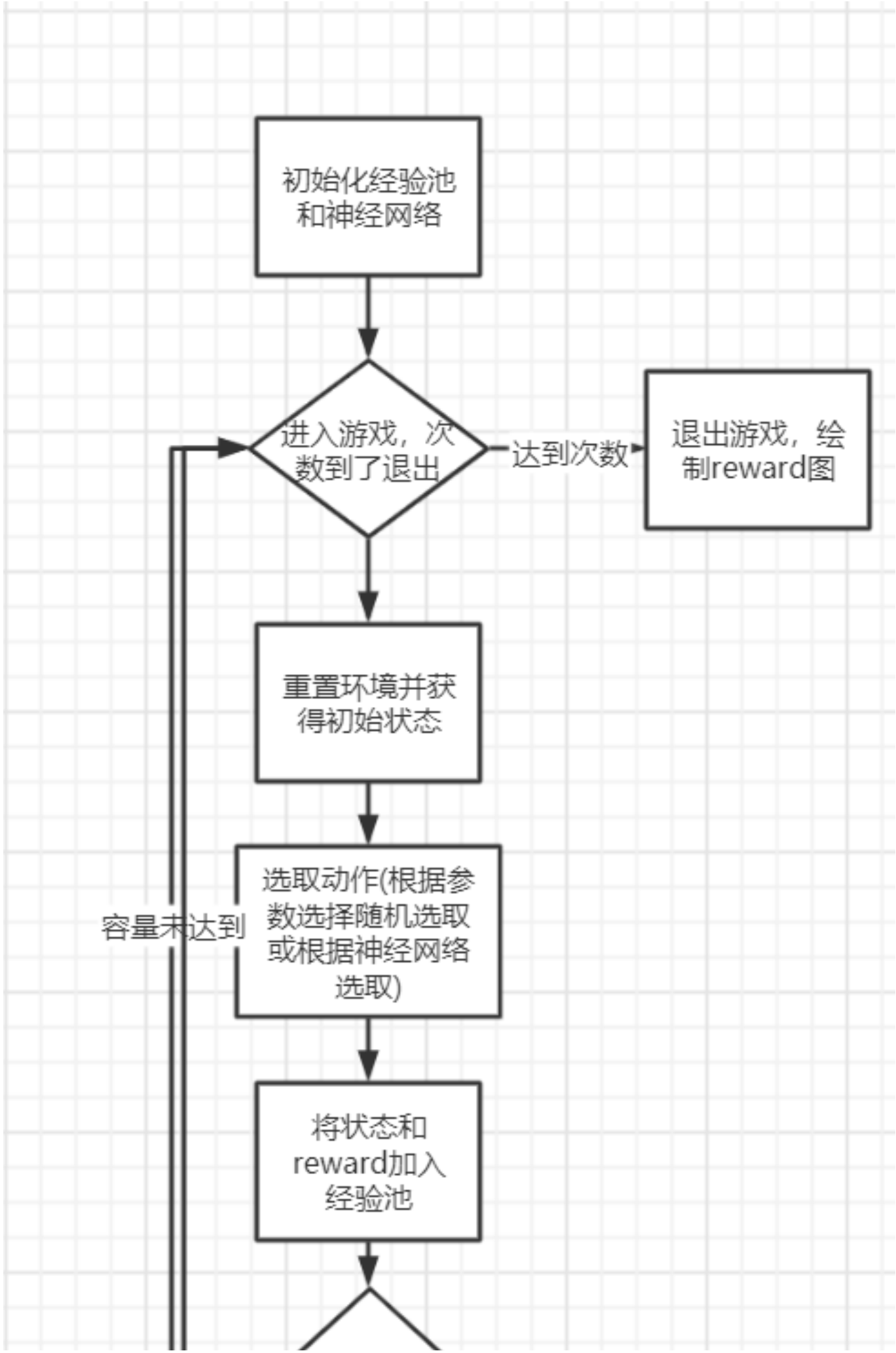
1. 算法原理

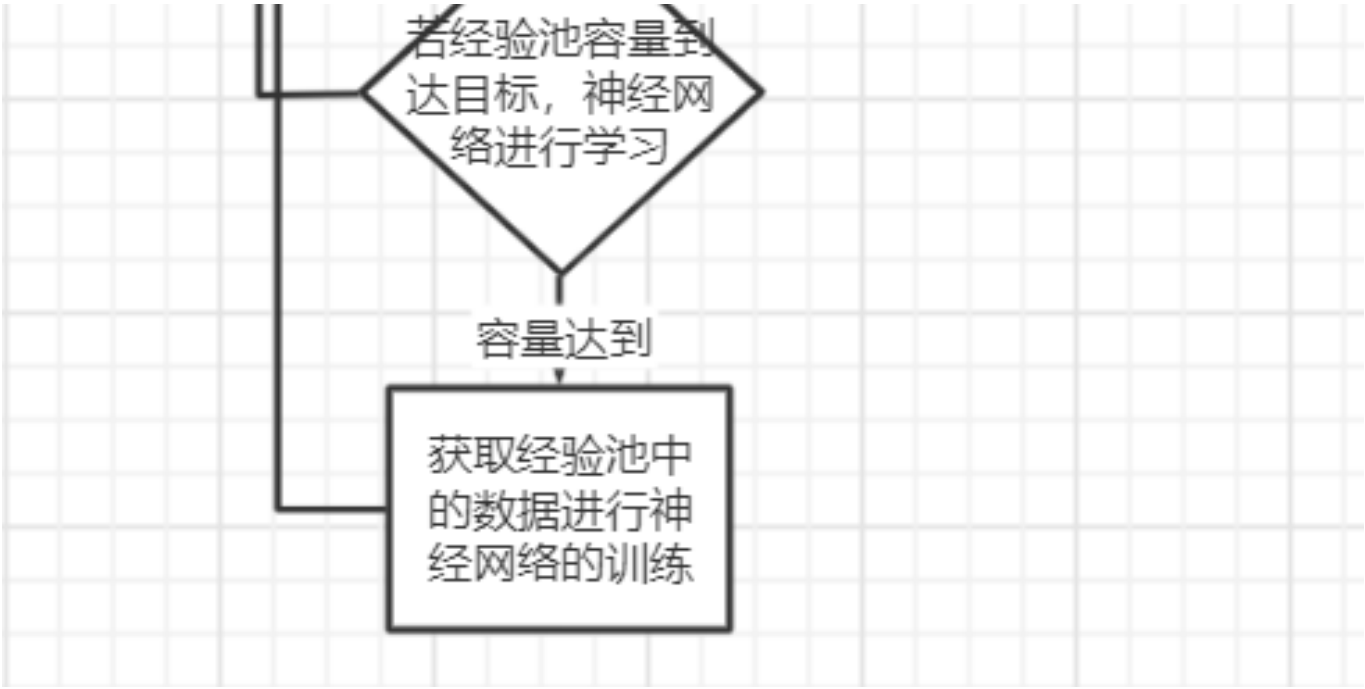
- 从决策方式来看，强化学习可以分为基于策略的方法(policy-based)和基于价值的方法(value-based)。基于策略的方法直接对策略进行优化，使制定的的策略能够获得最大的奖励。基于价值的强化学习方法中，智能体不需要制定显式的策略，它维护一个价值表格或价值函数，通过这个价值表格或价值函数来选取价值最大的动作。

- Q-Learning 算法就是一种value-based的强化学习算法。Q(s,a)是状态价值函数，表示在某一具体初始状态s和动作a的情况下，对未来收益的期望值。 Q-Learning算法维护一个Q-table，Q-table记录了不同状态下s(s∈S)，采取不同动作a(a∈A)的所获得的Q值。 探索环境之前，初始化Q-table，当agent与环境交互的过程中，算法利用贝尔曼方程（ballman equation）来迭代更新Q(s,a)，每一轮结束后就生成了一个新的Q-table。agent不断与环境进行交互，不断更新这个表格，使其最终能收敛。最终，agent就能通过表格判断在某个转态s下采取什么动作，才能获得最大的Q值。

- 这种算法存在很大的局限性。在现实中很多情况下，强化学习任务所面临的状态空间是连续的，存在无穷多个状态，这种情况就不能再使用表格的方式存储价值函数。为了解决这个问题，我们可以用一个函数Q(s,a;w)来近似动作-价值Q(s,a)，称为价值函数近似Value Function Approximation，我们用神经网络来生成这个函数Q(s,a;w)，称为Q网络（Deep Q-network），w是神经网络训练的参数。

2.流程图


初始化经验池和神经网络
进入游戏，次数到了退出
达到次数
退出游戏，绘制reward图
重置环境并获得初始状态
选取动作(根据参数选择随机选取或根据神经网络选取)
容量未达到
将状态和reward加入经验池



3.关键代码展示

(1)神经网络的构建:继承Module，构建两层神经网络，并重写前向传播

```
class QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(QNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        #构建两层的神经网络
        # TODO write another linear layer here with
        # inputsize "hidden_size" and outputsize "output_size"
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        #前向传播
        x = torch.Tensor(x)
        x = F.relu(self.fc1(x))
        # TODO: calculate output with layer fc2
        x = F.relu(self.fc2(x))
        return x
```

(2)构建经验池和相关操作

```
class ReplayBuffer:
    def __init__(self, capacity):#初始化经验池
        self.buffer = []
        self.capacity = capacity

    def len(self):#返回经验池中经验数量
        return len(self.buffer)
```

```
    def push(self, *transition):#加入经验池
        if len(self.buffer) == self.capacity:
            self.buffer.pop(0)
        self.buffer.append(transition)

    def sample(self, n):#从经验池中随机获得经验
        index = np.random.choice(len(self.buffer), n)
        batch = [self.buffer[i] for i in index]
        return zip(*batch)

    def clean(self):#清空经验池
        self.buffer.clear()
```

(3)选择动作

```
    def choose_action(self, obs):#选择动作
        # epsilon-greedy
        #action=0

        if np.random.uniform() <= self.eps:
            #随机选择
            # TODO: choose an action in [0, self.env.action_space.n) randomly
            action = np.random.randint(0,self.env.action_space.n)
            #self.env.action_space.n=2
        else:
            #选取神经网络输出值大的动作
            #pass
            # TODO: get an action with "eval_net" according to observation "obs"
            obs_tmp=torch.FloatTensor(obs)
            with torch.no_grad():#no_grad不保存为反向传播储备的值
               vals=self.eval_net(obs_tmp)#输出值
            action=vals.argmax().item()#选取神经网络输出值大的动作
        return action
```

(4)学习时，从经验池中获取部分数据，先进行前向传播，计算目标值，利用目标值进行反向传播对神经网络优化

```
    def learn(self):
        if self.eps > args.eps_min:
            self.eps *= args.eps_decay

        if self.learn_step % args.update_target == 0:
            self.target_net.load_state_dict(self.eval_net.state_dict())
        self.learn_step += 1

        obs, actions, rewards, next_obs, dones =
self.buffer.sample(args.batch_size)#获取经验学习
        actions = torch.LongTensor(actions)  # LongTensor to use gather latter
        dones = torch.FloatTensor(dones)
```

```
        rewards = torch.FloatTensor(rewards)


        # TODO: calculate q_eval with eval_net and q_next with target_net

        # TODO: q_target = r + gamma * (1-dones) * q_next
        # TODO: calculate loss between "q_eval" and "q_target" with loss_fn
        # TODO: optimize the network with self.optim


        # TODO: calculate q_eval with eval_net and q_next with target_net
        q_eval =
self.eval_net(np.array(obs)).gather(1,actions.unsqueeze(1)).squeeze(1)
        q_next = torch.max(self.target_net(np.array(next_obs)), dim = 1)[0]

        q_target = rewards + args.gamma * (1-dones) * q_next#目标值

        loss = self.loss_fn(q_eval, q_target)#计算损失
        self.optim.zero_grad()#清空上一轮梯度
        loss.backward()#反向传播
        self.optim.step()#优化反向传播
```

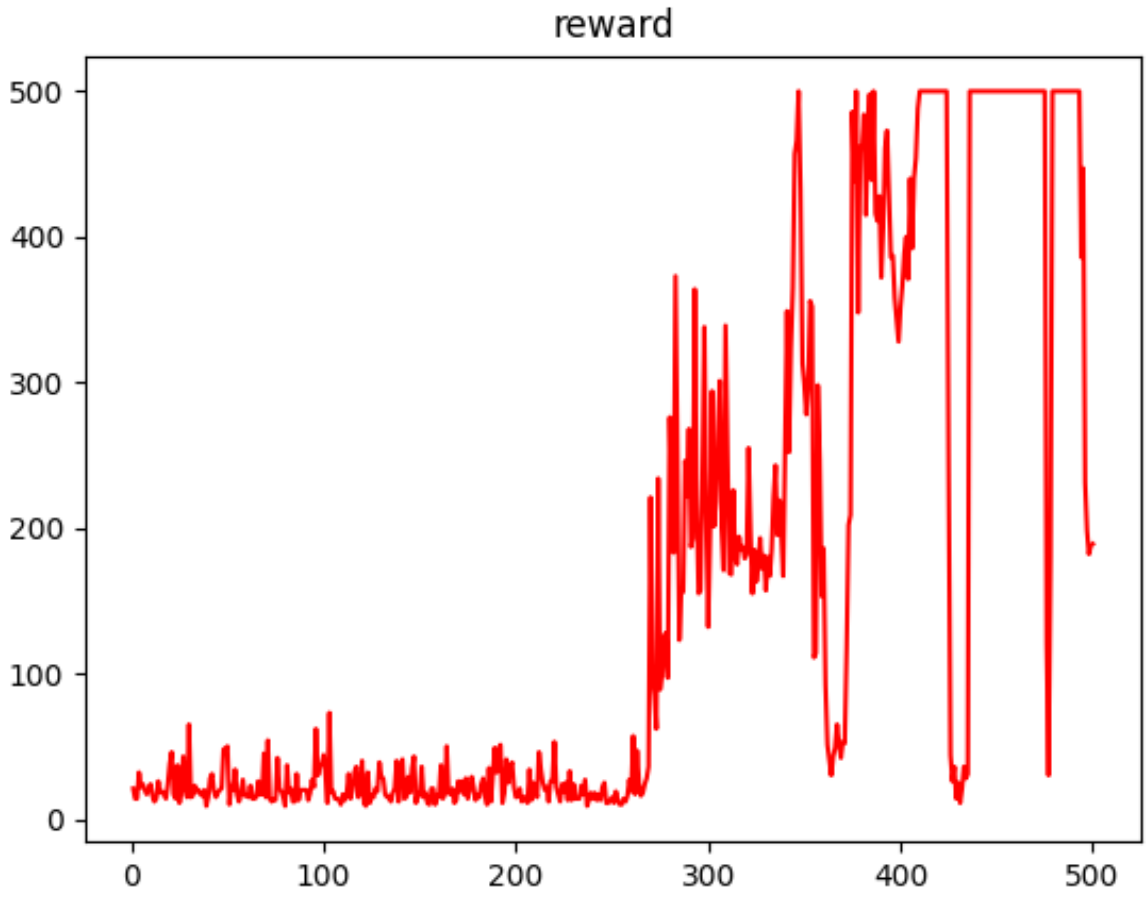
4.创新点&优化

无

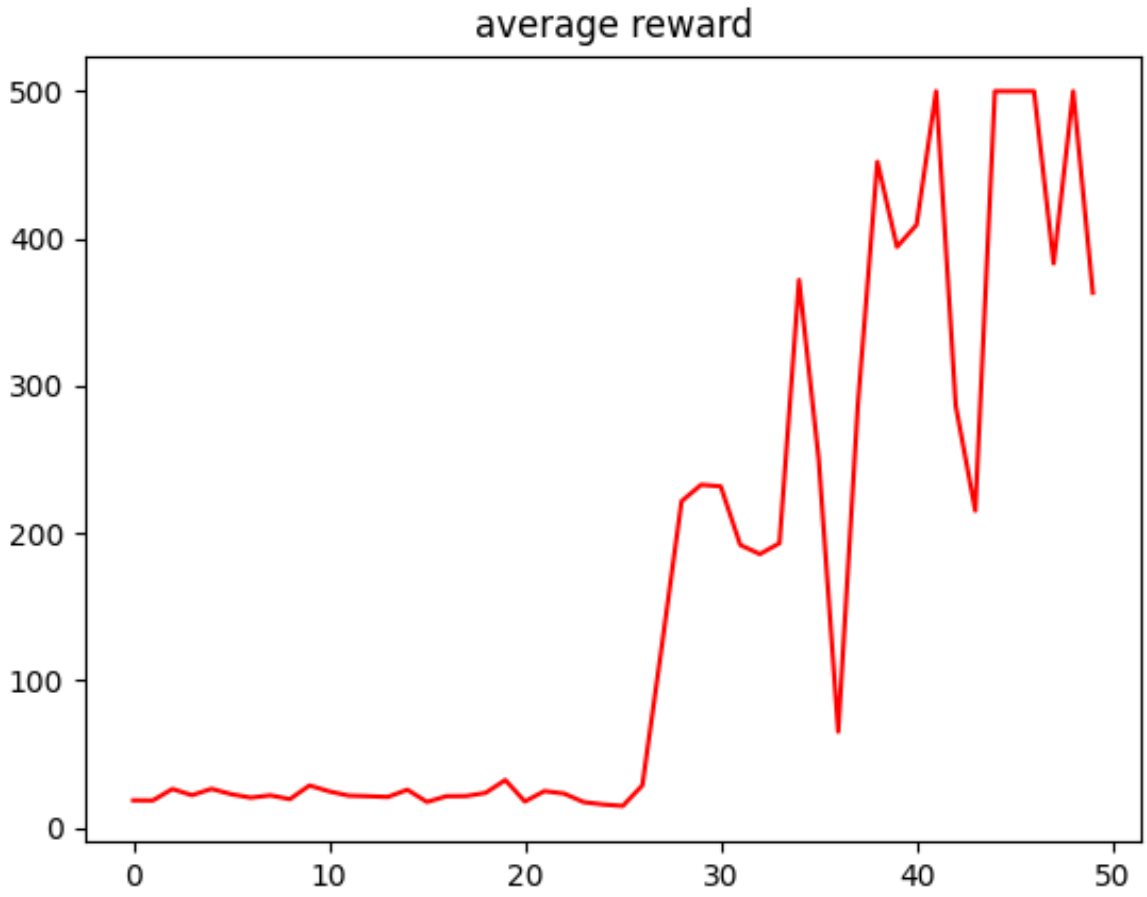
三、实验结果及分析

1. 实验结果展示示例(实验结果放入result文件夹中)

500次游戏每一步游戏返回的rewrad值

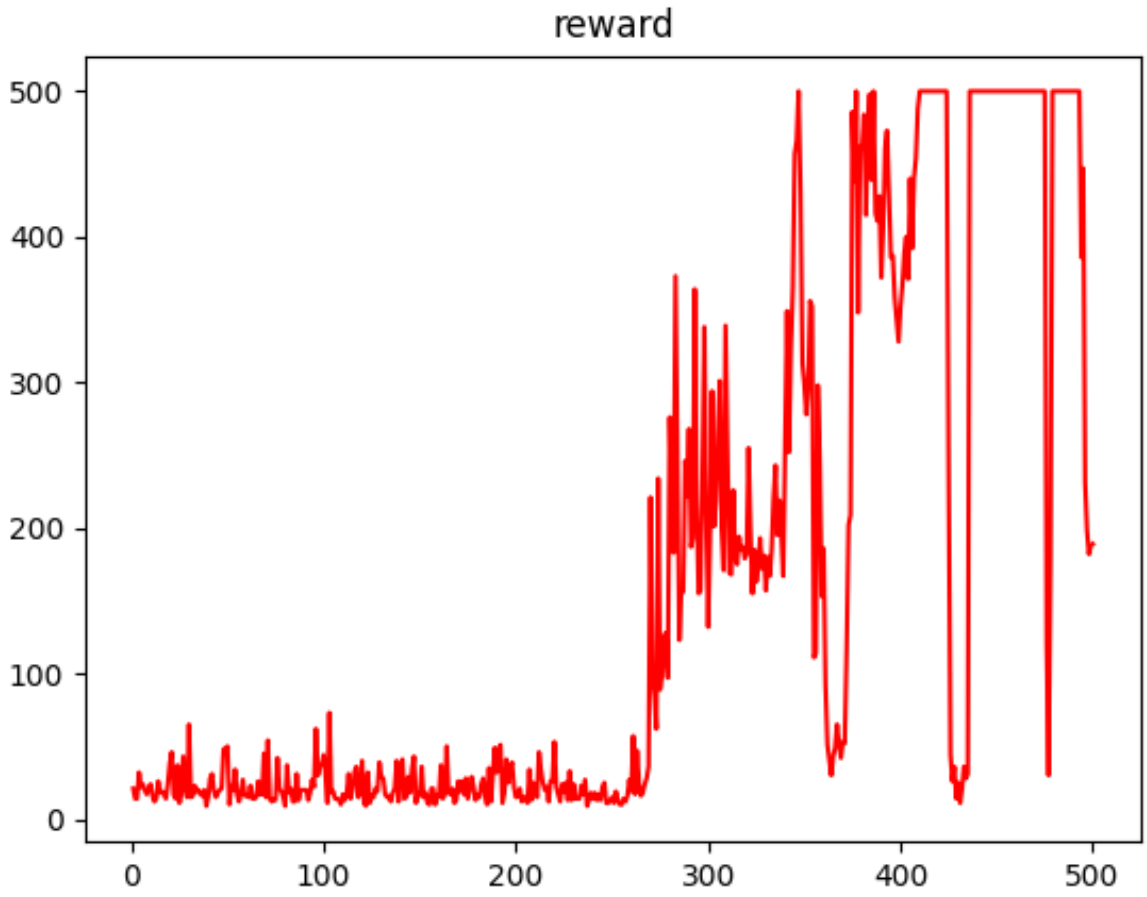


每10局的平均reward值



2、评价指标展示及分析

500次游戏每一步游戏返回的rewrad值

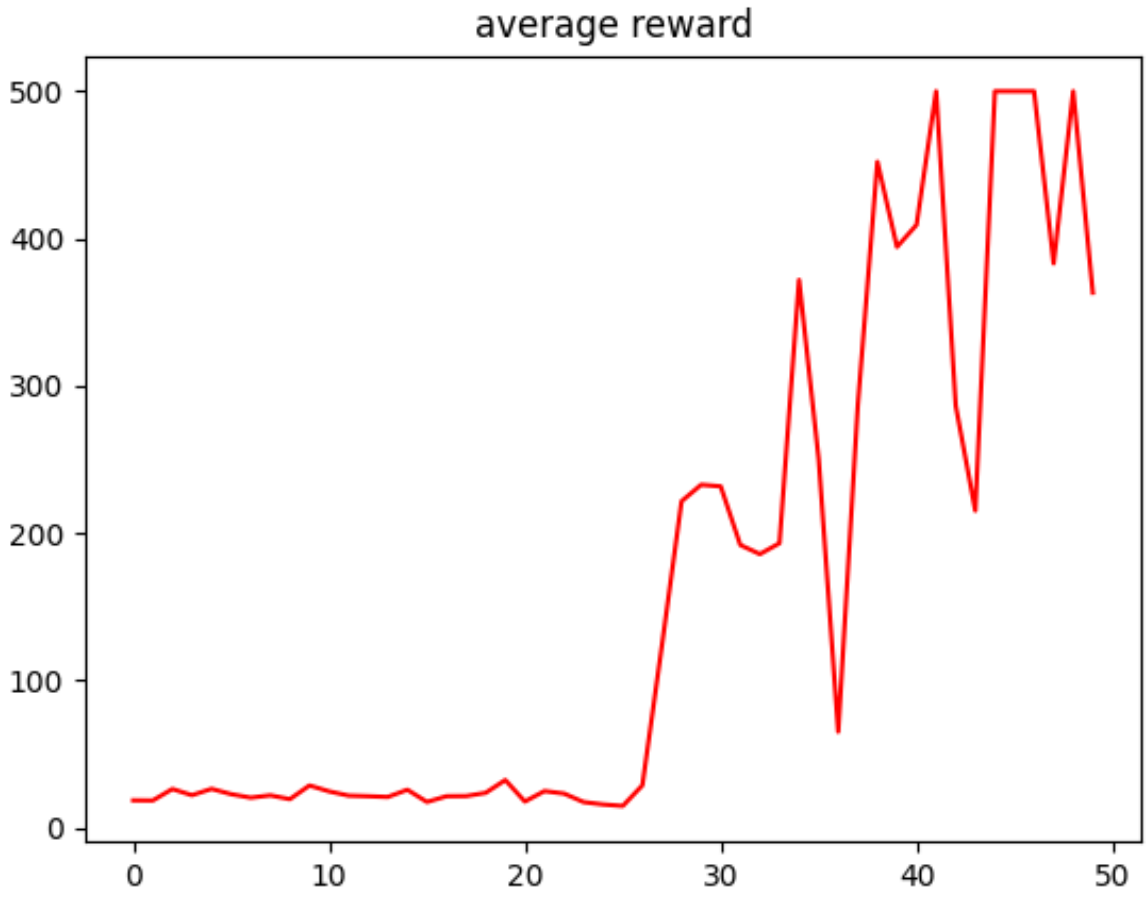


分析： (1)前200多步的reward的值较低，在大约280步左右开始显著上升，这是通过前200多步的经验继续学习优化神经网络，使得神经网络可以根据环境值输出较好的结果

(2)后200多步reward的值出现了骤升和骤降的现象，但总体处于较高的范围，出现了骤升和骤降的现象的原因查询资料可能是后期随着策略的变化，奖励/观测量产生异常剧变或者是过拟合

(3)在四百多步以后出现了多次reward值连续多步达到500的情况

每10局的平均reward值



分析: 可以看到400多步时出现了多次连续十步的reward值步达到500的情况，说明经过学习后能连续多次的表现出良好的效果

四、参考资料

实验课PPT