# DQN算法在CartPole上的实现

# --基于tensorflow

目录

[1.DQN算法 1](#_Toc22995)

[1.1算法简介和比对 1](#_Toc31399)

[1.2算法原理 2](#_Toc7057)

[1.2.1 经验回放 2](#_Toc1361)

[1.2.2目标网络 2](#_Toc7815)

[1.2.3目标网络的更新方式 2](#_Toc23476)

[2. 项目环境简介 3](#_Toc9517)

[2.1 Gym简介 3](#_Toc13172)

[2.2 CartPole-v1环境简介 3](#_Toc16252)

[3. 代码具体实现流程 4](#_Toc20989)

[3.1 经验回放池的实现 4](#_Toc9502)

[3.2 网络构造 4](#_Toc16508)

[3.3 算法控制流程 4](#_Toc21009)

[3.4 网络参数更新 5](#_Toc19569)

[4. 算法改进 6](#_Toc360)

[4.1 Double DQN算法 6](#_Toc8684)

[4.2 Dueling DQN算法 6](#_Toc16145)

[4.3 优化游戏环境 7](#_Toc29890)

[5. 结果分析 7](#_Toc20677)

[5.1 训练模型分析 7](#_Toc6188)

[5.2 加载预训练模型 8](#_Toc30611)

## **1.DQN算法**

## 1.1算法简介和比对

针对强化学习算法，在DQN被提出前，普遍采用为Q-learning算法，该算法采用一个Q-table来记录每个状态下的动作值，当状态控件和动作控件较大时，所需要的存储控件也会较大，导致该算法无法使用，因此Q-learning算法只能用来解决离散低维状态空间类问题;而DQN

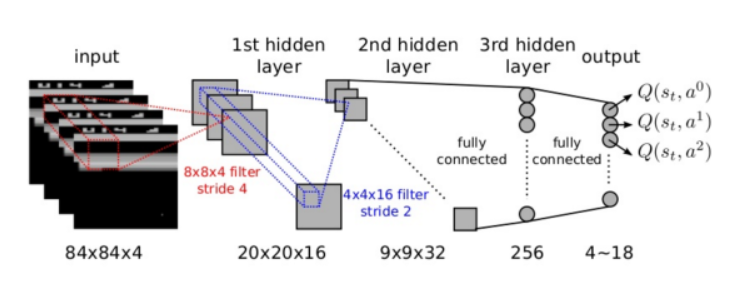
算法的核心是使用人工神经网络来代替Q-label,即动作价值函数，网络的输入为状态信息，输出为每个动作的价值，从而解决Q-learning中当状态维数过高时产生的维数灾难问题。

## 1.2算法原理

DQN算法作为一种off-policy算法，当出现异策，自益和函数近似时，无法保证其收敛性，容易出现不稳定或训练困难等问题，因此DQN引入了经验回放和目标网络两大概念

### 1.2.1 经验回放

经验回放从字面理解，就是一种让经验概率分布变得稳定的技术，从而提高训练的稳定性，而为什么要使用经验回放，对于Atari游戏，DQN算法将整个游戏的像素作为神经网络的输入，具体网络结构如图



游戏的特点导致像素级别的关联度高，只在某一处特别小的像素有变化，因此不同时序的样本关联度高，使得网路学习困难

而经验回放就可以解决这一问题，通过存储和回放两大步骤，存储将经验以(st,at,rt+1,st+1,done)的形式存储在经验池中，而回放则按照某种规则从经验池中采取一条或多条经验数据，采取经验回放可以打破数据之间的相关性，使得数据满足独立同分布，从而减少参数更新的方差，从而提高收敛速度，从而充分发挥off-policy的优势，使得behior policy用于搜集经验数据，而target policy只专注于价值最大化即可。

### 1.2.2目标网络

目标网络不同于原有的神经网络，它是在原有的神经网络之外重新搭建一个结构完全相同的网络，原先的网络被成为评估网络，而新搭建的网络被称为目标网络，在学习过程中，使用目标网络进行自益得到回报的评估值，作为学习目标，在更新过程中，只更新评估网络的权重，而不更新目标网络的权重，由于在目标网络没有变化的一段时间内回报的估计时相对固定的，因此目标网络的引入增加了学习的稳定性。

### 1.2.3目标网络的更新方式

硬更新：一定次数后将评估网络权重复制给目标网络的更新方式为硬更新，即wt<-we，wt表示目标网络的权重，we表示评估网络的权重

软更新：引入学习率，将旧的目标网络参数和新的评估网络参数直接做加权平均后的值赋值给目标网络

# 项目环境简介

## 2.1 Gym简介

Gym可以理解为一个仿真环境，其中内置了许多仿真游戏，不同游戏所用的网格，规则和奖励都不同，因此适合用来为强化学习做测试，并提供了页面渲染，以可视化查看效果

gym的核心接口为Env,作为统一的环境接口，Env包含几个核心方法:

.reset(self):重置环境状态，返回观察，如果回合结束，就要调用此函数，以重置环境信息

.step(self,action):执行动作action推进一个时间步长，返回observation,reward,done,info

.reward表示获得的奖励

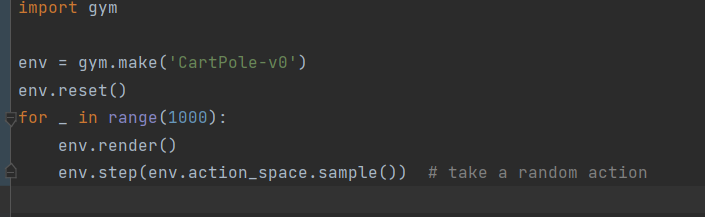
.done表示当前回合是否结束

.info返回一些诊断信息

.render(self,mode=’human’,close=False):重新绘制环境的一帧

.close(self):关闭环境，并清除内存

以下的代码描述了如何使用gym创建本项目需要的CartPole-v01环境，并刷新每个时间步长环境画面，对当前环境状态采取随机动作，并在循环结束后关闭仿真环境。



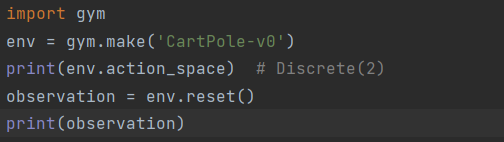
## 2.2 CartPole-v1环境简介

CartPole 是gym提供的一个基础的环境，即车杆游戏，游戏里面有一个小车，上有竖着一根杆子，每次重置后的初始状态会有所不同。小车需要左右移动来保持杆子竖直，为了保证游戏继续进行需要满足以下两个条件：

1.杆子倾斜的角度 angle 不能大于15°

小车移动的位置 x 需保持在一定范围（中间到两边各2.4个单位长度）

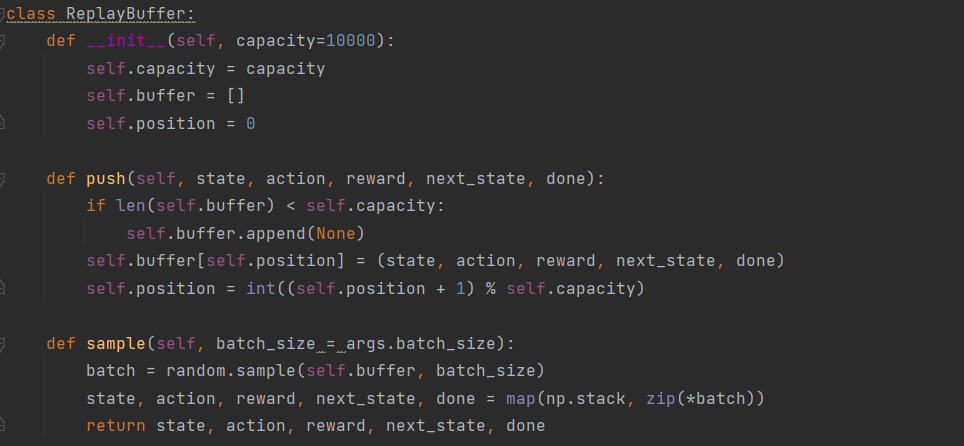
对于 CartPole-v1 环境，其动作是两个离散的动作左移（0）和右移（1），环境包括小车位置、小车速度、杆子夹角及角变化率四个变量,代码如下所示



# 代码具体实现流程

## 经验回放池的实现

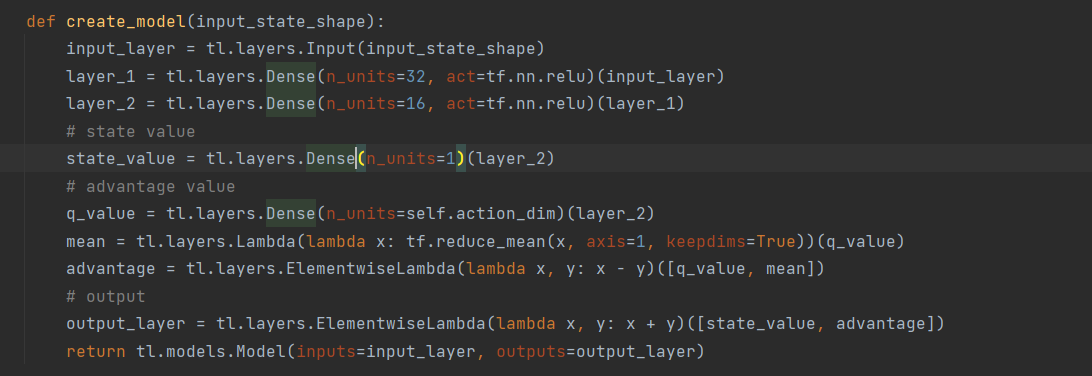
首先定义一个经验回放池，其容量为10000，函数push就是将智能体Agent与游戏环境交互的信息添加到经验池中，这里采用循环队列的方式来实现，每当需要数据来更新算法时，使用sample从经验队列中挑选一个batch\_size的数据，使用zip函数将每一个数据打包到一起，再对每一列的数据使用stack函数，在转化为列表后返回即可



## 网络构造

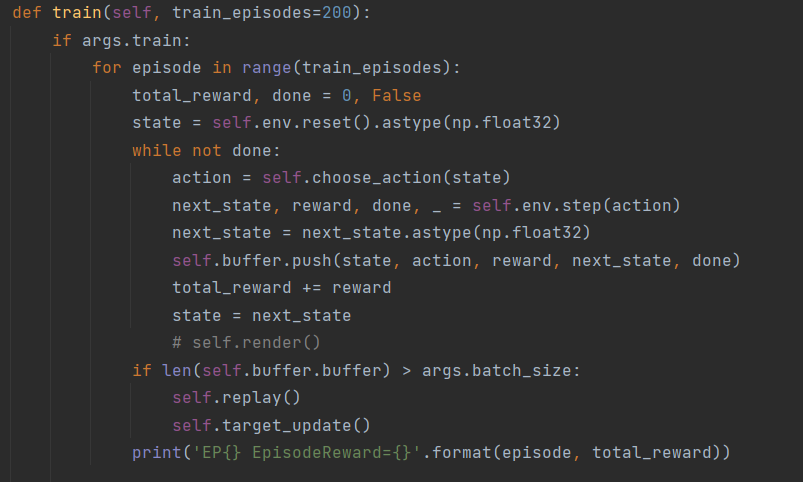
本次强化学习代码的搭建，在上次使用tensorflow的基础上，使用了tensorlayer，以方便调用强化学习的一些api，易于修改和扩展，减少开发中的重复工作。

接下来使用tensorlayer定义初步的网络模型，用creat\_model模型来生成网络模型，核心时创建当前网络model和目标网络target\_model,从上述DQN算法的介绍中，我们可以知道目标网络起到的一个“靶子”的作用，是用来评估当前的target值使用的，所以我们将其设置为评估模式，相应的调用eval()函数即可;而model网络是我们要训练的网络，调用train()将其设置为训练模式即可



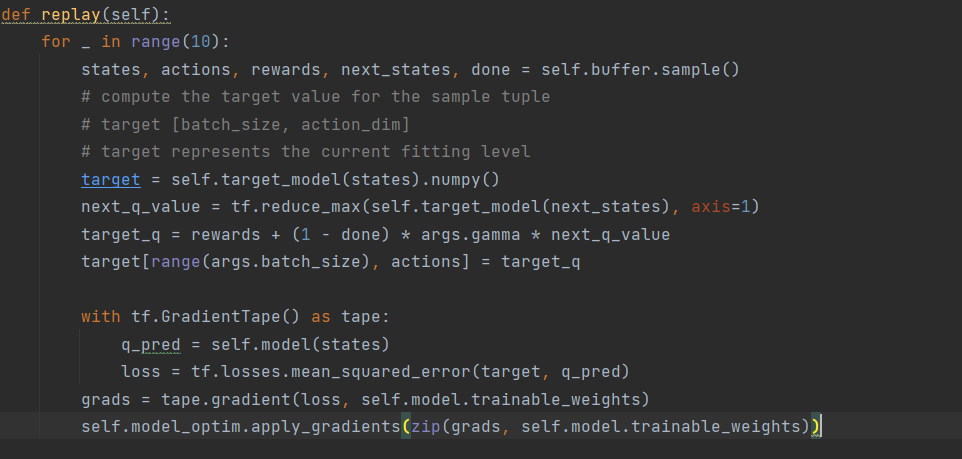
## 算法控制流程

首先利用2.1中提到的gym相关接口，获取当前环境的状态，并进行奖励记录等擦欧总，并不断将当前batch\_size和经验池长度进行比较，当经验池的长度大于batch\_size时，就开始调用replay()函数，用来更新网络model的网络参数，最后调用target\_update()函数将model网络参数复制赋给target\_model模型



## 网络参数更新

该部分为DQN的核心代码,在replay()函数中，我们循环更新当前网络十次，其作用就是改变两个网络的更新频率，有利于网络收敛，对于具体更新的部分，DQN相对于Q-learning就是将Q表格换为了神经网络，在神经网络中首先需要将状态输入神经网络，通过向前计算得到，首先获得当前的动作价值，target表示的就是根据当前的网络参数计算得到的动作价值，再获取当前网络参数的下一个状态的所有动作，然后使用reduce\_max()函数找出最大的动作价值，然后第9行和第十行利用下一个状态最大的价值来计算出target\_q,然后更新target，并且在上述计算target时一直在使用target\_model()网络，而target网络只有在评估网络状态时才用。接着使用q\_pred=self.model(states)网络来获取当前网络的状态，也就是公式中的Q(s,a,w)，利用MSE函数计算其损失函数，最后更新model网络。



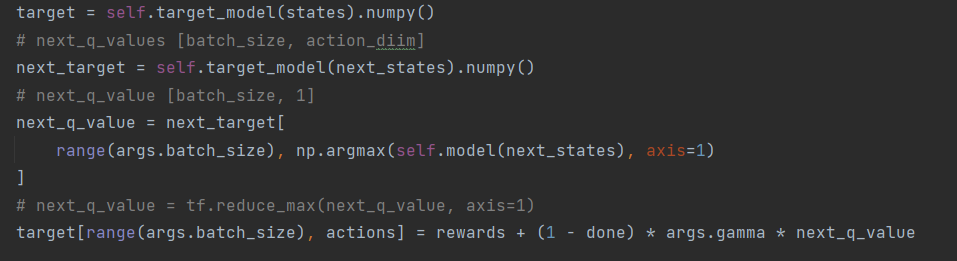
# 算法改进

## Double DQN算法

从上述DQN的分析可以看出，DQN仍存在一些问题，比如对于目标Q值的计算，因为Q-learning本身有固有的缺陷--过估计，过估计是指得值函数比真实函数要大，其根源在于DQN是一种off-policy的方法，每次学习时，不是使用下一次交互的真实动作，而是使用当前认为价值最大的动作来更新目标函数，而Double DQN的改进方法是将动作的选择和评估分别用不同的值函数来实现，而在Nature DQN中正好我们提出了两个Q网络。所以计算 TD Target 的步骤可以分为下面两步：

1. ：通过当前Q估计网络（Q Estimation 网络）获得最大值函数的动作a:  
   
2. :然后利用这个选择出来的动作 amax(s′,w) 在目标网络 (Q Target) 里面去计算目 Target Q值：

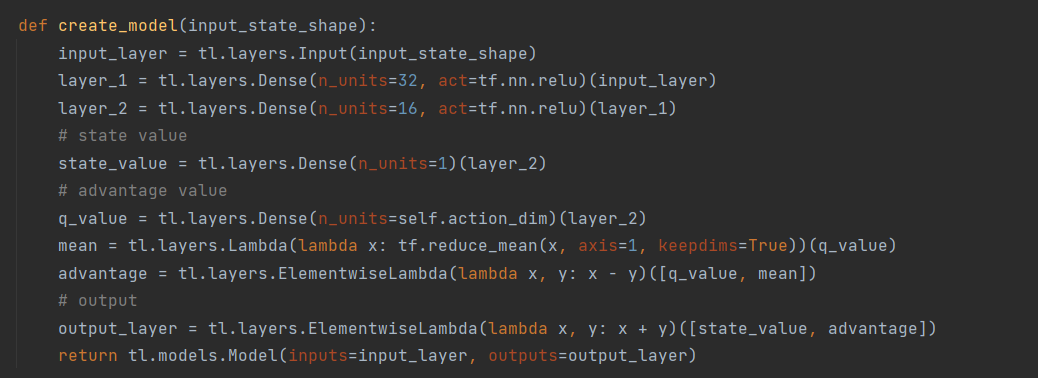
因此综合起来Double DQN和DQN中唯一不同的地方在于Q值的估计，其余流程均相同，代码展示如下



## Dueling DQN算法

在DQN算法中，神经网络的输出Q值代表动作价值，那么单纯的动作价值评估可能会不准确，因为Q(s,a)的值既和State有关，又和action有关，但是这两种相关的程度不同，而程序希望能够反映出两者的差异，而Dueling-DQN算法就是从网络结构上改进了DQN，神经网络输出的函数可以分为状态价值函数和优势函数，然后利用这两个函数通过神经网络来逼近，状态价值函数q(s,a)表示在状态s下选取动作a所能获取的价值;而优势函数表示动作价值相比于当前状态的值，如果优势大于0，则说明该动作比平均动作好，如果优势小于0，则说明当前动作还不如平均动作好，这样那些比平均动作好的动作将会有更大的输出，从而加速网络收敛过程。

Dueling DQN与DQN的不同处就在于网络结构，其余流程完全一样，代码如下



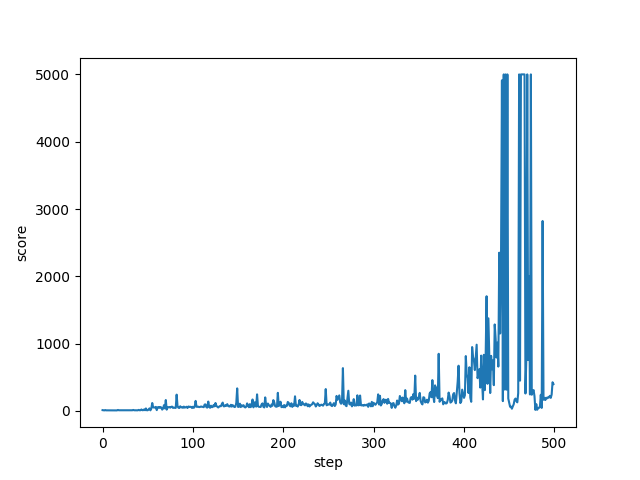
## 优化游戏环境

本次实验环境基于CartPole-v0,在实验过程中，发现分数最高只能500分，再继续训练成绩反而降低，经过多方面资料查询后，发现在tensorflow下gym环境中，该游戏最高得分为500分，超过500分后游戏重启，因此影响了强化学习的效率，因此对该游戏环境进行重新配置，修改最大得分和最大迭代次数，代码如下

# 结果分析

## 训练模型分析

在本次实验中，我采用了经过改进后的模型，训练了500次后，将模型保存，如下利用plt绘制的500次训练奖励变化图



可以看出在第450次时，奖励即可达到最大分5000，在保存模型时，设置判断条件。当到达5000分时直接保存模型即可

## 加载预训练模型

加载上述训练的模型，训练该游戏(若想快速得到结果关闭游戏显示窗口即可),可以看出在20次游戏中，绝大多数都达到了预先设置的最高分5000分，还有部分也基本都突破了千分，只有一个低分在1000分左右，说明之前的预训练模型还是较稳定的

