倒立摆强化学习算法研究

1 引言

1.1 研究问题

倒立摆，Inverted Pendulum ，是典型的多变量、高阶次 ，非线性、强耦合、自然不稳定系统。倒立摆系统的稳定控制是控制理论中的典型问题 ，在倒立摆的控制过程中能有效反映控制理论中的许多关键问题 ，如非线性问题、鲁棒性问题、随动问题、镇定、跟踪问题等。因此倒立摆系统作为控制理论教学与科研中典型的物理模型 ，常被用来检验新的控制理论和算法的正确性及其在实际应用中的有效性。从 20 世纪 60 年代开始 ，各国的专家学者对倒立摆系统进行了不懈的研究和探索。

倒立摆控制系统是一个复杂的、不稳定的、非线性系统，是进行控制理论教学及开展各种控制实验的理想实验平台。对倒立摆系统的研究能有效的反映控制中的许多典型问题：如非线性问题、鲁棒性问题、镇定问题、随动问题以及跟踪问题等。通过对倒立摆的控制，用来检验新的控制方法是否有较强的处理非线性和不稳定性问题的能力。同时，其控制方法在军工、航天、机器人和一般工业过程领域中都有着广泛的用途，如机器人行走过程中的平衡控制、火箭发射中的垂直度控制和卫星飞行中的姿态控制等。

倒立摆的控制目标是：倒立摆的控制问题就是使摆杆尽快地达到一个平衡位置，并且使之没有大的振荡和过大的角度和速度。当摆杆到达期望的位置后，系统能克服随机扰动而保持稳定的位置。

1.2 研究思路

现有算法是PID和DQN。PID算法缺少通用性，只要改变实验条件，就要重新设定参数。DQN算法十分不好用，我使用了DQN后发现它的效果不好。于是我想解决这个问题。

还有一种算法叫Sarsaλ的算法，它是回合制更新的，这使得这个算法可以考虑长远利益。我想把这个算法和DQN结合，来改善DQN的性能。

1.3 研究目的

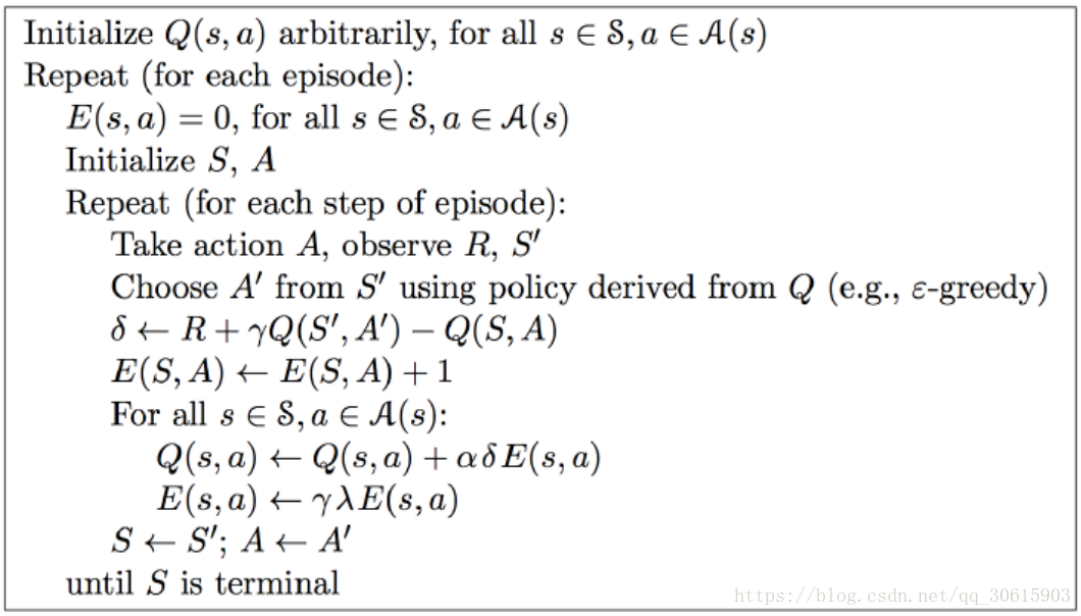
测试、对比DQN和Sarsaλ结合后和原版DQN的性能，改进DQN的速度。

2 研究现状

2.2 Sarsaλ

sarsaλ除了在更新时计算q值的动作是实际将要采用的动作之外，还有一个更新全路径的机制。其工作时的伪代码如图。其本质是认为了每一步动作产生的影响不止于其所导致的下一状态，而是影响了其后的每一个状态，因此在进行完某步之后，应当更新之前路径上所有状态。。考虑到更近和出现更多的状态有更大的影响，因此对每个状态-动作二元组用E来表达其对当前状态的影响程度，根据影响程度更新全路径。这种方法最大的优势便是若某个局部不

优但在全局中十分优秀的动作被执行，即使在本回合不能立即获得奖励进而更新网络，却能在将来这个动作的效果显现时得到回报而更新。另外。lambda和gamma都是较大的常数，能够对即使在数十甚至数百个step前的动作做出有效的更新。这种方式较为有效地避免了智能体陷入局部最优的陷阱中。


Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
$E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Initialize S, A
Repeat (for each step of episode):
Take action A, observe R, S'
Choose A' from S' using policy derived from Q (e.g., ε-greedy)
$\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
$E(S,A) \leftarrow E(S,A) + 1$
For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
$E(s,a) \leftarrow \gamma \lambda E(s,a)$
$S \leftarrow S'$; $A \leftarrow A'$
until S is terminal

2.2 DQN

dqn作为最基础的基于价值的强化学习策略，存在着诸多可以完善的点。首先，原始dqn认为其中的状态是无后效性的，就是说只有状态本身有意义而采用何种路径到达该状态没有影响，一个动作影响的方式只有由他导致的下一状态。这导致当获得了一个回报之后，这个回报向前传播到真正产生这个回报的状态-动作二元组的过程是缓慢而低效的。例如，考虑某动作的奖励在n个step后才能获得，则在此设计中需要n个episode才能使得这个奖励影响到产生它的状态-动作二元组，并且其再沿路径向前传播的过程中，每一步都要乘上一个极小的学习率，实质上导致了dqn要求某状态-动作二元组带来的收益必须很快发生而不能有大的滞后。这导致了dqn极易收敛到局部最优，过分追求某动作带来的，即时或者几乎即时的收益而忽略了长期的影响。这一缺点在保持立杆在中部的回报需要数百个step才能得到的平衡杆问题中是几乎致命的。

3 研究内容

3.1 算法改进

在原DQN算法中加入Q表和路径追踪表。

```
self.q_table = pd.DataFrame(columns=[0, 1], dtype=np.float64)
self.eligibility_trace = self.q_table.copy()
```

加入检查状态函数，如果某个状态不存在，就会在Q表中加入这一种状态。不用一次把所有状态加入，来节约空间。

```
def check_state_exist(self, state):
    if state not in self.q_table.index:
        self.q_table = self.q_table.append(
            pd.Series(
```

```
                [0] * 2,
                index = self.q_table.columns,
                name = state,
            )
        )
        self.eligibility_trace = self.eligibility_trace.append(
            pd.Series(
                [0] * 2,
                index = self.eligibility_trace.columns,
                name = state,
            )
        )
        self.match_table = self.match_table.append(
            pd.Series(
                [0],
                index = self.match_table.columns,
                name = state,
            )
        )
```

加入remember函数，当到达一种新状态，就把这种状态和其奖励记录在Q表里，并根据路径追踪衰减值修改Q表中的奖励。

```
def remember(self, s, a, r, s_):
    self.check_state_exist(self.to_str(s))
    self.eligibility_trace.loc[self.to_str(s), a] += 1
    self.q_table += self.eligibility_trace * r
    self.eligibility_trace *= 0.988
    self.match_table.loc[self.to_str(s), 0] = s_
```

当一轮游戏结束后，使用restore函数，把Q表中的数据储存进DQN的memory。然后再清零Q表。

```
def restore(self):
    for index, row in self.q_table.iterrows():
        if row[0] == 0:
            action = 1
        else:
            action = 0
    self.store_transition(np.array(self.to_list(index)), action, row[actio
n], np.array(self.match_table.loc[index, 0]))
    self.q_table = self.zero_table.copy()
    self.eligibility_trace = self.zero_table.copy()
    self.match_table = self.zero_match_table.copy()
```

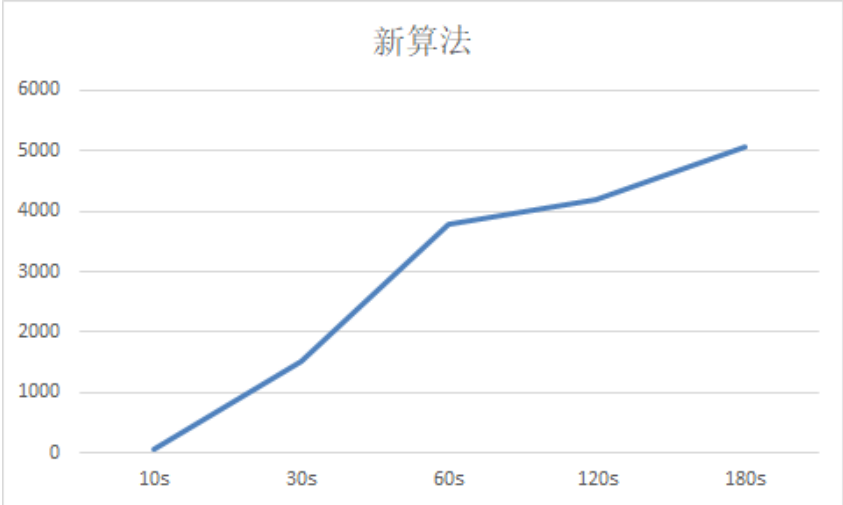
3.2 参数调试

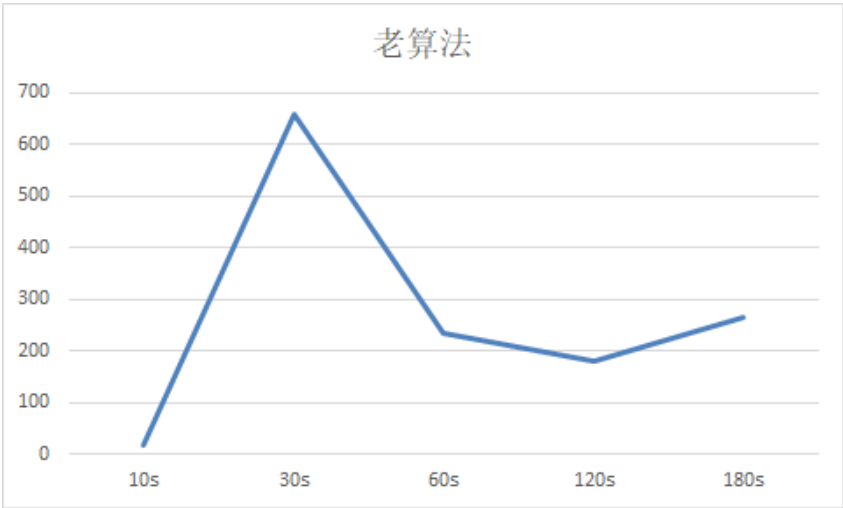
改进后的新算法在learning_rate=0.01,γ=0.9，λ=0.998时取得了较好的效果。

4 研究结果

4.1 速度分析

我分别用新算法（DQN+Sarsaλ）和老算法（DQN）进行了5次训练，用程序记录了其在10s、30s、60s、120s、180s时的得分，并取平均，得到了下面两张图表。





注意纵坐标数量级。

从图中我们可以看出，新算法的得分明显高于老算法，并且新算法的得分总是在增长，而老算法不稳定，有增长也有下跌。

总之，可以得出结论，DQN+Sarsaλ的算法，明显比原版DQN快。

为什么新算法这么快呢？这里提出一个猜想。

观察了原版DQN控制倒立摆的演示，我发现其实倒立摆很少倒下，但是经常以非常慢的速度向两边偏移，最终因为出界而游戏结束。结合奖励机制思考，也许神经网络不知道有一种操作方法就是先使倒立摆倾斜，再把它推回中央，最后平衡住倒立摆。DQN是单步更新的算法，每训练一次只能学会预测某一帧的情况，然而一秒钟有30 帧画面，所以凭借DQN做出先倾斜再回推这种高难度动作是很难的。把倒立摆故意向一端倾斜是一个扣分动作，DQN不会主动做出这个动作，因为它不知道通过这个动作可以获得更长远的利益。

使倒立摆倾斜一个小角度来回到中心这一动作，在刚刚做出时是几乎没有收益的，其收益需要数十step之后才开始显现，且该收益本身较小。该收益在dqn中沿路径传播时，每到上一step都会被乘上学习率，考虑到学习率learning_rate取值为0.01，该收益在传播数次后就接近于0，更为严重的是这种回溯往往在一个episode中发生次数较少，甚至需要数十个episode后

最初的回报的影响才能到达产生它的状态-动作二元组。因此这个动作的q值会往往小于局部最优的保持直立而不回归中心，且q值需要大量时间才能传到真正产生了回报的状态-动作二元组。q值的传播微弱而缓慢。dqn无法在短时间内收敛到全局最优。

然是通过把DQN和Sarsaλ结合，每个step都能影响到其中所有动作，所以算法就有了更好的计算长远利益的能力。我观察了新算法的训练过程。再两三次出界之后，新算法就学会了使倒立摆倾斜从而推回中央的技巧，到了第60、70轮，倒立摆就像静止了一样，几乎看不到不稳定的情况了。

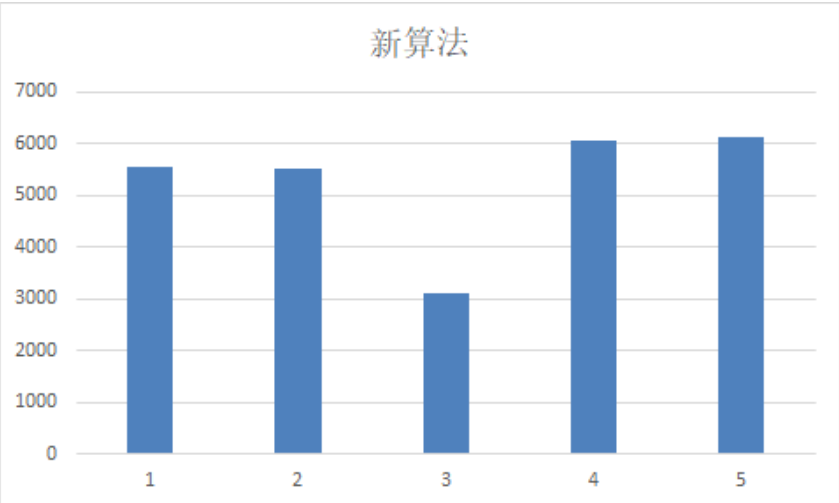
这是利用了衍生于sarsaλ的更新方式，考虑路径上每个状态-动作二元组对其后路径的影响。这时由回到中部这一动作产生的回报较为高效地在每个episode中传播，事实上，回到中部这一动作带来的收益在获得后，立刻传播到了产生它的状态-动作二元组，并且每向前传播一次，q值衰减极小（lambda=0.988，gamma=0.9,因而只衰减到约90%）。q值高效而快速地传播，故而智能体快速收敛到全局最优。

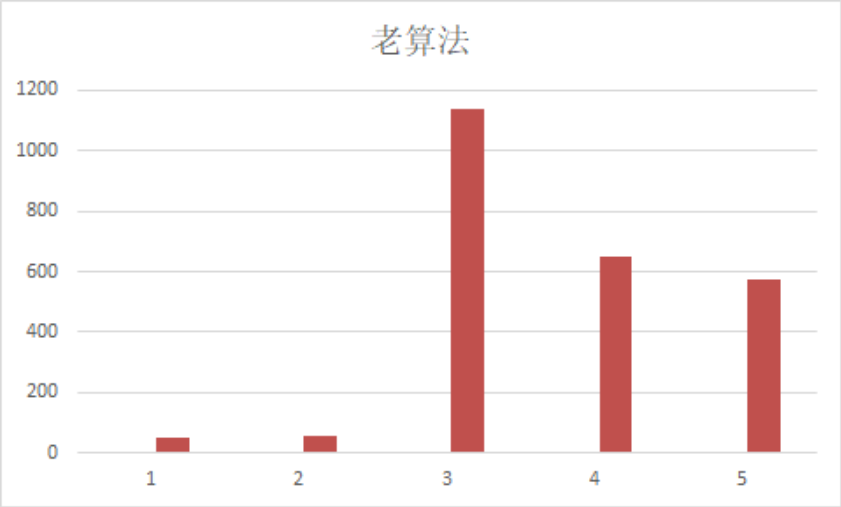
4.2 可靠性分析

由于电脑性能不行，我把每一轮的步数限制在5000步，达到5000步后强制结束，运行5000步的满分为6250。

分别用两种算法训练5次，每次2分钟，记录在2分钟时的得分，比较得分是否出现了较大的波动。并求出标准差、平均数、变异系数。

训练2分钟成绩稳定性统计						标准差	平均	变异系数
	1	2	3	4	5			
新算法	5541	5535	3121	6048	6116	1233.171	5272.2	0.233901
	1	2	3	4	5			
老算法	49	53	1138	651	576	457.8791	493.4	0.928008





如图所示，新算法（DQN+Sarsaλ）相比老算法（DQN）算法要稳定。虽然新算法的成绩标准差为1233，大于老算法的标准差457。但是因为新算法的平均分更高，所以新算法的变异系数为0.23，远小于老算法的0.92。变异系数是标准差除以平均数，用来衡量数据的离散程度。通过变异系数可以看出，新算法比老算法要稳定很多。

4.3 结果对比

这一节将会分析两种算法的训练过程，对训练结果进行对比。

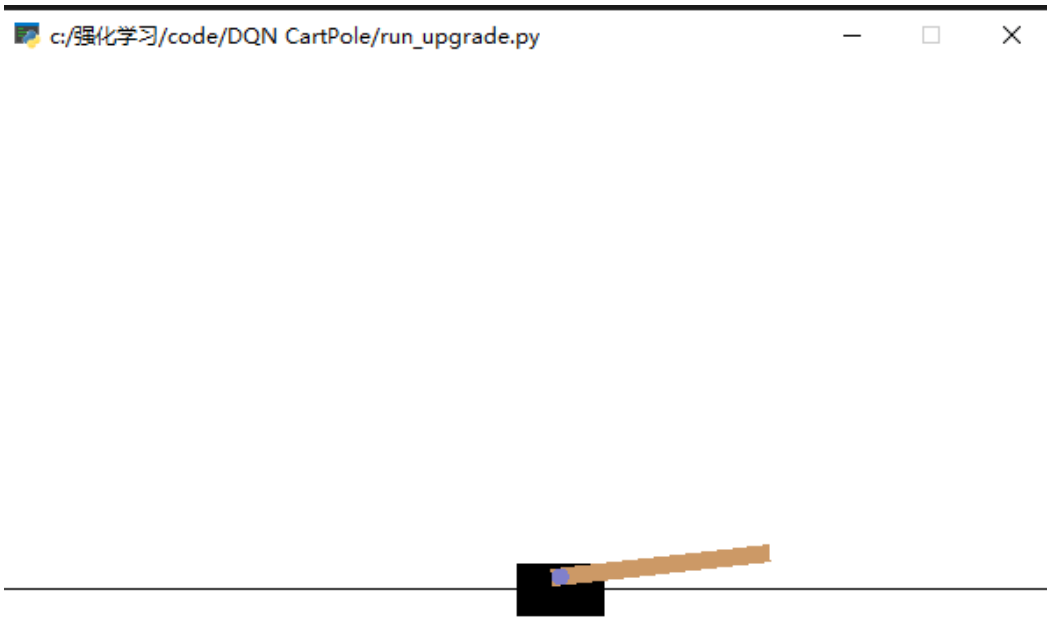
原版DQN算法

开始训练时，倒立摆向右倒下。

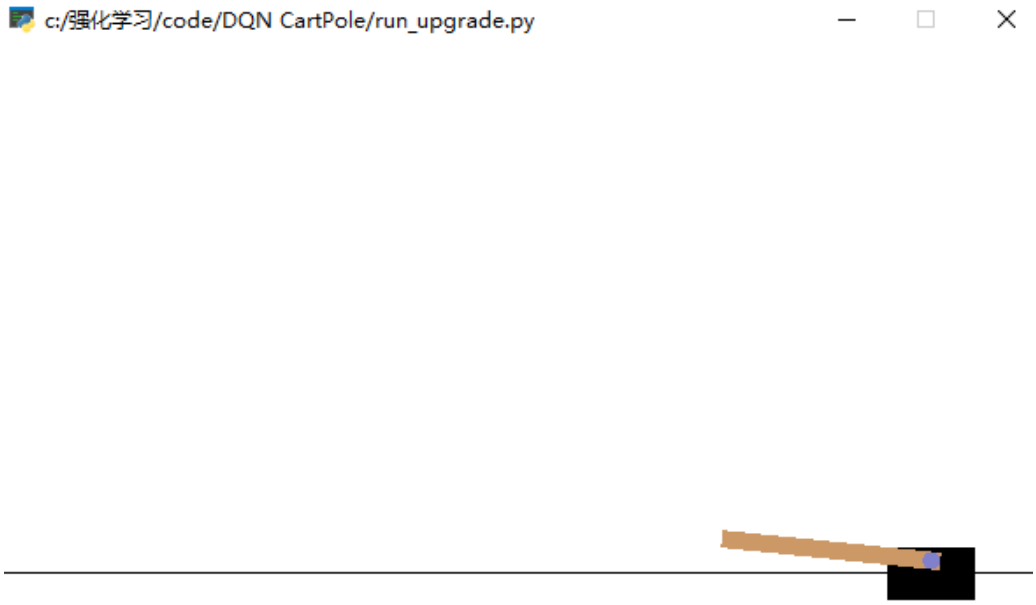
接着，经过几次训练，算法已经可以控制倒立摆一段时间不倒，但是其总会向一个方向偏移，最终出界。

训练过很多次后，倒立摆基本上不会倒下，但是它会保持一点点的倾斜角度，始终不能被纠正，最终出界。

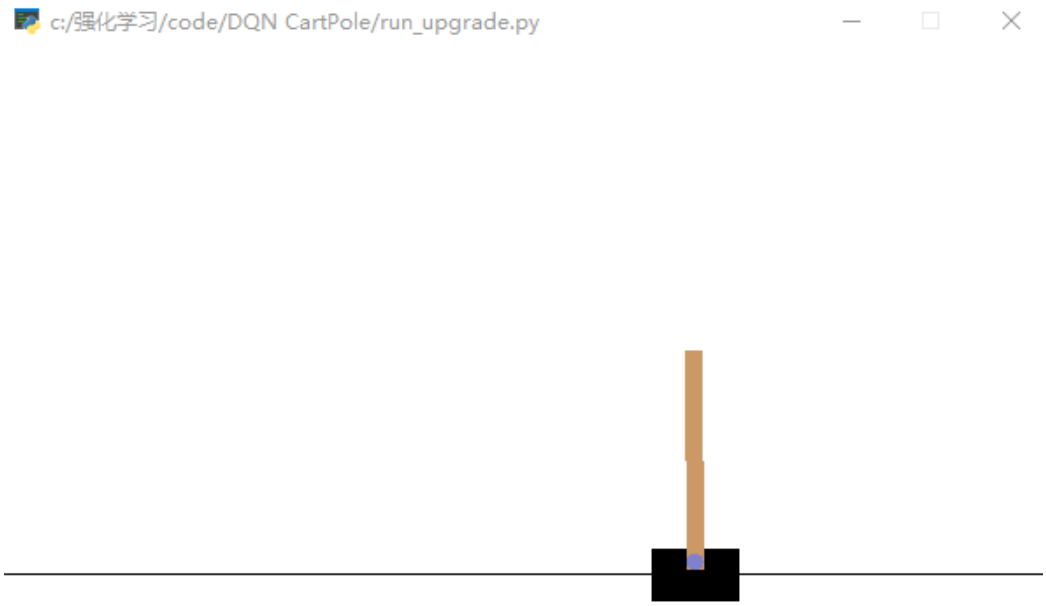
DQN+Sarsaλ



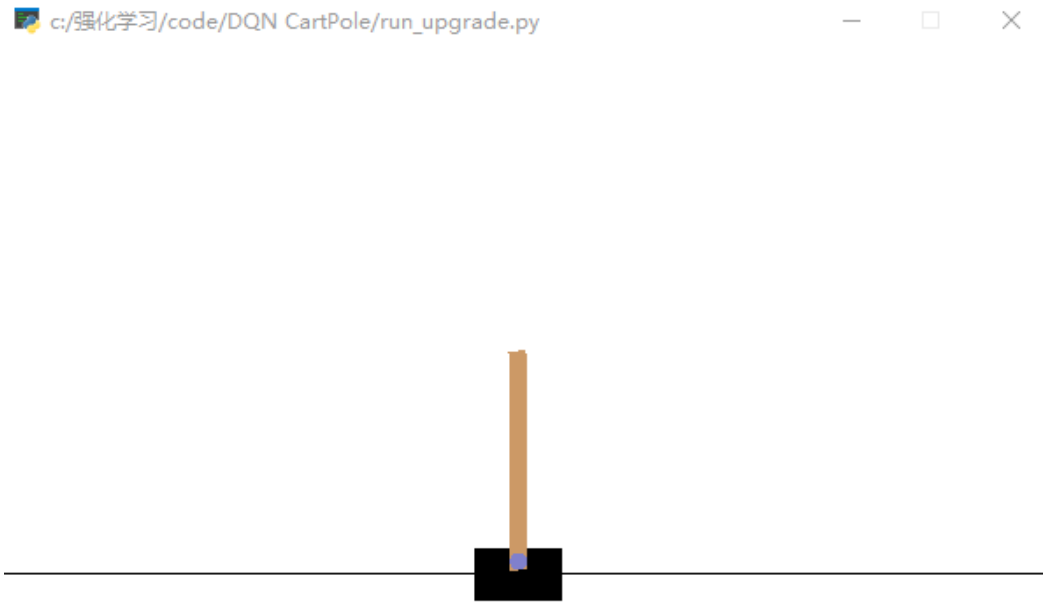
开始训练，十分容易倒。



从这个图中，可以看出这一算法很想把偏离的倒立摆纠正回来，可是用力过猛，结果向左倒了。



有时算法会陷入局部最优，在中间偏右的位置立住。虽然非常稳定，但是得分只有3000到4000，因为偏离了中心。不过这个得分已经比原版DQN好太多了。



这次算法没有陷入局部最优，经过200多次的训练，稳稳地立在最中间，由于随机干扰会出现一些波动，但都能很快修正。平均得分在5800以上。

5 结论和展望

sarsa算法在本项目中被证明是更优于dqn的一种算法，它收敛更加快速和稳定。

dqn在本次测试中，较快速地收敛到了局部最优，能够保持直立却不能稳定在中心，此现象在超过四千个episode后仍然未改变，推断dqn在本任务中陷入了局部最优

sarsa快速地收敛，能长时间地保持在中心直立，并且在后续的训练中，成绩稳定而不发生突然的下降。认为sarsa有效地收敛到了全局最优。

这些对比说明了sarsa能够在较高学习率下稳定快速收敛，且更不易陷入局部最优，这主要是由于结合了sarsaλ的算法通过对路径上的每个状态记录重要程度，然后整体性地按回合更新，避免了dqn每步直接更新导致的陷入局部最优和收敛过程中参数的震荡。

目前对神经网络的考察缺乏有效的方法，只能给出猜测，然后通过实验验证。例如在本项目中，对于sarsa优于dqn的原因只给出了猜想，并未详细地，形式化地解释和证明这种差距背后的机理。在将来的研究中，除了不断提出，改进新算法，也有必要加强神经网络的可解释性。做到知其然更知其所以然，相信当我们对神经网络有了深刻的理解之后，我们目前遇到的大量问题（过拟合，参数震荡，超参调节，对网络结构的选择）等问题给出详细的解释，使得对神经网络的研究摆脱经验化的局面。

6 参考文献

[1]高阳.强化学习研究综述[D].自动化学报.cnki.com.cn,2004-1

[2]刘全,翟建伟,章宗长,钟珊,周倩,章鹏,徐进.深度强化学习综述[J].计算机学报,2018,41(01):1-27.

[3]唐振韬,邵坤,赵冬斌,朱圆恒.深度强化学习进展:从AlphaGo到AlphaGo Zero[J].控制理论与应用,2017,34(12):1529-1546.

[4]马朋委. Q_learning强化学习算法的改进及应用研究[D].安徽理工大学,2016.

[5]赵婷婷,孔乐,韩雅杰,任德华,陈亚瑞.模型化强化学习研究综述[J/OL].计算机科学与探索:1-11[2020-05-22].http://kns.cnki.net/kcms/detail/11.5602.tp.20200331.1819.002.html.

[6]王欣,王芳.基于强化学习的动态定价策略研究综述[J].计算机应用与软件,2019,36(12):1-6+18.

[7]张汝波.强化学习理论_算法及应用[EB/OL].cnki.com.cn,2000-10.

7 致谢

感谢国家，感谢党，感谢父母，老师，同学。