Project Name

序号	学号	姓名	专业班级	组号
1	3180105630	董嘉华	计科 1803	1
2	3180105508	张佳文	计科 1802	1
3	3180103770	张雯琪	计科 1805	1
4	3180105354	潘子曰	计科 1801	1

1. Project Introduction

内容包括：（这部分内容不要太长，讲清楚即可）

（1） 选题（选择的游戏环境，包括游戏环境的安装）

利用 A3C、DDPG 算法训练智能体玩 CartPole 游戏，选择环境为 OpenAI Gym。

（2） 游戏带来的反馈：

在该游戏中，一根杆通过非驱动关节连接到小车上，小车沿无摩擦的轨道滑动。初始状态（推车位置、推车速度、杆的角度和杆子顶端的速度）随机初始化为 +/-0.05。通过对车施加 +1 或 -1（车向左或向右移动）的力对该系统进行控制。杆开始的时候是直立的，游戏目标是防止杆倒下。杆保持直立过程中的每个时间步都会得到 +1 的奖励。

A. **游戏终止条件：**当杆倾斜 15 度以上或小车与中间位置相隔 2.4 个单位时游戏结束。
B. **智能体观测到的 observation：**推车位置、推车速度、杆的角度和杆子顶端的速度。
C. **智能体采取的动作：**推车左右移动
D. **智能体获得的奖励：**游戏没有终止，累计奖励加 1，游戏结束，即时性奖励为 0

（3） 工作简介，即要做什么事情，使用什么算法（算法个数），该算法的特点（简单介绍该算法，包括其优缺点以及适用范围）

实现用 OpenAI 训练出使用 A3C 算法的智能体进行 CartPole 游戏，流程包括建立主要的智能体监管、建立工作智能体、实现 A3C 算法、训练智能体以及将模型表现可视化。

（4） 开发环境及系统运行要求，包括所用的开发工具、开发包、开源库、系统运行要求等

选择的环境是较为简单的 OpenAI Gym

2. Technical Details

内容包括：

（1） 工程实践当中所用到的理论知识阐述

（2） 具体的算法，请用文字、示意图或者是伪代码等形式进行描述（不要贴大段的代码）

（3） 程序开发中重要的技术细节，比如用到了哪些重要的函数？这些函数与算法中哪些数学知识相对应？函数实现了什么样的功能？或者哪些函数是该算法的亮点（收敛性、计算速度等等）？自己编写了哪些重要的功能函数（你的创新）？等等

（一）：A3C

(1)理论阐述：

A3C 算法是在 Actor-Critic 框架的基础上引入了异步训练的思想，在提升性能的同时大大加快了训练速度。其基本思想就是对输出的动作进行好坏评估，如果动作被认为是好的，那么就调整行动网络（Actor Network）使该动作出现的可能性增加。反之如果动作被认为是坏的，则使该动作出现的可能性减少。通过反复的训练，不断调整行动网络找到最优的动作。

A3C 算法的创新和厉害之处在于：A3C 算法为了提升训练速度采用异步训练的思想，利用多个线程。每个线程相当于一个智能体在随机探索，多个智能体共同探索，并行计算策略梯度，对参数进行更新。或者说同时启动多个训练环境，同时进行采样，并直接使用采集的样本进行训练，这里的异步得到数据，相比 DQN 算法，A3C 算法不需要使用经验池来存储历史样本并随机抽取训练来打乱数据相关性，节约了存储空间，并且采用异步训练，大大加倍了数据的采样速度，也因此提升了训练速度。与此同时，采用多个不同训练环境采集样本，样本的分布更加均匀，更有利于神经网络的训练。

(2)算法分析：

A. 建立基线

在模型中通过创建随机的智能体，在环境中做出一些随机行为，最后得到平均值作为基线正确判断模型的实际性能以及评估模型的度量标准。

B. 异步算法

获取全局网络参数，通过遵循最小化（t_max：到终极状态的步长）步长数的局部策略与环境进行交互，计算价值损失（=折扣奖励-损失函数)**2）和策略损失（-log(P(s)) * A(s)），从损失中得到梯度，用梯度更新全局网络，重复。

C. 运行算法

在每一个 episode 中，进行如下操作，基于现有框架得到策略（行为概率分布），根据策略选择行动，如果智能体已经做了一些操作（args.update_freq）或者说智能体已经达到了终端状态（结束），那么用从局部模型计算得到的梯度更新全局模型，重复。

(3)技术细节：

A.通过随机抽取样本建立基线以获得对模型评估的度量标准

B.建立 Memory 类来实现对于模型数据更简单的追踪，提高程序的可读性。

C.A3C 算法比较便利的是采用了异步的思想，虽然在程序运行中一个单独的 python 过程不能并行多个线程，从技术上讲不能称为真正的异步，但可以在 I/O 密集型操作过程转换上下文从而同时运行它们

(二)：DDPG

(1)理论阐述：

DDPG 相比 DQN 是一种 model free（无环境模型）, off-policy（产生行为的策略和进行评估的策略不一样）的强化学习算法。DDPG (Deep Deterministic Policy Gradient)算法也是 model free, off-policy 的，且同样使用了深度神经网络用于函数近似。但与 DQN 不同的是，DQN 只能解决离散且维度不高的 action spaces 的问题。而 DDPG 可以解决连续动作空间问题。另外，DQN 是 value based 方法，即只有一个值函数网络，而 DDPG 是 actor-critic 方法，即既有值函数网络(critic)，又有策略网络(actor)。

这两个网络之间的联系是这样的：首先环境会给出一个 obs，智能体根据 actor 网络做出决策 action，环境收到此 action 后会给出一个奖励 Rew，及新的 obs。这个过程是一个 step。此时要根据 Rew 去更新 critic 网络，然后沿 critic 建议的方向去更新 actor 网络。接着进入下一个 step。如此循环下去，直到训练出了一个好的 actor 网络。

(2)算法分析：

A：建立价值评判和决策网络，采用相同的结构，都为三层全连接网络，其中的激活函数为 relu 来提高网络的非线性特征。

B：设定噪声 Ornstein Uhlenbeck noise 和 brownian motion noise，对于前 200 个训练进行处理，提供更多探索度,具体来看，Ornstein Uhlenbeck noise 的噪音为 np.sqrt(self.delta) * self.sigma，brownian motion noise 的噪音为 self.ou_a * (self.ou_mu - prev_ou_level) * self.delta+

self.brownian_motion_log_returns()+prev_ou_level，从而更好的提升了模型的性能。

C：训练时，采用单线程，根据游戏结束状态决定是否继续游戏，决策时，调用决策（predict）函数根据当前观测状态进行决策，为了提高程序的性能，同时增强效率，仅对于前面部分的训练添加噪音。

之后，计算相应的收益和，并根据决策行为的结果，提升评判网络和行为决策网络，这里值得注意的是，相比 A3C，DDPG 采用了单步调整的方式，在每一步游戏时都进行优化，提升了效率（这里也因此存在相应的问题，如梯度的不稳定），再将结果加入到统计中，训练的算法大致得到完成

(3)技术细节：

1、注意到，这个 DDPG 工程的噪音具有创新性，一方面，采用了不同的噪音，另一方面对于噪音的使用范围进行了设定，使得整个模型更加优化。

2、对于收益，模型进行了特殊的处理，即 reward discounting，在不断优化的同时，对收益提出了更高的要求，看起来 reward 在后期保持不变，实际上在一直增大，这也是在保证一定训练效率的同时，提升性能的关键。（这部分较难想到，但十分精髓）

3、对于本模型中采取的网络均为全连接层，而未使用卷积等网络，根据经验来看，卷积网络实际上能够达到更优的效果，经过尝试后，确实如此，但为了保证运行的稳定性（性能受限），最终仍保留了全连接层的设计。

3. Experiment Results

用图文并茂的形式给出实验结果，如系统界面、操作说明、运行结果（这里可以截图，视频另附）等，并对实验结果进行总结和说明（如果有多算法，请比较）。

（一）A3C：

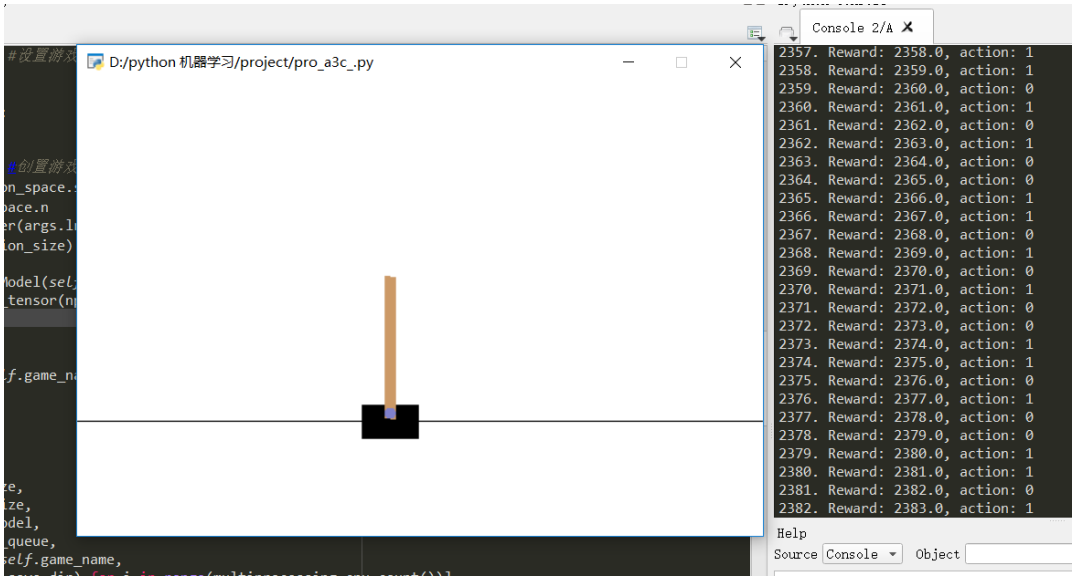
1.系统界面（部分）

```
124
125 class MasterAgent():   #总训练的类（主智能体）
126   def __init__(self):  #训练对象实例化
127     self.game_name = 'CartPole-v0'  #设置游戏
128     save_dir = args.save_dir
129     self.save_dir = save_dir
130     if not os.path.exists(save_dir):
131       os.makedirs(save_dir)
132
133     env = gym.make(self.game_name)  #创置游戏
134     self.state_size = env.observation_space.shape[0] #初始化
135     self.action_size = env.action_space.n
136     self.opt = tf.train.AdamOptimizer(args.lr, use_locking=True) #设置优化器为Adam
137     print(self.state_size, self.action_size)
138
139     self.global_model = ActorCriticModel(self.state_size, self.action_size)  # global network
140     self.global_model(tf.convert_to_tensor(np.random.random((1, self.state_size)), dtype=tf.float32))
141
142   def train(self):   #训练方法
143     if args.algorithm == 'random':
144       random_agent = RandomAgent(self.game_name, args.max_eps) #创造训练器
145       random_agent.run()
146       return
147
148     res_queue = Queue()
149
150     workers = [Worker(self.state_size,
151                       self.action_size,
152                       self.global_model,
153                       self.opt, res_queue,
154                       i, game_name=self.game_name,
155                       save_dir=self.save_dir) for i in range(multiprocessing.cpu_count())]
156
157     for i, worker in enumerate(workers):
158       print("Starting worker {}".format(i))
159       worker.start() #进行运作
```

2.操作说明

输入'1'进行训练，输入‘2’运行程序。如果想要得到不会倾倒的 pole，需要先训练再运行。

3.运行结果



经过了不到 1min 的训练之后，可以看到每次的反馈值可以达到 2000+，系统已经极其稳定，杆子只有细微地抖动。

（二）DDPG：

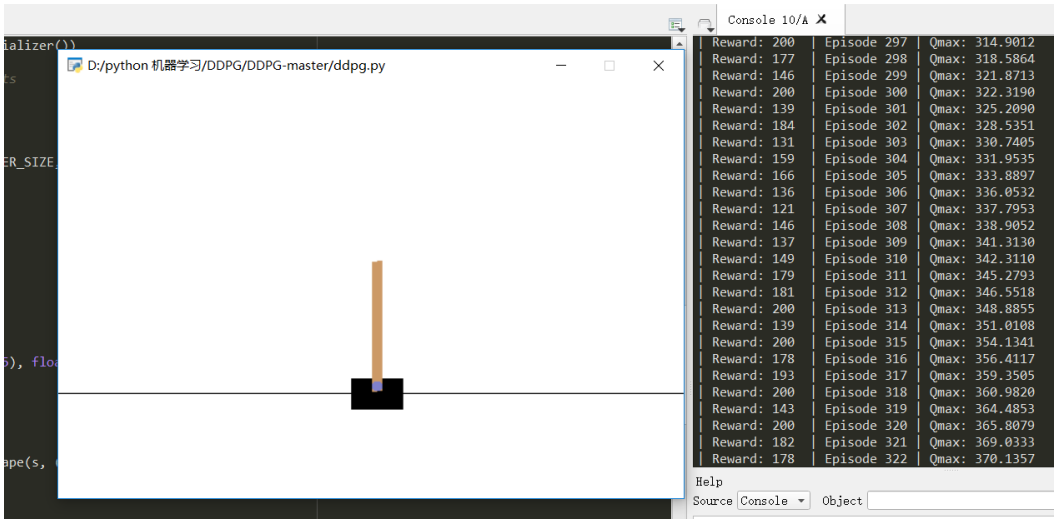
1.系统界面（部分）

```
def main(_):
    with tf.Session() as sess:
        env = gym.make(ENV_NAME)
        np.random.seed(RANDOM_SEED)
        tf.set_random_seed(RANDOM_SEED)
        env.seed(RANDOM_SEED)

        print(env.observation_space)
        print(env.action_space)

        state_dim = env.observation_space.shape[0]

        try:
            action_dim = env.action_space.shape[0]
            action_bound = env.action_space.high
            # Ensure action bound is symmetric
            assert (env.action_space.high == -env.action_space.low)
            discrete = False
            print('Continuous Action Space')
        except :  #原来的对象抛出处理不了这里的异常，此处更换为全部处理
            action_dim = env.action_space.n
            action_bound = 1
            discrete = True
            print('Discrete Action Space')

        actor = ActorNetwork(sess, state_dim, action_dim, action_bound,
                             ACTOR_LEARNING_RATE, TAU)

        critic = CriticNetwork(sess, state_dim, action_dim,
                               CRITIC_LEARNING_RATE, TAU, actor.get_num_trainable_vars())

        noise = Noise(DELTA, SIGMA, OU_A, OU_MU)
        reward = Reward(REWARD_FACTOR, GAMMA)
```

2.操作说明

点击运行进行训练，在 ddpg.py 文件中 s2, r, terminal, info = env.step(action)之上添加一行代码 env.render()即可输出 Cartpole 图像。

3.运行结果



经过训练之后发现当 Qmax>800 后图像稳定性基本可以达到要求，详见视频。

（三）算法比对：

因为在 ddpg 中，采用了 reward discount 的方式，使得单纯的 reward 比较失去意义（两者的度量出现不同），且 ddpg 的表现形式限定在了 200，经过测试，为了精准比较两个算法的收敛速度、计算效率，采用了测试达到稳定 200 的时间（即 3 次连续 200 及以上）。

测试结果：A3C：平均 40s，DDPG:平均 1min20s

从测试结果来看，A3C 因为异步的结构，测试效率要显著高于 DDPG，DDPG 虽然采用了步步优化的方式，但是也正因此带来了更多偏差，不过 A3C 同样存在一些问题，在测试过程中，我们发现 A3C 容易因为一些 Worker 的偏差而导致整体模型的 dead,回到非常差的训练状态，有时又难以进步。

相比之下，DDPG 的稳定性更高，而且因为其采用的 reward discount 的方式，避免了单次训练时间过长（模型难以输掉游戏）带来的时间损失，这一点十分具有借鉴意义。

References:

A. https://github.com/tensorflow/models/blob/master/research/a3c_blogpost/a3c_cartpole.py

B. https://github.com/liampetti/DDPG/blob/master/ddpg.py

备注：

代码中请给出较为详细的注释。

大作业以及前面的作业请在老师给出的截止日期前（好像是 21 号）提交