# [DPPO]:再看我的影分身之术（附代码及代码分析）

[](https://www.zhihu.com/people/zhang-si-jun-52)

[张斯俊](https://www.zhihu.com/people/zhang-si-jun-52" \t "/home/admin/文档\\x/_blank)

愿成为一把梯子，助你跨过无数的坑。

DPPO和A3C的思路其实是一致的，希望用多个智能体同时和环境互动，并对全局的PPO网络进行更新。

在A3C，需要跑数据并且计算好梯度，再更新全局网络。这是因为AC是一个在线的算法，所以在更新的时候，产生数据的策略和更新的策略需要时同一个网络。所以不能把worker产出的数据，直接给全局网络计算梯度用。

但PPO解决了离线更新策略的问题，所以DPPO的工人只需要提供数据给全局网络，由全局网络从数据中直接学习。

回到A3C中的例子：

A3C的班长并不直接学习，学习的是其他同学，由其他同学把自己的学习心得告诉班长；

而DPPO中，其他同学相当于帮助班长在原文中划重点，把画好的重点给班长亲自学习。

如果想对A3C有更多了解，可以看看专栏前一篇文章。

### 线程之间的通信

和A3C一样，DPPO同样使用多线程来学习。

全局网络，就是PPO，他的功能是学习；

工人，在全局网络的“指导”下，和环境互动，并且把数据保存起来提供给全局网络学习。

你可能会发现，这两项工作之间是不能同时的。

当全局网络在学习的时候，工人们需要等待全局网络学习完，才能干活；工人们在干活的时候，全局网络就需要等待工人提供数据。

那怎么协调这两者的工作呢？

就需要用到线程之间的通信——事件(event)

### 事件

可以把每一条线程想象成一根水管，线程按照代码运行，就像水在水管里面运行一样。

如果某些线程需要等待其他的线程工作完成以后，再继续进行。那么，这些水管就要安装上开关。

安装开关的方法很简单，只需要在需要安装的位置，加上这么一句代码，让代码执行进入等待状态。

event.wait()

线程运行到这里，将会暂停。称为该线程被“阻塞”了。那么怎么才能通过这个开关，继续运行下去呢。先可以想象有一个管理开关的工人，该工人会看“信号灯”，根据信号灯的情况决定是否让线程继续下去。

有两个操作，可以改变信号灯。 event.clear()，信号为False，表示“不能”让程序进行。 event.set(), 信号为Ture，表示“可以”让程序进行下去。

这样，用两个事件，就能解决两组不同线程的通信问题。

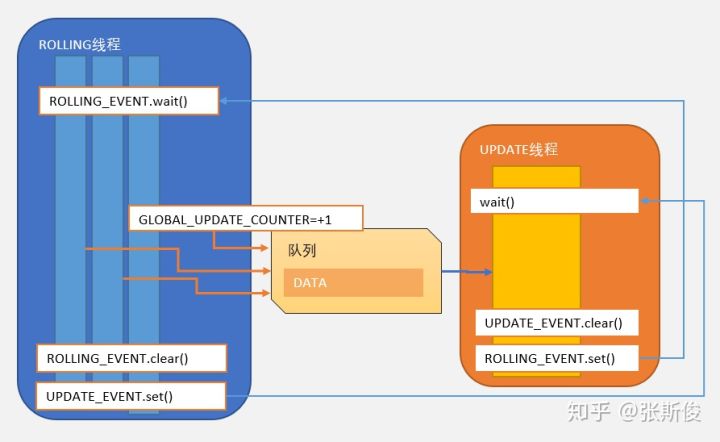
假如，有两组线程A，B。要求当A在运行时，B等待；当A运行完毕，B开始运行，A等待。那么可以这样做。

注意，这里是两组，只要线程的功能和需求相同，就可以把这些线程看成一组。由一组开关进行协调即可。就像的工人线程，完全可以看成一组线程，由一套开关控制。

1. 首先在A,B两组不同线程的适当位置，插入“开关”：分别是A\_event.wait(), B\_event.wait()，这时候，两个开关的信号灯都默认为False，就是阻塞状态。
2. 把A\_event的信号设置为True；B\_event信号设置为False：A\_event.set(), B\_event.clear()，这样，当进程start之后，A进程运行会继续，B进程会在B\_event.wait()被阻塞，等待A进程运行。
3. A进程运行到最后，需要B进程启动。A\_event.clear(),B\_event.set()。这时候，B进程在B\_event.wait()位置，看到信号灯设置为True，就会继续执行。但这时候，A并不会立即停止，通常A是一个多次的循环，这时候A会重新开始，并执行到A\_event.wait()的位置，由于这时候，信号为False。所以A就不再执行下去了，需要等待信号灯变更。
4. 同理，需要在B最后，设置信号。A\_event.set(), B\_event.clear()。这样AB两个线程就会交替进行下去。

### 原理简述

和A3C一样，并没有很费脑子理解的概念。简单来说，就是两组线程交替执行。



要注意，每个worker产生的数据，将会存在同一个队列。可以留意全局变量： GLOBAL\_UPDATE\_COUNTER

每个线程中，智能体和环境交互的一步，全局变量GLOBAL\_UPDATE\_COUNTER都会+1。当GLOBAL\_UPDATE\_COUNTER达到batch时，每个线程各自整理自己的数据，然后加入到队列中。

当UPDATE线程开始后，便会使用队列中的数据进行更新，更新的方法和PPO是相同的。

### 示例代码

所以，DPPO的核心就是多线程以及多线程之间的通信。为了大家能够更好理解整个结构。我把源代码进行了简化。以便大家更好的理解DPPO的结构。

tensorflow提供的示例代码：

[https://github.com/tensorlayer/tensorlayer/blob/master/examples/reinforcement\_learning/tutorial\_DPPO.py​](https://link.zhihu.com/?target=https://github.com/tensorlayer/tensorlayer/blob/master/examples/reinforcement_learning/tutorial_DPPO.py" \t "/home/admin/文档\\x/_blank)

[github.com](https://link.zhihu.com/?target=https://github.com/tensorlayer/tensorlayer/blob/master/examples/reinforcement_learning/tutorial_DPPO.py" \t "/home/admin/文档\\x/_blank)

带注释版：

[https://github.com/louisnino/RLcode/blob/master/tutorial\_DPPO.py​github.com](https://link.zhihu.com/?target=https://github.com/louisnino/RLcode/blob/master/tutorial_DPPO.py" \t "/home/admin/文档\\x/_blank)

简单的线程结构代码：

[https://github.com/louisnino/RLcode/blob/master/theading\_demo.py​github.com](https://link.zhihu.com/?target=https://github.com/louisnino/RLcode/blob/master/theading_demo.py" \t "/home/admin/文档\\x/_blank)

首先来看看主流程：

和A3C一样，需要先实例化需要用到的对象：工人们和PPO算法。 注意，在示例代码中，只用了一行代码就完成了，非常值得学习。

workers = [Worker(wid=i) for i in range(N\_WORKER)]

然后需要两个事件，一个用于控制ROLLING线程，也就是工人干活的线程。一个用于UPDATE线程。 会先使用ROLLING现程，再使用UPDATE线程，所以把ROLLING线程用set函数设置为恢复，UPDATE线程用clear函数设置为关。

UPDATE\_EVENT,ROLLING\_EVENT = threading.Event(), threading.Event()

UPDATE\_EVENT.clear()

ROLLING\_EVENT.set()

开启线程

#开启rolling线程

for worker in workers: #三个rolling线程

t = threading.Thread(target=worker.work) #线程的功能就是执行work函数

t.start()

threads.append(t)

#开启update线程

threads.append(threading.Thread(target=GLOBAL\_PPO.update,))

threads[-1].start() #启动最后加入的线程，就是update线程

注意，必须先把对象实例化，才能使用他们各自的函数。另外最后一句，很机智地使用了[-1]表示最后一个加入列表的元素，也就是PPO的update进程，然后start。

把进程都放到协调器中，在进程中，控制协调器开关就可以了。

现在看看worker.work函数：

while not COORD.should\_stop():

for \_ in range(EP\_LEN): #开始新的EP

if not ROLLING\_EVENT.is\_set():

ROLLING\_EVENT.wait()

QUEUE.put(self.wid)

GLOBAL\_UPDATE\_COUNTER += 1

if GLOBAL\_UPDATE\_COUNTER >= MIN\_BATCH\_SIZE: #更新大于

ROLLING\_EVENT.clear()

UPDATE\_EVENT.set()

if GLOBAL\_EP >= EP\_MAX:

COORD.request\_stop()

break

先来看看这几行代码。

1. 判断是否应该终止所有进程了。如果更新的次数（GLOBAL\_EP）大于设定好的次数(EP\_MAX)。那么就把所有线程停止了。所以需要先判读一下，是否已经停止了。
2. 和其他算法一样，开始一步一步和环境进行互动了。
3. 判读ROLLING\_EVENT的标志是否是set，也就是是否能通过。否则，在这里等待。
4. 在这里我简化了原来的示例代码，直接把工人的id放入到队列中。在DPPO中，这部分应替换为智能体和环境互动，并保存数据到各自buffer的代码。
5. 每个智能体，每进行一步，多一条数据，全局变量GLOBAL\_UPDATE\_COUNTER+1。所以GLOBAL\_UPDATE\_COUNTER相当于所有worker在上次更新至今，产生数据的条目数量。
6. 当条目数量大于设置值(MIN\_BATCH\_SIZE)，就可以开始更新了。所以把UPDATE\_EVENT用set函数，由阻塞设置为恢复。这时候，UPDATE线程已经在等待位置等候了，标志变更后，就马上开始更新。ROLLING\_EVENT用clear函数，设置为阻塞。
7. ROLLING线程并不会马上停下，会一直进行直到遇到wait函数。所以会继续执行下面的判断，如果更新次数大于设定值(EP\_MAX)，就通知协调器停下。

现在来看看PPO.update函数。

while not COORD.should\_stop():

if GLOBAL\_EP <= EP\_MAX:

UPDATE\_EVENT.wait()

'''

这里用输出表示更新

'''

print("====update====")

print("GLOBAL\_EP",GLOBAL\_EP)

print("GLOBAL\_UPDATE\_COUNTER:",GLOBAL\_UPDATE\_COUNTER)

print("update\_old\_pi")

print("Queuesize:",QUEUE.qsize())

print([QUEUE.get() for \_ in range(QUEUE.qsize())])

print("update Critic")

print("update Actor")

print("=====END======")

GLOBAL\_UPDATE\_COUNTER = 0

GLOBAL\_EP += 1

UPDATE\_EVENT.clear()

ROLLING\_EVENT.set()

如果已经明白了work函数，那么update函数也就很清晰了。

1. 判断是否应该停止所有线程。
2. 设置等待位置。
3. 这里应该进行PPO的更新操作，这里用打印代替。
4. 把UPDATE\_EVENT用clear函数设置为阻塞，开始ROLLING进程。

### 总结：

DPPO也没什么烧脑的理论，就是PPO在影分身的应用。

到了这里，专栏的第二阶段已经结束。第三阶段，将会介绍多人强化学习。包括Alpha Zero和 CFR(遗憾最小化算法)。