# 理解Actor-Critic的关键是什么？(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。

现在，终于开始学习鼎鼎大名的Actor-Critic了！

虽然AC算法是如此有名，又如此重要，终于要开始学习了，是不是有点小激动，但又怕学不动呢？

如果前面的基础已经扎实了，理解和实现Actor-Critic一点也不难。

### 什么是AC

关于AC，很多书籍和教程都说AC是DQN和PG的结合。个人觉得道理是这么个道理，但其实是不够清晰，也很容易产生误读，甚至错误理解AC。至于是在哪里容易产生误读，我会在讲解的时候为你说明。

照我的观点来说，PG利用带权重的梯度下降方法更新策略，而获得权重的方法是蒙地卡罗计算G值。

蒙地卡罗需要完成整个游戏过程，直到最终状态，才能通过回溯计算G值。这使得PG方法的效率被限制。

那可不可以更快呢？相信大家已经想到了，那就是改为TD方法。

但修改为TD方法还有一个问题需要解决，就是：在PG，需要计算G值；那么在TD中，应该怎样估算每一步的Q值呢？

有没有发现？

熟悉的问题：其实这个问题在学DQN的时候也遇到过。 熟悉的套路：用上万能的神经网络解决。

也就是说，Actor-Critic，其实是用了两个网络：

两个网络有一个共同点，输入状态S: 一个输出策略，负责选择动作，把这个网络称为Actor； 一个负责计算每个动作的分数，把这个网络称为Critic。

大家可以形象地想象为，Actor是舞台上的舞者，Critic是台下的评委。

Actor在台上跳舞，一开始舞姿并不好看，Critic根据Actor的舞姿打分。Actor通过Critic给出的分数，去学习：如果Critic给的分数高，那么Actor会调整这个动作的输出概率；相反，如果Critic给的分数低，那么就减少这个动作输出的概率。

所以依我的观点来看，AC不是PG+DQN，而应该说AC是TD版本的PG。

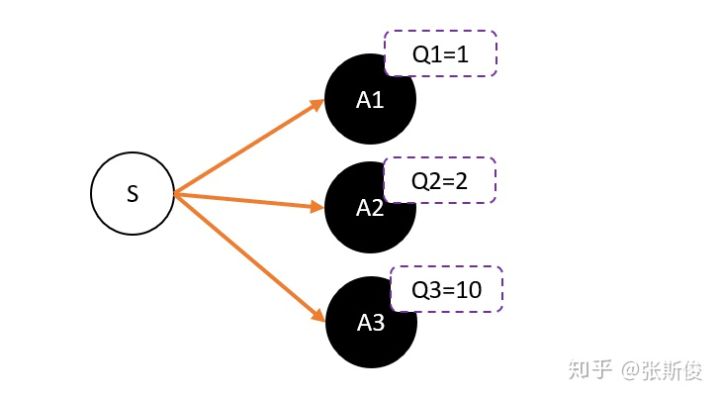
### TD-error

注意:这是AC的重点。很多同学在这里会和DQN搞乱，也就是容易产生误解的地方。在DQN预估的是Q值，在AC中的Critic，估算的是V值。

你可能会说，为什么不是Q值呢？说好是给动作评价呢。

可以直接用network估算的Q值作为更新值，但效果会不太好。

原因可以看下图：



假设用Critic网络，预估到S状态下三个动作A1，A2，A3的Q值分别为1,2,10。

但在开始的时候，采用平均策略，于是随机到A1。于是用策略梯度的带权重方法更新策略，这里的权重就是Q值。

于是策略会更倾向于选择A1，意味着更大概率选择A1。结果A1的概率就持续升高...

这就掉进了正数陷阱。明明希望A3能够获得更多的机会，最后却是A1获得最多的机会。

这是为什么呢？

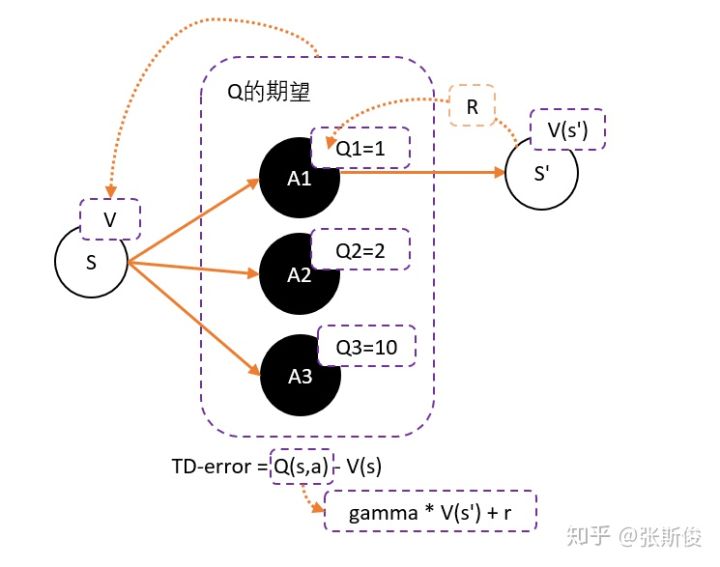
这是因为Q值是一个正数，如果权重是一个正数，那么相当于提高对应动作的选择的概率。权重越大，调整的幅度将会越大。

其实当有足够的迭代次数，这个是不用担心这个问题的。因为总会有机会抽中到权重更大的动作，因为权重比较大，抽中一次就能提高很高的概率。

但在强化学习中，往往没有足够的时间让去和环境互动。这就会出现由于运气不好，使得一个**很好**的动作没有被采样到的情况发生。

要解决这个问题，可以通过减去一个baseline，令权重有正有负。而通常这个baseline，选取的是权重的平均值。减去平均值之后，值就变成有正有负了。

而Q值的期望(均值)就是V。



所以可以得到更新的权重：Q(s,a)-V(s)

随之而来的问题是，这就需要两个网络来估计Q和V了。但马尔科夫告诉，很多时候，V和Q是可以互相换算的。

Q(s,a)用gamma \* V(s') + r 来代替，于是整理后就可以得到：

gamma \* V(s') + r - V(s) —— 把这个差，叫做TD-error

这个和之前DQN的更新公式非常像，只不过DQN的更新用了Q，而TD-error用的是V。

眼尖的同学可能已经发现，如果Critic是用来预估V值，而不是原来讨论的Q值。那么，这个TD-error是用来更新Critic的loss了！

没错，Critic的任务就是让TD-error尽量小。然后TD-error给Actor做更新。

现在再总结一下TD-error的知识：

1. 为了避免正数陷阱，希望Actor的更新权重有正有负。因此，把Q值减去他们的均值V。有：Q(s,a)-V(s)

2. 为了避免需要预估V值和Q值，希望把Q和V统一；由于Q(s,a) = gamma \* V(s') + r - V(s)。所以得到TD-error公式： TD-error = gamma \* V(s') + r - V(s)

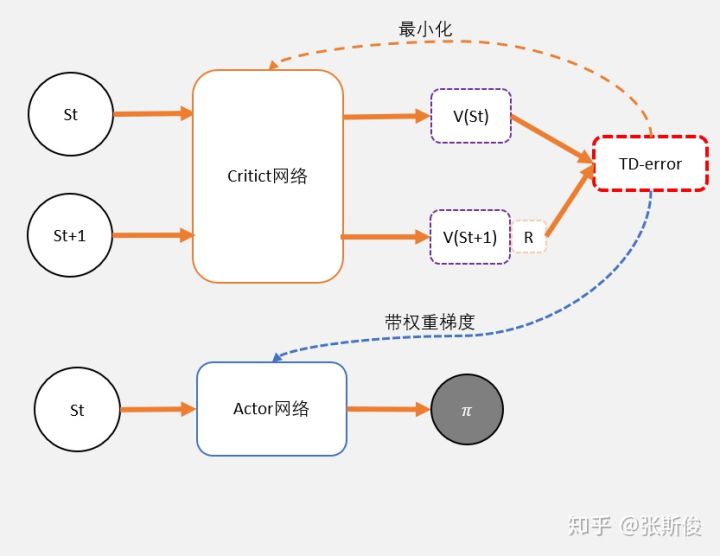
3. TD-error就是Actor更新策略时候，带权重更新中的权重值；

4. 现在Critic不再需要预估Q，而是预估V。而根据马可洛夫链所学，知道TD-error就是Critic网络需要的loss，也就是说，Critic函数需要最小化TD-error。

大家看，其实AC没有很难，关键是对TD-error的理解。理解了TD-error，就能串起Actor和Critic两者。

### 算法

1. 定义两个network：Actor 和 Critic
2. 进行N次更新。
   1. 从状态s开始，执行动作a，得到奖励r，进入状态s'
   2. 记录的数据。
   3. 把数据输入到Critic，根据公式： TD-error = gamma \* V(s') + r - V(s) 求 TD-error，并缩小TD-error
   4. 把数据输入到Actor，计算策略分布 。



### 代码解释

以下，就用tensorflow的AC代码作为示例，一起看看应该如何实现。

tensorflow示例代码：

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

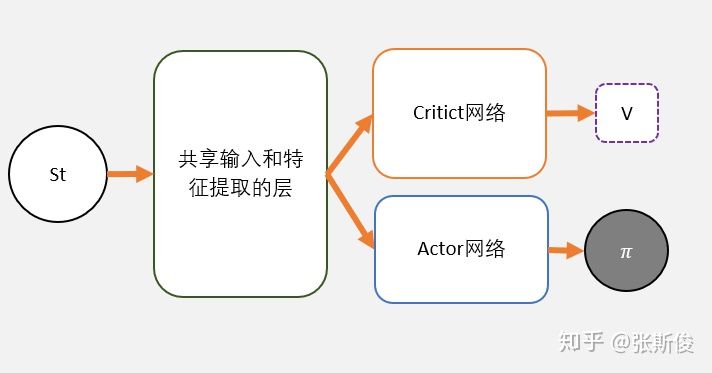
如果看代码有困难，可以看我的带注释版本。希望能帮助到你。

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

**更新流程** 可以把更新流程和PG做对比： 在PG，智能体需要从头一直跑到尾，直到最终状态才开始进行学习。 在AC，智能体采用是每步更新的方式。

但要注意，需要先更新Critic，并计算出TD-error。再用TD-error更新Actor。

在示例代码中，Actor 和 Critic两个网络是完全分离的。但在实现的时候，很多时候会把Actor和Critic公用网络前面的一些层, 例如state是一张图片，先通过几层的CNN进行特征的提取，再分别输出Actor的动作概率分布和Critic的V值。



### 修改reward

if done: r = -20

在更新流程中，有这么一行代码。意思是：如果已经到达最终状态，那么奖励直接扣20点。这是为什么呢？

首先要明确，这个CartPole游戏最终目的，是希望坚持越久越好。所以大家可以想象这么一个过程：在某个濒死状态s下，选择动作a，进入结束状态s，收获r，在CartPole中，这个reward为 1.0。

但并不希望游戏结束，希望智能体能在濒死状态下“力挽狂澜”！

于是把reward减去20，相当于是对濒死状态下，选择动作a的强烈不认同。通过-20大幅减少动作a出现的概率。

再进一步，reward会向前传播，让智能体濒死状态之前状态时，不选择会进入濒死状态的动作，努力避免进入濒死状态。

所以说，reward是一个主观因素很强的数值。当环境返回的reward不能满足的的时候，完全可以进行reward的修改，让智能体更快学习。

### Critic 的学习

def learn(self, s, r, s\_):

v\_ = self.model(np.array([s\_]))

with tf.GradientTape() as tape:

v = self.model(np.array([s]))

## TD\_error = r + lambd \* V(newS) - V(S)

td\_error = r + LAMBDA \* v\_ - v

loss = tf.square(td\_error)

grad = tape.gradient(loss, self.model.trainable\_weights)

self.optimizer.apply\_gradients(zip(grad, self.model.trainable\_weights))

这里重点关注td-error的计算。

### Actor 的学习

Actor 的学习本质上是PG的更新，也就是加权的学习。代码中用了cross\_entropy\_reward\_loss函数。

\_exp\_v = tl.rein.cross\_entropy\_reward\_loss(logits=\_logits, actions=[a], rewards=td[0])

其实这个函数就是在PG中的带参数交叉熵损失函数。

def cross\_entropy\_reward\_loss(logits, actions, rewards, name=None):

cross\_entropy = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(labels=actions, logits=logits, name=name)

return tf.reduce\_sum(tf.multiply(cross\_entropy, rewards))

其功能是完全一样的，只不过tensorlayer把这个函数提出来而已。

### 总结

要掌握AC算法，关键是理解TD-error的来历。不然Critic要最小化TD-error，Actor要把TD-error带参数更新，听上去就会很懵。

理解TD-error本质是Q(s,a)-V(s)来的，而Q(s,a)转为由V(s')+r的形式表示，整个思路就会非常清晰。

AC很强，强在它有无限的潜力。AC出来之后，就有很多厉害的算法基于AC框架，这就包括了下篇要说的，OpenAI都拿它做基准的PPO算法。