





听见下雨的声音

    
首页 分类 关于 归档 标签

【David Silver强化学习公开课之七】Policy Gradient

 发表于 2016-08-03 |  分类于 [project experience](#) |  |  1619

本文是David Silver强化学习公开课第七课的总结笔记。这一课主要讲了将policy看成某个参数 θ 的函数，即将policy形式变成状态和动作的概率分布函数，在policy函数可微的情况下能够通过对参数求导来优化policy。

【转载请注明出处】chenrudan.github.io

本文是David Silver强化学习公开课第七课的总结笔记。这一课主要讲了将policy看成某个参数 θ 的函数，即将policy形式变成状态和动作的概率分布函数，在policy函数可微的情况下能够通过对参数求导来优化policy。

本课视频地址:[RL Course by David Silver - Lecture 7: Policy Gradient Methods](#)

本课ppt地址:http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/pg.pdf

文章的内容是课程的一个总结和讨论，会按照自己的理解来组织。个人知识不足再加上英语听力不是那么好可能会有一些理解不准的地方，欢迎一起讨论。

建了一个强化学习讨论qq群，有兴趣的可以加一下群号595176373或者扫描下面的二维码。



1.内容回顾

在第一课的内容中，曾经提到过agent的分类，将value function、policy、model(environment)进行组合可以得到model-based、policy-based、model-free、value-based、actor critic五种类型。其中value-based是说已知policy情况下学习value function，就像第四课的内容。policy-based是指没有显式的值函数形式，而需要学习policy。actor critic则是需要同时学习学习值函数和policy的形式。

带有随机性的policy往往比确定性的policy效果要好，就如第一课中所说的确定性的policy通常会被过利用。Policy-based类型能学习到带有随机性的policy，并且对于高维的action更有效。后者是因为在用value-based的方法时，常常要最大化当前的值函数，会需要大量的计算。

在之前的课程中，policy都是基于greedy或者 ϵ -greedy的方法直接从值函数中获得。本课主要内容是为policy引入参数，变成在某个状态和某组参数下选择某个动作的概率分布 $\pi_{\theta}(s,a)=P[a|s,\theta]$ ，直接求解策略，从已有的sample的experience中学习如何让policy变得更好。为什么要求解含参数的policy也是跟上节课一样的原因，现实

问题中有大量的state和action，无法针对每个state每个action都有一个确定的policy，因此需要一定的泛化能力，面对没有见过的state或者action有一定的决策能力。

将Policy表达成参数 θ 的目标函数，有如下几种形式，start value是针对拥有起始状态的情况下求起始状态 s_1 的reward，average value针对不存在起始状态而且停止状态也不固定的情况，在这些可能的状态上计算平均的reward，或者只计算immdiate reward的期望。其中 $d^{\pi_{\theta}}(s)$ 指状态分布函数。

start value	average value	average reward per time-step
$J_1(\theta) = V^{\pi_{\theta}}(s_1)$	$J_{avV}(\theta) = \sum_s d^{\pi_{\theta}}(s) V^{\pi_{\theta}}(s)$	$J_{avR}(\theta) = \sum_s d^{\pi_{\theta}}(s) \sum_a \pi_{\theta}(a s) R(s,a)$

2.Monte-Carlo Policy Gradient

假设 π_{θ} 是可微的，且梯度为 $\nabla_{\theta} \pi_{\theta}(s,a) = \pi_{\theta}(s,a) \frac{\nabla_{\theta} \pi_{\theta}(s,a)}{\pi_{\theta}(s,a)} = \pi_{\theta}(s,a) \nabla_{\theta} \log \pi_{\theta}(s,a)$ ，第二项称为score function。如果假设不同形式的policy，就能得到不同形式的score function，例如假设policy表达式与特征线性组合（特征向量为 $\phi(s,a)$ ）的指数成线性关系 $\pi_{\theta}(s,a) \propto e^{\phi(s,a)^T \theta}$ ，求出的score function为 $\nabla_{\theta} \log \pi_{\theta}(s,a) = \phi(s,a) - E_{\pi_{\theta}}[\phi(s,\cdot)]$ 。按照我的理解，从公式角度来说，第一项是当前输入特征向量与期望的差值，第二项是输入特征向量的期望，如果当前输入特征向量出现次数多而且表现比较好，但是距离期望比较远，那就要调整policy来更加趋近于当前特征向量，这个差值代表了当前特征向量与期望之差，距离远调整的就多，距离近调整的就少。

从上面的目标函数形式中我觉得目标是在优化reward，就还是在优化值函数，只是这里 θ 不是值函数的参数，而是policy的参数。如果目标函数对参数求导，可以得到policy的gradient的表达式为 $\nabla_{\theta} J(\theta) = E_{\pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(s,a) Q^{\pi_{\theta}}(s,a)]$ 。第一项衡量了policy朝当前选择(某个状态+某个动作)偏移的程度，第二项衡量了当前选择的好坏。

从而推导出Monte-Carlo Policy Gradient的形式，首先更新参数的方法是随机梯度下降+policy gradient，gradient中的动作值函数取值用执行过程中的return来代替。

- 随机初始化 θ
- 针对每个episode $s_1, a_1, r_2, \dots, s_{T-1}, a_{T-1}, r_T$ π_{θ}
- ==从t=1到T-1
- ====更新 $\theta = \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) v_t$

3.Actor-Critic Policy Gradient

上面的方法是用episode中反馈的return当做是动作值函数的采样，如果不当成采样而是采用上节课的value function approximation的方法，即迭代更新policy又更新值函数。policy的gradient稍微做了一点变化 $\nabla_{\theta} \log \pi_{\theta}(s,a) Q_w(s,a)$ ，具体流程如下

- 初始化起始状态 s, θ
- 基于初始policy采样得到下一次执行的动作 a
- 每走一步
- ==得到奖赏 r ，和转移到下一个时刻的状态 s'
- ==基于下一个时刻的状态再选择一个动作 $a' \pi_{\theta}(s', a')$
- ==求值函数参数梯度 $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
- ==更新policy参数 $\theta = \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(s,a) Q_w(s,a)$
- ==更新值函数参数 $w = w + \beta \delta \phi(s,a)$
- ==更新动作和状态 $a = a', s = s'$

这节课实在是太抽象了T^T。。没有啥例子，完全无法直观的感受一下policy如何表达成 θ 的函数形式。

文章目录

站点概览

- 1. 1.内容回顾
- 2. 2.Monte-Carlo Policy Gradient
- 3. 3.Actor-Critic Policy Gradient

函数

Learning and Planning(对Environment建立模

[文章目录](#) [站点概览](#)

Disqus 无法加载。如果您是管理员，请参阅[故障排除指南](#)。

- [1. 1.内容回顾](#)
- [2. 2.Monte-Carlo Policy Gradient](#)
- [3. 3.Actor-Critic Policy Gradient](#)