# 十分钟入门策略梯度算法(Policy Gradient Algorithms)

Yanjie Ze, June 2021

# 1 什么是Policy Gradient?

Policy Gradient（策略梯度）是一种解决强化学习问题的算法。如果你还没有深入了解过强化学习这个领域，建议先阅读这篇博客：“A (Long) Peek into Reinforcement Learning » Key Concepts”，来获得强化学习的问题定义和关键概念。

## 1.1 变量定义

下面这张表是之后要用到的变量定义，都是RL中比较常见的概念，在此不赘述。

| Symbol | Meaning |
|---|---|
| $s \in \mathcal{S}$ | States. |
| $a \in \mathcal{A}$ | Actions. |
| $r \in \mathcal{R}$ | Rewards. |
| $S_t, A_t, R_t$ | State, action, and reward at time step $t$ of one trajectory. I may occasionally use $s_t, a_t, r_t$ as well. |
| $\gamma$ | Discount factor; penalty to uncertainty of future rewards; $0 < \gamma \leq 1$. |
| $G_t$ | Return; or discounted future reward; $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. |
| $P(s', r\|s, a)$ | Transition probability of getting to the next state $s'$ from the current state $s$ with action $a$ and reward $r$. |
| $\pi(a\|s)$ | Stochastic policy (agent behavior strategy); $\pi_\theta(.)$ is a policy parameterized by $\theta$. |
| $\mu(s)$ | Deterministic policy; we can also label this as $\pi(s)$, but using a different letter gives better distinction so that we can easily tell when the policy is stochastic or deterministic without further explanation. Either $\pi$ or $\mu$ is what a reinforcement learning algorithm aims to learn. |
| $V(s)$ | State-value function measures the expected return of state $s$; $V_w(.)$ is a value function parameterized by $w$. |
| $V^\pi(s)$ | The value of state $s$ when we follow a policy $\pi$; $V^\pi(s) = \mathbb{E}_{a\sim\pi}[G_t\|S_t = s]$. |
| $Q(s, a)$ | Action-value function is similar to $V(s)$, but it assesses the expected return of a pair of state and action $(s, a)$; $Q_w(.)$ is a action value function parameterized by $w$. |
| $Q^\pi(s, a)$ | Similar to $V^\pi(.)$, the value of (state, action) pair when we follow a policy $\pi$; $Q^\pi(s, a) = \mathbb{E}_{a\sim\pi}[G_t\|S_t = s, A_t = a]$. |
| $A(s, a)$ | Advantage function, $A(s, a) = Q(s, a) - V(s)$; it can be considered as another version of Q-value with lower variance by taking the state-value off as the baseline. |

## 1.2 Policy Gradient

强化学习的目的是：找到一个agent的最优policy，以此获得最优的reward。

Policy Gradient方法的目的就是直接建模与优化policy。policy一般用一个参数$\theta$来建模，即$\pi_\theta(a|s)$。我们要优化的目标也依赖于这个policy。

**那么，优化的目标是什么呢** 即reward function，定义如下：

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\pi(s)V^\pi(s) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s)Q^\pi(s, a)$$

其中，$d^{\pi}(s)$是在policy $\pi$下的Markov chain的稳态分布。第二个等号的成立是来自V和Q的定义。

（为了方便起见，接下来的内容都将$\pi_\theta$省略为$\pi$，但是我们应该注意policy的参数是$\theta$。）

**上面提到的稳态分布是什么意思呢？**想像一下你在Markov chain上随波逐流，当时间趋于无穷时，你到达某个state的概率会不再变化：

$$d^{\pi}(s) = lim_{t\to\infty} P(s_t = s|s_0, \pi_\theta)$$

这就是稳态分布了。

很自然地，我们期望policy-based的RL算法在连续的状态空间或动作空间中更加有效，因为value-based的RL算法（如Q-learning）在连续空间中的时间花费太大了。

而在policy-based算法中，使用gradient ascent，即梯度上升，我们可以改变$\theta$，向让我们的目标函数$J(\theta)$更大的方向前进，这个方向当然就是梯度$\nabla J(\theta)$。

## 1.3 Policy Gradient Theorem

计算$\nabla J(\theta)$的是比较tricky的，因为它既依赖于action的选择（由policy直接决定），又依赖于稳态分布d(s)（由policy间接决定）。考虑到environment处于unknown的状态，即这些状态转移的稳态分布是很难估计的。我们遇到困难了。

幸运的是，**Policy Gradient Theorem**拯救了一切！

它提供了一个我们的目标函数的变体，不包含稳态分布d(s)的导数，并且简化了$\nabla J(\theta)$的计算很多：

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \sum_{s\in S} d^{\pi}(s) \sum_{a\in A} Q^{\pi}(s,a)\pi_\theta(a|s) \\ &\propto \sum_{s\in S} d^{\pi}(s) \sum_{a\in A} Q^{\pi}(s,a)\nabla_\theta \pi_\theta(a|s)\end{aligned}$$

可以看到，第二行的式子不含d(s)的导数，只有policy的导数。

### 1.4 Policy Gradient Theorem的证明

这一部分在Sutton的那本《强化学习》里有写，因为比较复杂，在这里不重复摘抄。

# 2 Policy Gradient Algorithms

---

近年来，很多policy gradient算法被提出，下面介绍一些。

## 2.1 REINFORCE

REINFORCE，一种Monte-Carlo policy gradient算法，依赖于用Monte-Carlo估计的return来更新参数$\theta$。

REINFORCE可以work是因为sample gradient的期望和真实的gradient一样。

梯度公式：

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] \\ &= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)] \qquad ; \text{Because } Q^\pi(S_t,A_t)=\mathbb{E}_\pi[G_t|S_t,A_t]\end{aligned}$$

我们可以通过采样的trajectory来获得$G_t$（即discounted reward的和，可见前面的变量定义表）。由于一次更新需要一个完整的trajectory，REINFORCE被称为Monte-Carlo方法。

算法流程：

1. 随机初始化$\theta$。
2. 根据policy $\pi_\theta$生成一个trajectory: $S_1, A_1, R_2, S_2, A_2, \ldots, S_T$
3. 对于$t = 1, 2, \ldots, T$:
    a. 估计$G_t$
    b. 更新参数: $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta ln\pi_\theta(A_t|S_t)$

一个广泛采用的REINFORCE的变体是给$G_t$减去一个baseline value，来减少梯度估计的方差。比如，常用的方法是在action value上减去state-value。详情可以看看参考文献[3]。

## 2.2 Actor-Critic

在Policy Gradient里，两个主要部分是policy model和value function。学习一个value function来辅助policy的更新是很有意义的。这就是Actor-Critic算法。

Actor-critc算法由两个模型组成，可以选择共享参数：

- critic更新value function参数w，根据算法不同，可能是$Q_w(a|s)$或是$V_w(s)$，state-action value和state-value。
- actor更新策略参数$\theta$。

让我们看一下一个简单的action-value actor-critic算法：

1. 随机初始化$s, \theta, w$; 采样$a \sim \pi_\theta(a|s)$
2. 对于t=1...T:
    a. 采样reward $r_t \sim R(s,a)$ ,下一个state是 $s' \sim P(s'|s,a)$
    b. 采样下一个action $a' \sim \pi_\theta(a'|s')$
    c. 更新策略参数: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta ln \pi_\theta(A_t|S_t)$
    d. 计算对于action-value在t时刻的correction（TD error）：

    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s,a)$

    然后用它来更新action-value function的参数：

    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
    e. 更新：$a \leftarrow a', s \leftarrow s'$

其中，$\alpha_\theta, \alpha_w$是学习率，人为设置。

# 3 总结

---

这篇文章初步讲了policy gradient的理论来源：policy gradient theorem，并且介绍了两个最经典的policy gradient算法：REINFORCE和Actor-critic。

还有A3C, A2C, DPG, DDPG, D4PG, MADDPG, TRPO, PPO等一系列算法，以后逐步更新。

参考文献：

[1] policy gradient algorithms

[2] A (Long) Peek into Reinforcement Learning » Key Concepts

[3] Fundamentals of Policy Gradients