策略梯度

Skychen
dzhchxk@126.com

Outline

- 介绍
- 蒙特卡罗策略梯度
- Actor-Critic 策略梯度

从Value-based到Policy-based

- 在上一节中，我们使用参数θ 逼近值函数或动作值函数
  - $V_\theta(s) \approx V^\pi(s)$
  - $Q_\theta(s,a) \approx Q^\pi(s,a)$

- 然后策略会从值函数推出
  - E.g. using ϵ-greedy

- 本节我们直接优化策略:
  - $\pi_\theta(s,a) = P[a|s,\theta]$

策略优化

- 用τ表示一组状态-行为序列：$s_0, a_0, \ldots, s_T, a_T$
- 用$R(\tau)$表示轨迹τ的累积回报，$\mathrm{P}(\tau;\theta)$表示此轨迹出现的概率
- 目标函数：$\mathrm{J}(\theta) = \sum_\tau P(\tau;\theta)R(\tau)$
  - 所有轨迹的累积回报的期望

- 明确了目标函数，策略优化就是一个最优化问题

$$\theta_* = \underset{\theta}{arg\max} J(\theta)$$

- Gradient-based 方法
  - 梯度下降
  - 共轭梯度
  - …

- 不用梯度的搜索方法
  - 爬山算法
  - 遗传算法
  - …

基于有限差分的梯度近似

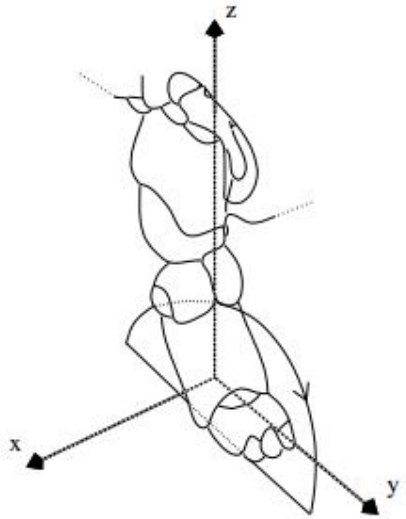
- For each dimension $k \in [1, n]$
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ϵ in kth dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where u_k is unit vector with 1 in kth component, 0 elsewhere

- Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

Example： Training AIBO to Walk



- Goal: learn a fast AIBO walk (useful for Robocup)
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time

策略梯度的推导

- 用τ表示一组状态-行为序列：$s_0, a_0, \dots, s_T, u_T$
- 用$R(\tau)$表示轨迹τ的回报，$\mathrm{P}(\tau;\theta)$表示此轨迹出现的概率
- **目标函数：$\mathbf{J}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\tau}} \boldsymbol{P}(\boldsymbol{\tau};\boldsymbol{\theta})\boldsymbol{R}(\boldsymbol{\tau})$**

- 对目标函数求导：
  - $\nabla J(\theta) = \nabla \sum_{\tau} P(\tau;\theta)R(\tau)$
  - $= \sum_{\tau} \nabla P(\tau;\theta)R(\tau)$
  - $= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla P(\tau;\theta)R(\tau)$
  - $= \sum_{\tau} P(\tau;\theta) \frac{\nabla P(\tau;\theta)}{P(\tau;\theta)} R(\tau)$
  - $= \sum_{\tau} P(\tau;\theta) \nabla lnP(\tau;\theta)\, R(\tau)$

策略梯度的推导

- $\nabla J(\theta) = \sum_\tau P(\tau;\theta)\nabla lnP(\tau;\theta)\ R(\tau)$
- 表明：策略梯度变成求$\nabla lnP(\tau;\theta)\ R(\tau)$的**期望**

- 在当前策略下通过采样m条轨迹来逼近策略梯度
  - $\nabla J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \nabla \ln P(\tau_i;\theta) R(\tau)$

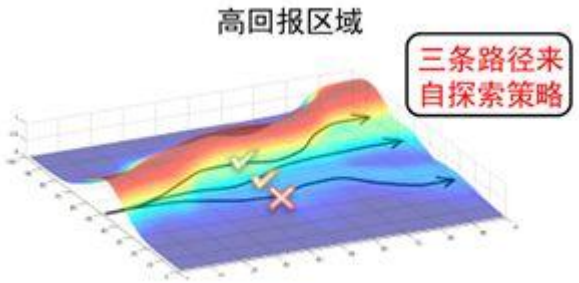
策略梯度的直观理解

- $\nabla J(\theta) \approx \frac{1}{m}\sum_t \nabla \ln P(\tau;\theta) R(\tau)$

- 第一项$\nabla \ln P(\tau;\theta)$是轨迹$\tau$的概率随着参数$\theta$变化最陡峭的方向。参数在该方向更新时，若沿着正方向，则该轨迹τ的概率会变大

- 第二项$R(\tau)$控制了参数更新的方向和步长。为正则参数更新后该轨迹的概率越大，为负则降低概率，抑制该轨迹的发生

- 因此，**策略梯度会增加高回报路径的出现概率**



策略梯度的推导

- $\nabla J(\theta) = \sum_\tau P(\tau;\theta)\nabla lnP(\tau;\theta)\, R(\tau)$

- 接下来解决似然率的梯度问题，即如何求$\nabla lnP(\tau;\theta)$
  - $\mathrm{P}(\tau;\theta) = \prod_{t=0}^{T} P(s_{t+1}|s_t,a_t) * \pi_\theta(a_t|s_t)$
  - 其中$P(s_{t+1}|s_t,a_t)$表示环境动态，与策略无关

  - $\nabla lnP(\tau;\theta) = \nabla \ln\left(\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t) * \pi_\theta(a_t|s_t)\right)$
  - $= \nabla[\sum_t \ln P(s_{t+1}|s_t,a_t) + \sum_t \ln \pi_\theta(a_t|s_t)]$
  - $= \sum_t \nabla \ln \pi_\theta(a_t|s_t)$
- 最终：
- $\nabla J(\theta) = \sum_\tau P(\tau;\theta)(\sum_t \nabla \ln \pi_\theta(a_t|s_t) R(\tau))$

Monte-Carlo Policy Gradient (REINFORCE)

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
        end for
    end for
    return θ
end function

使用基准降低方差

- 我们常在回报中引入基准b来减小方差
  - $\nabla J(\theta) \approx \frac{1}{m}\sum \nabla \ln P(\tau;\theta)(R(\tau)-b)$

- 引入基准b后不影响策略梯度的方向
- 对b的要求：与动作a无关

$$\nabla J(\boldsymbol{\theta}) = \sum_s \mu_\pi(s) \sum_a \Big(q_\pi(s,a) - b(s)\Big) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta})$$

$$\sum_a b(s) \nabla_{\boldsymbol{\theta}} \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} \sum_a \pi(a|s,\boldsymbol{\theta}) = b(s) \nabla_{\boldsymbol{\theta}} 1 = 0$$

如何选择Baseline

- 一种自然的选择是$b(s_t) = V^{\pi_\theta}(s_t)$
- Self-critical
- Moving average of R
- 。。。开放性学术问题

Actor-Critic 算法

- REINFORCE不足：
  - 采样完整轨迹，方差大，效率低
- Actor-Critic 算法：
  - 结合策略梯度和时序差分学习，从状态s开始的总回报可以通过当前动作的即刻回报和下一个状态的值函数来近似
  - Actor(演员)：策略函数$\pi_\theta(s,a)$
  - Critic(评论员)：值函数$V_\phi(s)$
  - 可以进行单步更新，不需要等到回合结束

Actor-Critic 算法

- 策略函数和值函数在训练过程中同时学习
- 从t时刻开始的总回报用下式近似：
  - $\hat{G}(\tau_{t:T}) = r_{t+1} + \gamma V_{\phi}(s_{t+1})$
- 两个学习目标：
  - $\min_{\phi} \left(\hat{G}(\tau_{t:T}) - V_{\phi}(s_t)\right)^2$ 更新ϕ使得值函数接近估计的真实回报
  - $\theta \leftarrow \theta + \alpha\gamma^t \left(\hat{G}(\tau_{t:T}) - V_{\phi}(s_t)\right) \nabla \ln \pi_{\theta}(a_t|s_t)$将值函数作为baseline，更新策略参数

Actor-Critic 算法

算法 15.8: actor-critic 算法

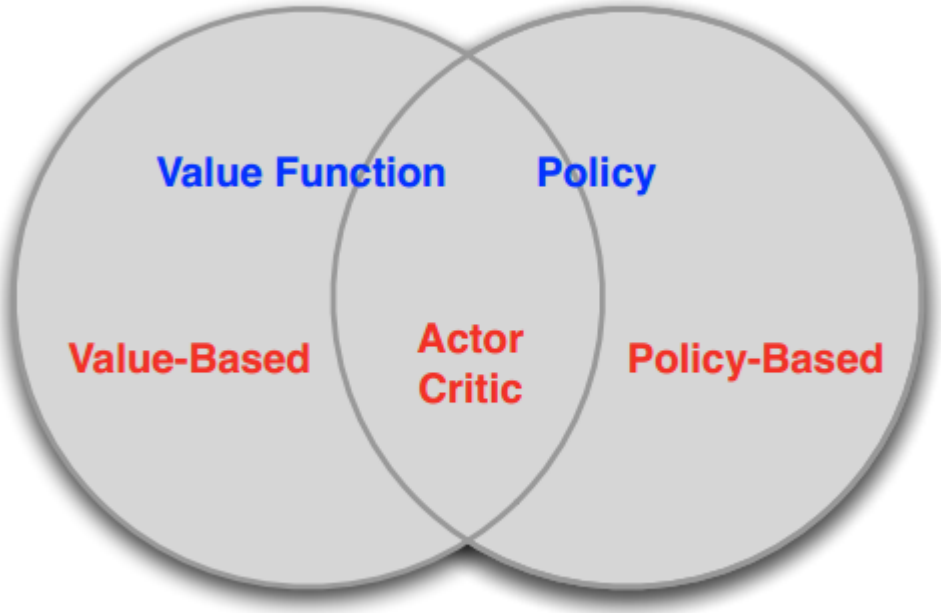
输入: 状态空间 $\mathcal{S}$，动作空间 $\mathcal{A}$;
可微分的策略函数 $\pi_\theta(a|s)$;
可微分的状态值函数 $V_\phi(s)$;
折扣率 γ，学习率 $\alpha > 0, \beta > 0$;

随机初始化参数 θ, ϕ;
repeat
  初始化起始状态 s;
  $\lambda = 1$;
  repeat
    在状态 s，选择动作 $a = \pi_\theta(a|s)$;
    执行动作 a，得到即时奖励 r 和新状态 s';
    $\delta \leftarrow r + \gamma V_\phi(s') - V_\phi(s)$;
    $\phi \leftarrow \phi + \beta\delta \frac{\partial}{\partial \phi} V_\phi(s)$;
    $\theta \leftarrow \theta + \alpha\lambda\delta \frac{\partial}{\partial \theta} \log \pi_\theta(a|s)$;
    $\lambda \leftarrow \gamma\lambda$;
    $s \leftarrow s'$;
  until s 为终止状态;
until θ 收敛;

输出: 策略 π_θ

Value-based and policy-based

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



Value-Based VS. Policy-Based

- Value-based的典型算法是DQN，policy-based是policy gradient
- 更新频率不同：value-based每个action都可以更新，policy-based要等到回合结束。
- 收敛性不同：policy更容易收敛，value-based 方法值函数中的一些小小改变可能会使得策略发生较大改变，从而收敛性较差。Policy容易收敛到局部最优
- 策略的随机性不同：policy能学习到随机性策略，Value-Based 需要用贪婪的方法得到策略，是近确定性策略