Model-free RL

- model-free的特点是不知道状态转移矩阵P，也就是没有办法推出哪个state做出action后会转移到哪个state，这个特点也更符合现实的情况，所以model-free的研究是现在的大热
- model-free的一个数据被称为一个episode，包括 $\{s_1, a_1, r_1, s_2, a_2, r_2 \ldots\ldots, s_t, a_t, r_t\}$ 或者有一些情况下，只存在最终奖励 r_t （比如游戏的输赢）
- 注意Model-free的情境下只解出state-value是不够的，必须要解出action-value才能算解出
- model-free的思路：先传统的model-free的Prediction、Control，再讲现代的model-free的Prediction和Control做出的改变

一、传统Model-free的解法

1. Prediction

注意一个事情，Agent走到 s_{t+1} 的时候更新 s_t 的公式

（1）Monte Carlo policy evaluation（MC）

- MC是一类方法，主要思想就是通过多次取样实验算平均值来近似表达真实值
- $v^{\pi}(s_t) = \frac{G_t}{N}$ 其中 G_t 是 N 次实验的总Return值
- Incremental MC则是用迭代的方式进行估计，可以用于无止境的exploration：
  对于第 n 次实验，do：

$$n+=1$$

$$v^{\pi}(s_t)+=\frac{G_t - v^{\pi}(s_t)}{n}$$

  理解：每一次相当于是都对新的return做一次平均再加到总的return上去
  其中 $\frac{1}{n}$ 也时常被一个参数 α 取代，这样的话通过设置 α 的值就可以达到“老的episode”被遗忘的效果

（2）Temporal Difference learning（TD）

- TD方法就像SR一样，是n-look-ahead类型，从TD(0)开始一直到TD(n)
- 以TD(0)的公式为例：

$$v^{\pi}(s_t)+=\alpha * (R_{t+1} + \gamma * v^{\pi}(s_{t+1}) - v^{\pi}(s_t))$$

  它的公式实际上就是只加上了向前走一个state的MC，TD(n)方法就与MC方法没有任何区别了
- TD方法的好处在于它可以在episode还没结束时就可以对策略 π 进行评估

- **TD target**是指 $R_{t+1} + \gamma * v^{\pi}(s_t + 1)$ 的部分
  TD error是指 $R_{t+1} + \gamma * v^{\pi}(s_{t+1}) - v^{\pi}(s_t)$ 的部分

2. Control

- 传统Model-free的Control主要为value-based的方法，其中又可以分为on-policy和off-policy两类
- on-policy的意思是采样的策略与评估的策略是同一个策略，off-policy则是两个不同的策略
  off-policy有如下优势：
  可以在exploration下学习到最好的算法，而不是exploitation
  可以reuse老的策略
  exploration的意思是探索新的方法，exploitation的意思是只选择被证明是好的方法
  比如你现在要吃饭，你是去尝试一家新的餐厅还是吃你最喜欢吃的老餐厅
  对于正在训练中的Agent来说，我们希望他是能够尝试新餐厅的，不好吃的话就不去了就可以，而不是一直选择老餐厅
- 只讲两个代表算法：SARSA和Q-learning，SARSA是on-policy的，Q-learning是off-policy的

（1）SARSA

- init $Q(s, a)$
- 对于每一个episode，do：

$$Q(s_t, a_t) += \alpha * (R_{t+1} + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

  注意，SARSA在根据Q函数选择执行动作时，采取的是 ϵ-greedy 的策略：
  即有 ϵ 的概率选择随机的动作，$1 - \epsilon$ 的概率选择最佳的动作（ ϵ 这个超参数设置的越小，越倾向于选择最佳的动作）
- 做完所有episode后，从 $Q^*(s, a)$ 中提取到最佳策略 π^*

(2) Q-learning

- init $Q(s, a)$
- 对于每一个episode，do：

$$Q(s_t, a_t) += \alpha * (R_{t+1} + \gamma * max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

  理解：每一步都向着最优的方向迭代，收敛速度快
- 做完所有episode后，从 $Q^*(s, a)$ 中提取到最佳策略 π^*

二、现代Model-free的解法

因为现代数据过于庞大、Q函数本身是一个look-up table，在内存里已经存不下了，需要新的方法来估计value；解决的方法就是function approximation，用机器学习的方法来解RL

- 建模：
  state向量 $X(s)=(f_1,f_2,......,f_n)$ ：把一个state拆分成由若干特征表示的向量
  $q^\pi(s,a,w)$ 、$v^\pi(s,w)$、$\pi(a|s)$ ：两个value function都由参数 w 进行估计，policy function不变

1. Value-based

（1）Linear

- 假设q函数与状态s符合线性关系 $q^\pi(s,a,w)=W^TX$
- 构建损失函数 $J(W)=E[(q_{best}(s_{t+1},a_{t+1},W)-W^TX)^2]$
  根据两种不同的Prediction，MC和TD，可以有两种形式的 q_{best} ，分别为 G_t （MC）、$R+\gamma*q(s_{t+1},at+1,W)$ （TD），其使用场景与MC和TD完全一致
- 梯度下降不断迭代更新W，$W=W-\alpha*\nabla_w J(W)$
  注意MC的话是一个episode一更新，TD的话是每走一个state就一更新
  本质上这样去更新梯度是符合SGD的
- 线性的方法是一定会收敛到optimal

（2）Neural Network

- 从过程上看，跟Linear的方法的唯一的区别就是用神经网络去拟合 q^π
- 但是，因为加入了神经网络，泛化能力变强的同时会引入一些神经网络优化的困难
- 问题之一：样本间不具有独立性，这会导致神经网络参数剧烈震荡，样本间最好是独立同分布的
- 问题之二：q_{best} 也是根据同样的参数W算出来的，也就是目标值也会一起更新，导致有时候W追不上目标值的更新速度

DQN

- 使用卷积网络提取游戏的帧作为样本
- 使用Replay Buffer，储存每一次的s、a、r作为一个样本，每次从Replay Buffer中提取若干样本进行训练
- 目标值的梯度固定为 W^- ，等待梯度更新若干步后，再进行同步

2. Policy-based

- policy-based相比于value-based会有一些好处
- 比如policy本身可以是stochastic的，而value-based中提取出来的policy是deterministic的
- policy-based可以保证收敛，并且更加effective当action很多时（高维）
- 但是他的问题就是：极有可能陷入局部最优点、并且估计policy时会有较高的variance

（1）建模

- 我们建模最好的策略 π^* 是最好的参数 θ^* 的函数
- 构建目标函数 $J(\theta) = E_{\tau \sim \pi}[G_t]$ ，其中： $\tau \sim \pi$ 的意思是根据 π 这个策略采样出来的episode τ ，我们的目标函数含义则是：在 π 这个策略下采样出来的所有轨迹的奖励的均值，显然我们希望目标函数越大越好
- 所以我们采用梯度上升的方法来更新参数，即 $\theta = \theta + \alpha * \nabla_\theta J(\theta)$

（2）计算梯度

- 有些可导的目标函数可以直接操弄梯度来更新，有些则不行
- 对于可以计算梯度的，公式为：

$$\nabla_\theta J(\theta) = E_\tau[R(\tau) \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t^i|s_t^i)]$$

理解： $\nabla_\theta log\pi_\theta(a_t^i|s_t^i)$ 表示方向，如果我想增加动作a发生的概率，我应该往这个梯度方向增加；而R则表示增加多少，高R的动作必然增加的多

- 推导：

(2) 推导：

① multi-step 问题转化：

Denote a state-action trajectory from one episode as $\tau = (s_0, a_0, r_1, \ldots s_{T-1}, a_{T-1}, r_T, s_T) \sim (\pi_\theta, P(s_{t+1}|s_t, a_t))$

Denote $R(\tau) = \sum_{t=0}^{T-1} R(s_t, a_t)$ as the sum of rewards over a trajectory τ

The policy objective is

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} R(s_t, a_t)\right] = \sum_\tau P(\tau;\theta)R(\tau)$$

where $P(\tau;\theta) = \mu(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)$ denotes the probability over trajectories when executing the policy π_θ

Then our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

---

② 导数公式

极大似然：估计哪个参数能解释数据，但如果分布假设的不对，估计也是歪的 就说白了

衡量一个模型(θ)是否适合观测到的数据

Assume policy π_θ is differentiable whenever it is no-zero and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$

Likelihood ratios exploit the following tricks

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a)\frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a)\nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

$(\ln x)' = \frac{1}{x}$

The score function is $\nabla_\theta \log \pi_\theta(s, a)$

$$\begin{aligned}\therefore \quad \nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\ &= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\ &= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)}\nabla_\theta P(\tau;\theta)R(\tau) \\ &= \sum_\tau P(\tau;\theta)R(\tau)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\ &= \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)\end{aligned}$$

③ MC estimate

Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)\nabla_\theta \log P(\tau_i;\theta)$$

④ 进一步拆解

Decompose $\nabla_\theta \log P(\tau;\theta)$

初始state为s_0的prob

s_t下采取a_t后转移至s_{t+1}的prob

s_t下采取a_t的prob

$$\begin{aligned}\nabla_\theta \log P(\tau;\theta) &= \nabla_\theta \log\left[\mu(s_0)\prod_{t=0}^{T-1}\pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)\right] \\ &= \nabla_\theta\left[\log \mu(s_0) + \sum_{t=0}^{T-1}\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)\right] \\ &= \sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t|s_t)\end{aligned}$$

（3）不可导的算法

(1) CEM (cross-entropy method)

Algorithm 1 CEM for black-box function optimization

for iter $i = 1$ to N **do**
  $\mathcal{C} = \{\}$
  for parameter set $e = 1$ to N **do**
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$ → 从$P_{\mu^{(i)}}$这个分布里采样一串θ
    execute roll-outs under $\theta^{(e)}$ to evaluate $J(\theta^{(e)})$
    store $(\theta^e, J(\theta^{(e)}))$ in $\mathcal{C}$
  end for
  跟交叉熵一样，信息量的多少
  $\mu^{(i+1)} = \arg\max_\mu \sum_{k \in \hat{\mathcal{C}}} \log P_\mu(\theta^{(k)})$
  where $\hat{\mathcal{C}}$ are the top 10% of $\mathcal{C}$ ranked by $J(\theta^{(e)})$
end for

在$\mu^{(i)}$分布下J(θ)最有多少

更新$\mu^{(i)}$，然后再看一下J(θ)能到多少

(2) FD (finite differient)

1. To evaluate policy gradient of $\pi_\theta(s, a)$
2. For each dimension $k \in [1, n]$
   1. estimate kth partial derivative of objective function by perturbing (扰乱) θ by a small amount ϵ in kth dimension

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

根据导数定义来估计那一维的偏导数

where u_k is unit vector with 1 in kth component, 0 else where
3. uses n evaluations to compute policy graident in total n dimensions
4. though noisy and inefficient, but works for arbitrary policies, even if policy is not differentiable.

（4）解决方差过大

方差会大的原因是，你还是通过采样的方式获取的数据，每一个episode都有运气成分

- Temporal causality
  公式中从t=0一直到T-1时刻都乘上的是最终的奖励R，但其实R只与之前的动作有关，与之后的动作无关。所以我们将公式改写为：$\nabla_\theta J(\theta) = E_\tau[\sum_{t=0}^{T-1} G_t * \nabla_\theta log\pi_\theta(a_t^i|s_t^i)]$
- Baseline
  方差大的话我就减去一个常数使方差变小即可，这个常数要满足与a无关只与s有关（要保证始终是无偏的），最优解可以计算但是太复杂，一般我们就用 $v^\pi(s)$ 来代替
  所以公式改写为：$\nabla_\theta J(\theta) = E_\tau[(R(\tau) - v^\pi(s)) \sum_{t=0}^{T-1} \nabla_\theta log\pi_\theta(a_t^i|s_t^i)]$

3. Actor-Critic

- 动机：纯policy-based的方法采样效率低、震荡大；纯value-based的方法则没有stochastic的策略，不够灵活
- actor-critic方法由两部分组成，两部分是两个网络（两个参数）
  actor只负责根据policy做动作，优化目标仍然是刚才的 $J(\theta)$ ，只不过步幅从奖励变成了由critic“评价”的优势函数 $A(s_t, a_t)$
  critic只负责评判policy的动作，优化目标是TD的损失函数 $J(\phi)$

- actor：

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} A(s_t, a_t) \nabla_\theta log\pi_\theta(a_t^i|s_t^i)$$

  其中 $A(s_t, a_t) = R_{t+1} + \gamma * v^\phi(s_{t+1}) - v^\phi(s_t)$
- critic：

$$L(\phi) = (R_{t+1} + \gamma * v^\phi(s_{t+1}) - v^\phi(s_t))^2$$

- 先更新critic，再更新actor