PPO

传统强化学习都是在维护表格，也就是要列举出在所有状态下应该采取的行动。这面临两个问题。一是当状态太多时，需要大量样本，并且存储表格状态也需要大量空间；二是只适合状态连续的场景，对于温度、GDP等指标则需要离散化成不同的状态（单个指标离散化会造成一定的精度损失，当多个指标需要离散化时会加重这种损失）。

更好的方法是构造方程来预测给定状态下的v和q，也就是构造 $\hat{v}(s,w) \approx v_\pi(s)$ ，训练目标就是 $min\ l(w) = \sum_s \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2$ ， $\mu(s)$ 表示被访问的概率。如果采用梯度下降，则更新公式为：

$$w = w - \eta \nabla l(w) = w + \eta \nabla \sum_s \mu(s) \cdot 2[v_\pi(s) - \hat{v}(s,w)] \nabla \hat{v}(s,w)$$

又已知 $v_\pi(s) = E[G_t|S_t = s] = E[R_{t+1} + \gamma G_{t+1}|S_t = s] = R_{t+1} + \gamma v_\pi(S_{t+1})$ ，只需要知道下一个状态就可以对当前状态进行的w进行更新.

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S is terminal

PPO前身Policy gradient

之前介绍对于continuing task，有不考虑衰减的奖励 $G_t = \sum_{k=t+1}^{n} R_k$ ，考虑衰减的奖励 $G_t = \sum_{k=t+1}^{n} \gamma^{k-t} R_k$ 。此外，还有differential return，其中 $r(\pi)$ 是基于当前所有奖励算出的平均值，这样计算的好处是能减少方差。

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{r,s'} p(s', r|s, a)\Big[r - r(\pi) + v_\pi(s')\Big],$$

$$q_\pi(s, a) = \sum_{r,s'} p(s', r|s, a)\Big[r - r(\pi) + \sum_{a'} \pi(a'|s') q_\pi(s', a')\Big]$$

利用differential return，可以改写基于TD和梯度下降的Sarsa，$\hat{q}(S, A, w)$ 是一个回归模型，拟合真实的q值。

Differential semi-gradient Sarsa for estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$
Algorithm parameters: step sizes $\alpha, \beta > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)
Initialize average reward estimate $\bar{R} \in \mathbb{R}$ arbitrarily (e.g., $\bar{R} = 0$)

Initialize state S, and action A
Loop for each step:
    Take action A, observe R, S'
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\delta \leftarrow R - \bar{R} + \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
    $\bar{R} \leftarrow \bar{R} + \beta\delta$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

之前介绍的policy iteration是首先计算state-value值，再采用贪心的策略更新策略。现在想直接对策略进行优化，表示为 $\pi(a|s,\theta)$，其满足 $\pi(a|s,\theta) \geq 0, \sum_{a \in \mathcal{A}} \pi(a|s,\theta) = 1$ 。

首先想到可以用softmax函数，其中h函数可以用深度学习的回归模型来实现

$$\pi(a|s,\theta) = \frac{e^{h(s,a,\theta)}}{\sum_{b \in \mathcal{A}} e^{h(s,b,\theta)}}$$

训练目标：

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

其中 $d^\pi(s)$ 表示状态s被访问的概率，目标是最大化 $J(\theta)$，也就是希望访问的都是价值高的state，这里的v就是state-value，$v_\pi(S) = E[G_t|S_t = s]$。需要计算的也就是 $\nabla_\theta J(\theta)$

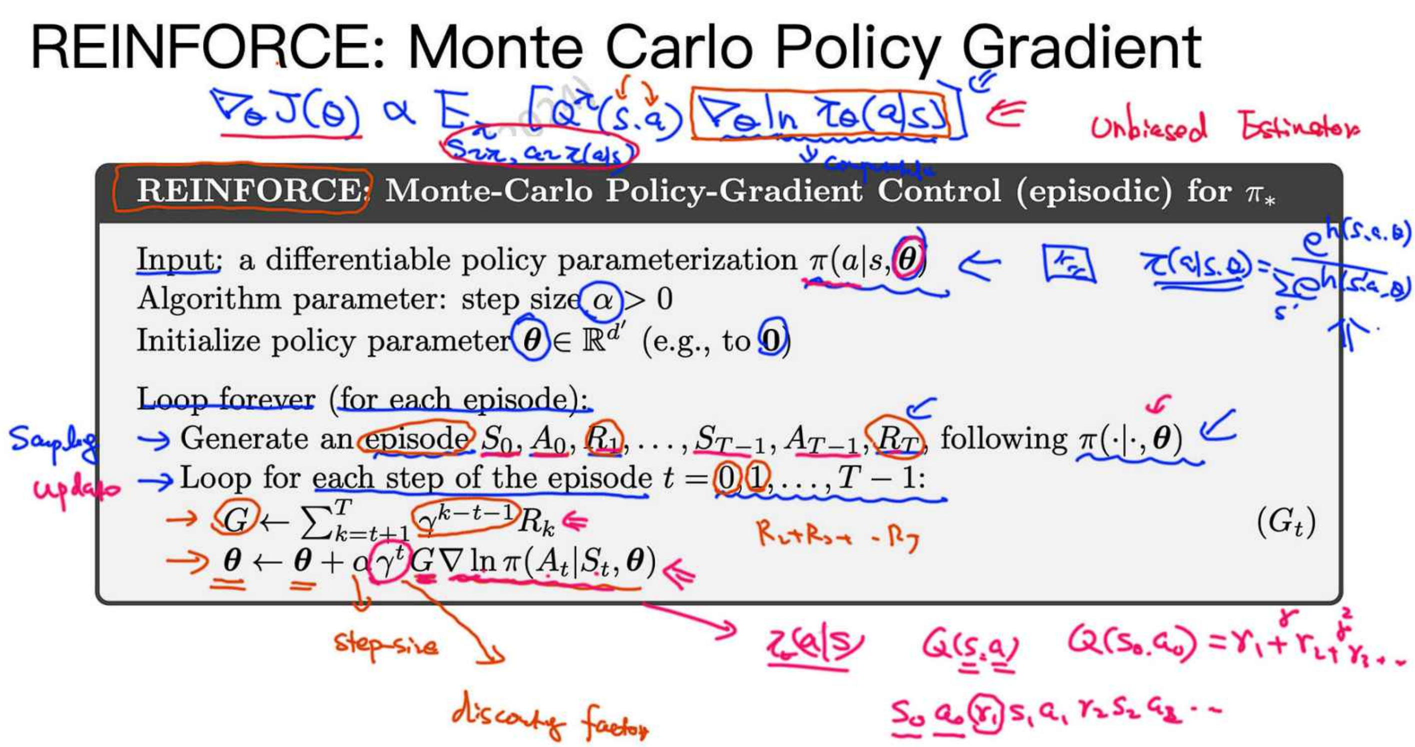
$$\theta = \theta + \eta\nabla_\theta J(\theta)$$

依照policy gradient theorem（这个证明比较复杂，这里不写了）：

$$\nabla_\theta J(\theta) = \nabla_\theta[\sum_{s\in S} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) q^\pi(s,a)] \propto \sum_{s\in S} d^\pi(s) \sum_{a\in\mathcal{A}} q^\pi(s,a) \nabla_\theta \pi_\theta(a|s) =$$

$$\sum_{s\in S} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} = E_{s\in S, a\in\pi_\theta(a|s)} q^\pi(s,a) \nabla_\theta ln\pi_\theta(a|s)$$

下图中用G近似替代了q，都是奖励函数。



面临问题:episode怎么生成，可能很长，episode生成和策略更新不可同时进行。

Off-policy policy gradient

之前介绍的是on-policy的，即采样和更新都是对同一个θ 进行，因此只能能同时做一件事。off-policy值利用参考的相似的θ 生成的episode更新目标的θ 。

采用Off-policy policy gradient，对应的梯度加一个距离系数，变成了：

$$\nabla_\theta J(\theta) = E_\beta[\frac{\pi_\theta(a|s)}{\beta(a|s)} q^\pi(s,a) \nabla_\theta ln(a|s)]$$

在PPO中，用π^{t-1} 作为β 。此外用Advantage function $A^\pi(s,a) = q^\pi(s,a) - v^\pi(s)$ 代替q，因为$v(s) = \sum_{a\in A} \pi(a|s) q(s,a)$,相当于减去平均值，使得方差更小更平滑，就得到了TRPO

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}},a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right]$$ maxmize

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s))] \leq \delta$$ s. t.

面临问题：存在限制条件，要求参考策略和目标策略距离接近，难以优化。PPO消除了限制条件

PPO

为了消除限制条件，首先定义 $r(\theta)=\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$ ，对 $r(\theta)$ 进行clip操作。实验证明，只对上区间做clip效果最好。

$$r(\theta)=1+\epsilon, if\ r(\theta)>1+\epsilon$$

$$J^{\text{CLIP}}(\theta)=\mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a),\text{clip}(r(\theta),1-\epsilon,1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a))]$$

$$J^{\text{CLIP'}}(\theta)=\mathbb{E}[J^{\text{CLIP}}(\theta)-c_1(V_\theta(s)-V_{\text{target}})^2+c_2H(s,\pi_\theta(.))]$$

红色项是因为采用了Advantage function（$A(s,a)=q(s,a)-v(s)$），需要一个模型 V_θ 来求出针对当前状态下的状态值，这里与policy采用同一个网络参数,因此放在一起优化，红色项最小化预测 V_θ 与真实 V_{target} 之间的距离；第三项为了exploration，采用熵的形式，提升模型可探索域的大小。

整体的训练流程就是基于 θ_{old} 生成一个episode，利用episode计算目标函数中的值，更新 θ .伪代码中 ϕ_0 一般与 θ_0 用同一个模型。

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for