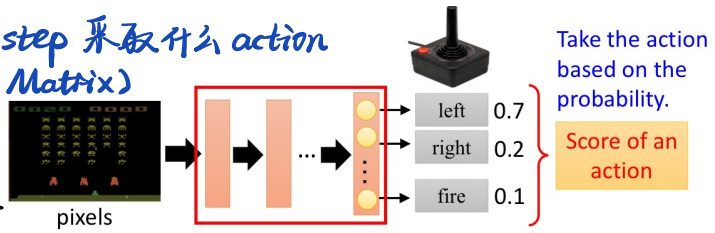
Policy Gradient

Basic Components Actor, Environment, Reward Function 只有 Actor 可以改变，其他两个固定不变

Policy of Actor

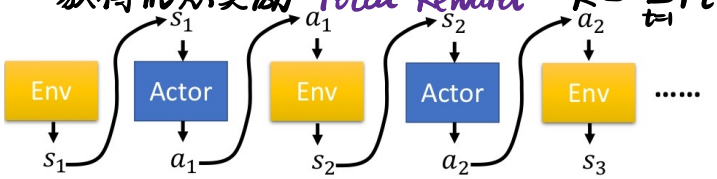
Policy π → A network with parameter θ 决定每个 time step 采取什么 action

- Input: Observation (Represented as a Vector or a Matrix)
- Output: 每个 neuron 代表一个 action (用概率表示)

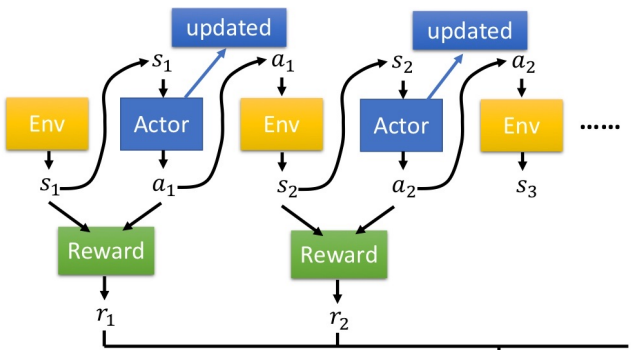


Actor, Environment, Reward

- 每个 time step (St), 采取一个 action at 获得 reward rt
- 一局游戏结束 → 1个 Episode 回合
  获得的总奖励 Total Reward $R = \sum_{t=1}^{T} r_t$



Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots, s_T, a_T\}$



Expected Reward

$$\bar{R}_\theta = \sum_\tau R(\tau) p_\theta(\tau) = E_{\tau \sim p_\theta(\tau)}[R(\tau)]$$

$$R(\tau) = \sum_{t=1}^{T} r_t$$

Policy → 希望最大化 Total Reward R

- Trajectory $\tau = \{s_1, a_1, s_2, a_2, \cdots s_T, a_T\}$
  → 获得这个 Trajectory 的概率 $P_\theta(\tau)$
  → $p_\theta(\tau) = p(s_1) p_\theta(a_1|s_1) p(s_2|s_1, a_1) p_\theta(a_2|s_2) p(s_3|s_2, a_2)$
  $= p(s_1) \prod_{t=1}^{T} p_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$
  同样的 state 采取相同 action 结果可能不相同 由 Environment 决定
- $R(\tau) = \sum_{t=1}^{T} r_t$ 调整 actor 的参数 θ，使 R 越大越好
  R 不是 scalar 而是 random variable Env 和 actor 有随机性
  $\bar{R}_\theta = \sum_\tau R(\tau) p_\theta(\tau) = E_{\tau \sim p_\theta(\tau)}[R(\tau)]$ R 的期望值

Policy Gradient $R(\tau)$ 不需要是可微的 Differentiable

$$\nabla \bar{R}_\theta = \sum_\tau R(\tau) \nabla p_\theta(\tau) = \sum_\tau R(\tau) p_\theta(\tau) \frac{\nabla p_\theta(\tau)}{p_\theta(\tau)}$$

$$= \sum_\tau R(\tau) p_\theta(\tau) \nabla \log p_\theta(\tau)$$

$$= E_{\tau \sim p_\theta(\tau)}[R(\tau) \nabla \log p_\theta(\tau)] \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n) \nabla \log p_\theta(\tau^n)$$

$$= \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n|s_t^n)$$

$\nabla f(x) = f(x) \nabla \log f(x)$

$p(s_{t+1}|s_t, a_t)$ 与 θ 无关 不需要作 Gradient

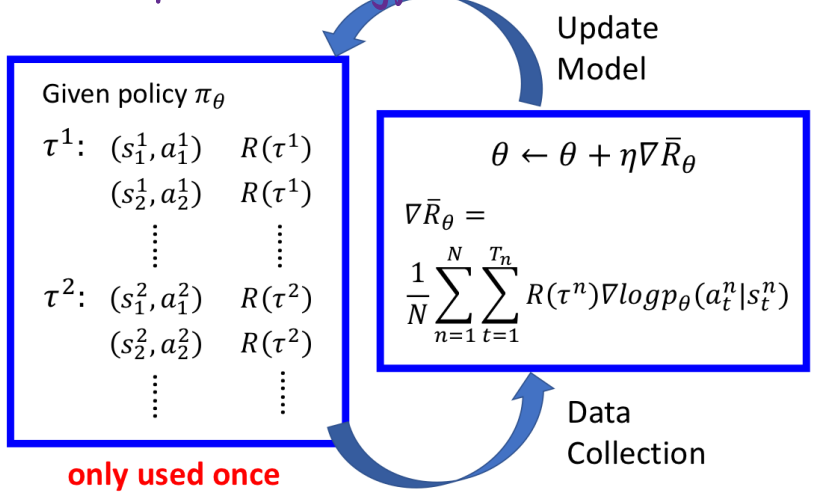
$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau) \nabla \log p_\theta(\tau)]$ $R>0$ → 提高概率

Policy Gradient 更新过程

- 用一个 actor (π_θ) 进行游戏
- 记录每一个 Episode 的 τ 只会使用一次
- N 个 Episode 以后，用这些 Data 来 Update
- 用更新后的 model 重复以上过程

$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

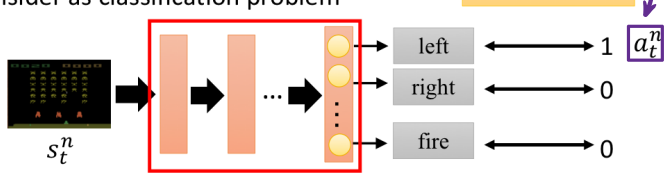
$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n|s_t^n)$$



Implementation

Ground Truth s_t^n a_t^n $R(\tau^n)$

Consider as classification problem



- Ground Truth：来自于 Sampling Process
  Sample 到在 s_t 下应当采取 a_t，以此作为 GT
- 分类问题的目标函数：最小化 Cross-Entropy
  最小化 Cross-Entropy ⇔ 最大化似然估计
- RL 相比普通 Classification 需要权重 $R(\tau^n)$
  → $R(\tau^n)$ 是 τ^n 这一整局游戏的 Total Reward
  → $R(\tau^n)$ 并非在 s_t 采取 a_t 获得的 reward

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p_\theta(a_t^n|s_t^n) \longrightarrow \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \nabla \log p_\theta(a_t^n|s_t^n)$$

TF, pyTorch ...

普通分类问题

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \log p_\theta(a_t^n|s_t^n) \longrightarrow \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla \log p_\theta(a_t^n|s_t^n)$$

RL 问题

Tip 1: Add a Baseline 加入 baseline b 使得 $R(\tau^n)$ 不总为正 → $R(\tau)$ 较小的 τ，其系数为负

$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$

It is possible that $R(\tau^n)$ is always positive.

$R(\tau^n)$ 可能总是 >0，理想状态：$R(\tau)$ 越大的 τ，概率增加越多
但有些未被 sample 到的 action 会因此概率下降
→ 加入 baseline b，$R(\tau) > b$ → 概率增加，否则概率下降

$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla \log p_\theta(a_t^n|s_t^n) \qquad b \approx E[R(\tau)]$$

Tip 2: Assign Suitable Credit → 将系数从 Total Reward → Cumulated Reward 并加入 γ

一个 Episode 的 $R(\tau^n) > 0$，不代表每个 action 都是有益的；反之，也不代表每个 action 都是负面的

→ 每个 $\log p_\theta(a_t^n|s_t^n)$ 的权重相同不恰当 之前的 action，state 已经发生，和本步无关

- 将 $R(\tau^n)$ 改为 $\sum_{t'=t}^{T_n} r_{t'}^n$，只考虑采取 a_t 后的 reward 之和（至该 Episode 结束之前）
- 加入 Discount Factor γ 随着时间推移，当前 a_t 对以后的 reward 影响会越来越小
- Advantage Function $A^\theta(s_t, a_t)$ 表示使用一个 model θ 和 Environment 作 interaction 得到 Cumulated Reward 的结果（r 的总和）

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}(R(\tau^n)-b)\nabla \log p_\theta(a_t^n|s_t^n)$$

$R(\tau^n)$ 由一个网络 Output 出来，可以是 state-dependent

$$\sum_{t'=t}^{T_n} r_{t'}^n \rightarrow \sum_{t'=t}^{T_n} \gamma^{t'-t} r_{t'}^n \quad \gamma < 1$$

→ 如果在 s_t 采取 a_t，相较于其他 action 会有多好 Estimated by Critic

- On-policy: The agent learned and the agent interacting with the environment is the same.
- Off-policy: The agent learned and the agent interacting with the environment is different.

用训练的 agent 和 Env 互动 VS 用其他 agent 和 Env 之间的 Data 训练

From on-policy to off-policy

Policy Gradient = On-policy

→ On-policy 缺点：Update 一次以后必须重新 Sample Training Data 的 Sample 效率太低，Update 一次就要重新 Sample

→ Off-policy：用 θ' 的 Data 训练 θ

θ' 是 fixed 的，只 Sample 一次可以 Update θ 多次重复使用 Data

$$\nabla \bar{R}_\theta = E_{\tau \sim p_\theta(\tau)}[R(\tau)\nabla \log p_\theta(\tau)]$$

- Use π_θ to collect data. When θ is updated, we have to sample training data again.
- Goal: Using the sample from $\pi_{\theta'}$ to train θ. θ' is fixed, so we can re-use the sample data.

Importance Sampling

无法从 p(x) 中 Sample，只能从 q(x) 中 Sample（q(x) 可以是任意 Distribution）

Importance Sampling

$$E_{x\sim p}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x^i)$$

x^i is sampled from $p(x)$

We only have x^i sampled from $q(x)$

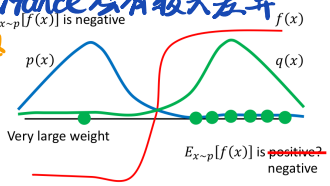
$$= \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

Importance weight

$$E_{x\sim p}[f(x)] = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

p(x) 和 q(x) 差异不能过大，否则 variance 会有较大差异

→ 需要 Sample 更多来弥补这个问题



$$\nabla \bar{R}_\theta = E_{\tau\sim p_{\theta'}(\tau)}\left[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)}R(\tau)\nabla \log p_\theta(\tau)\right]$$

Importance Weight 作修正

Gradient for Update

$$E_{(s_t,a_t)\sim \pi_\theta}[A^\theta(s_t,a_t)\nabla \log p_\theta(a_t^n|s_t^n)]$$

On-policy 的 Gradient $\nabla \bar{R}_\theta$

$$\nabla f(x) = f(x)\nabla \log f(x)$$

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim \pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

Off-policy 的 Objective Function

而是同样的 State Input 进 θ 和 θ'，Output 出的 action 分布的 JS 散度

Add Constraint 为了避免 pθ 和 pθ' 间的差异过大

TRPO (Trust Region Policy Optimization)

$$J_{TRPO}^{\theta'}(\theta) = E_{(s_t,a_t)\sim \pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

$$KL(\theta,\theta') < \delta$$

额外的 Constraint 难以计算

PPO (Proximal Policy Optimization) ↑

$$J_{PPO}^{\theta'}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta,\theta')$$

并非 θ 和 θ' 的 KL 散度

θ 不能和 θ' 差别过大

类似于正则项

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim \pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}A^{\theta'}(s_t,a_t)\right]$$

PPO Algorithm

- Initial policy parameters θ^0
- In each iteration
  - Using θ^k to interact with the environment to collect $\{s_t, a_t\}$ and compute advantage $A^{\theta^k}(s_t,a_t)$
  - Find θ optimizing $J_{PPO}(\theta)$

θ^k 是前一个 iteration 训练得到的 actor

θ 是当前 iteration 需要训练的 actor

$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta,\theta^k)$$

Update parameters several times

$$J^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)}\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}A^{\theta^k}(s_t,a_t)$$

每个 iteration 使用 θ^k 进行 Sample 以后，用 Data 对 θ 进行多次 Update（因为有了 Importance Sampling）

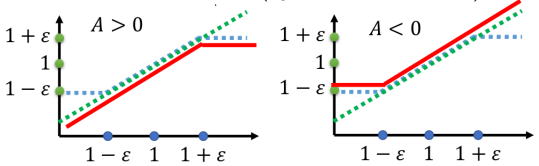
设置 KL 的最小值和最大值

- If $KL(\theta,\theta^k) > KL_{max}$, increase β
- If $KL(\theta,\theta^k) < KL_{min}$, decrase β

Adaptive KL Penalty

PPO2 Algorithm

$$J_{PPO2}^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}A^{\theta^k}(s_t,a_t),\ clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)},1-\varepsilon,1+\varepsilon\right)A^{\theta^k}(s_t,a_t)\right)$$



- 约束 θ^k 和 θ 不能相距太远
- 当 A>0，即 action 有正向效果，尽可能增大概率，但不能大过 1+ε

Policy Gradient → On-policy → Off-policy → Add constraint

Q-Learning Introduction

Critic = State Value Function $V^{\pi}(s)$ / State-Action Value Function $Q^{\pi}(s,a)$

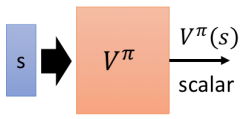
→ Value-based 方法：不学习 policy，而是 Critic

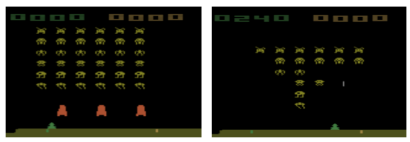
Critic

Critic 所 output 的 value 取决于被评估的 actor π（和 state s）

- 不直接决定 action
- 给定 actor π，评估 actor 有多好
- State Value Function $V^{\pi}(s)$

→ 使用 actor π 时，在 state s 以后获得的 cumulated reward 的期望





$V^{\pi}(s)$ is large　　$V^{\pi}(s)$ is smaller

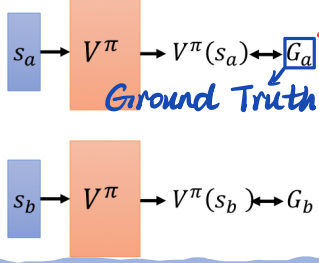
How to Estimate $V^{\pi}(s)$ — Monte-Carlo (MC) & Temporal-difference (TD) Approach

- Monte-Carlo (MC) based Approach

Critic 观看 π 玩游戏（玩到游戏结束）

After seeing s_a,
Until the end of the episode, the cumulated reward is G_a

After seeing s_b,
Until the end of the episode, the cumulated reward is G_b

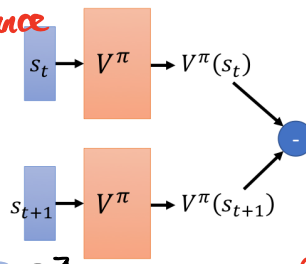


Larger variance
G_a 是很多步 reward 的总和

Ground Truth

- Temporal-Difference Approach

不需要等到整个 Episode 结束（应用更多）



$V^{\pi}(s_t) = V^{\pi}(s_{t+1}) + r_t$

$V^{\pi}(s_t) - V^{\pi}(s_{t+1}) \leftrightarrow r_t$

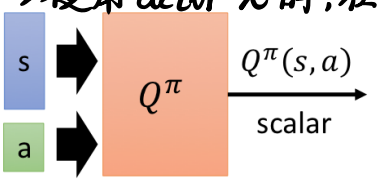
$V^{\pi}(s_t) \leftrightarrow r_t + V^{\pi}(s_{t+1})$

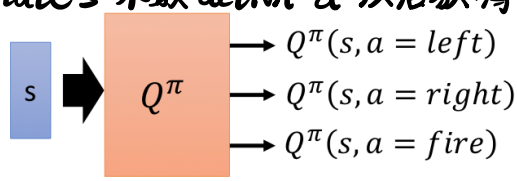
有能不准确

Smaller variance

Another Critic — State-Action Value Function $Q^{\pi}(s,a)$

→ 使用 actor π 时，在 state s 采取 action a 以后获得的 cumulated reward 的期望



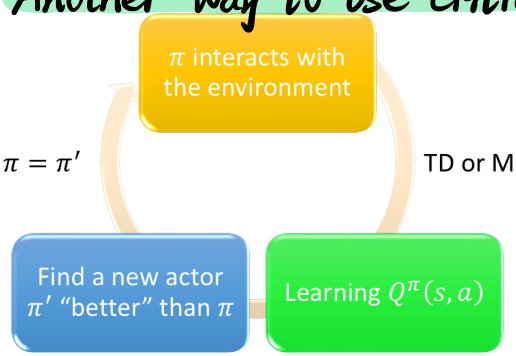


for discrete action only

→ 在 state s 强制采取 action a，后续用 π 继续完成该 Episode

→ 若 action 无法穷举（连续非离散），则只能使用左图的结构

Another Way to Use Critic: Q-Learning



- 先用初始的 π 和 Env 互动（TD 或 MC 方式评估都可以）
- 学习得到一个 State-Action Value Function $Q^{\pi}(s,a)$
- 用这个 $Q^{\pi}(s,a)$ 找到一个比 π 更好的 π'

→ Better 指的是：对于所有的 states，都有 $V^{\pi'}(s) \geq V^{\pi}(s)$

→ 如何找到 π'：$\pi'(s) = \arg\max_a Q^{\pi}(s,a)$

π' 不是个新的 actor，每次采取的 action 都是在当前 state s 下令 $Q^{\pi}(s,a)$ 具有最大值的 action（$a = \pi(s)$）不适用于连续 action

- 用 π' 代替 π 进行下一次循环 π' 完全取决于 Q，没有另外的参数

Tip 1: Target Network

在学习 Q Function 时，用到 TD 的概念，但在实际训练时，$Q^{\pi}(s_{t+1}, \pi(s_{t+1}))$ 是一直在变的

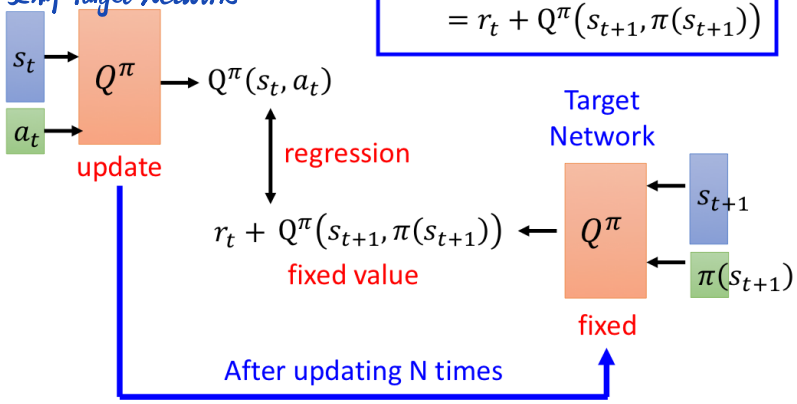
Regression 的目标在变化会造成训练困难

→ Target Network

Fix Target Network 的参数，更新 N 次参数以后，再用 Q 的参数更新 Target Network

$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

$$Q^{\pi}(s_t, a_t) = r_t + Q^{\pi}(s_{t+1}, \pi(s_{t+1}))$$



Tip 2: Exploration

- Policy Gradient 采取的 action 有随机性
- Q-learning 的 Policy 完全取决于 Q function

action 确定

$$a = \arg\max_a Q(s,a)$$

This is not a good way for data collection.

Epsilon Greedy

ε 随训练进行会不断 decay

$$a = \begin{cases} \arg\max_a Q(s,a), & \text{with probability } 1-\varepsilon \\ random, & \text{otherwise} \end{cases}$$

Boltzman Exploration

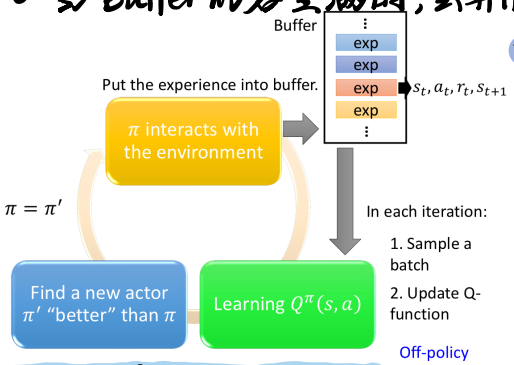
类似 Policy-Based

$$P(a|s) = \frac{exp\big(Q(s,a)\big)}{\sum_a exp\big(Q(s,a)\big)}$$

→ 根据 Q 的大小决定概率

Tip 3：Replay Buffer (s_t, a_t, r_t, s_{t+1})

- 将 policy 在每一步的 Experience 都放入 Buffer
- Buffer 中的 Experience 可能来自不同的 policy
- 当 Buffer 的容量满时，丢弃旧的 Experience



优势：
- 不需要使用完整的 trajectory，使用 Batch of Experience 的效率更高；
- Experience 来自不同的 policy，多样性更好，利于训练

Typical Q-Learning Algorithm

- Initialize Q-function Q, target Q-function $\hat{Q} = Q$
- In each episode
  - For each time step t
    - Given state s_t, take action a_t based on Q (epsilon greedy)
    - Obtain reward r_t, and reach new state s_{t+1}
    - Store (s_t, a_t, r_t, s_{t+1}) into buffer
    - Sample (s_i, a_i, r_i, s_{i+1}) from buffer (usually a batch)
    - Target $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$
    - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression) 每过 C 步更新一次
    - Every C steps reset $\hat{Q} = Q$ Target Network

Tips of Q-Learning

Double DQN Q值总是被高估

- Q value is usually over estimate

$$Q(s_t, a_t) \longleftrightarrow r_t + \max_a Q(s_{t+1}, a)$$

- Double DQN: two functions Q and Q' Target Network

$$Q(s_t, a_t) \longleftrightarrow r_t + Q'\left(s_{t+1}, \arg\max_a Q(s_{t+1}, a)\right)$$

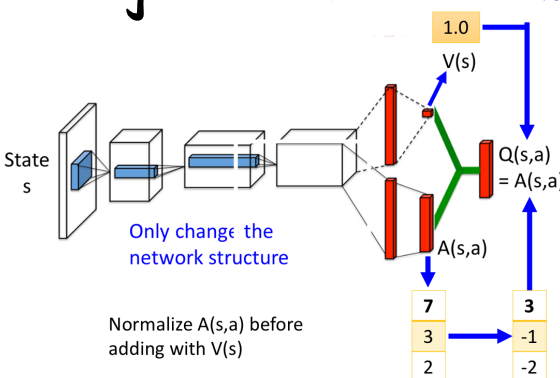
- Q值是被估测出的
- 倾向于选择Q值较大的 action

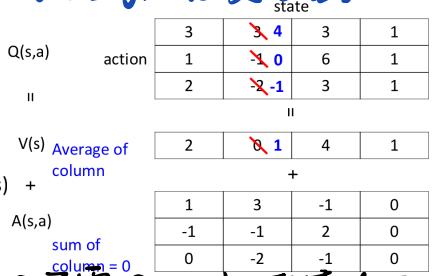
→ Double DQN

- 使用 Target 选择 Q值最大的 action
- 使用正在更新的 Q Network 计算 Q值

如果Q高估，则Q'会计算正确的Q值
如果Q'高估，则Q不会选择这个action

Dueling DQN 在某个State，改变 V(S) 可以改变 Q(S)

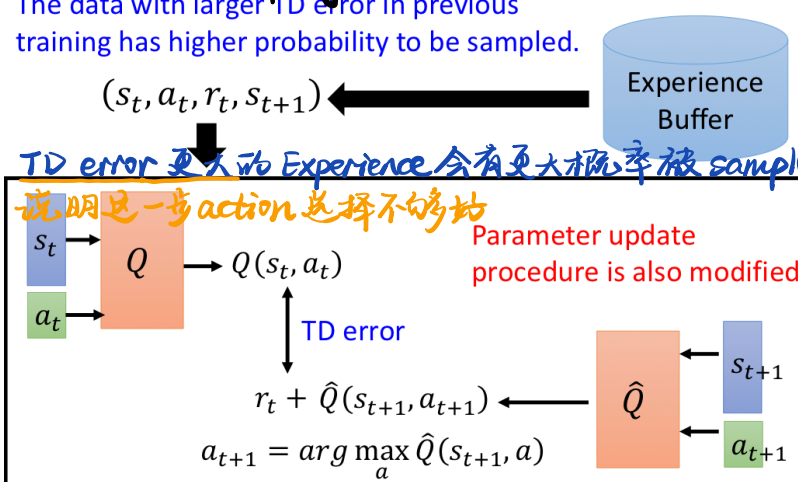




- 需要对 A(s,a) 的每个 column 加入 constraint
  → 倾向于 Update V(s)
- Normalize 包含在网络中
- V(s) 中的 value 相当于该列每个单元的平均值
- 不需 Sample 所有的 S-A 对 就可以更新 Q-Network 的参数
  → 更加 efficient 的方法
- 在加上 V(s) 前 Normalize A(s,a)

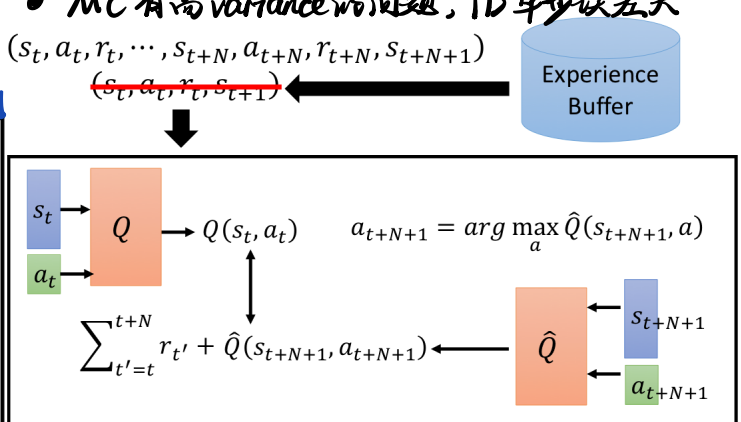
Prioritized Replay https://arxiv.org/abs/1511.05952?context=cs

The data with larger TD error in previous training has higher probability to be sampled.



Multi-Step 在MC和TD之间找到平衡

- 多个 step 版的 Experience
- MC 有高 variance 的问题，TD 单步误差大



Noisy Net

- Noise on Action (Epsilon Greedy)

$$a = \begin{cases} \arg\max_a Q(s,a), & \text{with probability } 1-\varepsilon \\ \text{random}, & \text{otherwise} \end{cases}$$

→ 对于相同的 state 也有可能采取不同的 action
→ 和真正的 policy 的方式不同 随机乱试

- Noise on Parameters 在每个 Episode 的开头加入 Noise

$$a = \arg\max_a \tilde{Q}(s,a)$$

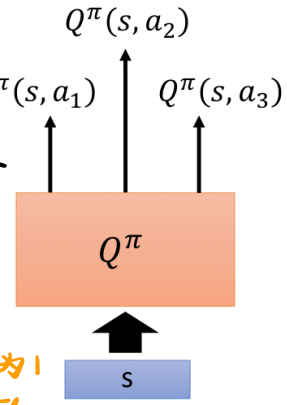
$Q(s,a) \longrightarrow \tilde{Q}(s,a)$ Add noise

每个 Episode 过程中 noise 不会变

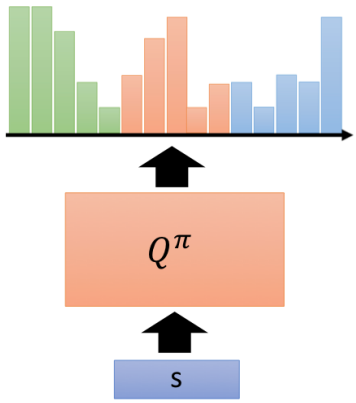
→ 对于相同的 State 采取相同的 action
→ State-dependent Exploration 有系统地试
→ 以一种被定的方式作 Exploration

Distributional Q-Function

- State-action Value Function $Q^\pi(s,a)$
  → 当使用actor π时，在state s 采取 action a 以后在当前 Episode 会获得的 cumulated reward 的期望
  → 返回的是一个value，可能会丢失信息，
  不同的 Distribution 可能具有相同的 $Q^\pi(s,a)$
- Distributional Q-Function
  → 返回的不是单个value，而是一个 Distribution
  - 每个action细分为多个 bin 每个bin都有一个概率，和为1
  - bin的均值为 Q-value 当 mean 相同时选择方差更小的
  → 不容易高估 Q value → 可以不用 Double-DQN
  划分 bin 相当于对 Q-value的取值做了剪裁

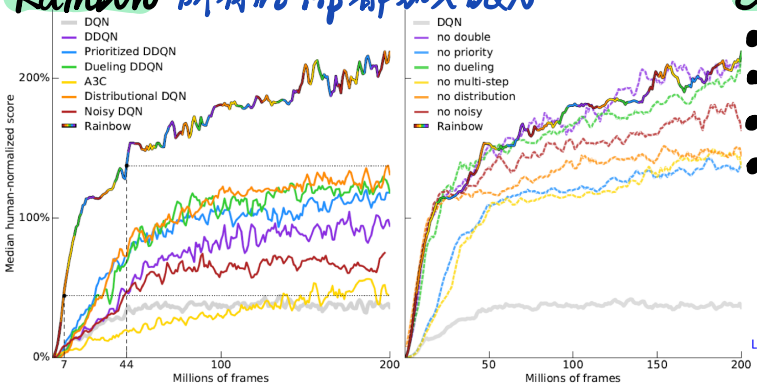


A network with 3 outputs



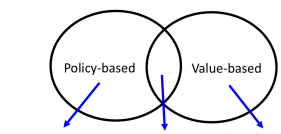
A network with 15 outputs
(each action has 5 bins)

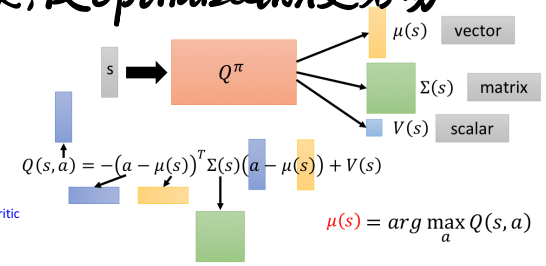
Rainbow 所有的Tip都加入DQN



Continuous Actions Action a 是一个连续 vector

- Sample一个action的集合，看哪个action的Q值最大
- 使用 Gradient Ascent 解决 Optimization 问题
- 设计一个 Network，使 Optimization 更容易
- 不使用 DQN





Actor-Critic

Review Policy Gradient & Q-Learning

- Policy Gradient

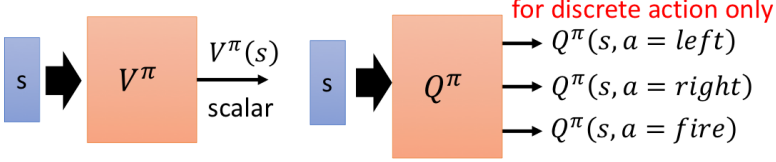
$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n - b\right)\nabla log p_\theta(a_t^n|s_t^n)$$

baseline

G_t^n : obtained via interaction

G_t^n 的估计非常不稳定 当前s执行a后方差很大

- Q-Learning
  → State Value Function $V^\pi(s)$
  → State-action Value Function $Q^\pi(s,a)$
  $V^\pi(s)$ 是 $Q^\pi(s,a)$ 的期望值 MC或TD估计



Actor-Critic $Q^{\pi_\theta}(s_t^n,a_t^n) - V^{\pi_\theta}(s_t^n)$

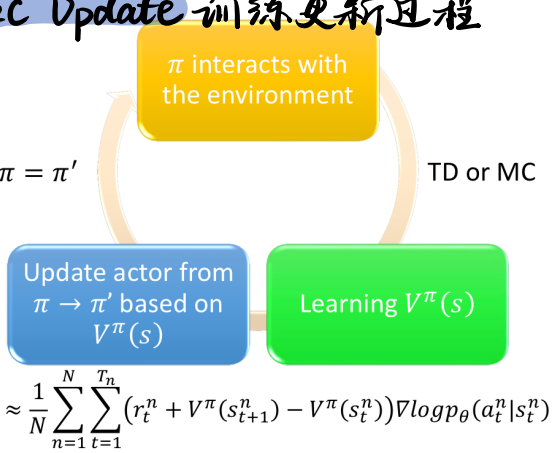
$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n - b\right)\nabla log p_\theta(a_t^n|s_t^n)$$

baseline

G_t^n : obtained via interaction

- 用 $Q^{\pi_\theta}(s_t^n,a_t^n)$ 代替 G_t^n $E[G_t^n] = Q^\pi(s_t^n,a_t^n)$
  G_t^n 是由 interaction 获得（不是期望）
- 用 $V^{\pi_\theta}(s_t^n)$ 代替 b (baseline)

A2C Update 训练更新过程



$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)\right)\nabla log p_\theta(a_t^n|s_t^n)$$

Advantage Actor-Critic

$$Q^\pi(s_t^n,a_t^n) - V^\pi(s_t^n)$$

$$Q^\pi(s_t^n,a_t^n) = E[r_t^n + V^\pi(s_{t+1}^n)] = E[G_t^n]$$

G_t^n 本身就不带期望，所以可直接替换为 $r_t^n + V^\pi(s_{t+1}^n)$

$$r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)$$

Estimate two networks? We can only estimate one.

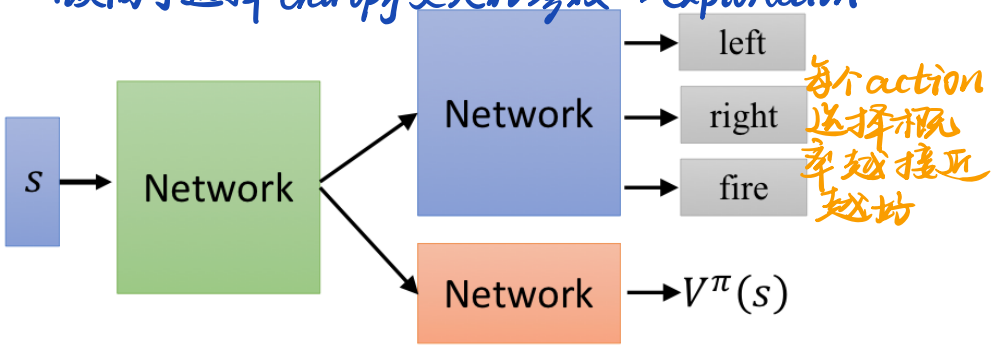
Only estimate state value

A little bit variance

- 只需估计一个 $V^\pi(s_t^n)$ Network 无需两个网络
- 只有 r_t^n 是随机变量 variance 相对 G_t^n 更小

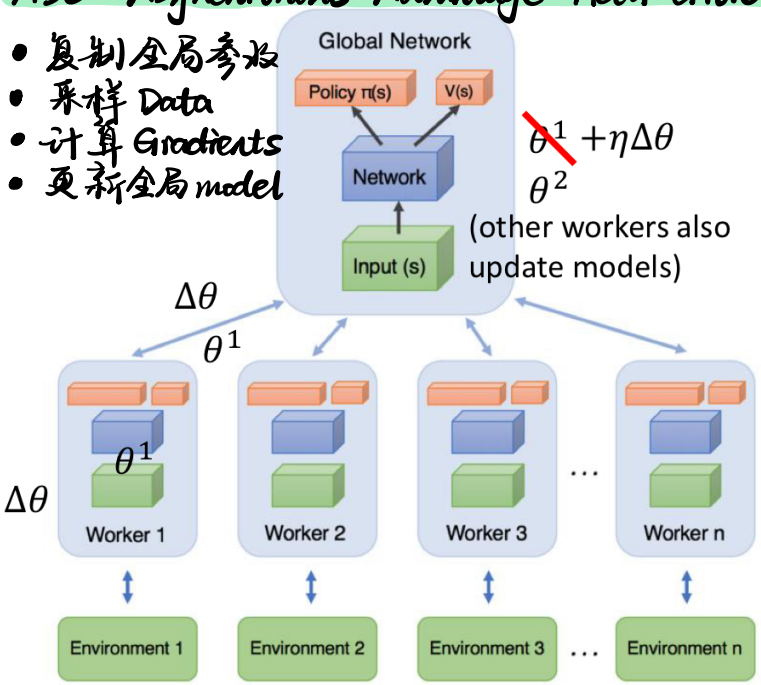
Tips

- actor $\pi(s)$ 和 critic $V^\pi(s)$ 的参数可以共享
- 使用 output entropy 作为 $\pi(s)$ 的 regularization
  倾向于选择 entropy 更大的参数 → exploration

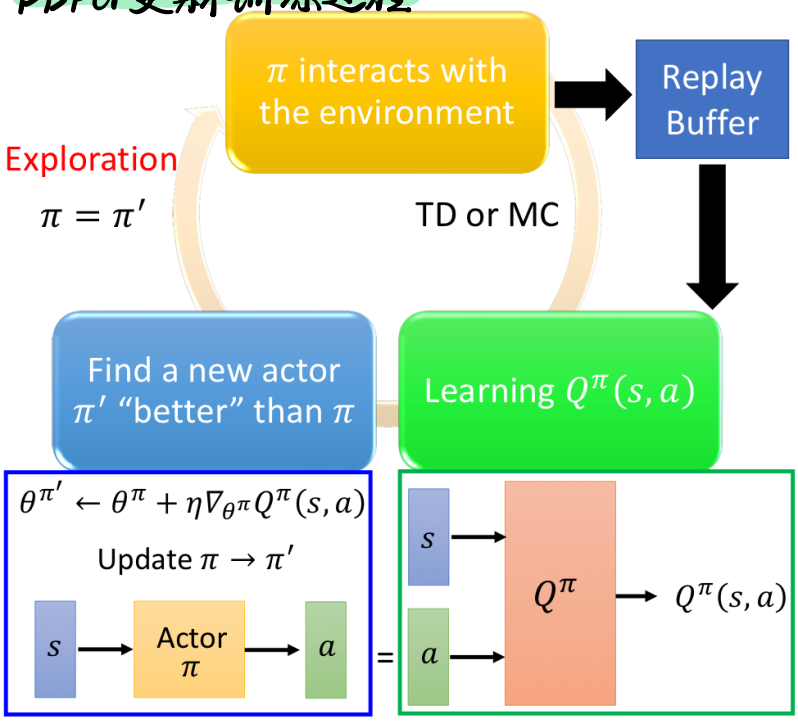


A3C : Asynchronous Advantage Actor Critic

- 复制全局参数
- 采样 Data
- 计算 Gradients
- 更新全局 model



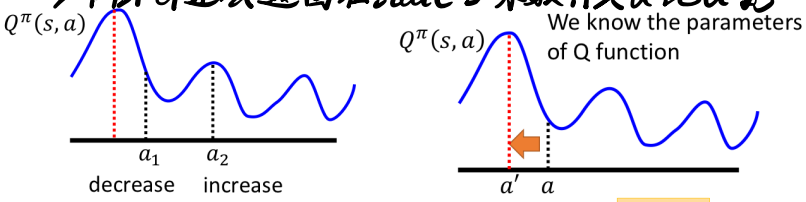
PDPG 更新训练过程



Pathwise Derivative Policy Gradient DDPG

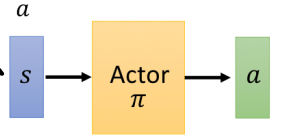
→ Original Actor-Critic 只返回采取 a 后的 $Q(s,a)$

→ PDPG 还会返回在 state s 采取什么 a' 比 a 好



- Action a 是一个连续 vector

$a = \arg\max_a Q(s,a)$



→ 用 Actor 这个 Network 来解决 optimization 问题

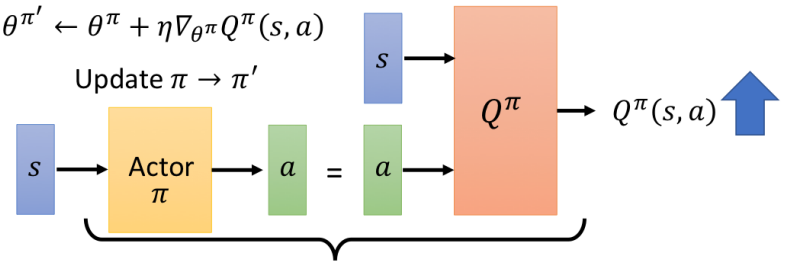
→ a 是 Actor 的 Output $\pi'(s) = \arg\max_a Q^\pi(s,a)$

Gradient ascent:

$\theta^{\pi'} \leftarrow \theta^\pi + \eta\nabla_{\theta^\pi} Q^\pi(s,a)$

Update $\pi \rightarrow \pi'$

Fixed



This is a large network

PDPG Algorithm

- Initialize Q-function Q, target Q-function $\hat{Q} = Q$, actor π, target actor $\hat{\pi} = \pi$
- In each episode
  - For each time step t

1. Given state s_t, take action a_t based on ~~π~~ (exploration)
   - Obtain reward r_t, and reach new state s_{t+1}
   - Store (s_t, a_t, r_t, s_{t+1}) into buffer
   - Sample (s_i, a_i, r_i, s_{i+1}) from buffer (usually a batch)
2. Target $y = r_i + \cancel{\max_a \hat{Q}(s_{i+1}, a)}\ \hat{Q}(s_{i+1}, \hat{\pi}(s_{i+1}))$
   - Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)
3. Update the parameters of π to maximize $Q(s_i, \pi(s_i))$
   - Every C steps reset $\hat{Q} = Q$
4. Every C steps reset $\hat{\pi} = \pi$