人工智能导论

主讲：王 博

人工智能与自动化学院

第五章 行为主义

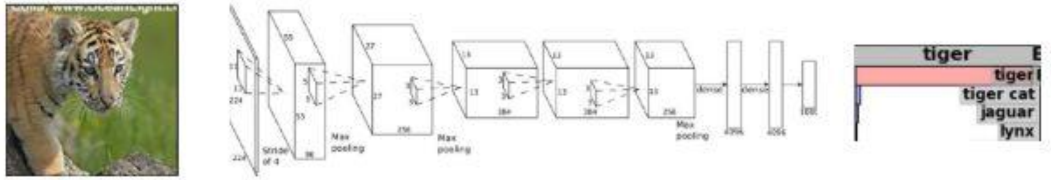
强化学习与智能优化

行为主义 – 目录

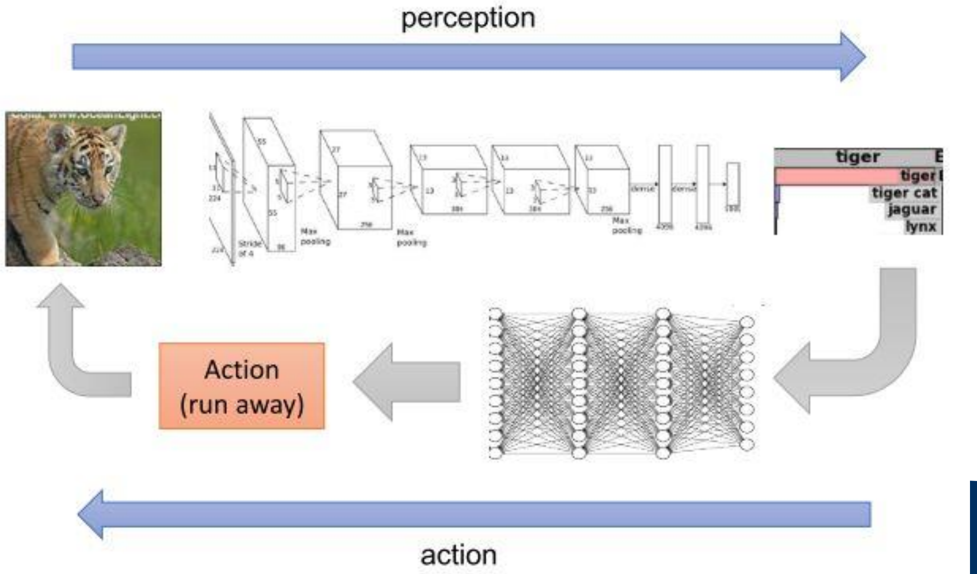
- 行为学派简介
- 经典强化学习思想与原理
- 深度强化学习简介
- 智能优化方法简介

深度强化学习框架

- 深度强化学习是深度学习和强化学习的结合
  - 深度学习的表达能力 → 策略和值函数的建模问题



  - 强化学习的决策能力 → 定义问题和优化目标



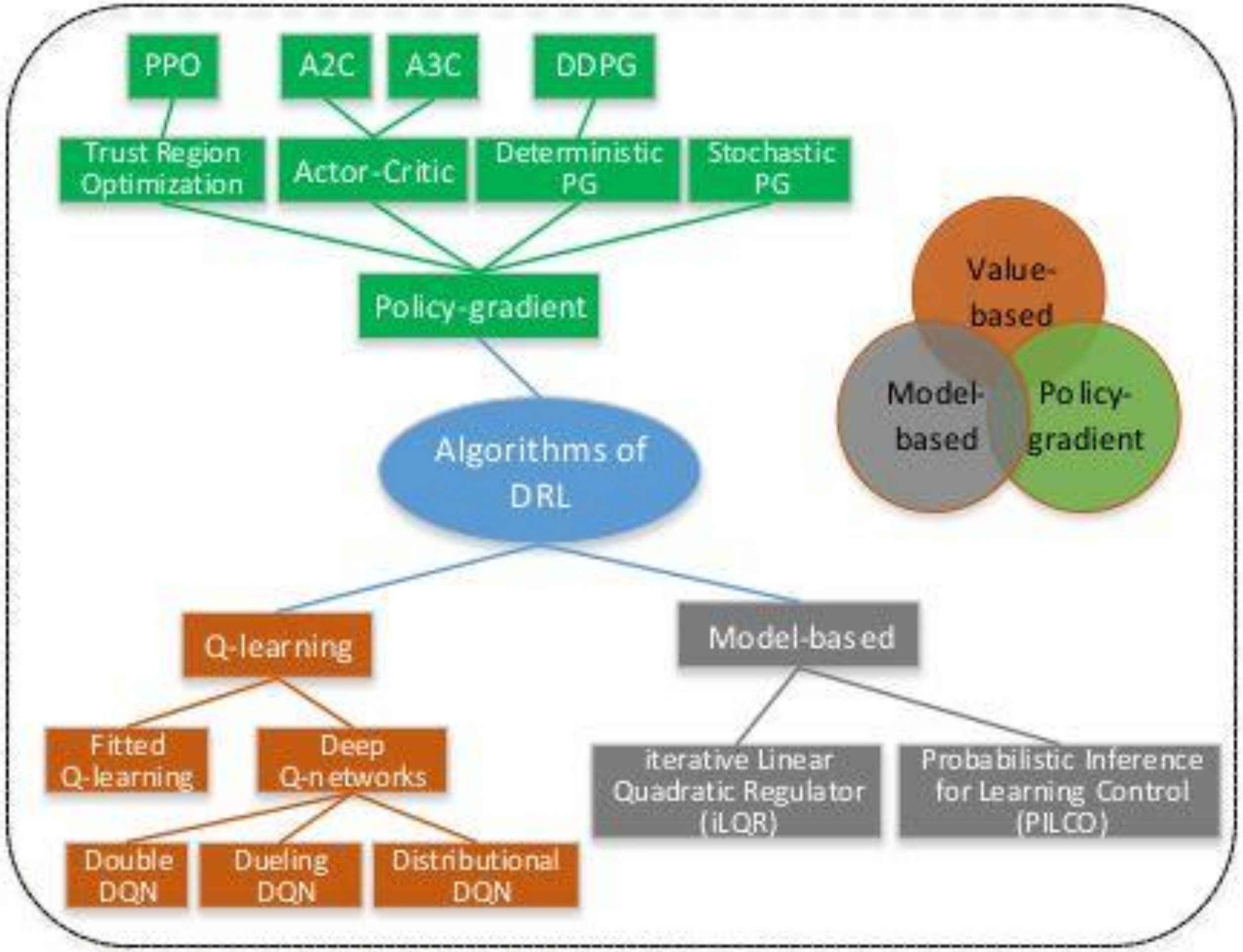
深度强化学习算法家族

基于值函数的方法

- 深度Q网络，DQN
  - Double DQN，Dueling DQN，Prioritized Replay

基于策略的方法

- 蒙特卡洛策略梯度，REINFORCE
- Actor-Critic算法，Asynchronous Advantage Actor Critic，A3C
- 近端策略优化，Proximal Policy Optimization，PPO
- 深度确定性策略梯度，Deep Deterministic Policy Gradient，DDPG



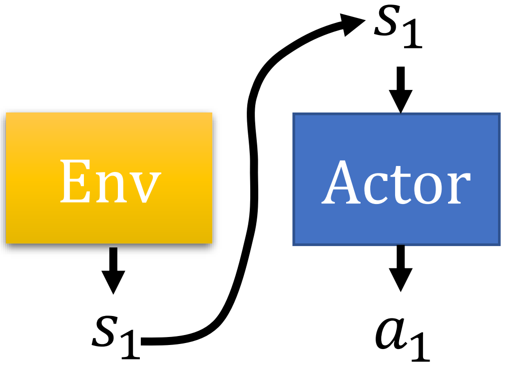
基于策略/值函数的强化学习

- 基于策略的
  - 在<u>策略空间</u>中搜索
  - 没有值函数

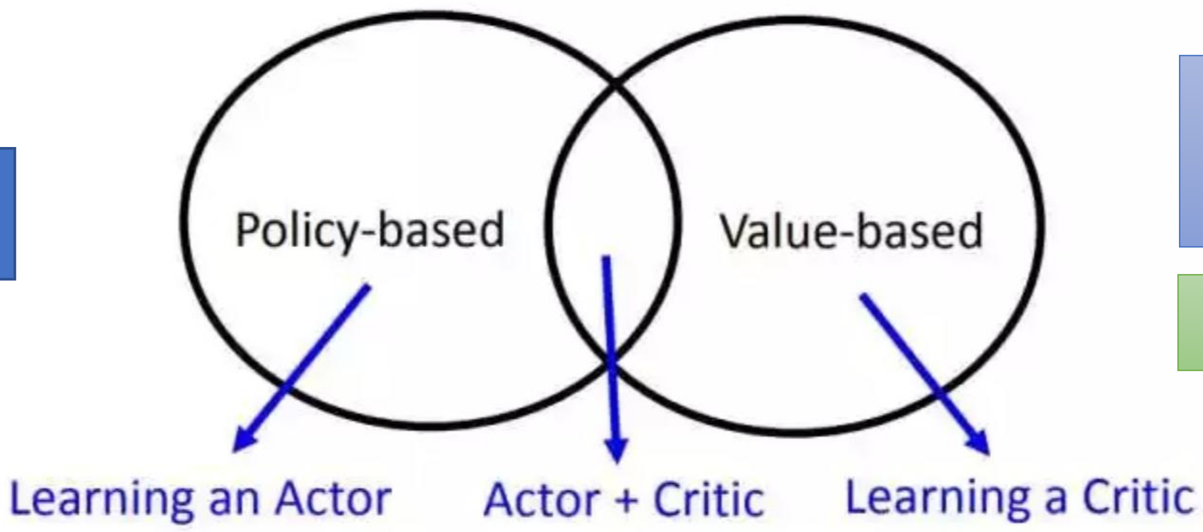
    （根据结果好坏直接调整策略）

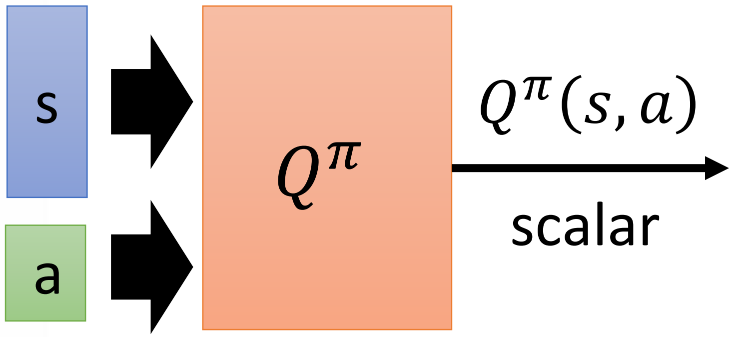
- 基于值函数的
  - 在<u>值函数空间</u>中搜索
  - 策略隐式表达

    （基于值函数选择动作）



Actor

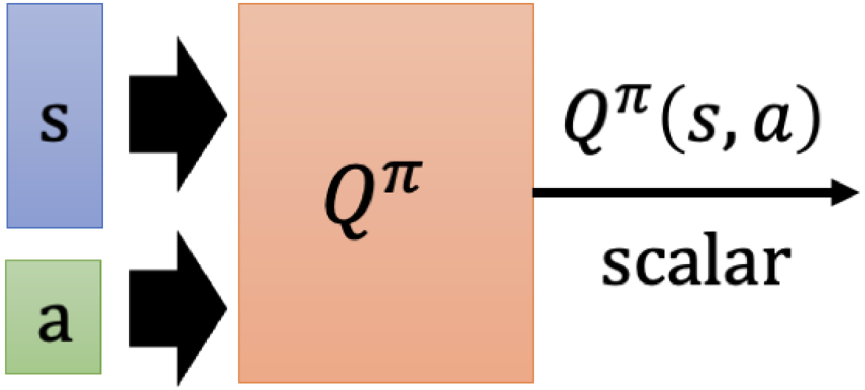


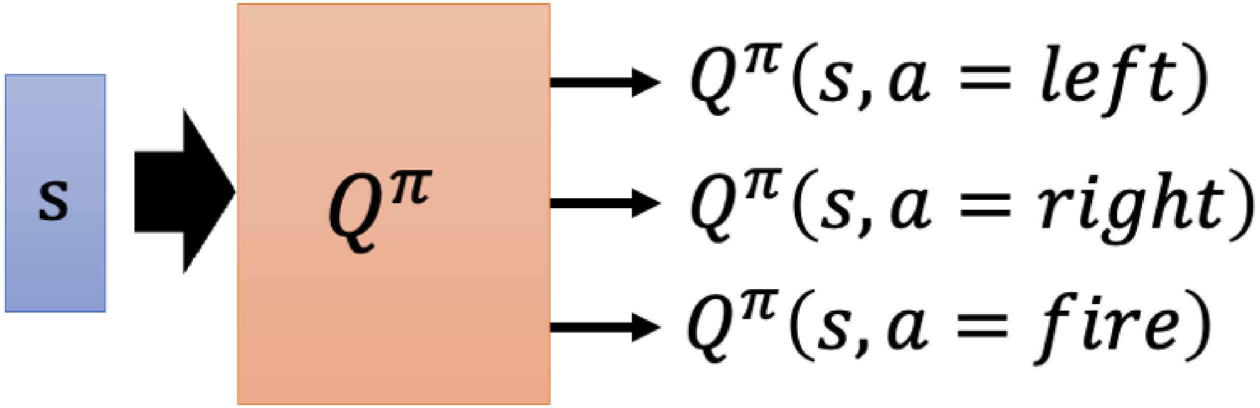


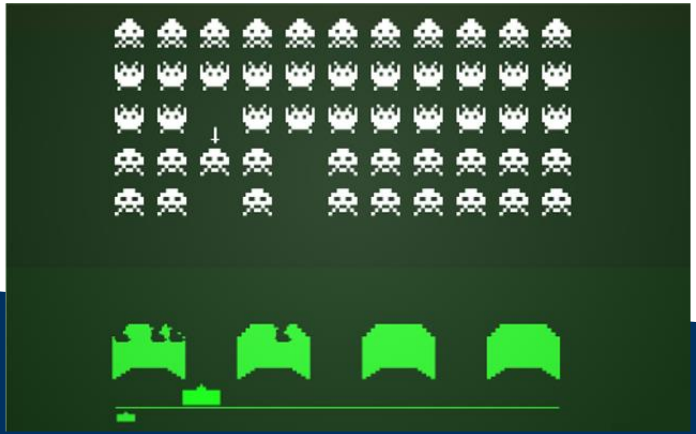
Critic

Critic的两种形式

- 状态动作值函数 $Q^{\pi}(s,a)$
  - 基于策略π，在状态s执行了动作a后获得的累积奖级*cumulated reward*







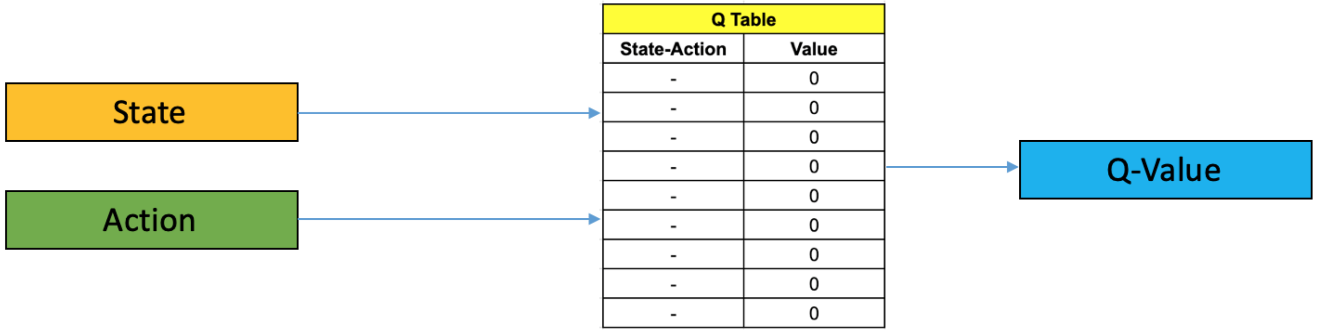
适用于离散动作

深度Q网络

- Q表的局限性：当状态和行为的组合不可穷尽时，无法通过查表的方式选取最优的Action。

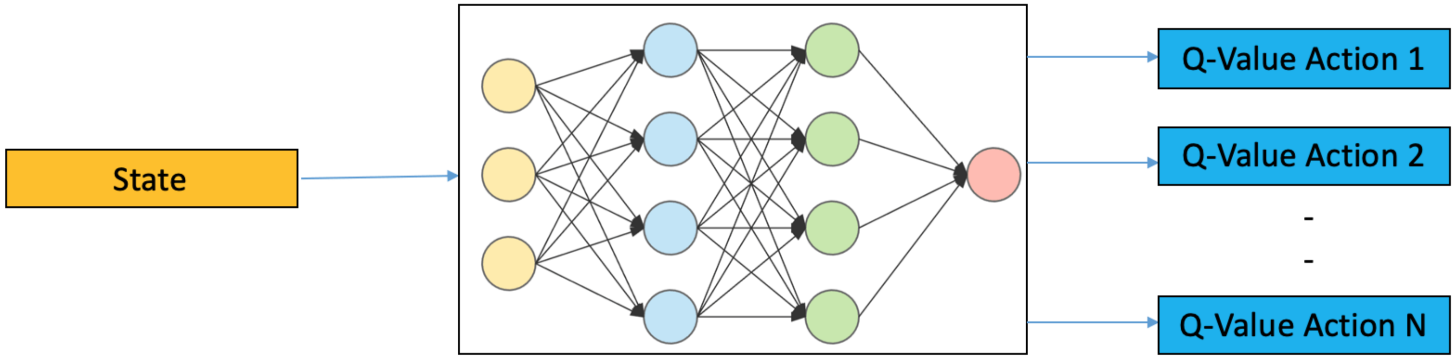
DQN：用(深度)神经网络拟合Q表：

- 转化为监督学习
- 目标：Q函数
- 样本：交互（试错）产生的数据



Q Table	
State-Action	Value
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

Q Learning

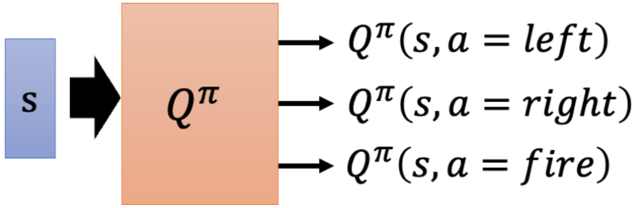


Deep Q Learning

深度Q网络

使用DQN模型代替Q表会遇到的问题：

- 交互得到的序列存在一定的相关性：
  - 监督学习要求样本独立同分布。
- 交互数据的使用效率：
  - 迭代需要样本数量较多，样本获取靠交互。



Structured Data

Size	#bedrooms	...	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
⋮	⋮		⋮
27	71244		1

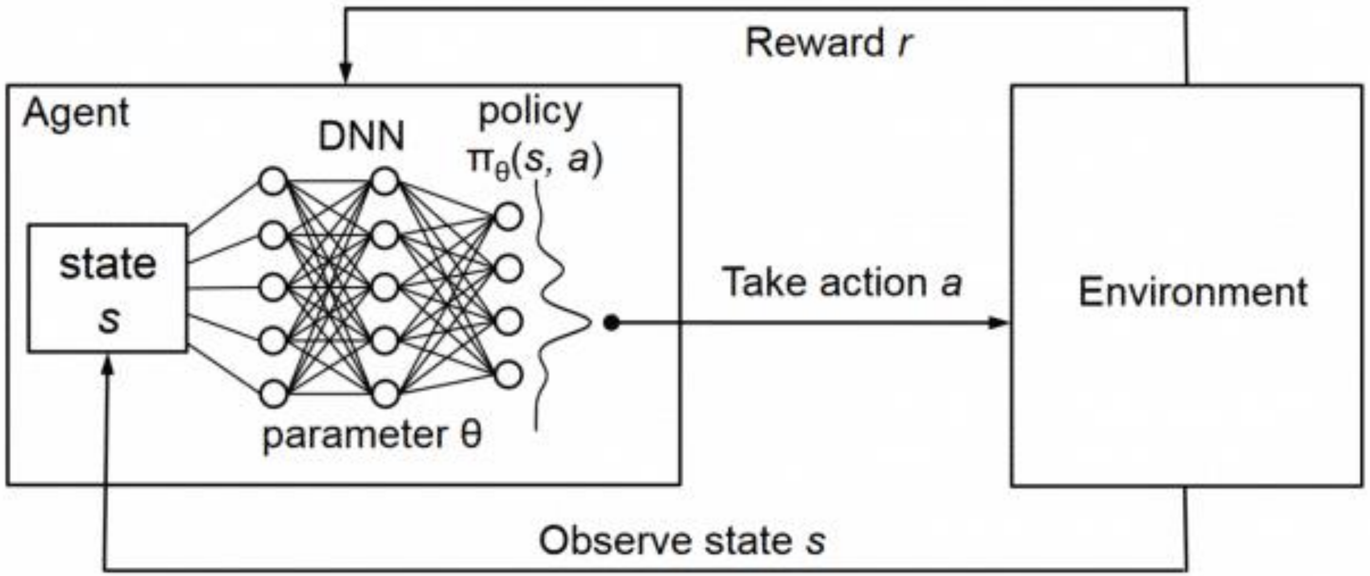
Unstructured Data


Audio


Image

Four scores and seven years ago...

Text

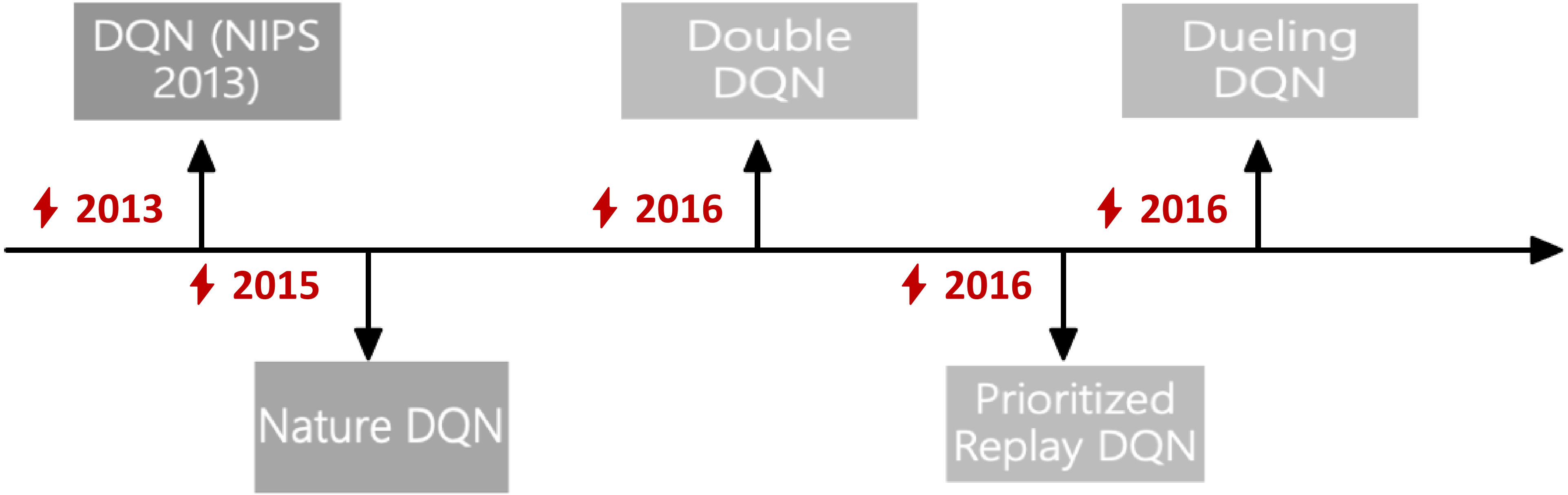


DQN的发展历程

[1] Playing atari with deep reinforcement learning, NIPS Deep Learning Workshop 2013, https://arxiv.org/abs/1312.5602

[3] Van Hasselt, et al. "Deep reinforcement learning with double q-learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 30. No. 1. 2016.

[5] Wang, Ziyu, et al. “Dueling network architectures for deep reinforcement learning.” International conference on machine learning, ICML, 2016.



[2] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

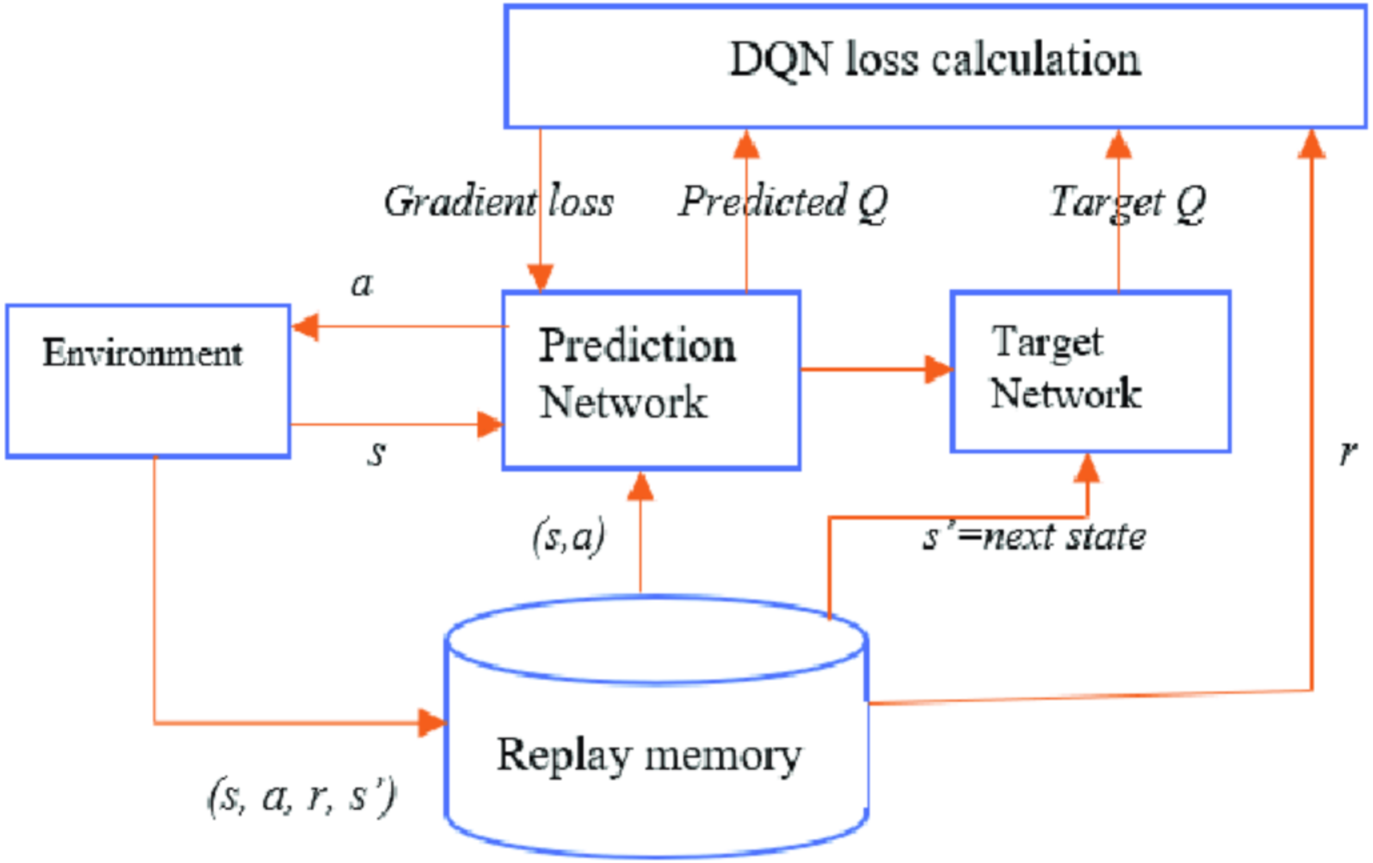
[4] Prioritized Experience Replay, ICLR 2016, https://arxiv.org/abs/1511.05952

Nature DQN

- 问题
  - 交互得到的序列存在一定的相关性：

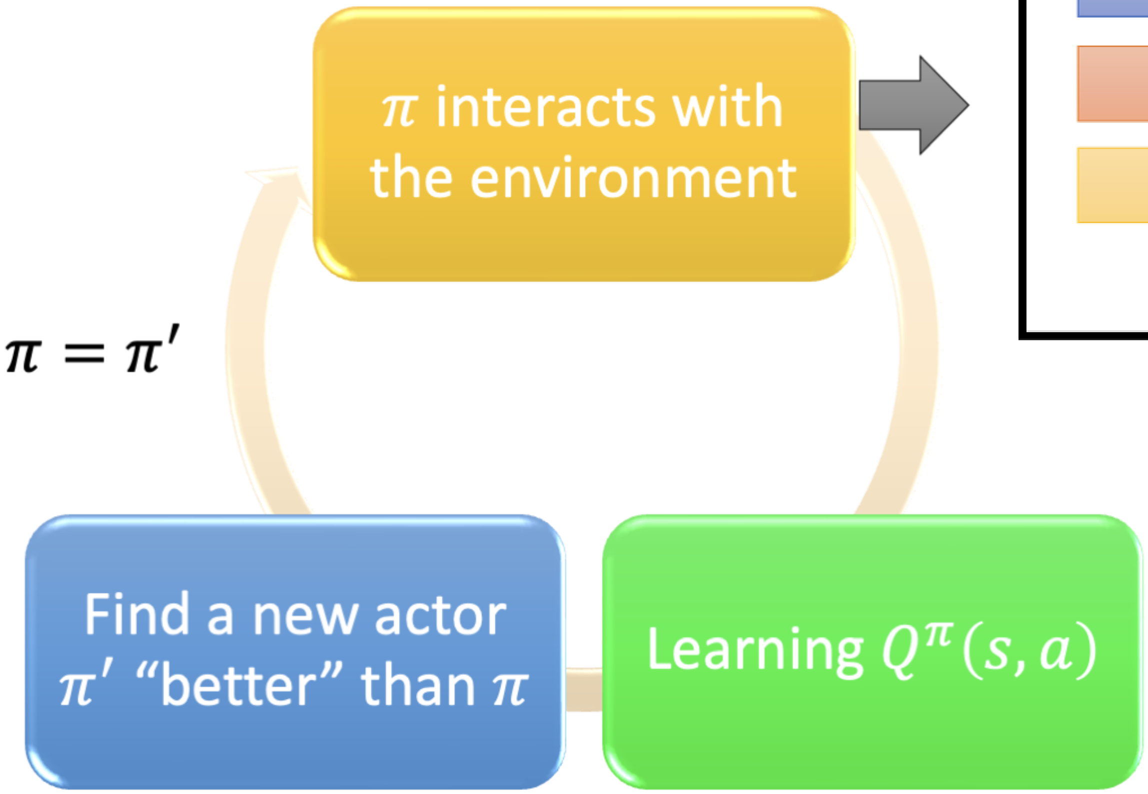
    ※监督学习要求样本独立同分布。
  - 交互数据的使用效率：

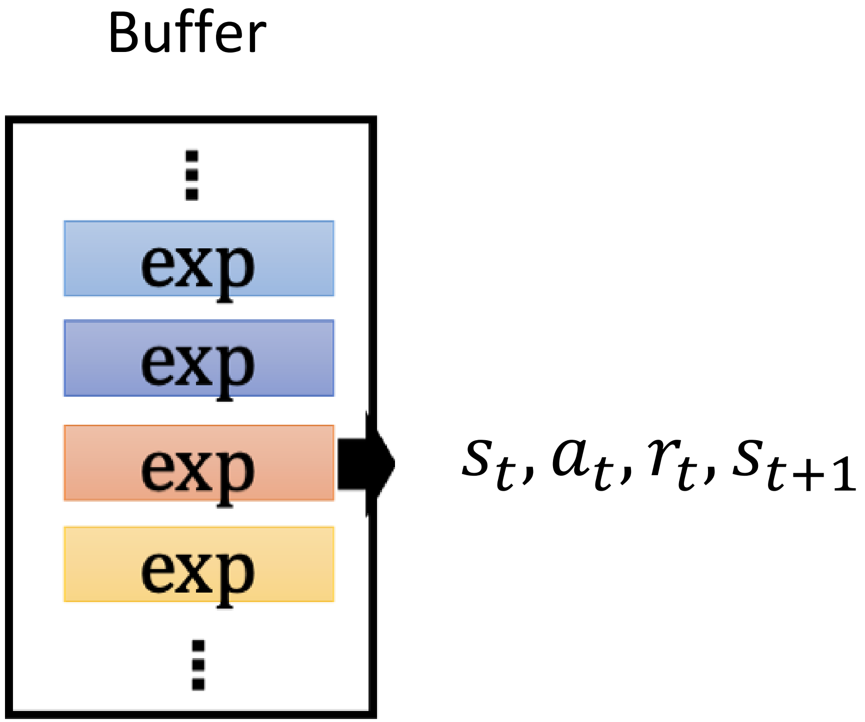
    ※迭代需要样本数量较多，样本获取靠交互。
- 特点：
  - 经验回放（Experience replay）
  - 固定target Q值



Replay Buffer

Put the experience into buffer.

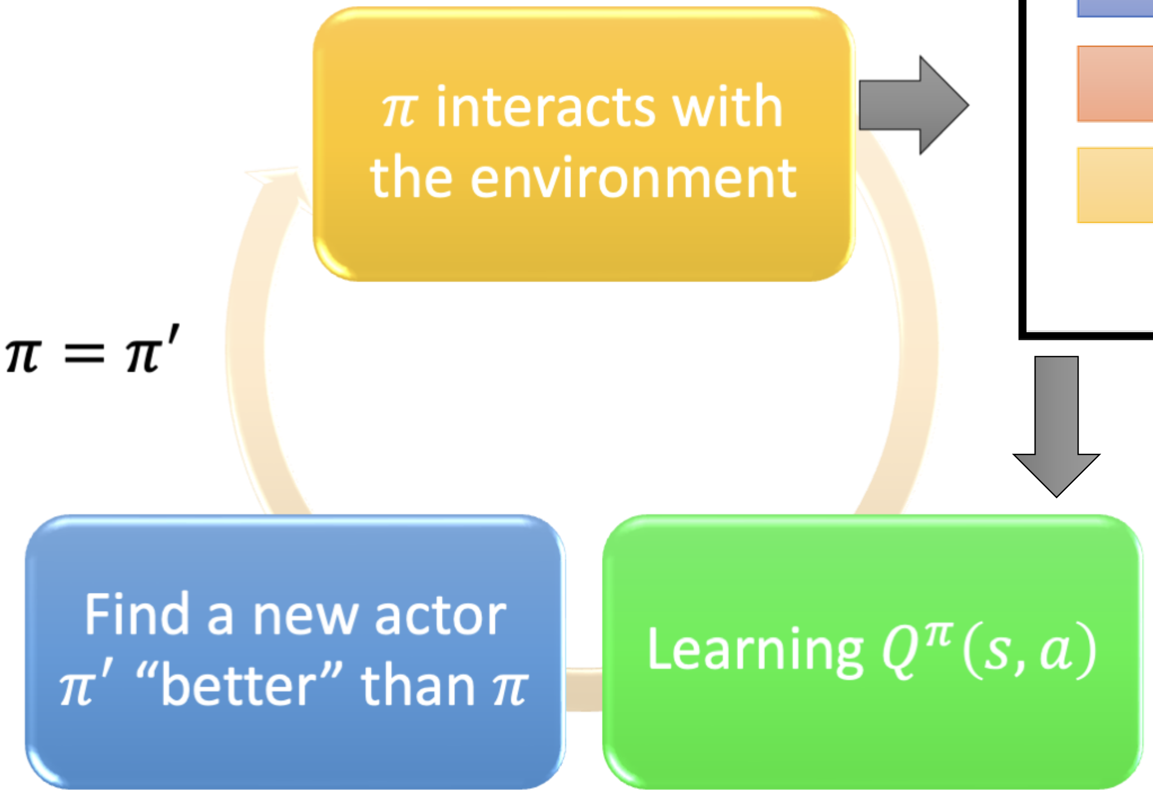




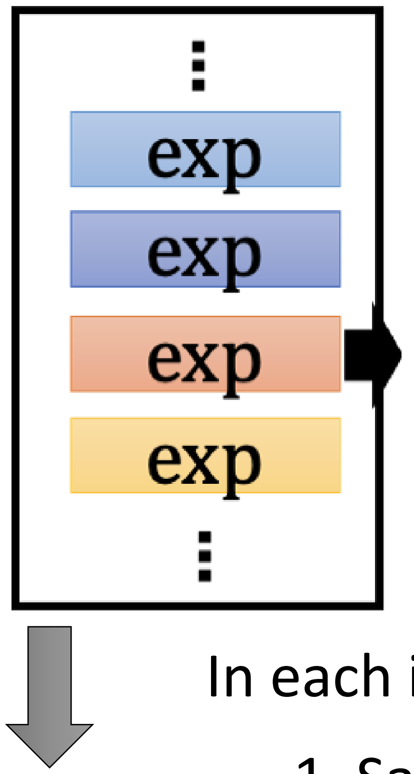
- The experience in the buffer comes from different policies.
- Drop the old experience if the buffer is full.

Replay Buffer

Put the experience into buffer.



Buffer



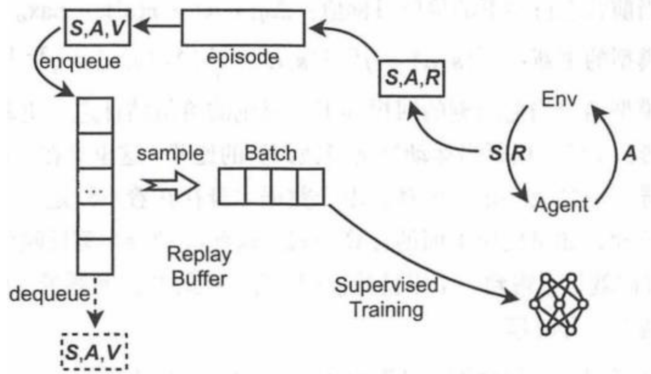


图 7-15 Replay Buffer 的结构图

s_t, a_t, r_t, s_{t+1}

In each iteration:

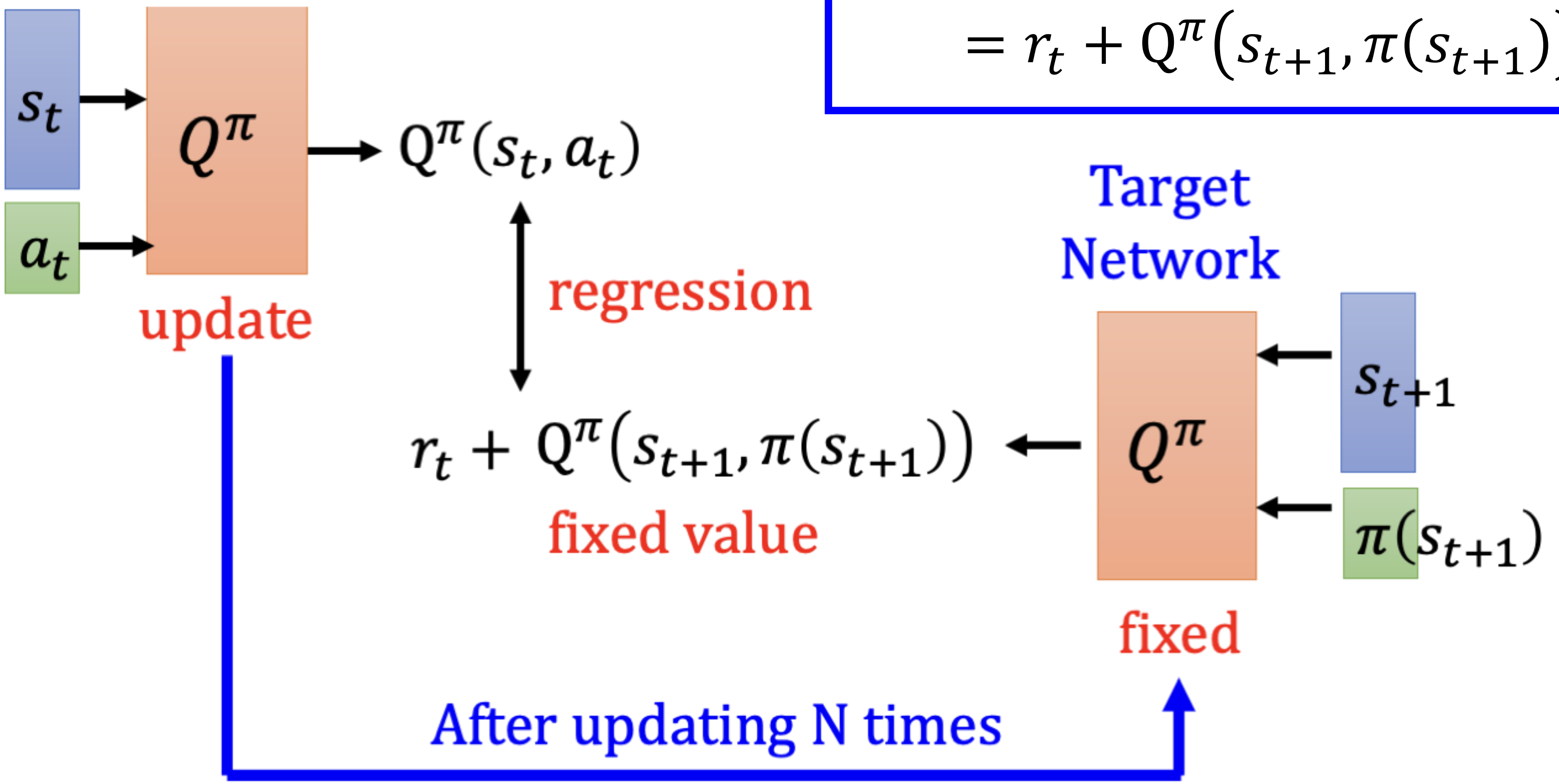
1. Sample a batch
2. Update Q-function

Off-policy

Target Network

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

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



Exploration

$s \to a_1$ $Q(s, a_1) = 0$ Never explore

$s \to a_2$ $Q(s, a_2) = \boxed{1}$ Always sampled

$s \to a_3$ $Q(s, a_3) = 0$ Never explore

- The policy is based on Q-function

$$\boldsymbol{a = arg\max_{a} Q(s, a)}$$

This is not a good way for data collection.

Epsilon Greedy

ε would decay during learning

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

Boltzmann Exploration

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

Typical DQN 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)
    - Every C steps reset $\hat{Q} = Q$

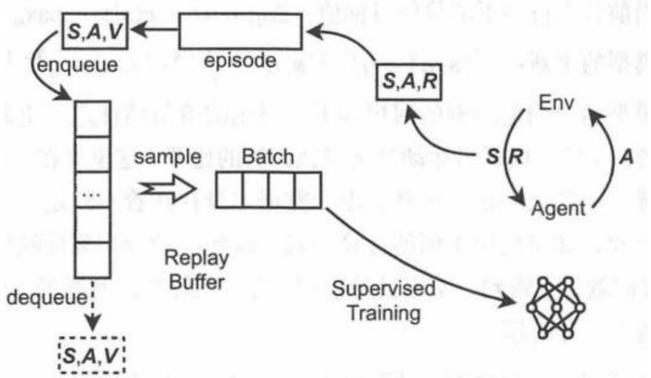


图 7-15 Replay Buffer 的结构图

Fixed Target

$$L(\theta) := \mathbb{E}_{(s,a,r,s')\sim\mathscr{D}}\left[\left(\underbrace{r(s,a) + \gamma \max_{a' \in \mathscr{A}} Q(s', a'; \theta^-)}_{\text{Target}} - \underbrace{Q(s,a;\theta)}_{\text{Prediction}}\right)^2\right]$$

DQN的改进

- **Double DQN**
  - 用当前Q网络计算最大Q值对应的动作，用目标Q网络计算这个最大动作对应的目标Q值，进而消除贪婪法带来的偏差。
- **Prioritized Replay DQN**
  - 对DQN的经验回放池按权重采样
  - 根据每个样本的TD误差绝对值$|\delta(t)|$，给定该样本的优先级正比于$|\delta(t)|$，将这个优先级的值存入经验回放池
- **Dueling DQN**
  - 通过优化神经网络的结构来优化算法，把网络输出：状态动作值函数$Q(s,a)$，分为优势函数$A(s,a)$和状态值函数$V(s)$

DQN存在的问题

基于值函数强化学习方法的共同问题：

- 无法表示随机策略
  - DQN在实现时采用了贪婪策略，无法实现按照概率执行各种候选动作的要求。
- 无法表示连续动作。DQN要求动作空间是离散的，且只能是有限个：
  - 某些问题中，动作是连续的，例如要控制在x y z方向的速度、加速度，这些值显然是连续的。
- 对受限状态下的问题处理能力不足
  - 真实环境下不同的状态由相同的特征表达。
- DQN输出值（各个动作的Q值）的微小改变会导致某一动作被选中或不选中，这种不连续的变化会影响算法的收敛

贪婪策略的缺陷

策略梯度 (Policy Gradient)

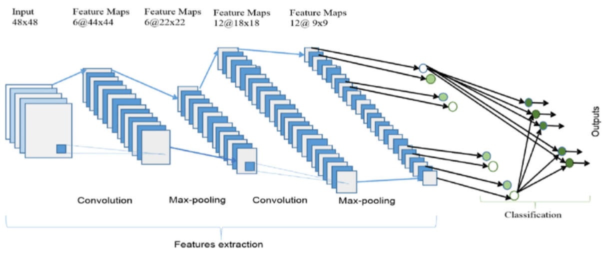
- 动机：基于值函数不能通吃所有场景。为什么一定要根据值函数选择动作？
- 好处：直接更新策略梯度
  - 连续动作空间
  - 随机策略
- 策略函数的近似表示：
  - 策略π可以被被描述为一个包含参数θ的函数

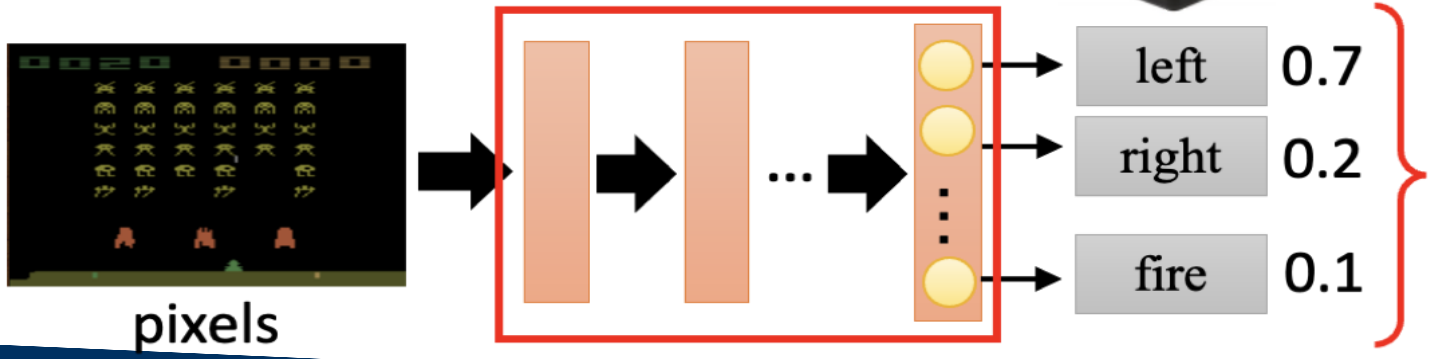
$$\pi_\theta(s,a) = P(a|s,\theta) \approx \pi(a|s)$$

- 策略表示为连续函数：最佳策略优化

Policy of Actor

- Policy π is a network with parameter θ
  - Input: the observation of machine represented as a vector or a matrix
  - Output: each action corresponds to a neuron in output layer



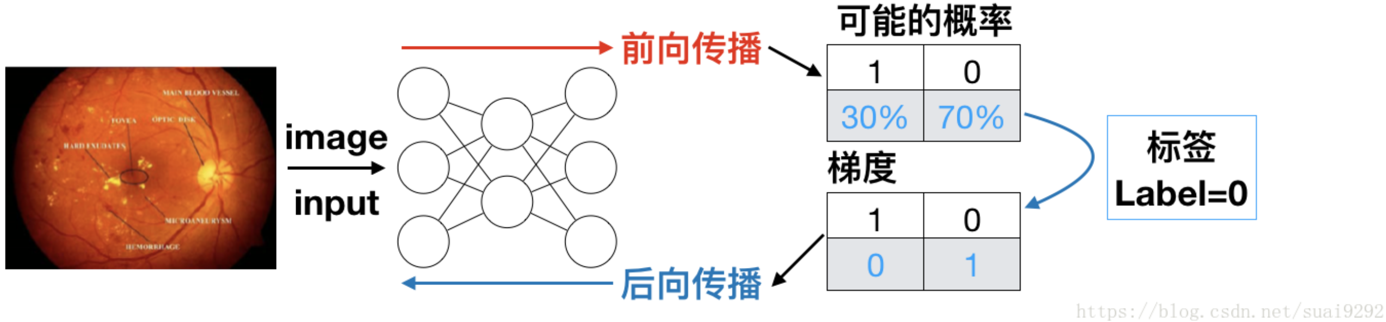


Take the action based on the probability.

Score of an action

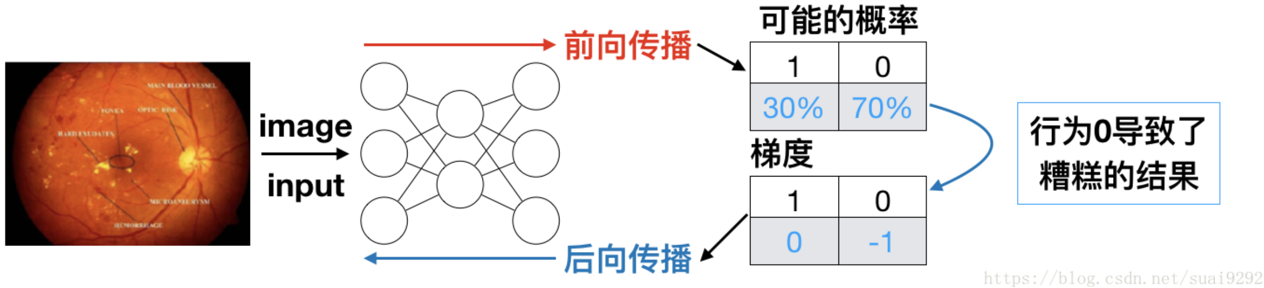
策略梯度 (Policy Gradient)

- 监督学习的权重调整：



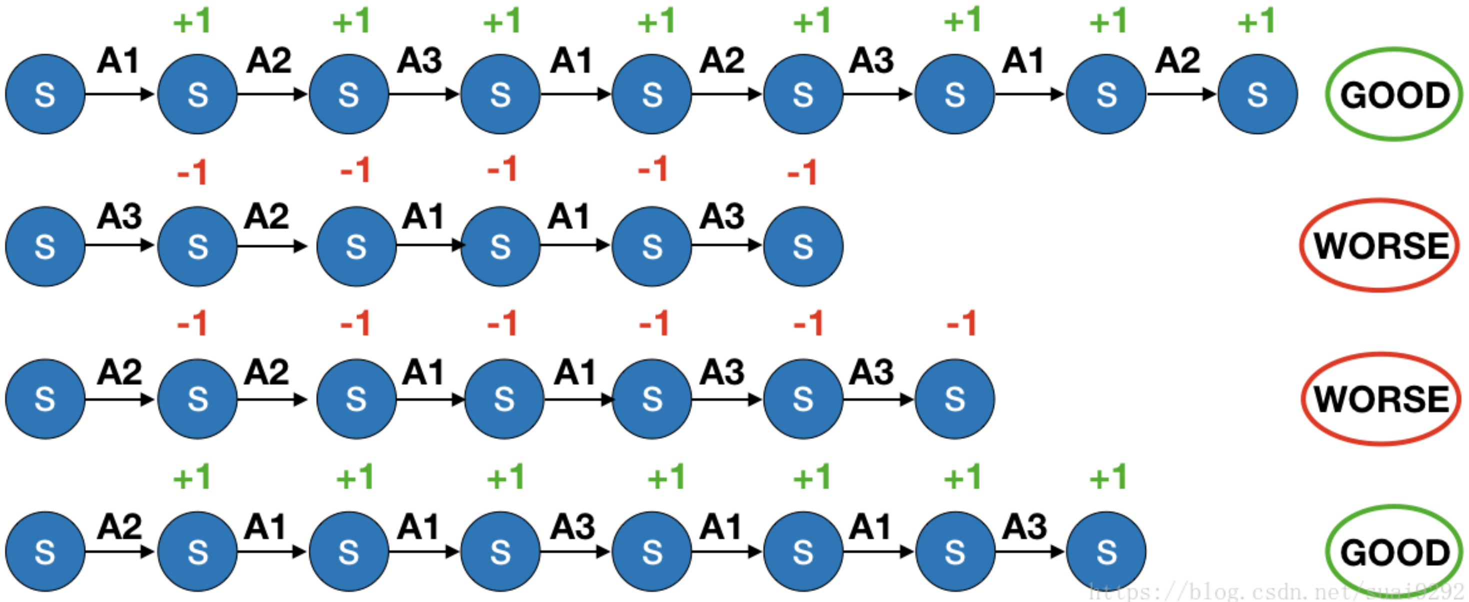
- 策略梯度强化学习的权重调整

前提：如果一系列行为最后导致的结果是差的，那么我们认为这一系列的行为都是不好的行为



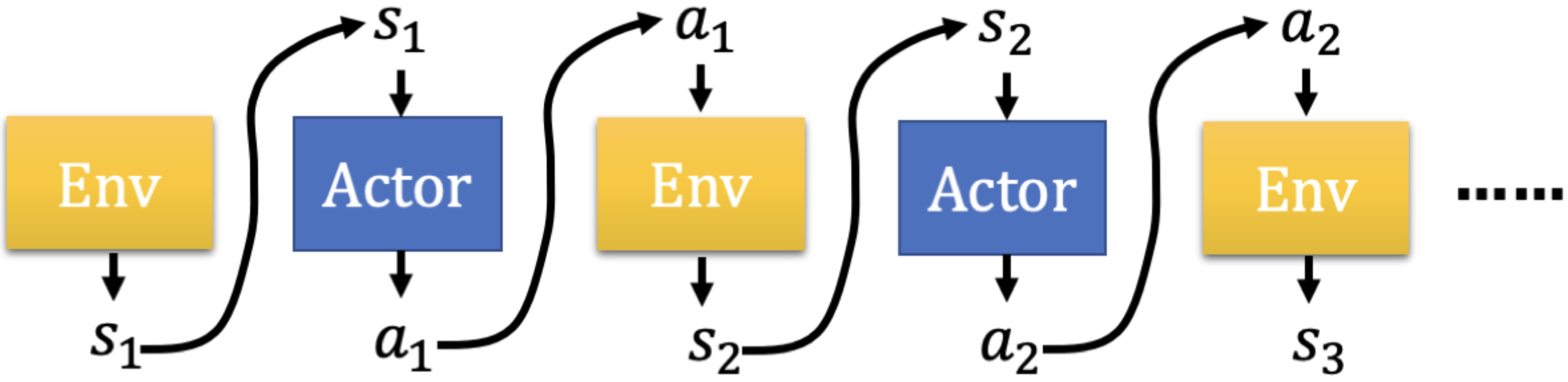
策略梯度（Policy Gradient）

- 以游戏为例
  - 所有获胜对局中的动作都认为是好的 → 正向更新
  - 所有失败对局中的动作都认为是不好的→ 负向更新



https://blog.csdn.net/zuaj0292

策略梯度（Policy Gradient）



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

$$p_\theta(\tau) = p(s_1)p_\theta(a_1|s_1)p(s_2|s_1,a_1)p_\theta(a_2|s_2)p(s_3|s_2,a_2)\cdots$$

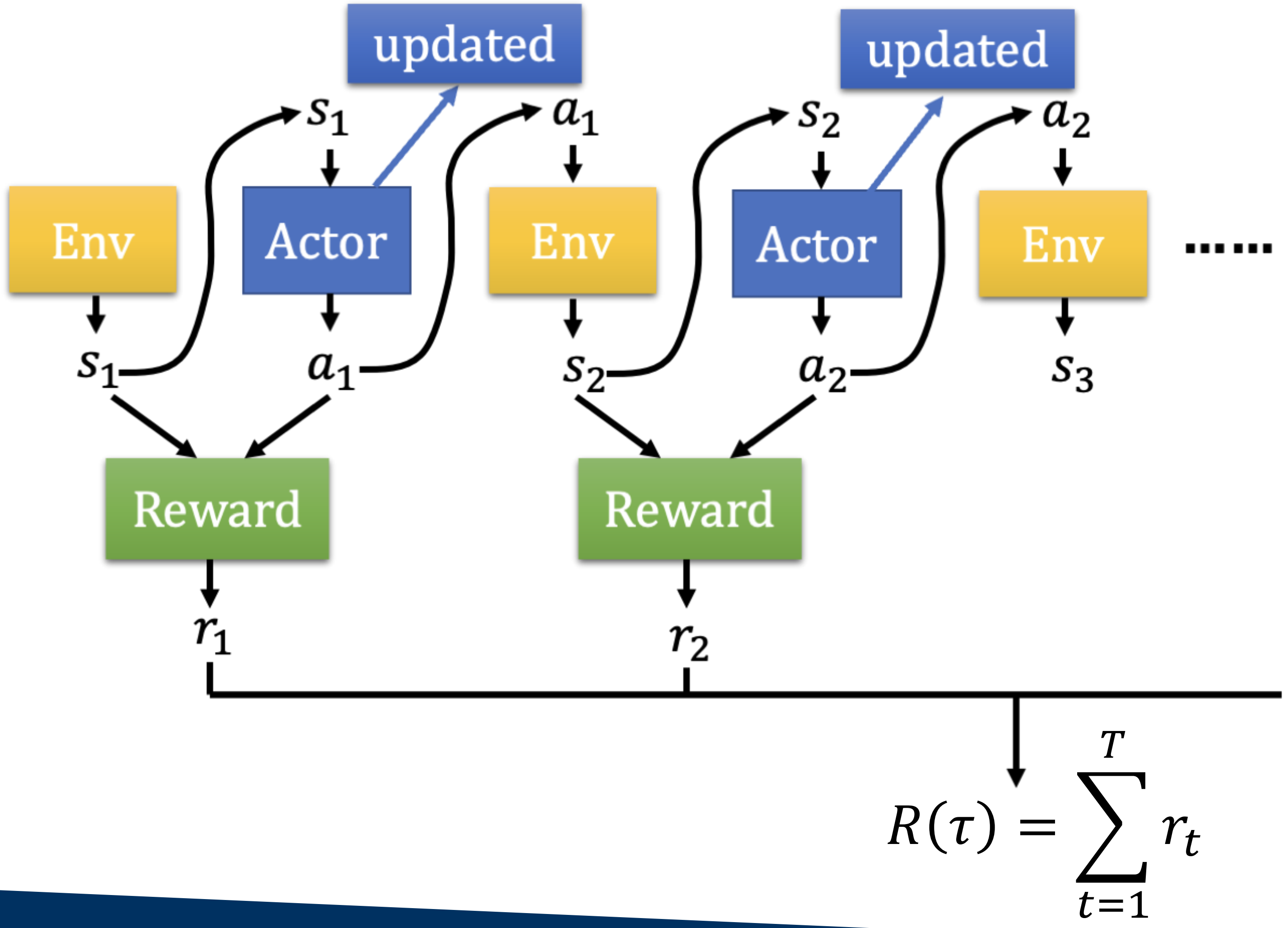
$$= p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

策略梯度（Policy Gradient）

Expected Reward

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

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



策略梯度（Policy Gradient）

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

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

$R(\tau)$ do not have to be differentiable, it can even be a black box.

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

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

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

$$p_\theta(\tau) = p(s_1) \prod_{t=1}^{T} p_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

策略梯度定理

- 策略梯度定理：对于任意MDP，不论是优化平均奖励还是初始状态奖励，目标对参数θ求梯度的形式都可以表示为：

$$\nabla_\theta J(\theta) = E_{s,\pi_\theta}[\nabla_\theta \boxed{log\pi_\theta(s,a)} \boxed{R_\pi(s,a)}]$$

分值函数
score function

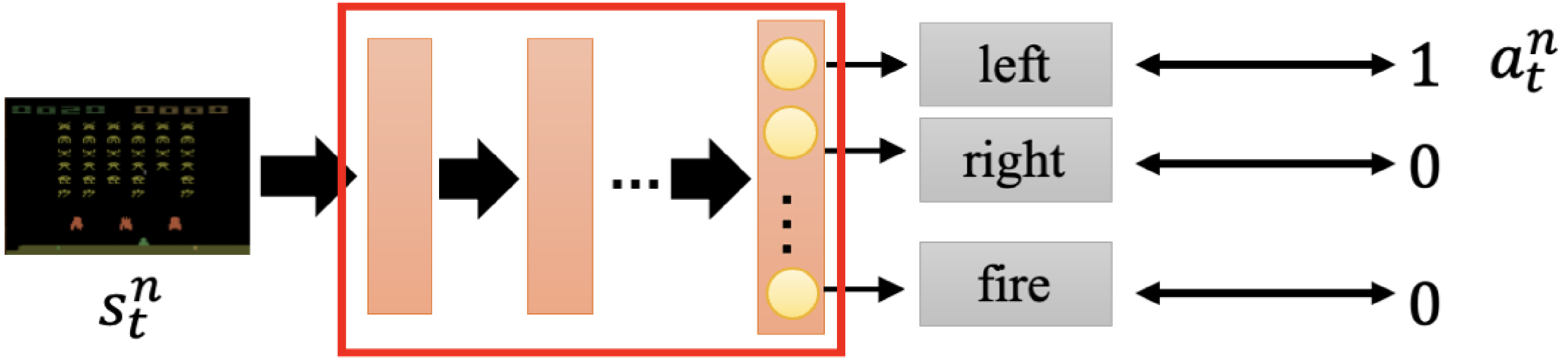
累积收益
根据优化目标变化

- 可优化的函数目标：
  - 优化初始状态收获的期望：$J_1(\theta) = V_{\pi\theta}(s_1) = E_{\pi\theta}(G1)$
  - 优化平均价值：$J_{avV}(\theta) = \sum_s d_{\pi\theta}(s) V_{\pi\theta}(s)$
  - 优化每一时间步的平均奖励： $J_{avR}(\theta) = \sum_s d_{\pi\theta}(s) \sum_a \pi_\theta(s,a) R_s^a$

分值函数

$s_t^n \quad a_t^n \quad R(\tau^n)$

Consider as classification problem



Classification: $$\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} log p_\theta(a_t^n|s_t^n) \xrightarrow{\text{TF, pyTorch ...}} \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n} \nabla log p_\theta(a_t^n|s_t^n)$$

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

估计累积收益$R_\pi(s,a)$

- 蒙特卡洛策略梯度算法 REINFORCE
  - 使用价值函数$v(s)$近似代替$G_\pi(s,a)$
  - 蒙特卡洛方法估计$v(s)$

- REINFORCE的采样梯度是无偏的。但是同样由于MC，导致REINFORCE梯度估计的方差很大，从而可能会降低学习的速率

for each step t

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s,a) v_t$$

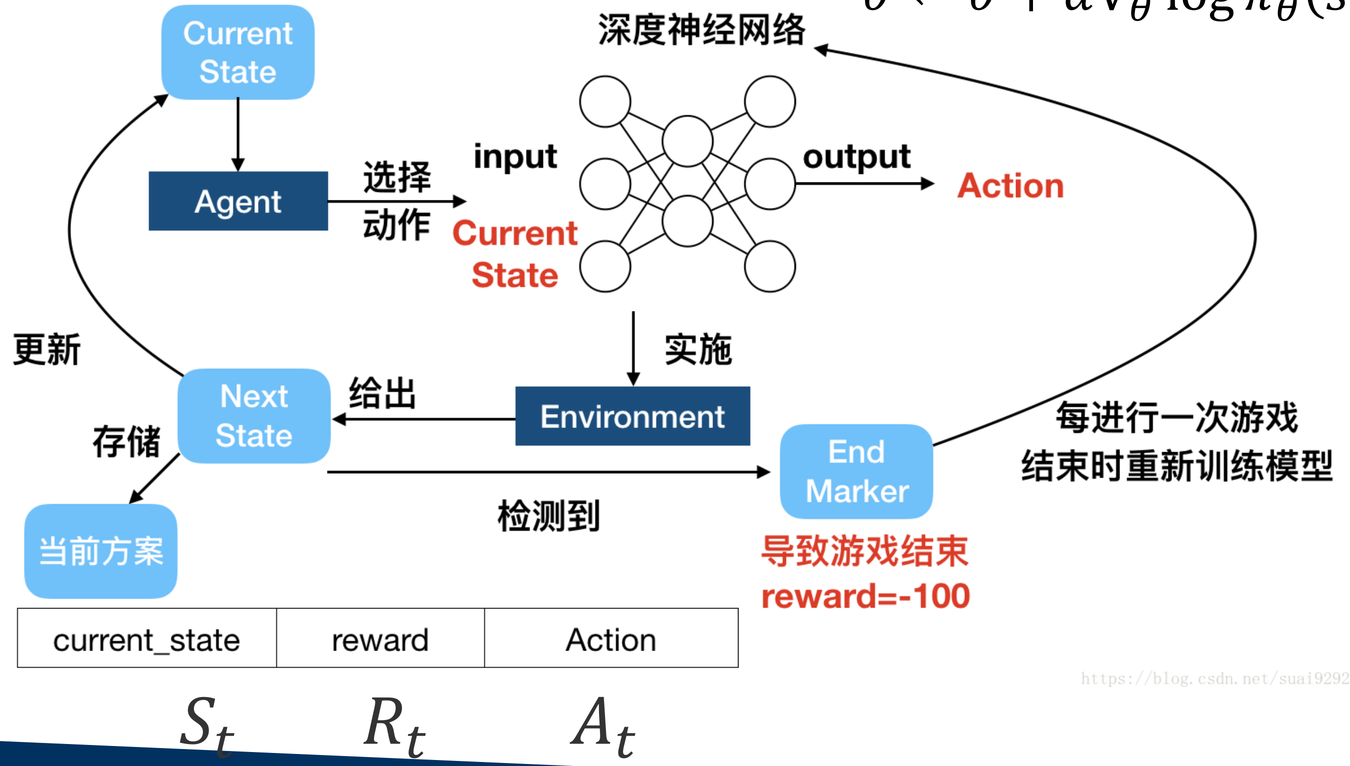
输入：

1) N个蒙特卡罗完整序列

2) 训练步长α

输出：

策略函数的参数θ



蒙特卡洛策略梯度算法REINFORCE

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

蒙特卡洛策略梯度的局限

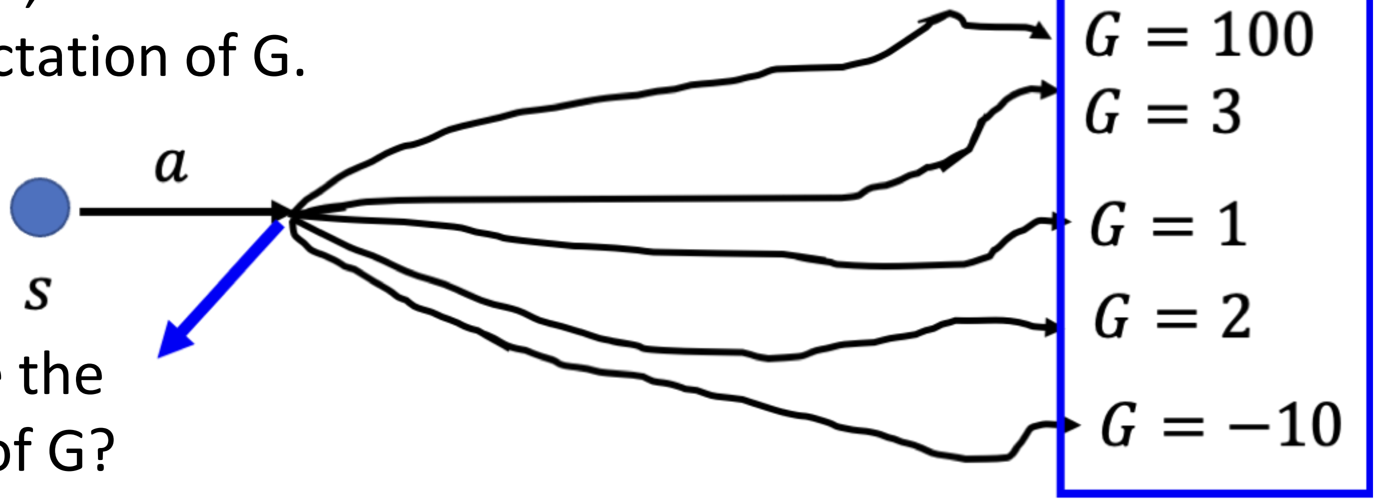
baseline

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

G_t^n : obtained via interaction

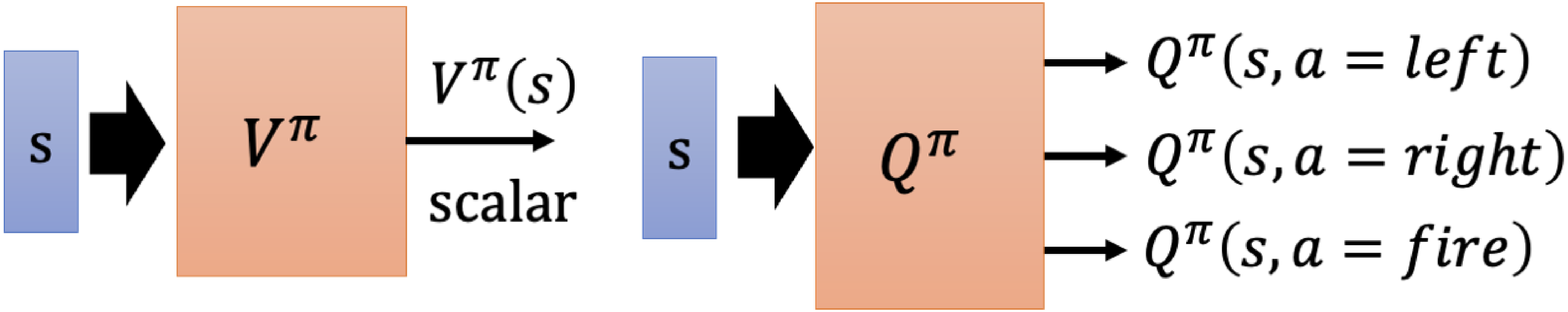
Very unstable

With sufficient samples,
approximate the expectation of G.



Can we estimate the
expected value of G?

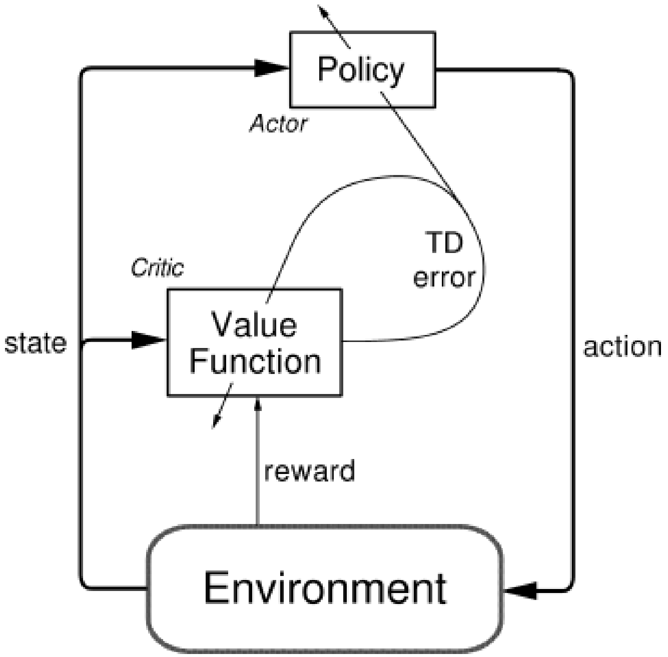
估计累积回报的方法



Estimated by TD or MC

能否结合两者? $E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n)$

Actor-Critic算法



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

$$V^{\pi_\theta}(s_t^n)$$

baseline

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

G_t^n : obtained via interaction

$$E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n)$$

Advantage Actor-Critic

Actor-Critic算法

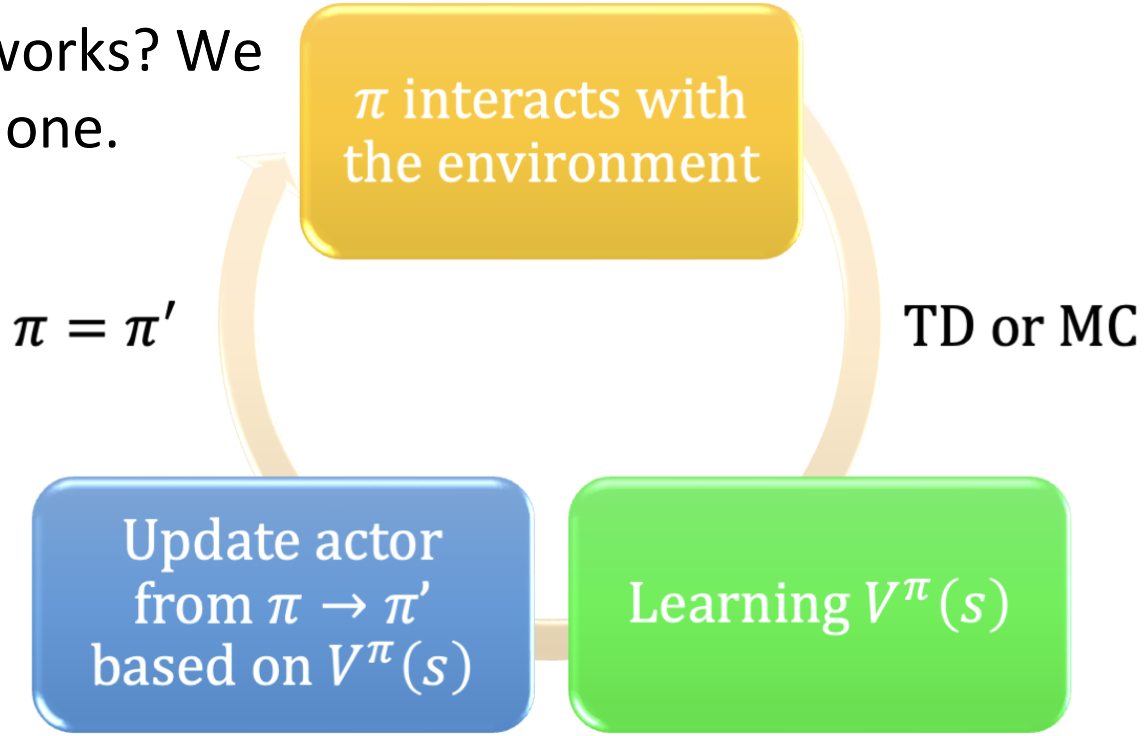
$$Q^{\pi}(s_t^n, a_t^n) - V^{\pi}(s_t^n)$$

Estimate two networks? We can only estimate one.

$$\Downarrow$$

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

Only estimate state value, suffering a little bit variance



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

Actor-Critic算法可选形式

- 基于状态价值（蒙特卡洛策略梯度）

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s,a)\, V(s,\omega)$$

- 基于动作价值（DQN）

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s,a)\, Q(s,a,\omega)$$

- 基于时序差分误差（时序差分学习）

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s,a)\, \delta(t)$$

- 基于$TD(\lambda)$误差（时序差分学习）

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s,a)\, \delta(t) E_r(t)$$

- 基于优势函数（Dueling DQN）

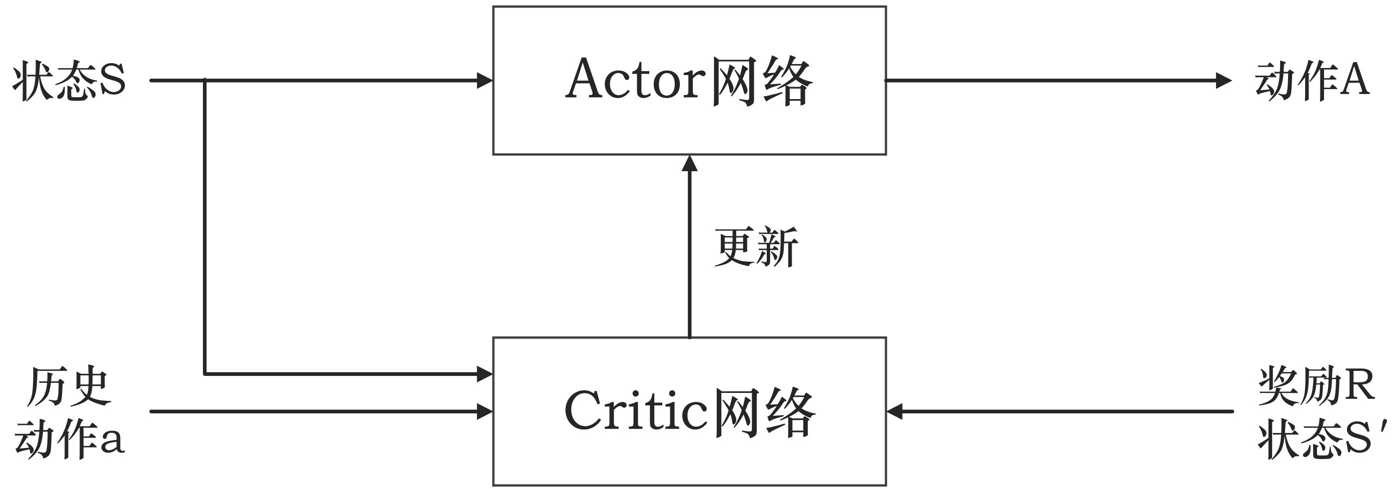
$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s,a)\, A(S,A,\omega,\beta)$$

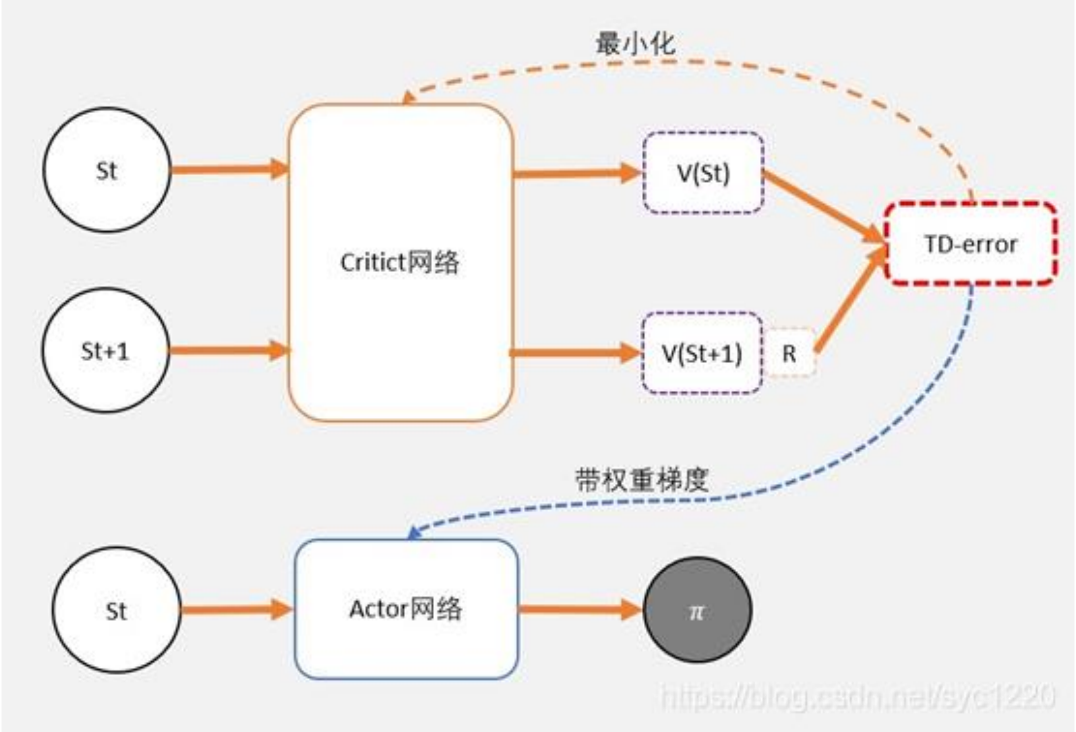
Actor-Critic算法

- 算法框图：
  - 以TD Actor-Critic为例

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(S_t, A)\delta_t$$

$$\delta_t = R + \gamma V(S') - V(S)$$





损失函数：$\sum\left(R + \gamma V(S') - V(S, \omega)\right)^2$

Actor-Critic算法流程

算法输入：迭代轮数T，状态特征维度n，动作集A，步长α, β，衰减因子γ，探索率ϵ，Critic网络结构和Actor网络结构。

输出：Actor 网络参数θ，Critic网络参数w

1. 随机初始化所有的状态和动作对应的价值Q

2. for i from 1 to T，进行迭代。

   a) 初始化S为当前状态序列的第一个状态，拿到其特征向量$\phi(S)$

   b) 在Actor网络中使用$\phi(S)$作为输入，输出动作A,基于动作A得到新的状态S',反馈R。

   c) 在Critic网络中分别使用$\phi(S)$，$\phi(S')$作为输入，得到Q值输出$V(S)$，$V(S')$

   d) 计算TD误差$\delta = R + \gamma V(S') - V(S)$

   e) 使用均方差损失函数$\sum(R + \gamma V(S') - V(S, w))^2$作Critic网络参数$w$的梯度更新

   f) 更新Actor网络参数θ：

$$\theta = \theta + \alpha \nabla_\theta log \pi_\theta(S_t, A)\delta$$

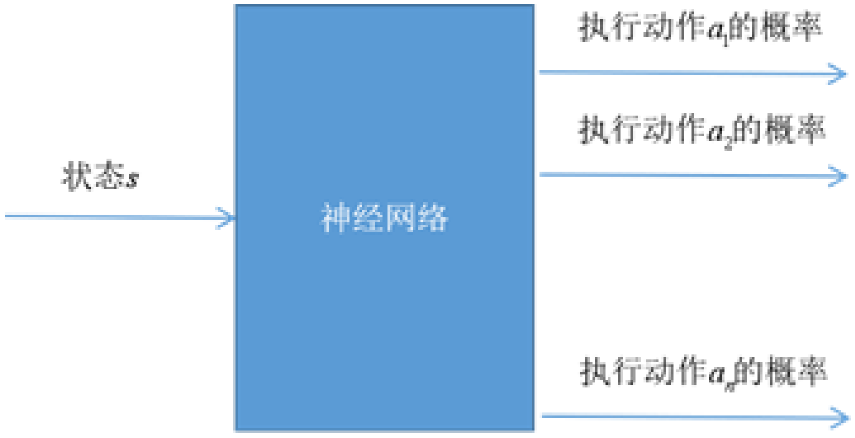
对于Actor的分值函数$\nabla_\theta log \pi_\theta(S_t, A)$,可以选择softmax或者高斯分值函数。

HUAZHONG UNIV

分值函数

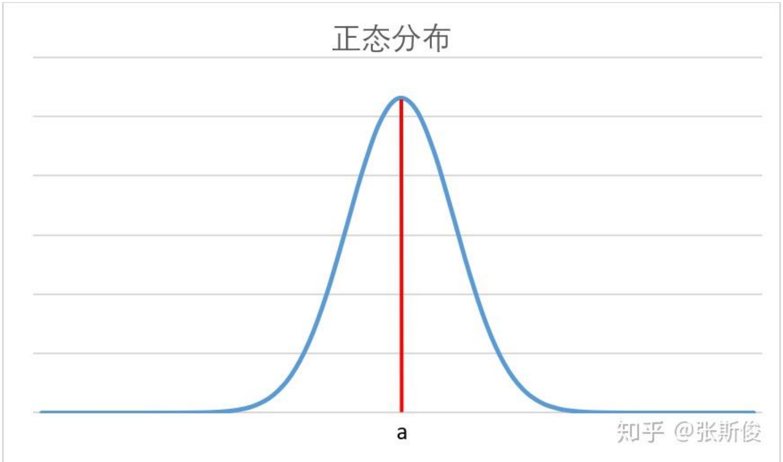
- 对于离散型动作空间：
  - 常用Softmax函数
  - $\pi_\theta(s,a) = \dfrac{e^{\phi(s,a)^T\theta}}{\sum_i e^{\phi(s,i)^T\theta}}$
  - 分值函数：

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - E_{\pi\theta}[\phi(s,\cdot)]$$



- 对于连续型动作空间：
  - 例子，高斯策略函数， $\pi_\theta(s,a) \sim N(\phi(s)^T\theta, \sigma^2)$
  - 分值函数：

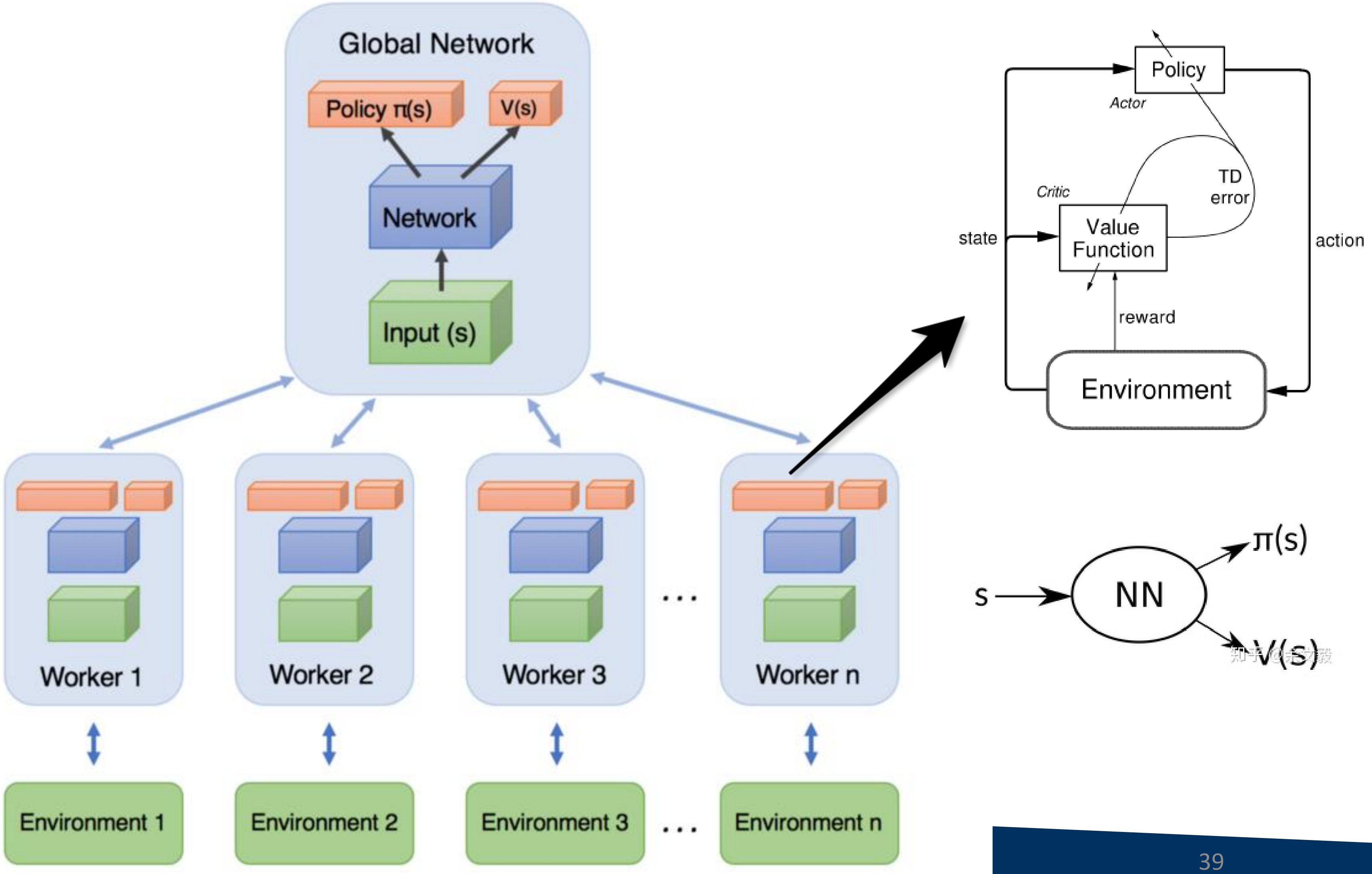
$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a-\phi(s)^T\theta)\phi(s)}{\sigma^2}$$



Asynchronous Advantage Actor-Critic (A3C)

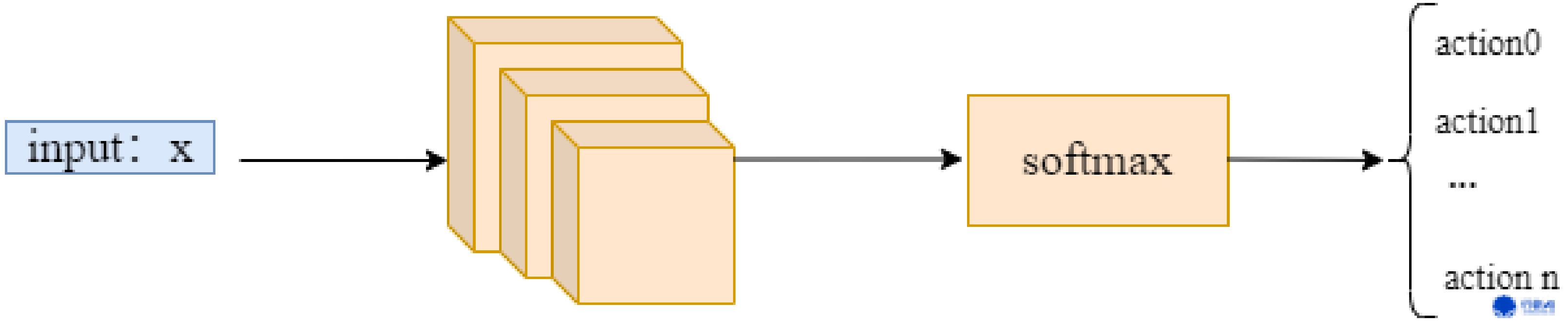
Asynchronous Advantage Actor-Critic (A3C)

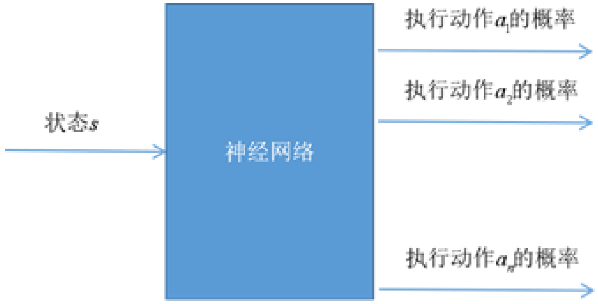
1. Copy global parameters
2. Sampling some data
3. Compute gradients
4. Update global models



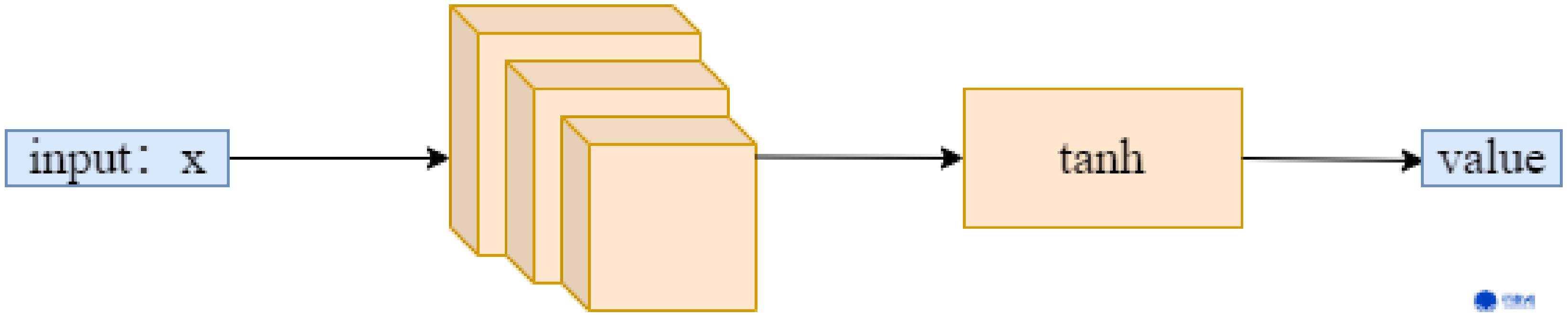
离散型vs.连续型动作

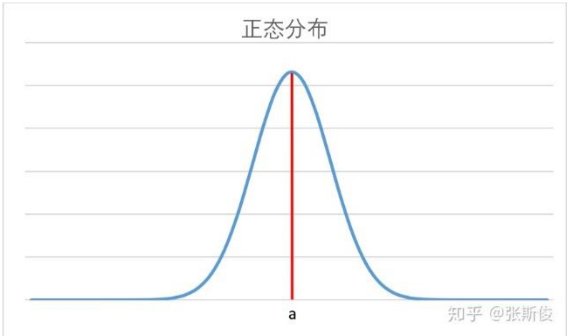
- 离散动作



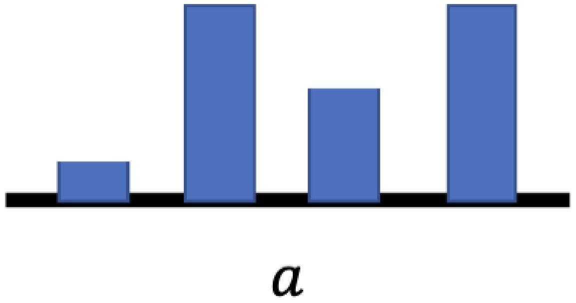


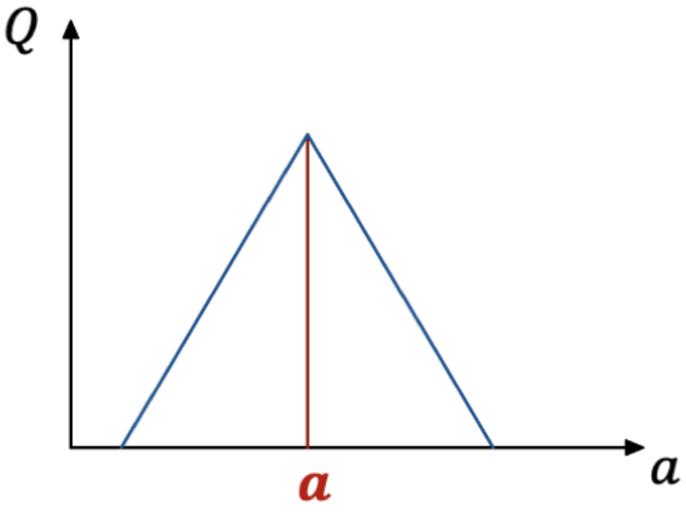
- 连续动作

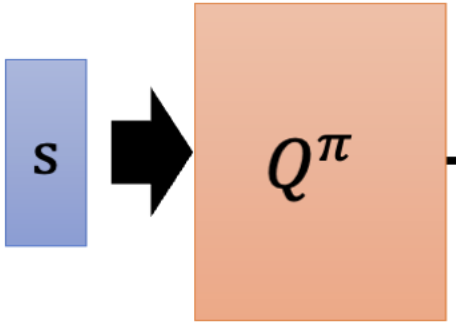




随机策略vs.确定性策略

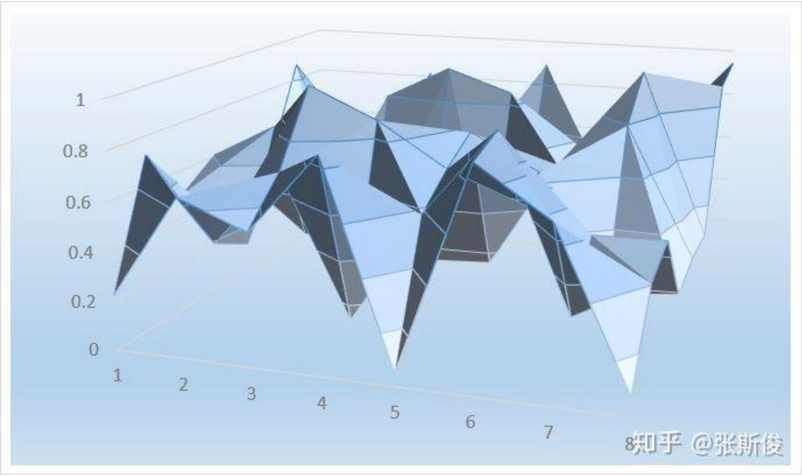




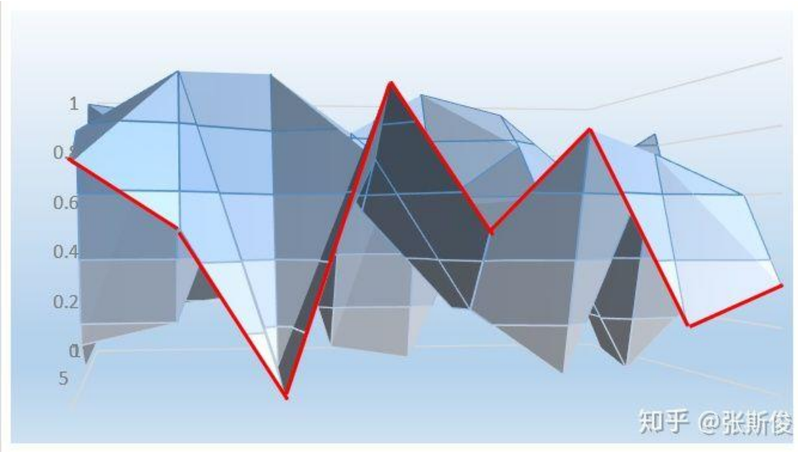


$a(a = \text{argmax}_a\, Q)$

Policy Net



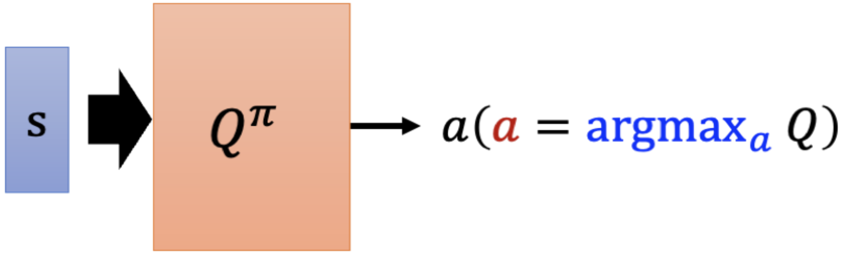
Q函数

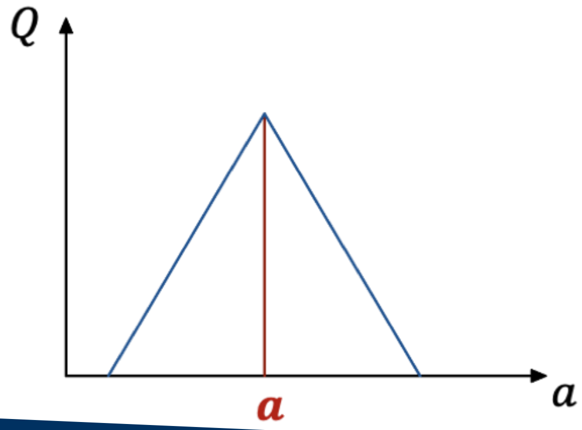


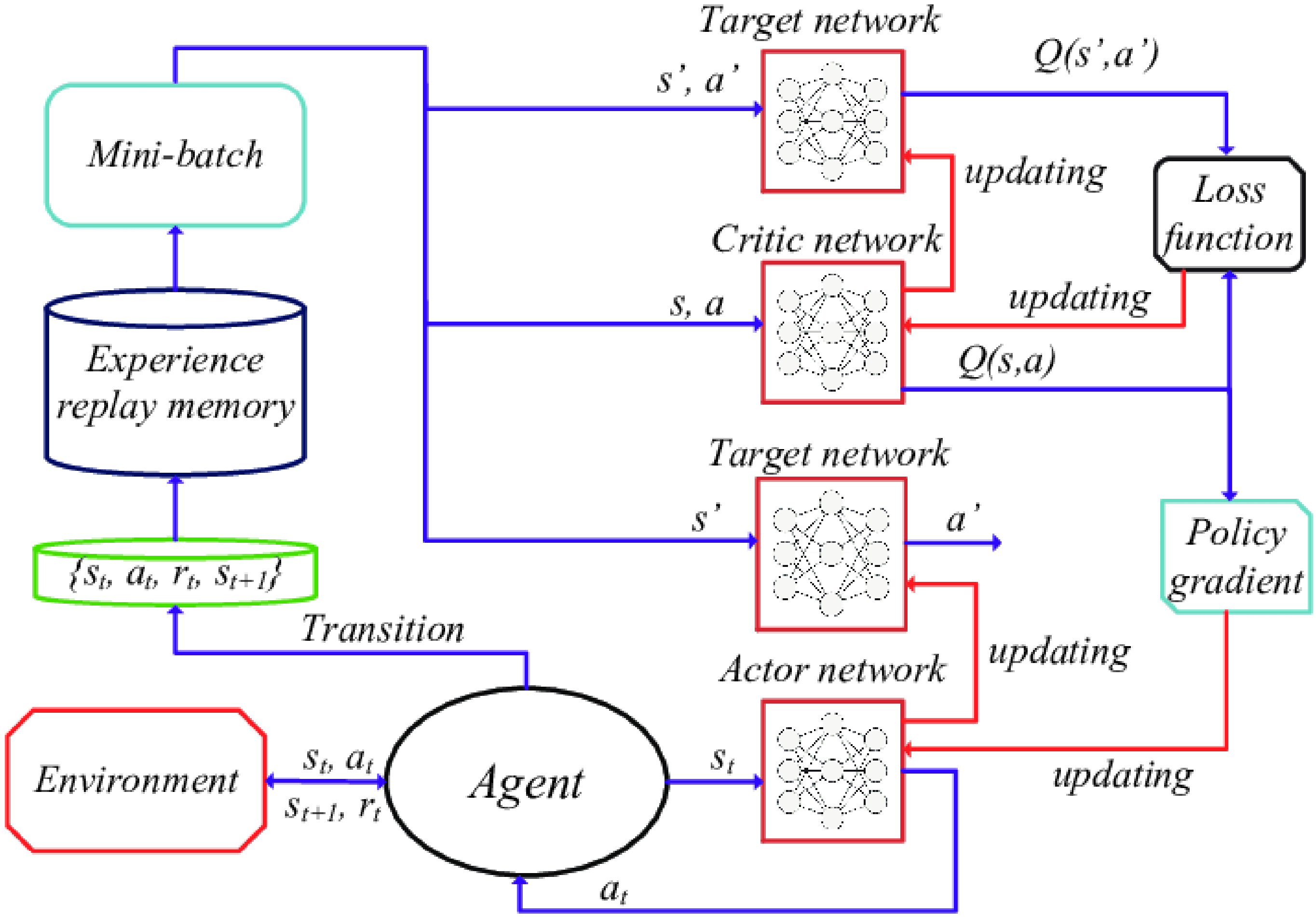
$Q(s, a)$

深度确定性策略梯度DDPG

- DDPG 相比于 DQN 主要是解决连续型动作的预测问题







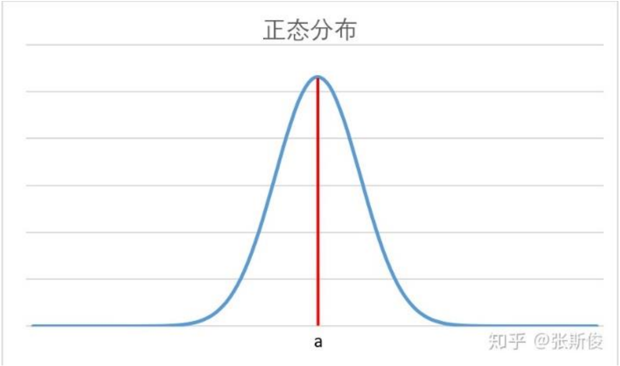
DDPG算法

1. Actor Online Networks：负责策略网络参数θ^{μ}的迭代更新，负责根据当前状态S选择当前动作A，用于和环境交互生成S', R。
2. Actor Target Networks：负责根据经验回放池中采样的下一状态S'选择最优下一动作A'。网络参数$\theta^{\mu'}$定期从θ^{μ}复制。
3. Critic Online Networks：负责价值网络参数w的迭代更新，负责计算负责计算当前Q值$Q(S, A, \theta^{Q})$。目标Q值$y_i = R + \gamma Q'(S', A', \theta^{Q'})$
4. Critic Target Networks：负责计算目标Q值中的$Q'(S', A', \theta^{Q'})$部分。网络参数$\theta^{Q'}$定期从θ^{Q}复制。

- 与DQN目标网络之间延迟复制现有网络不同的是，DDPG中采用soft update, 也就是缓慢地更新两个目标网络中的参数

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

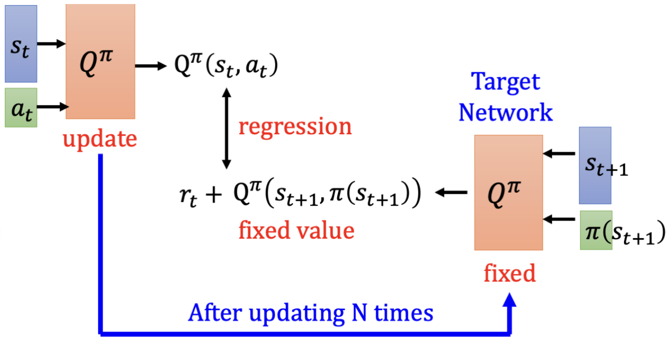
DDPG的一些要点



- 动作选择 $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$

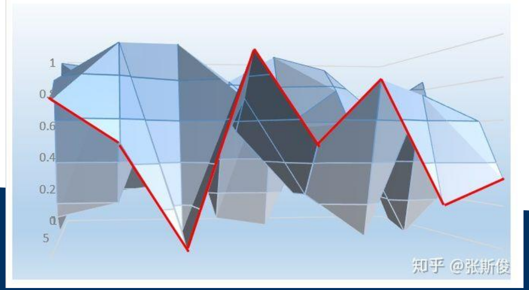
- Critic网络训练 $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$



- Actor网络训练 $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$

$$J(\theta^\mu) = \mathbb{E}[Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}]$$



DDPG算法

Algorithm 1 DDPG algorithm

Randomly initialize $\boxed{\text{critic network } Q(s,a|\theta^Q)}$ and $\boxed{\text{actor } \mu(s|\theta^\mu)}$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize $\boxed{\text{replay buffer } R}$
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $\boxed{a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t}$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition $\boxed{(s_t, a_t, r_t, s_{t+1})}$ in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \boxed{\gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})}$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\boxed{\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{aligned}}$$

  end for
end for

https://blog.csdn.net/qq_41871826