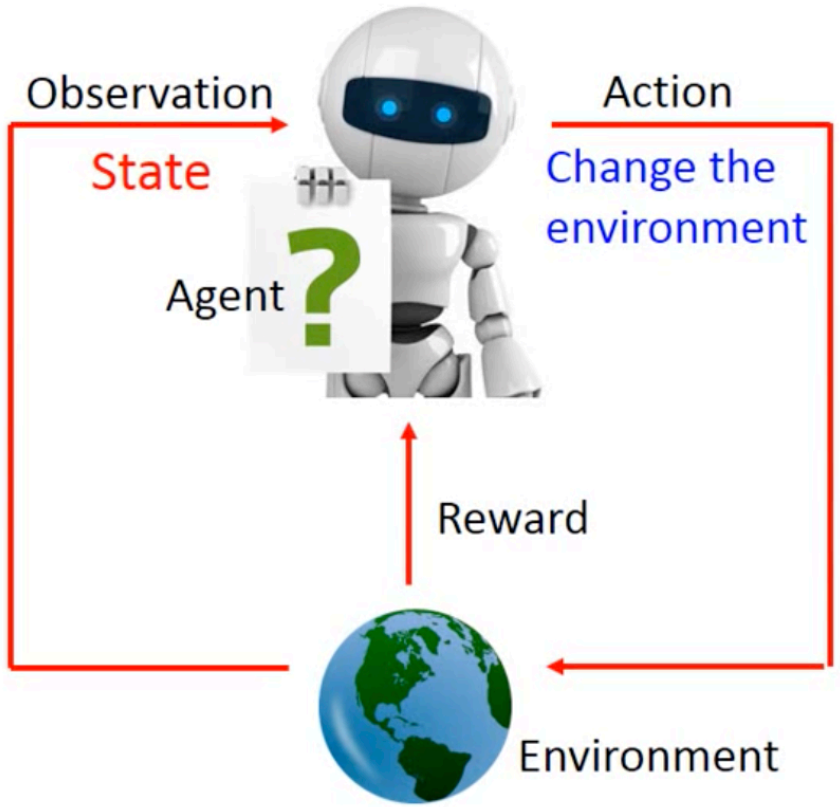
Reinforcement Learning

- ✓ Learning an Actor
- ✓ Learning an Critic
- ✓ Actor-Critic

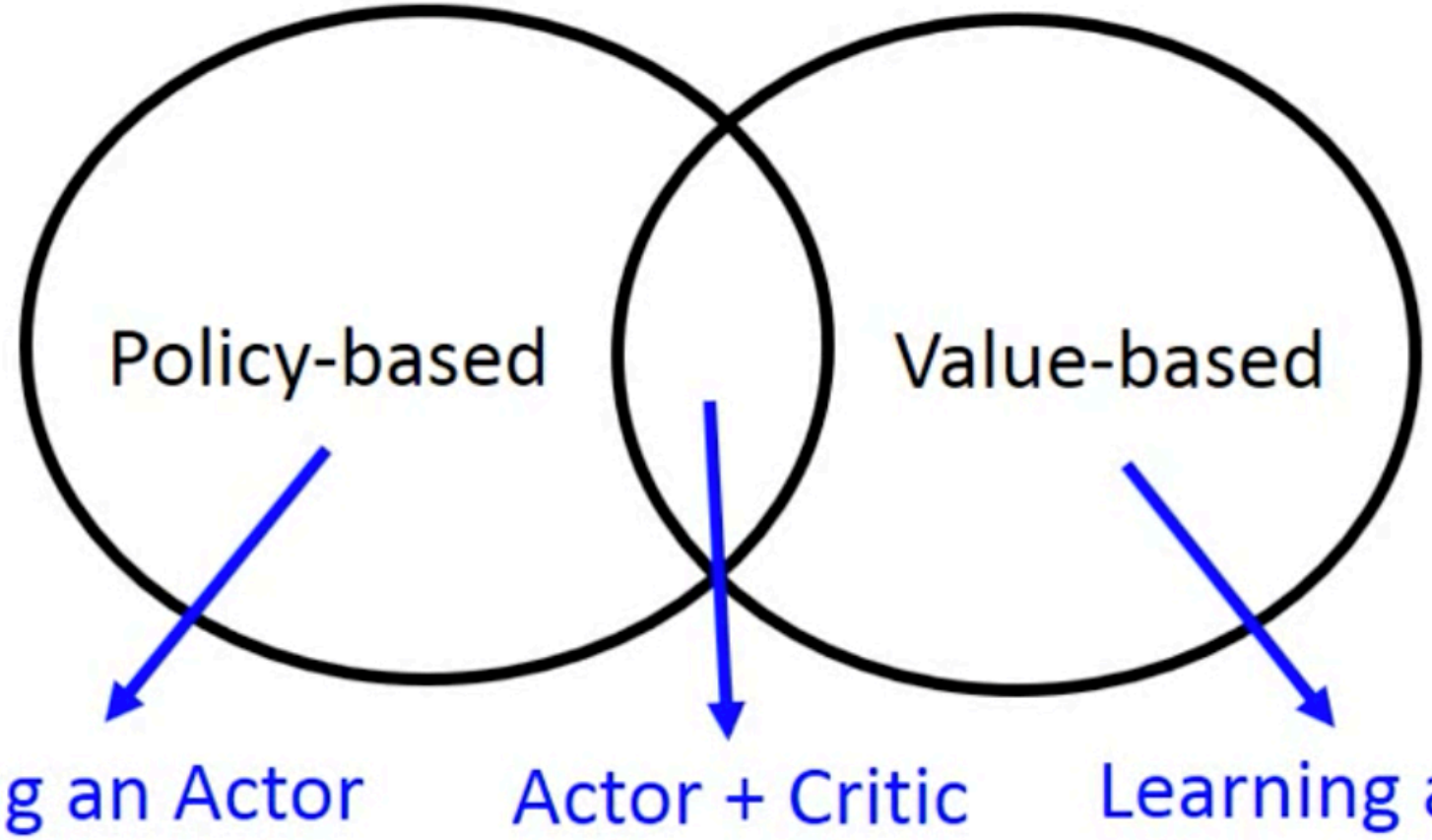
Scenario of Reinforcement Learning



Difficulties of Reinforcement Learning

- Reward delay
  - In space invader, only “fire” obtains reward
    - Although the moving before “fire” is important
  - In Go playing, it may be better to sacrifice immediate reward to gain more long-term reward
- Agent’s actions affect the subsequent data it receives
  - E.g. Exploration

Model free



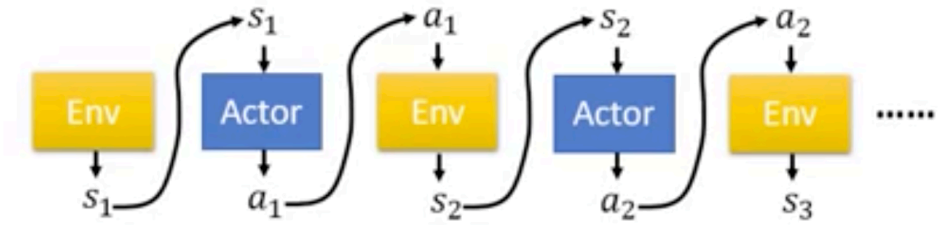
Asynchronous Advantage Actor-Critic (A3C)

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning", ICML, 2016

Learning an Actor——Policy Gradient

Policy Gradient

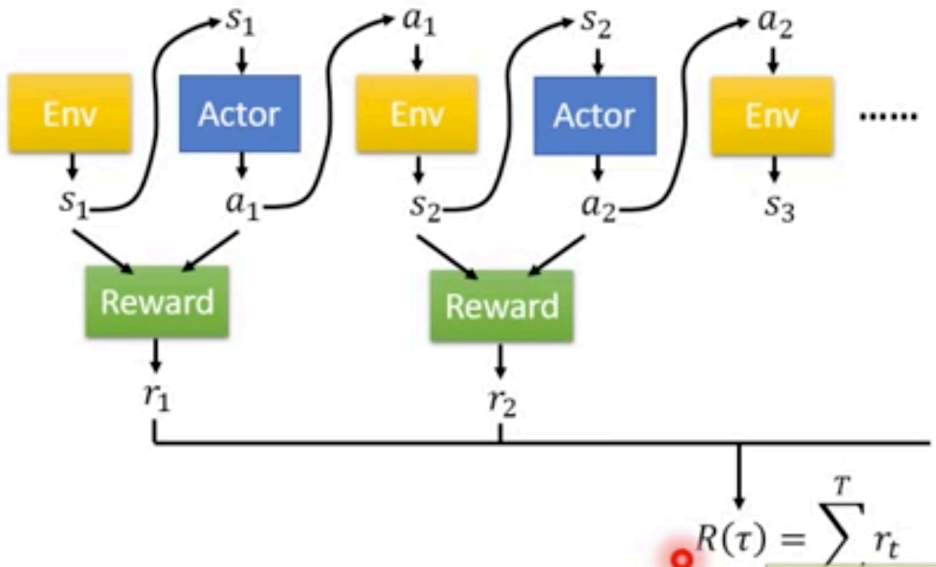
Actor, Environment, Reward



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

$$p_\theta(\tau)$$
$$= p(s_1)p_\theta(a_1|s_1)p(s_2|s_1,a_1)p_\theta(a_2|s_2)p(s_3|s_2,a_2)\cdots$$
$$= p(s_1)\prod_{t=1}^{T} p_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)$$

Actor, Environment, Reward

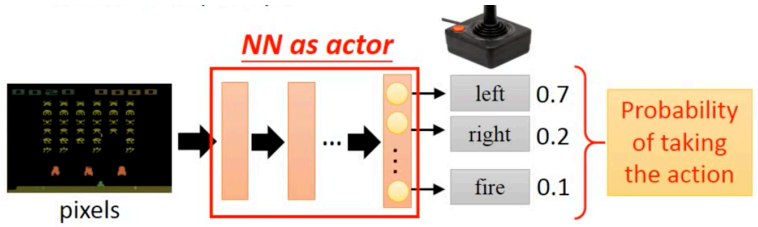


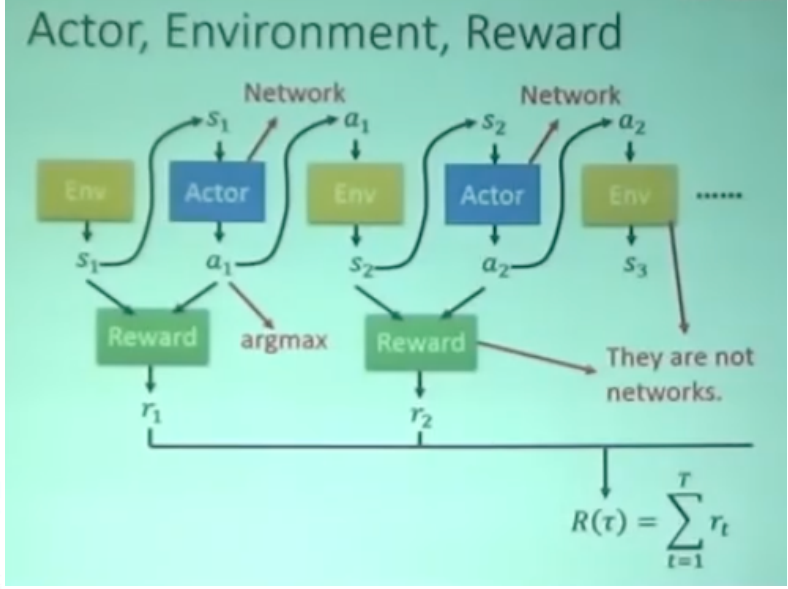
Expected Reward

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

Policy Gradient

Step1：Neural Network as Actor





- An episode is considered as a trajectory τ
  - $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$
  - $R(\tau) = \sum_{n=1}^{N} r_n$
  - If you use an actor to play the game, each τ has a probability to be sampled
    - The probability depends on actor parameter θ: $P(\tau|\theta)$

Step2：Goodness of Actor

大数定理

$$\bar{R}_\theta = \sum_\tau R(\tau) P(\tau|\theta) \approx \frac{1}{N} \sum_{n=1}^{N} R(\tau^n)$$

Sum over all possible trajectory

Use π_θ to play the game N times, obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$

Sampling τ from $P(\tau|\theta)$ N times

Policy Gradient

Step3：Pick the great function
Gradient Ascent
Part 1

Gradient Ascent

$$\bar{R}_\theta = \sum_\tau R(\tau)P(\tau|\theta) \quad \nabla\bar{R}_\theta = ?$$

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

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

$$= \sum_\tau R(\tau)P(\tau|\theta)\nabla logP(\tau|\theta) \qquad \frac{dlog(f(x))}{dx} = \frac{1}{f(x)}\frac{df(x)}{dx}$$

$$\approx \frac{1}{N}\sum_{n=1}^{N} R(\tau^n)\nabla logP(\tau^n|\theta)$$

Use π_θ to play the game N times,
Obtain $\{\tau^1, \tau^2, \cdots, \tau^N\}$

Gradient Ascent

$\nabla logP(\tau|\theta) = ?$

- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$

$$P(\tau|\theta) =$$

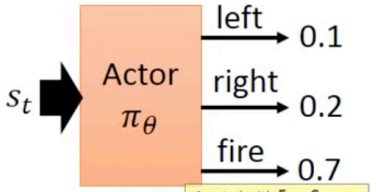
$$p(s_1)p(a_1|s_1,\theta)p(r_1,s_2|s_1,a_1)p(a_2|s_2,\theta)p(r_2,s_3|s_2,a_2)\cdots$$

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

not related to your actor

Control by your actor π_θ

$p(a_t = "fire"|s_t,\theta) = 0.7$



- $\tau = \{s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T\}$

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

$$logP(\tau|\theta)$$

$$= logp(s_1) + \sum_{t=1}^{T} logp(a_t|s_t,\theta) + logp(r_t,s_{t+1}|s_t,a_t)$$

$$\nabla logP(\tau|\theta) = \sum_{t=1}^{T} \nabla logp(a_t|s_t,\theta)$$

Ignore the terms not related to θ

Step3：Pick the great function

Gradient Ascent

Part 2

Gradient Ascent

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

$$\nabla log P(\tau|\theta) = \sum_{t=1}^{T} \nabla log p(a_t|s_t,\theta)$$

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

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

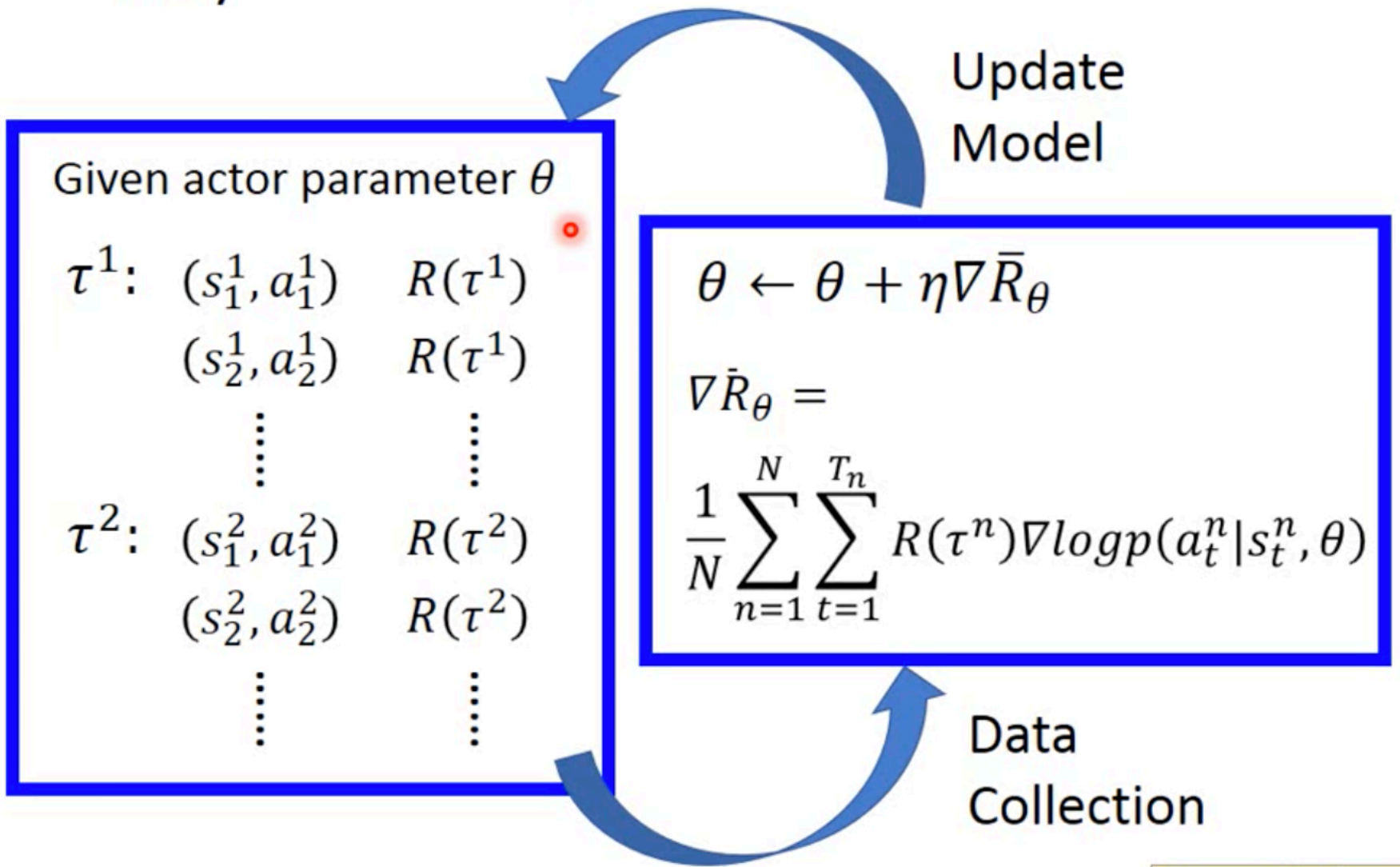
If in τ^n machine takes a_t^n when seeing s_t^n in

$R(\tau^n)$ is positive ⇒ Tuning θ to increase $p(a_t^n|s_t^n)$

$R(\tau^n)$ is negative ⇒ Tuning θ to decrease $p(a_t^n|s_t^n)$

It is very important to consider the cumulative reward $R(\tau^n)$ of the whole trajectory τ^n instead of immediate reward r_t^n

Policy Gradient



Policy Gradient

减去baseline 使得reward有正有负

Tip 1: Add a Baseline

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

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

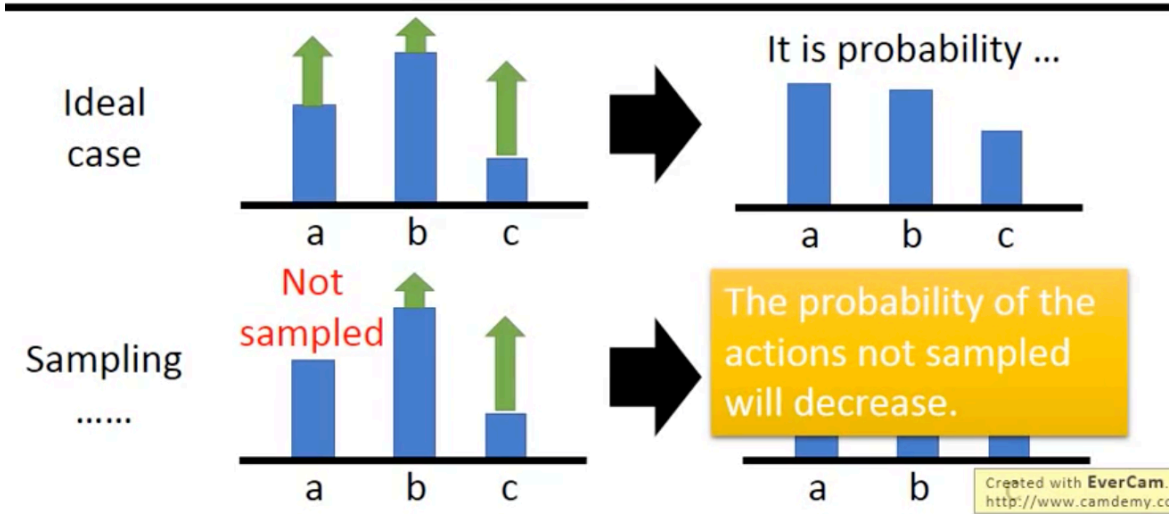
$$\nabla \bar{R}_\theta \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (R(\tau^n) - b) \nabla log p_\theta(a_t^n | s_t^n) \qquad b \approx E[R(\tau)]$$

Add a Baseline

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

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

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



Policy Gradient

Tip 2: Assign Suitable Credit

$\times 3$	$\times 3$	$\times 3$	$\times -7$	$\times -7$	$\times -7$
(s_a, a_1)	(s_b, a_2)	(s_c, a_3)	(s_a, a_2)	(s_b, a_2)	(s_c, a_3)
+5	+0	-2	-5	+0	-2

$R = +3$ $\quad R = -7$

$\times 3$	$\times -2$	$\times -2$	$\times -7$	$\times -2$	$\times -2$
(s_a, a_1)	(s_b, a_2)	(s_c, a_3)	(s_a, a_2)	(s_b, a_2)	(s_c, a_3)
+5	+0	-2	-5	+0	-2

$R = +3$ $\quad R = -7$

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

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

Add discount factor $\quad \gamma < 1$

Advantage Function $A^\theta(s_t, a_t)$

How good it is if we take a_t other than other actions at s_t.

Estimated by "*critic*" (later)

Can be state-dependent

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

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

Add discount factor $\quad \gamma < 1$

On-policy vs Off-policy

On-policy v.s. Off-policy

- 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.

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

- 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.

Policy gradient中要花很多时间进行采样（因为更新一次策略参数之后之前的样本就失效了，需要重新采样新的策略下的样本）

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 $q(x)$

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

修正两个distribution之间的差异

On-policy vs Off-policy

两个distribution之间的差异较大时，会有较大的方差

Issue of Importance Sampling

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

$Var_{x\sim p}[f(x)]$ $Var_{x\sim q}[f(x)\frac{p(x)}{q(x)}]$

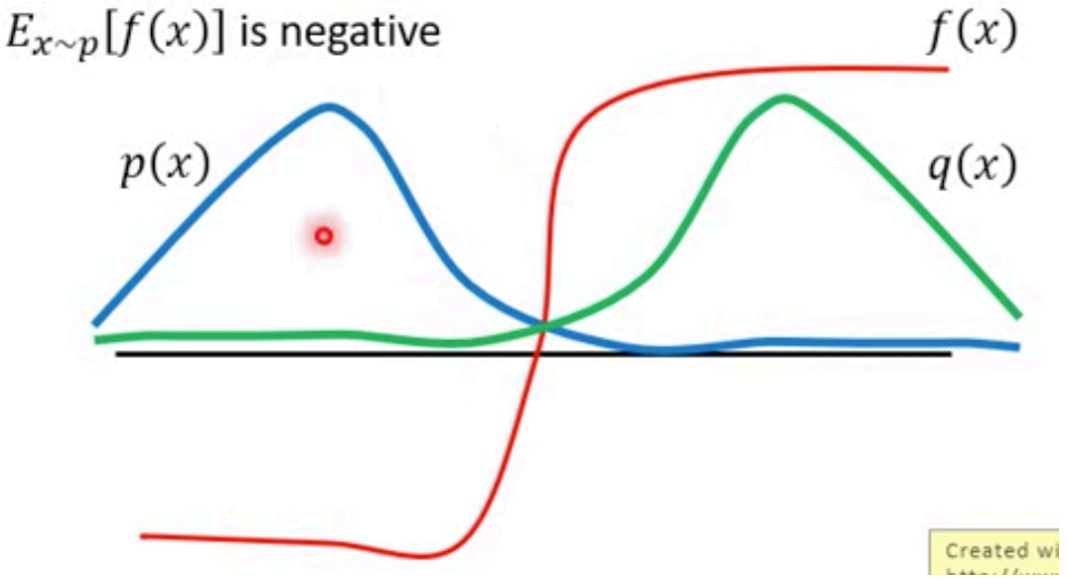
$$VAR[X] = E[X^2] - (E[X])^2$$

$$Var_{x\sim p}[f(x)] = E_{x\sim p}[f(x)^2] - \left(E_{x\sim p}[f(x)]\right)^2$$

$$Var_{x\sim q}[f(x)\frac{p(x)}{q(x)}] = E_{x\sim q}\left[\left(f(x)\frac{p(x)}{q(x)}\right)^2\right] - \left(E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]\right)^2$$

$$= E_{x\sim p}\left[f(x)^2\frac{p(x)}{q(x)}\right] - \left(E_{x\sim p}[f(x)]\right)^2$$

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



On-policy vs Off-policy

On-policy → Off-policy

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

- 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.

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

- Sample the data from θ'.
- Use the data to train θ many times.

Gradient for update

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

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

$A^{\theta'}(s_t,a_t)$ This term is from sampled data.

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

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

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

PPO / TRPO

为了避免两个分布差太多
需要增加constraint

PPO / TRPO

Proximal Policy Optimization (PPO)

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

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

衡量两个分布的相似度
计算action之间的距离

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

TRPO (Trust Region Policy Optimization)

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

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

原理类似，但是PPO比TRPO应用起来更简单
KL是计算action的距离

PPO

PPO algorithm

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

- 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 $J_{PPO}(\theta)$

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

Update parameters several times

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

Adaptive KL Penalty

KL值太大 说明惩罚项不够

PPO / PPO2

PPO algorithm

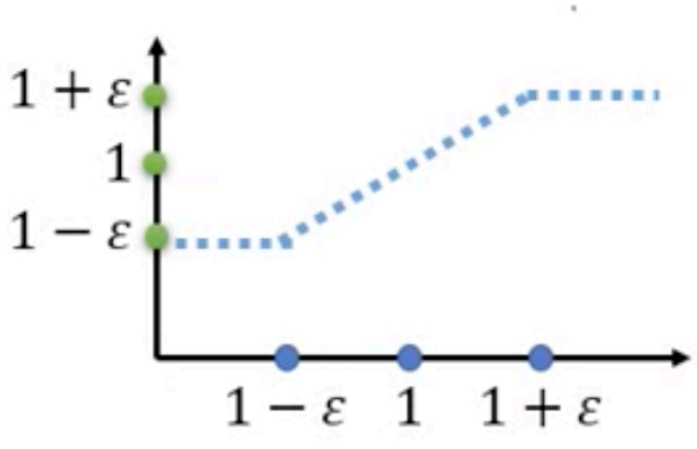
$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k)$$

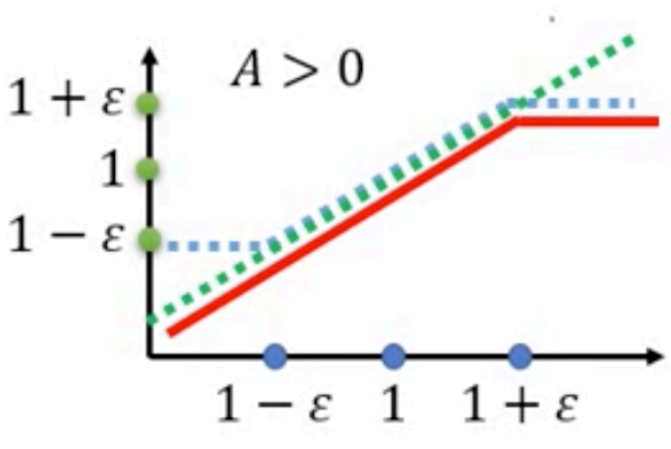
$$J^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)} \frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t)$$

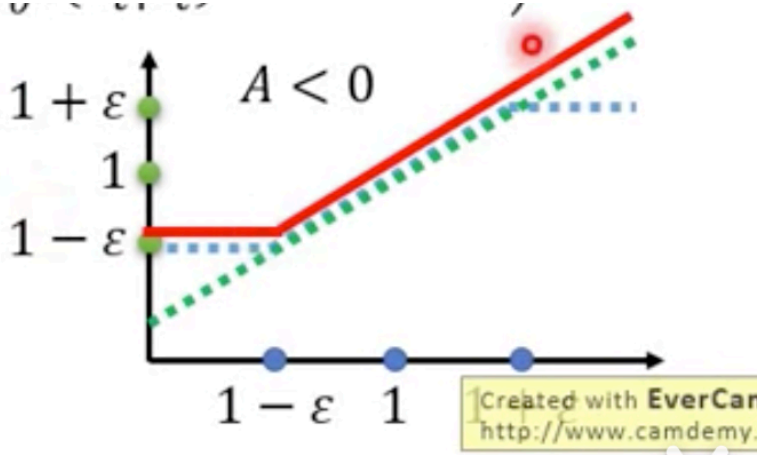
PPO2 algorithm

$$J_{PPO2}^{\theta^k}(\theta) \approx \sum_{(s_t,a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)} A^{\theta^k}(s_t, a_t),\right.$$

$$\left. clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta^k}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta^k}(s_t, a_t)\right)$$







Learning a Critic——Three kinds of Critics

Critic evaluates how good the actor is.

Critic——状态价值函数V

- State value function $V^{\pi}(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after visiting state s

How to estimate $V^{\pi}(s)$

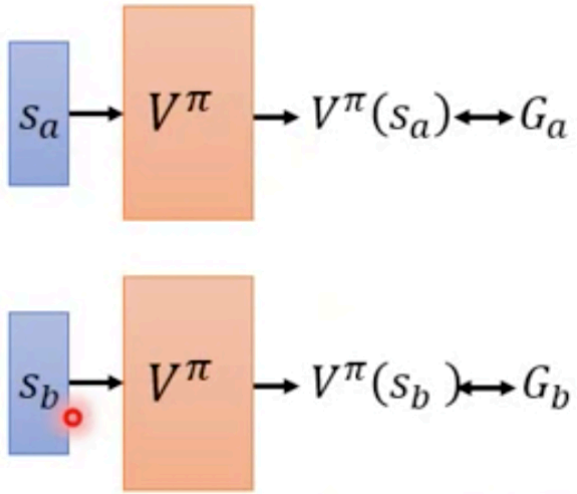
- **Monte-Carlo (MC) based approach**
  - The critic watches π playing the game

After seeing s_a,

Until the end of the episode, the cumulated reward is G_a

After seeing s_b,

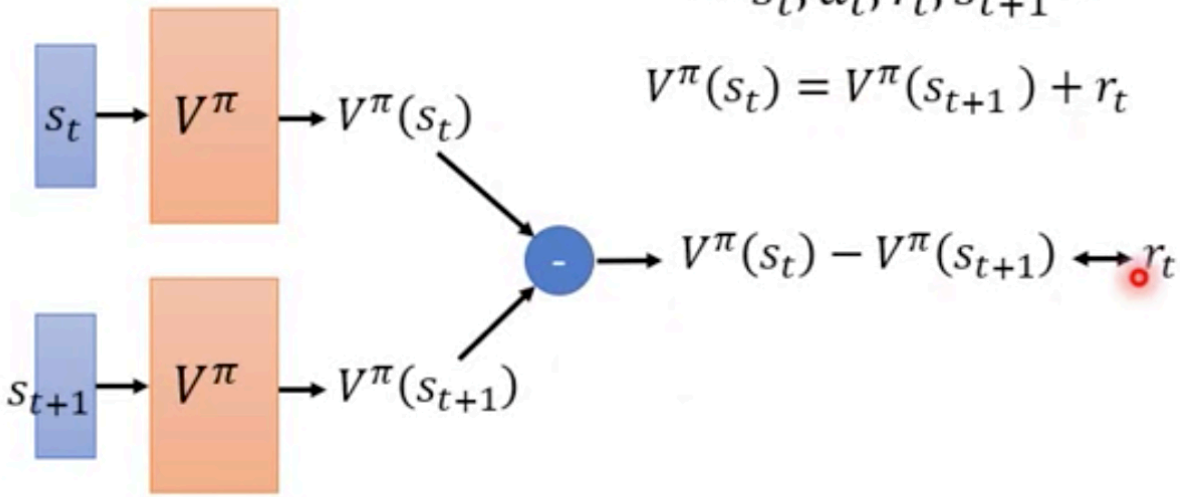
Until the end of the episode, the cumulated reward is G_b



- **Temporal-difference (TD) approach**

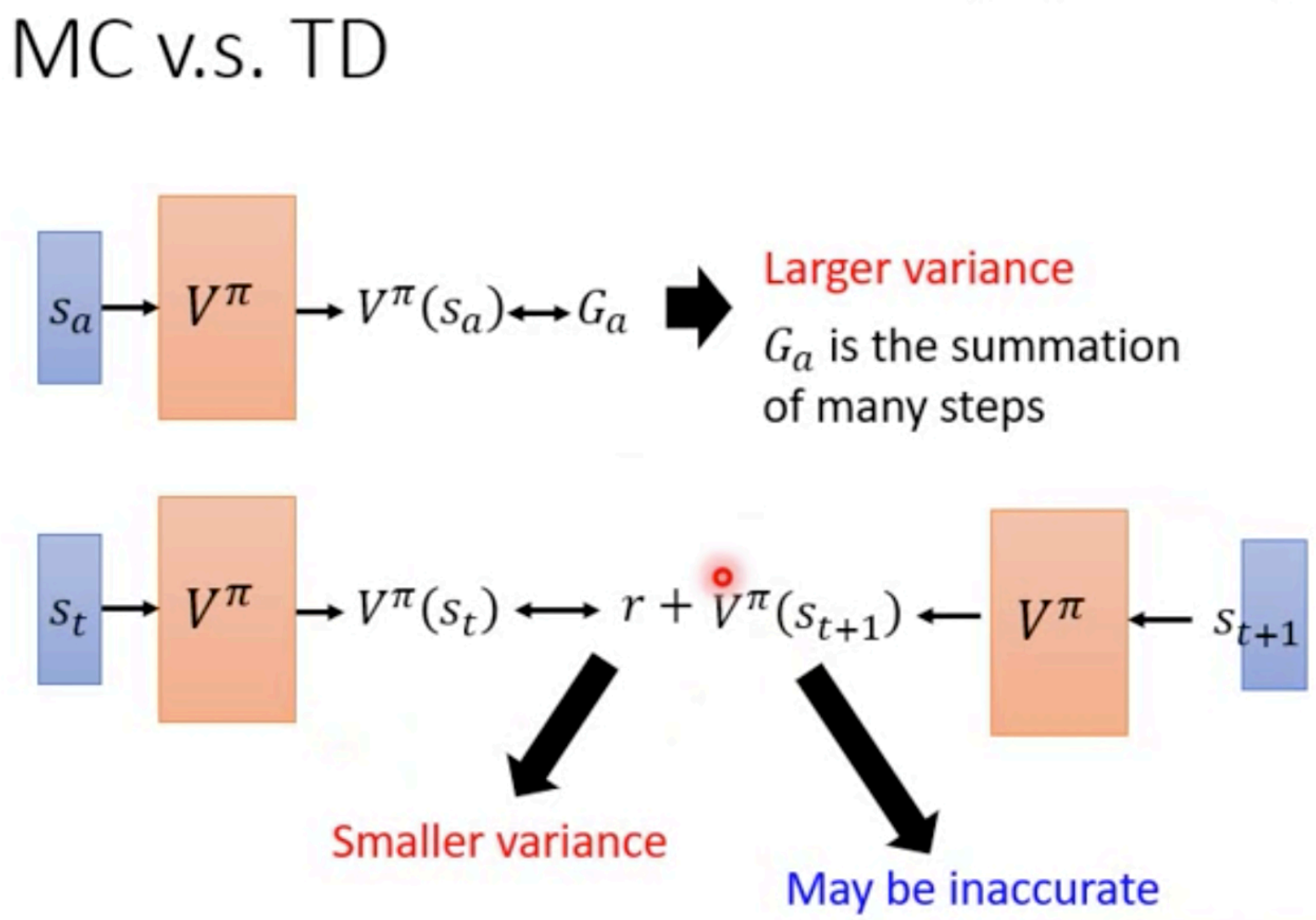
$$\cdots s_t, a_t, r_t, s_{t+1} \cdots$$

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



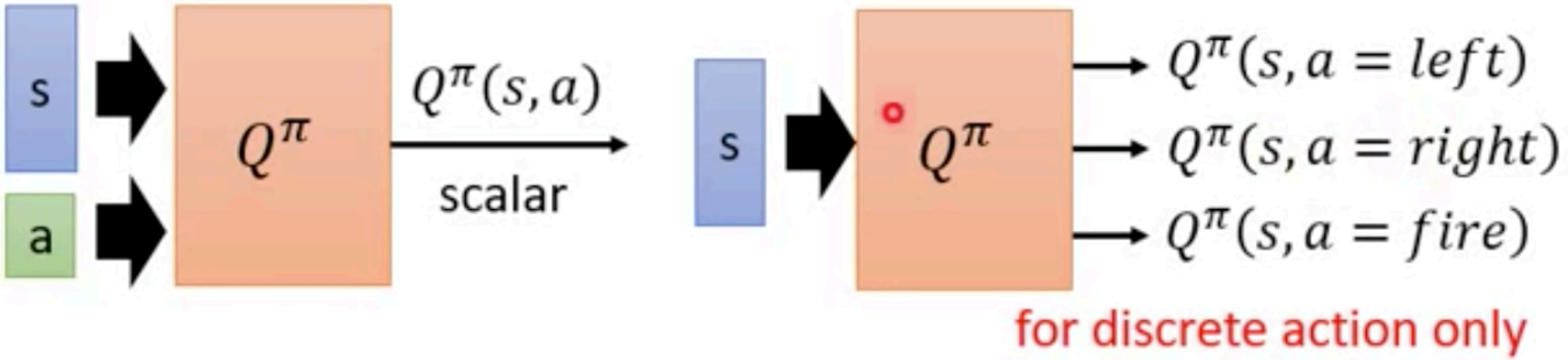
Some applications have very long episodes, so that delaying all learning until an episode's end is too slow.

Critic——状态价值函数V



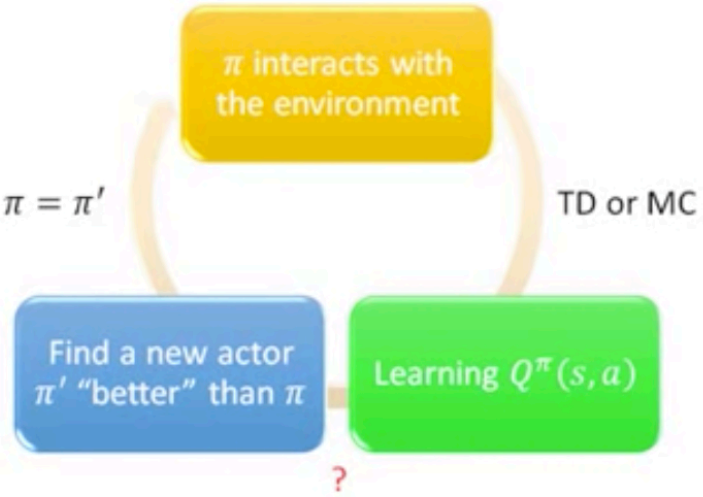
Critic——状态-动作价值函数Q

- State-action value function $Q^{\pi}(s,a)$
  - When using actor π, the *cumulated* reward expects to be obtained after taking a at state s



Critic——Q-Learning

Q-Learning



- Given $Q^{\pi}(s,a)$, find a new actor π' "better" than π
  - "Better": $V^{\pi'}(s) \geq V^{\pi}(s)$, for all state s

$$\pi'(s) = arg \max_{a} Q^{\pi}(s,a)$$

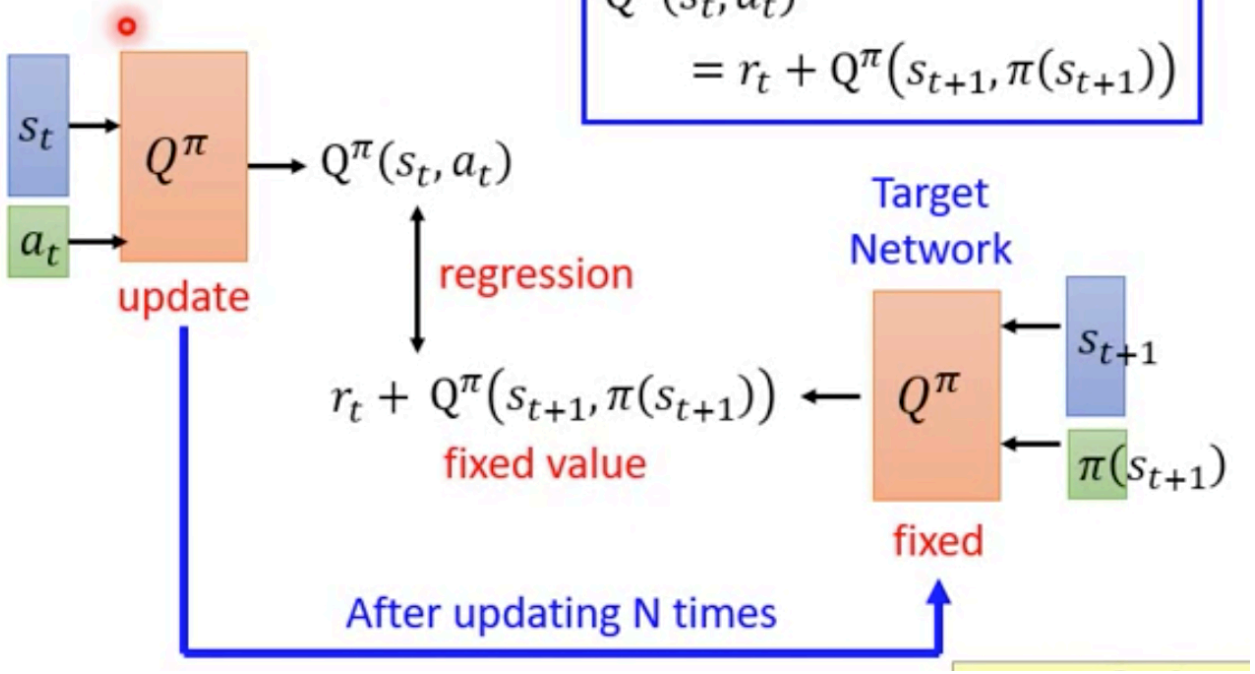
- π' does not have extra parameters. It depends on Q
- Not suitable for continuous action a (solve it later)

Critic——Q-Learning Tips

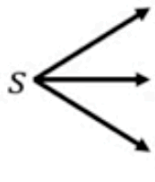
Target Network

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

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



Exploration



$Q(s,a) = 0$ Never explore

$Q(s,a) = 1$ Always sampled

$Q(s,a) = 0$ Never explore

- The policy is based on Q-function

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

This is not a good way for data collection.

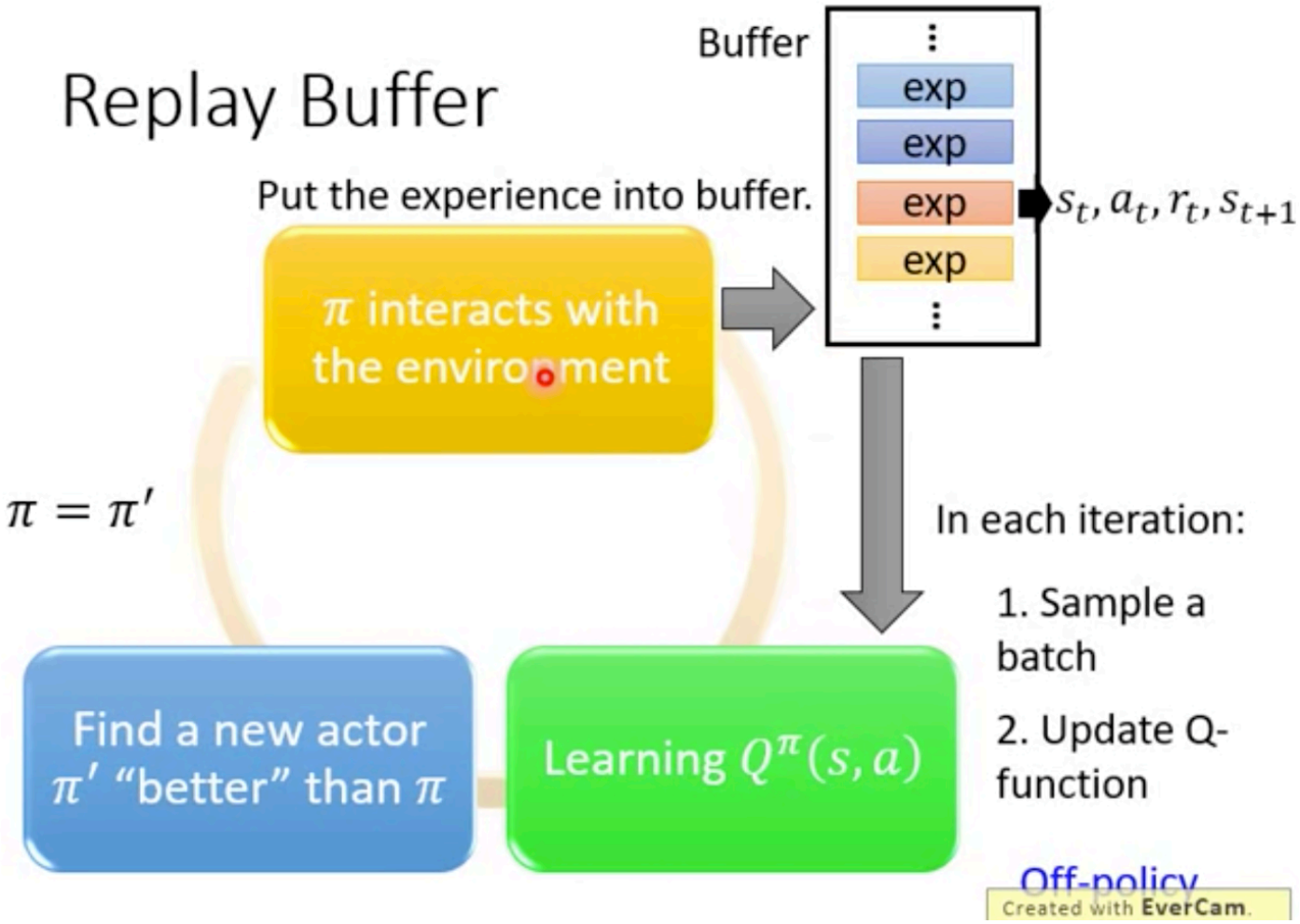
Epsilon Greedy ε would decay during learning

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

Boltzmann Exploration

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

Critic——Q-Learning Tips



Critic——Typical Q-Learning Algorithm

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 butter
    - Sample (s_i, a_i, r_i, s_{i+1}) from butter (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 $\hat{Q} = Q$

DQN Variants——Double DQN

- Q value is usually over estimate

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

Tend to select the action that is over-estimated

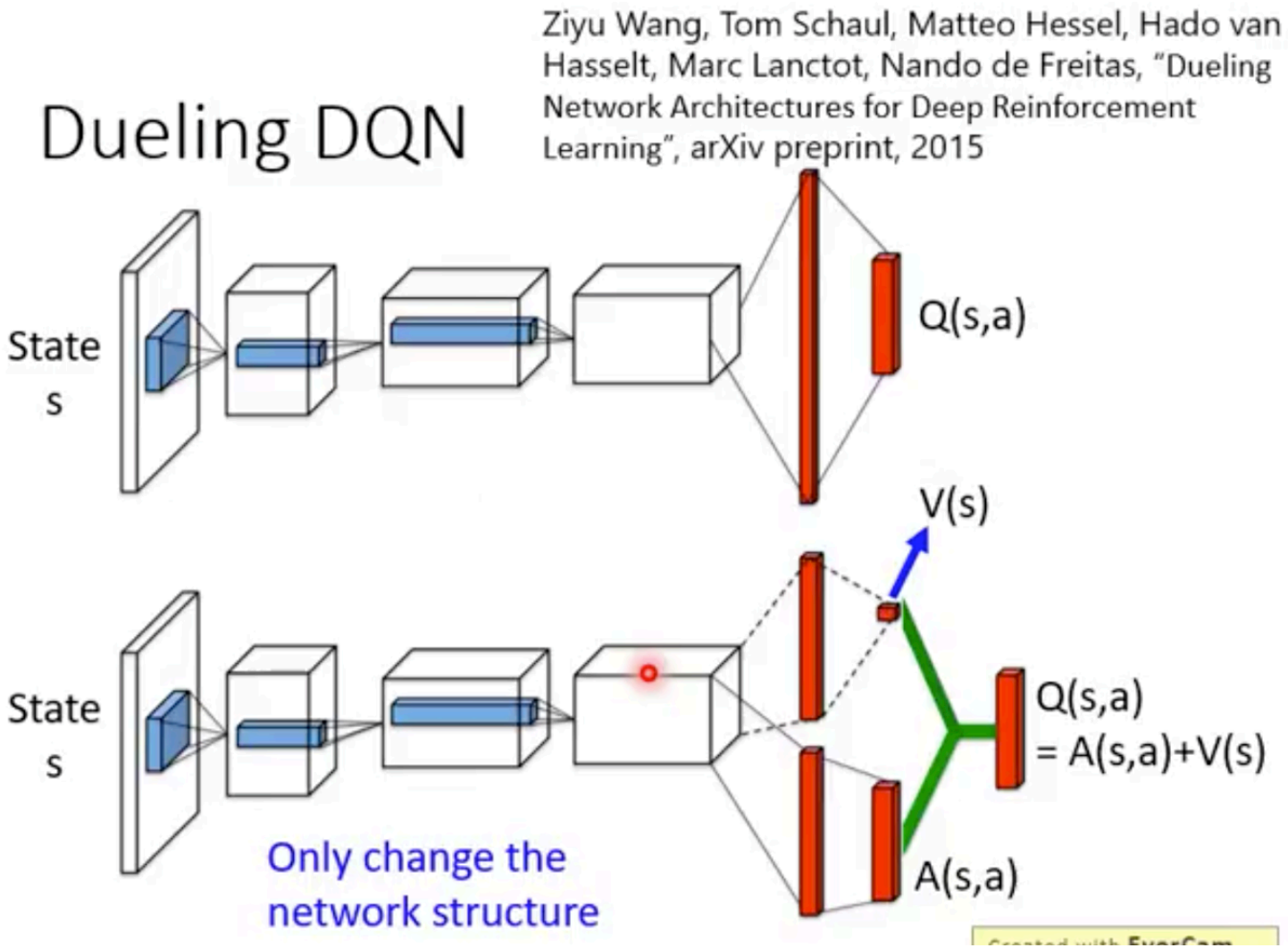
- Double DQN: two functions Q and Q' Target Network

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

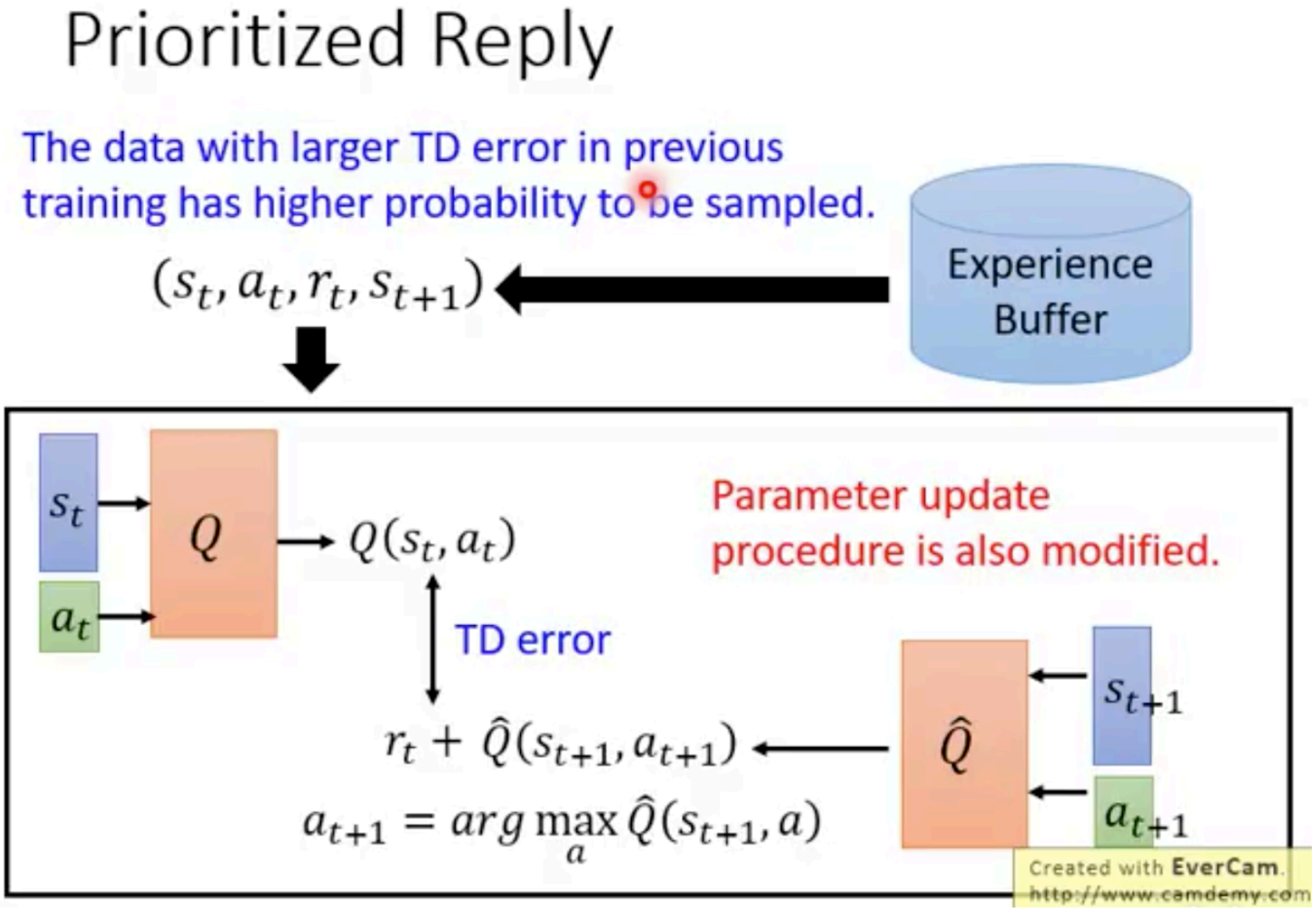
If Q over-estimate a, so it is selected. Q' would give it proper value.

How about Q' overestimate? The action will not be selected by Q.

DQN Variants——Dueling DQN



DQN Variants——DQN with Prioritized Replay



Q-Learning for Continuous Actions

Continuous Actions

- Action a is a *continuous vector*

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

Solution 1

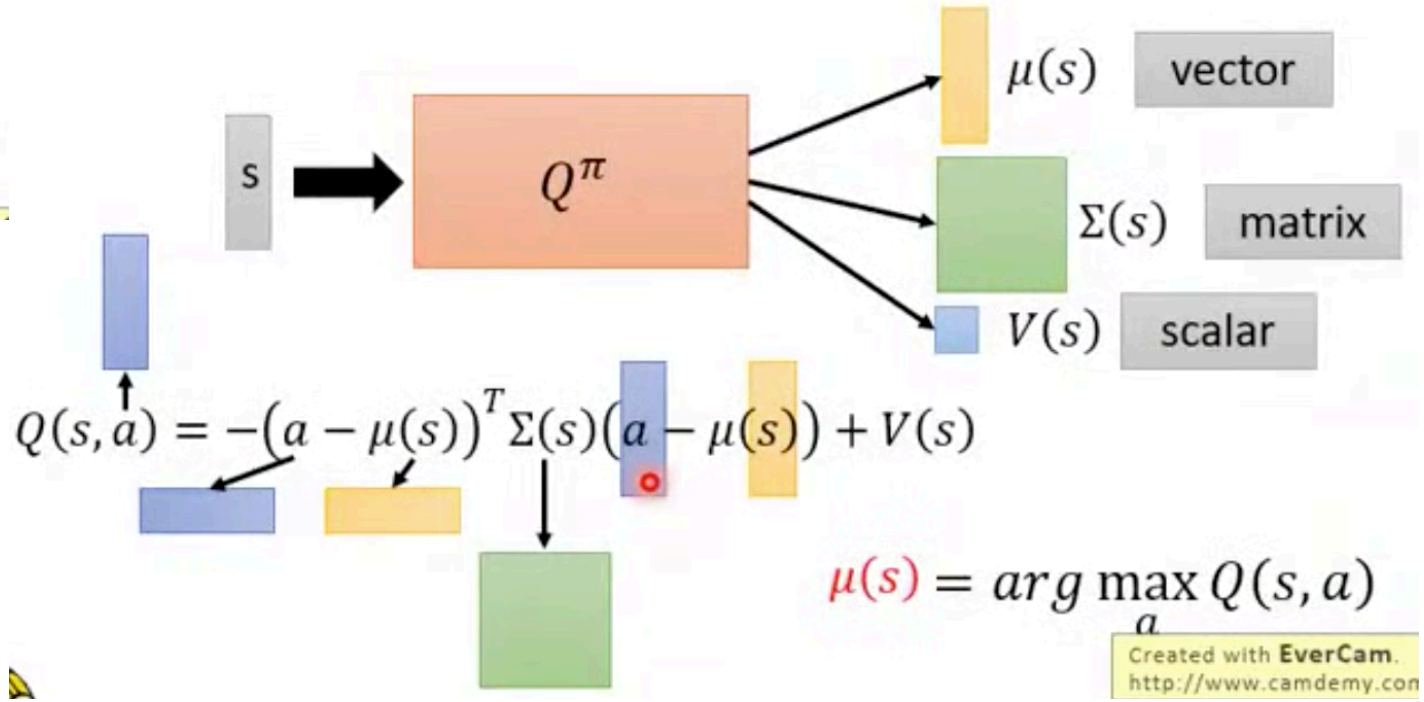
Sample a set of actions: $\{a_1, a_2, \cdots, a_N\}$

See which action can obtain the largest Q value

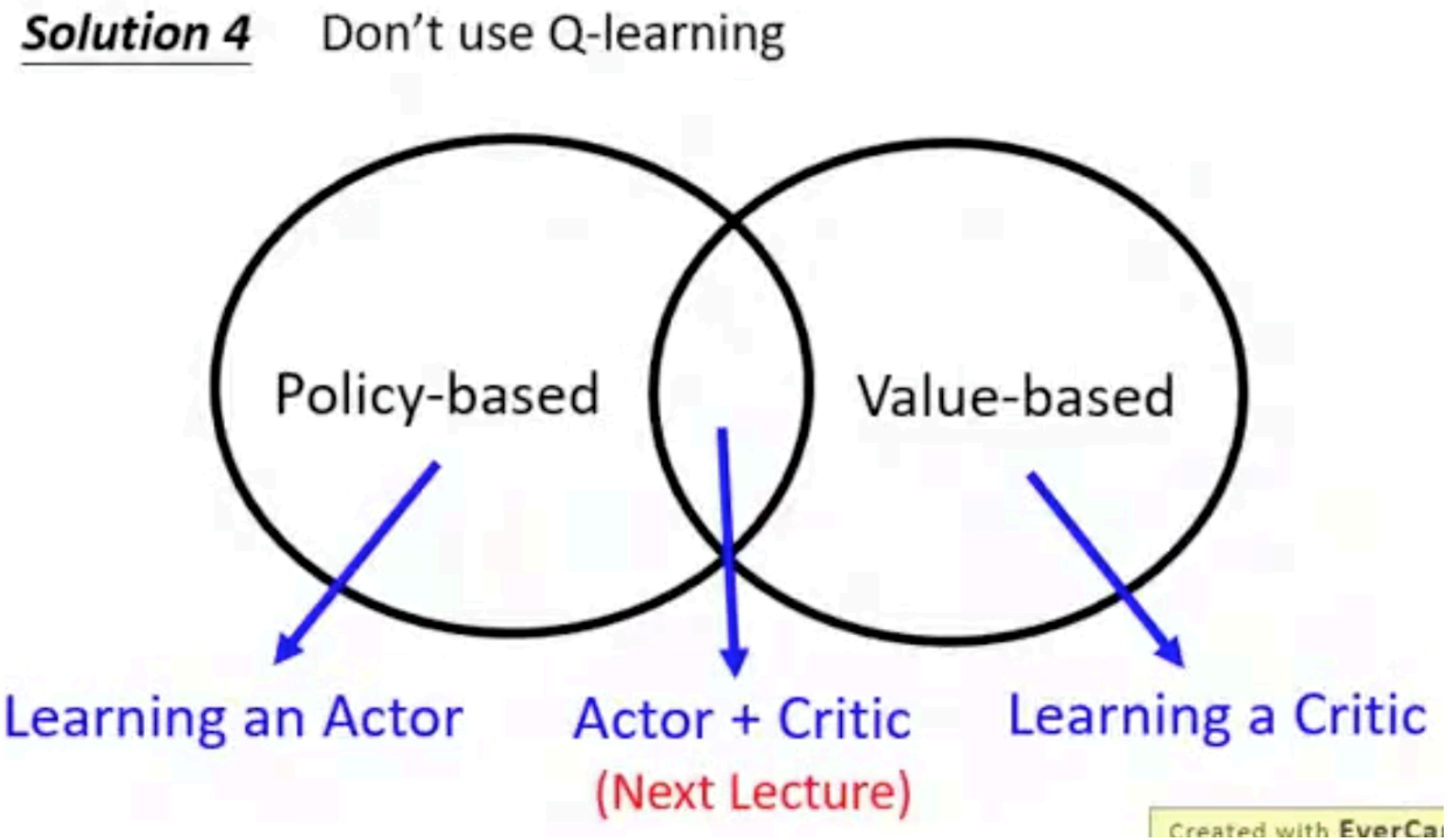
Solution 2

Using gradient ascent to solve the optimization problem.

Solution 3 Design a network to make the optimization easy.



Q-Learning for Continuous Actions



Actor-Critic

A3C、A2C

Review of Actor and Critic

Review – Policy Gradient

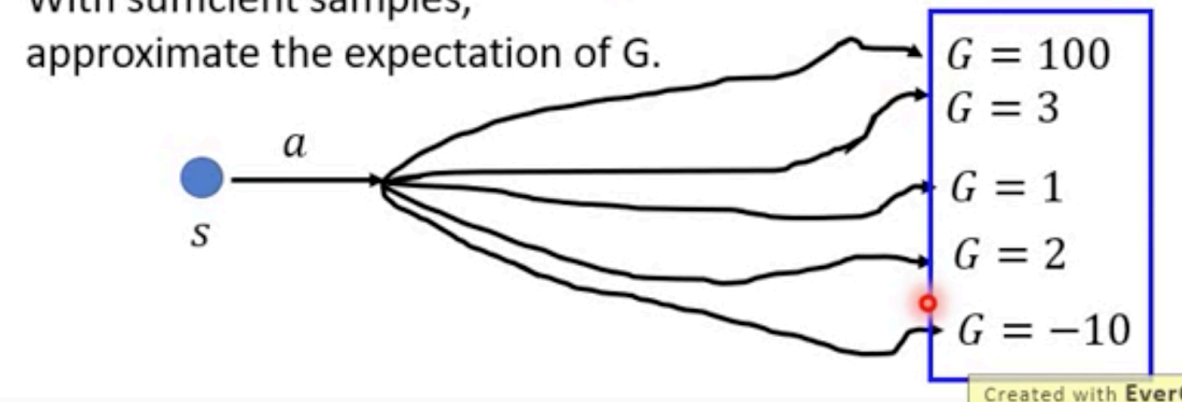
$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'=t}^{T_n}\gamma^{t'-t}r_{t'}^n - b\right)\nabla log p_\theta(a_t^n|s_t^n)$$

baseline

G_t^n : obtained via interaction

Very unstable

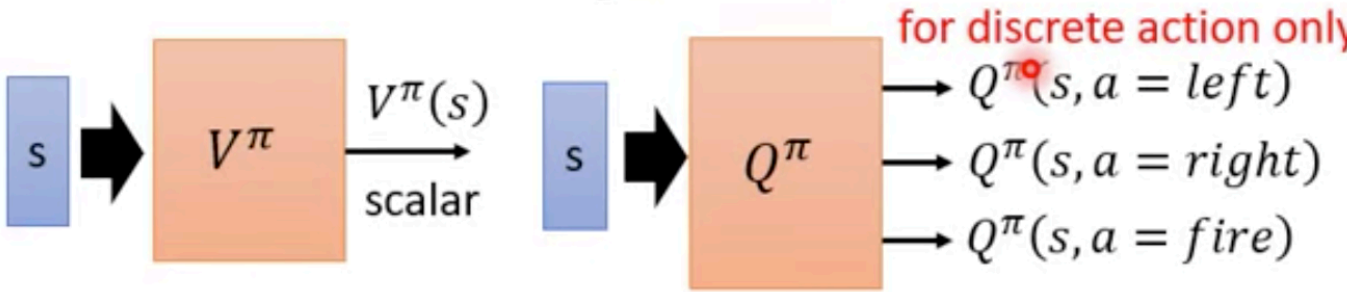
With sufficient samples, approximate the expectation of G.



可能会有很大的方差

Review – Q-Learning

- State value function $V^\pi(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after visiting state s
- State-action value function $Q^\pi(s,a)$
  - When using actor π, the *cumulated* reward expects to be obtained after taking a at state s



Estimated by TD or MC

Actor-Critic

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

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

baseline

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

G_t^n : obtained via interaction

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

Advantage Actor-Critic

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

Estimate two networks? We can only estimate one.

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

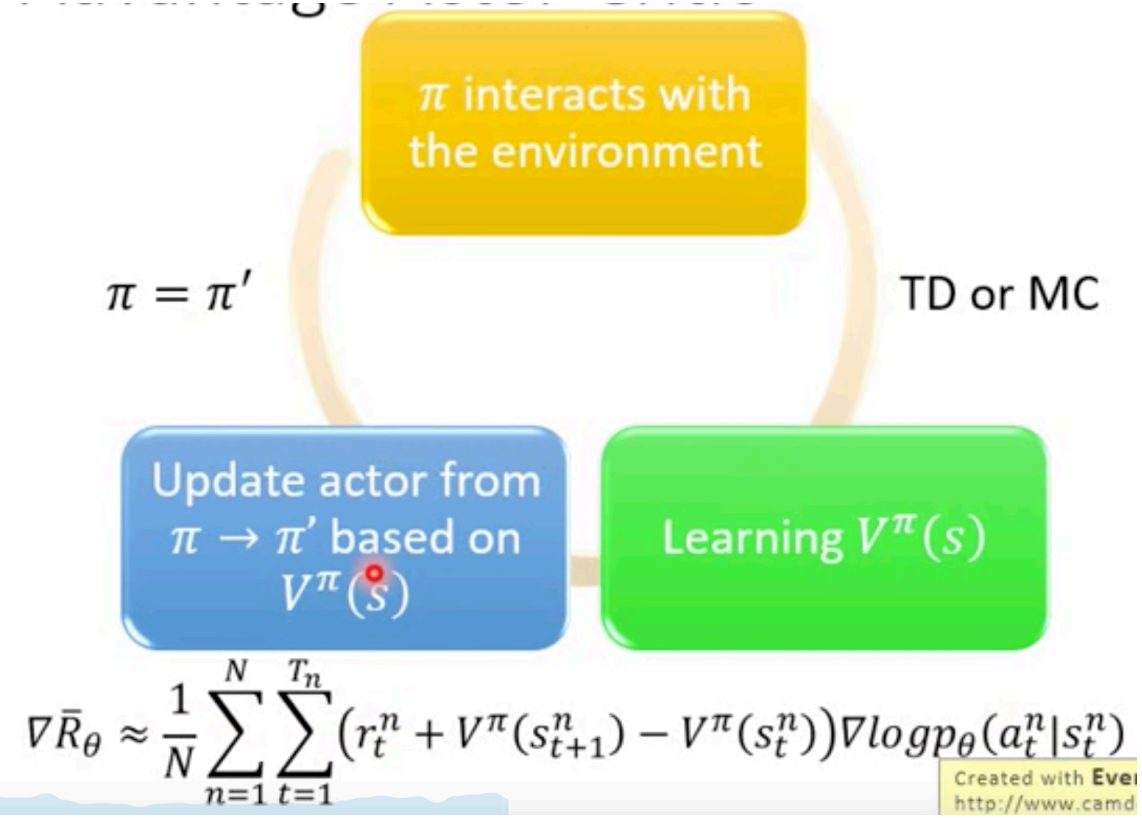
Only estimate state value

A little bit variance

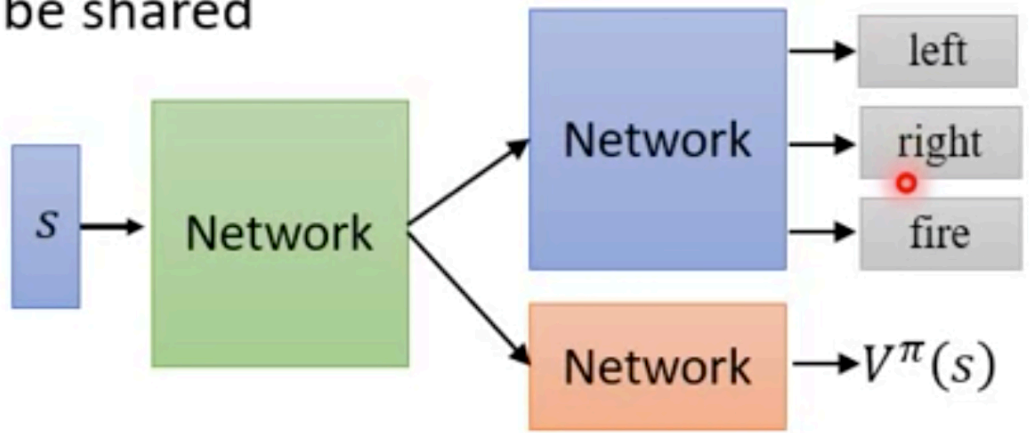
$$Q^{\pi}(s_t^n, a_t^n) = E[r_t^n + V^{\pi}(s_{t+1}^n)]$$

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

Actor-Critic



- Tips
  - The parameters of actor $\pi(s)$ and critic $V^{\pi}(s)$ can be shared



  - Use output entropy as regularization for $\pi(s)$
    - Larger entropy is preferred → exploration

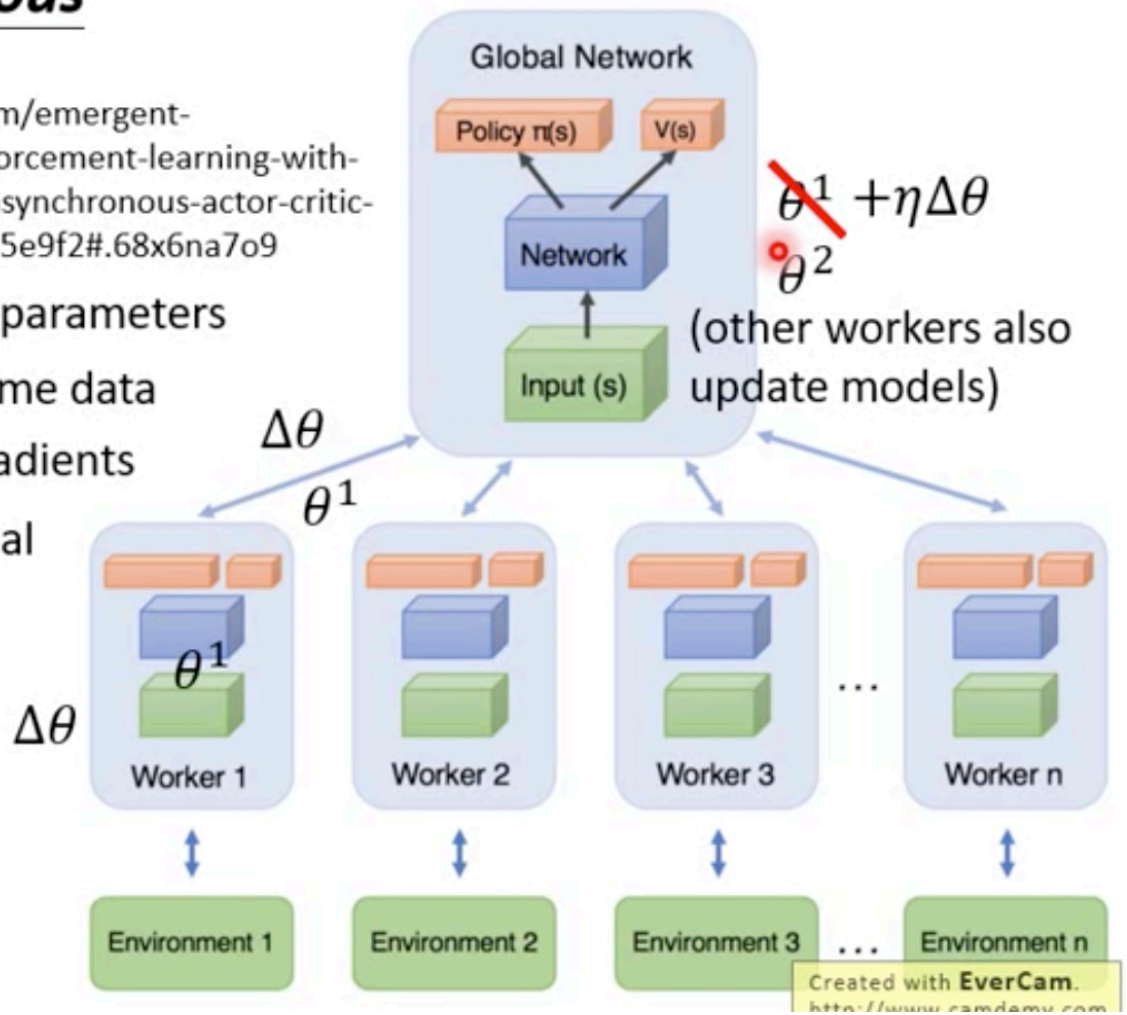
不同的action被采用的几率平均一些
在训练的时候会多尝试不同的action

Actor-Critic——A3C

Asynchronous

Source of image: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2#.68x6na7o9

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



Actor-Critic——Pathwise Derivative Policy Gradient

Q-Learning Algorithm ➡ ***Pathwise Derivative Policy Gradient***

- 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 ~~Q~~ 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 $y = r_i + \max_a \hat{Q}(s_{i+1}, a)$ ~~$\hat{Q}(s_{i+1}, a)$~~ $\hat{Q}\left(s_{i+1}, \hat{\pi}(s_{i+1})\right)$
- Update the parameters of Q to make $Q(s_i, a_i)$ close to y (regression)

3
- Update the parameters of π to maximize $Q\left(s_i, \pi(s_i)\right)$
- Every C steps reset $\hat{Q} = Q$

4
- Every C steps reset $\hat{\pi} = \pi$

Sparse Reward Problem