CSCE 636 Neural Networks (Deep Learning)

Lecture 14: Deep Reinforcement Learning (continued)

Anxiao (Andrew) Jiang

Based on the interesting lecture of Prof. Hung-yi Lee "Deep Reinforcement Learning"
https://www.youtube.com/watch?v=z95ZYgPgXOY

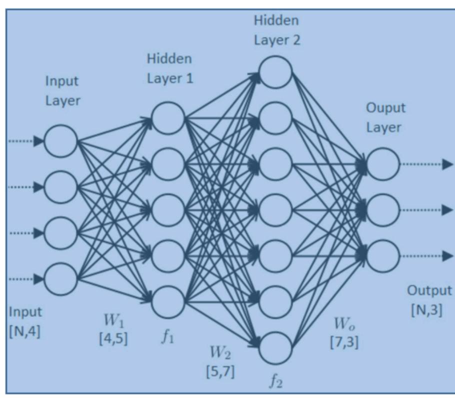
Policy-based Approach
(a.k.a. Policy Gradient Approach)

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

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

negative Cross entropy

hurt leg

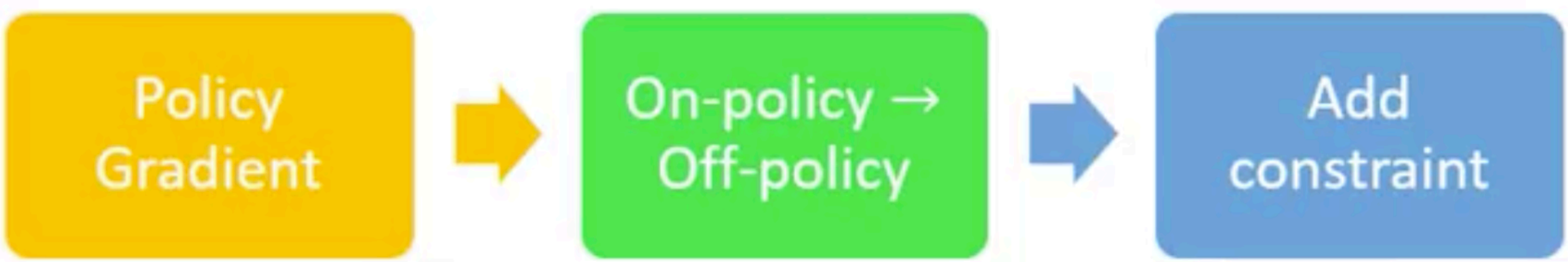


Actor

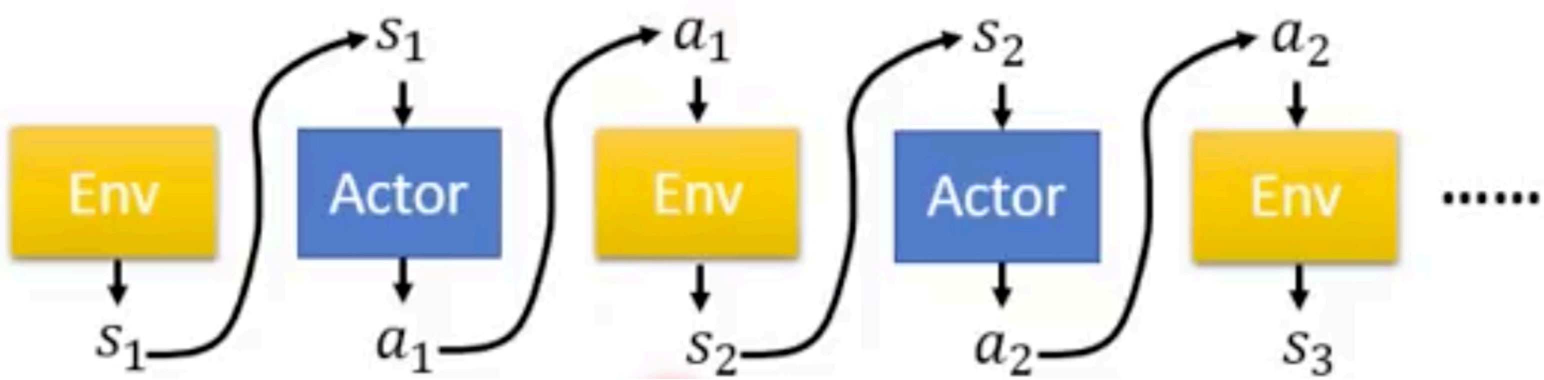
		Target output
train hard	0.4	1
train less hard	0.3	0
give up	0.3	0

Proximal Policy Optimization (PPO)

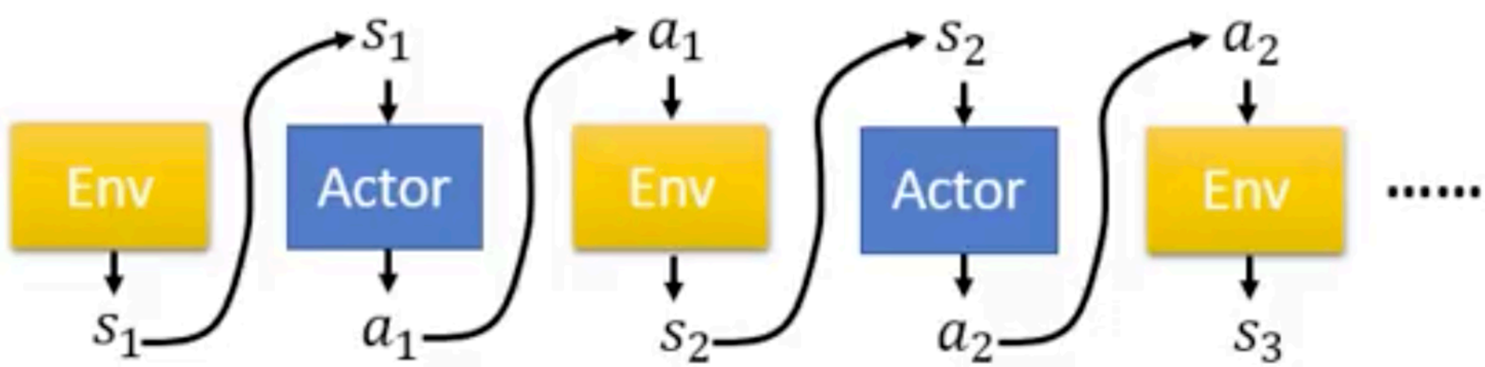
default reinforcement learning algorithm at OpenAI



Actor, Environment, Reward



Actor, Environment, Reward



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

$p_\theta(\tau)$

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

Policy Gradient

Given policy π_θ

$$
\begin{array}{lll}
\tau^1: & (s_1^1, a_1^1) & R(\tau^1) \\
 & (s_2^1, a_2^1) & R(\tau^1) \\
 & \vdots & \vdots \\
\tau^2: & (s_1^2, a_1^2) & R(\tau^2) \\
 & (s_2^2, a_2^2) & R(\tau^2) \\
 & \vdots & \vdots
\end{array}
$$

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

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

Policy Gradient

Given policy π_θ

$$
\begin{array}{lll}
\tau^1: & (s_1^1, a_1^1) & R(\tau^1) \\
 & (s_2^1, a_2^1) & R(\tau^1) \\
 & \vdots & \vdots \\
\tau^2: & (s_1^2, a_1^2) & R(\tau^2) \\
 & (s_2^2, a_2^2) & R(\tau^2) \\
 & \vdots & \vdots
\end{array}
$$

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

$$\nabla \bar{R}_\theta =$$

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

Data Collection

Policy Gradient

Given policy π_θ

$$
\begin{array}{lll}
\tau^1: & (s_1^1, a_1^1) & R(\tau^1) \\
 & (s_2^1, a_2^1) & R(\tau^1) \\
 & \vdots & \vdots \\
\tau^2: & (s_1^2, a_1^2) & R(\tau^2) \\
 & (s_2^2, a_2^2) & R(\tau^2) \\
 & \vdots & \vdots
\end{array}
$$

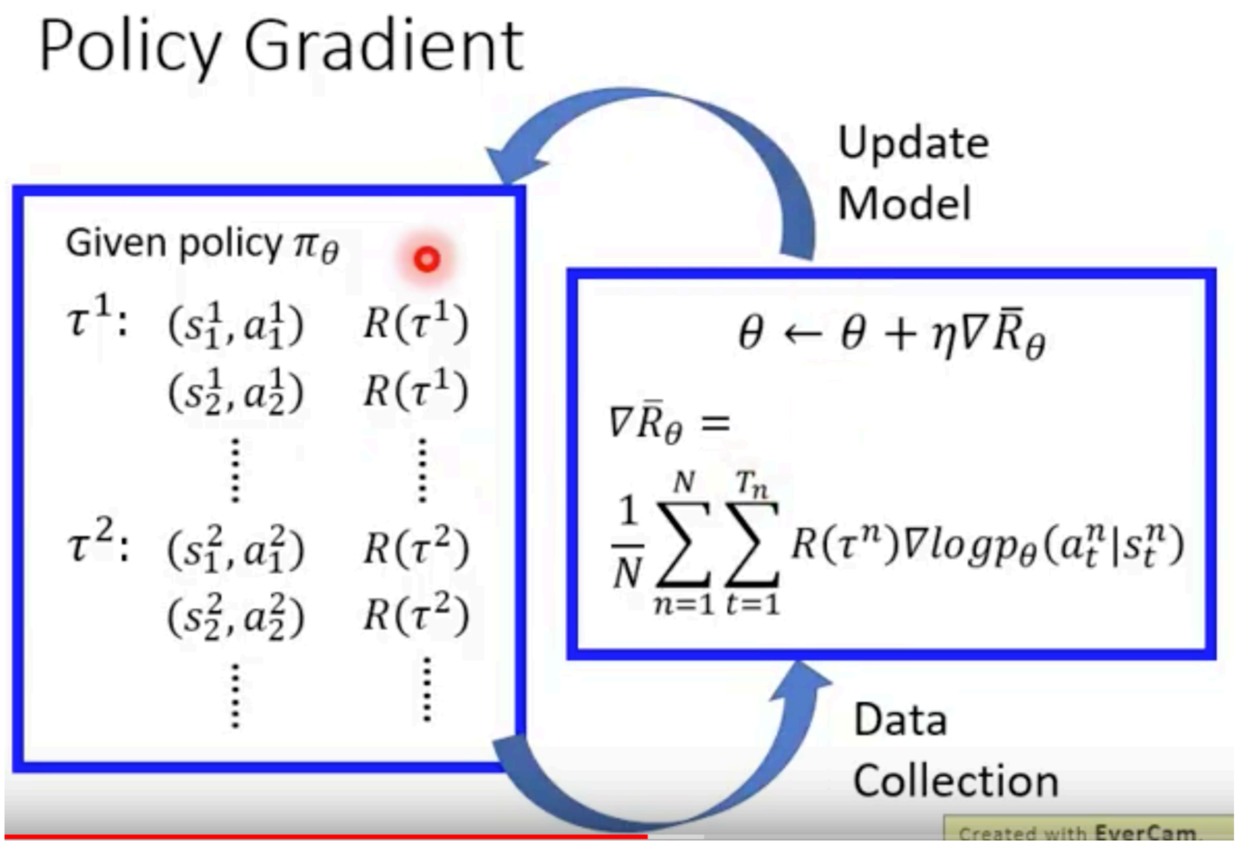
Update Model

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

$$\nabla \bar{R}_\theta =$$

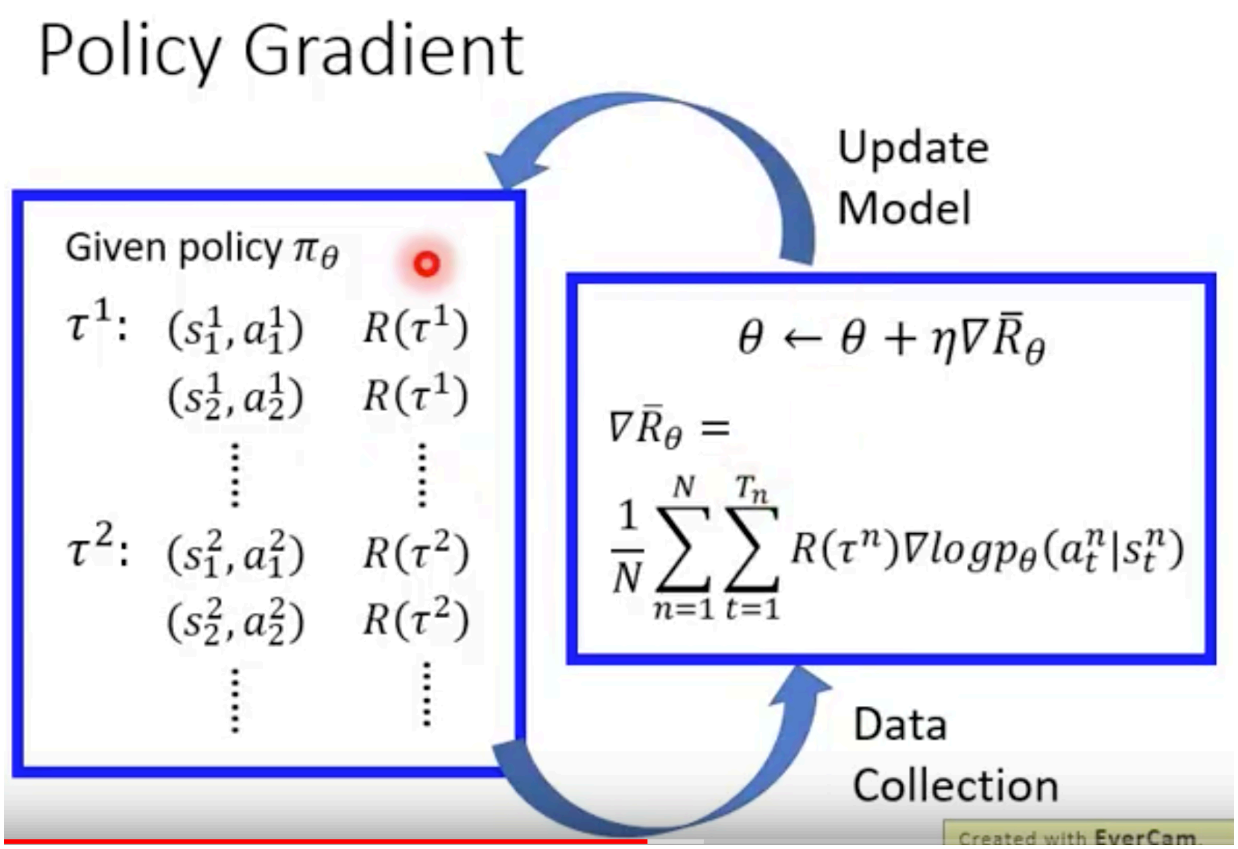
$$\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} R(\tau^n) \nabla log p_\theta(a_t^n | s_t^n)$$

Data Collection



Rigorously speaking, we should use the above data only once (to update the NN's weights only once).

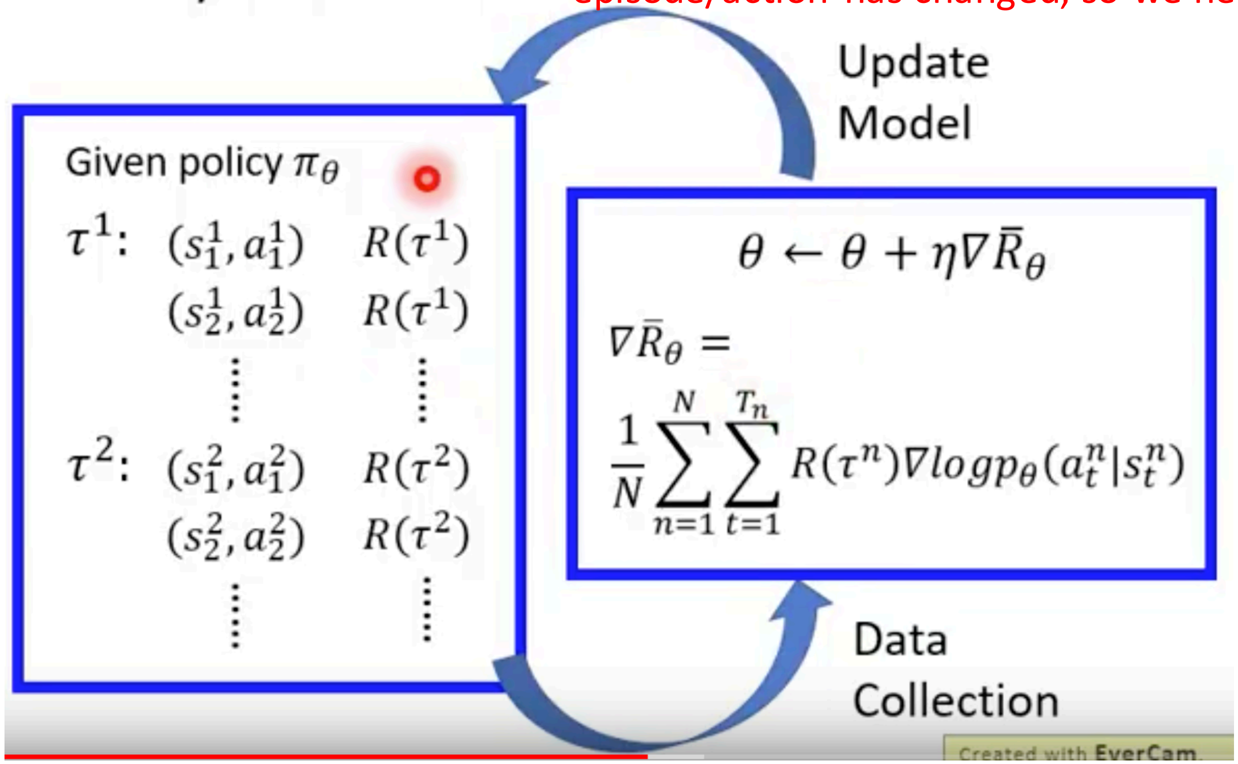
Then we need to collect data AGAIN!



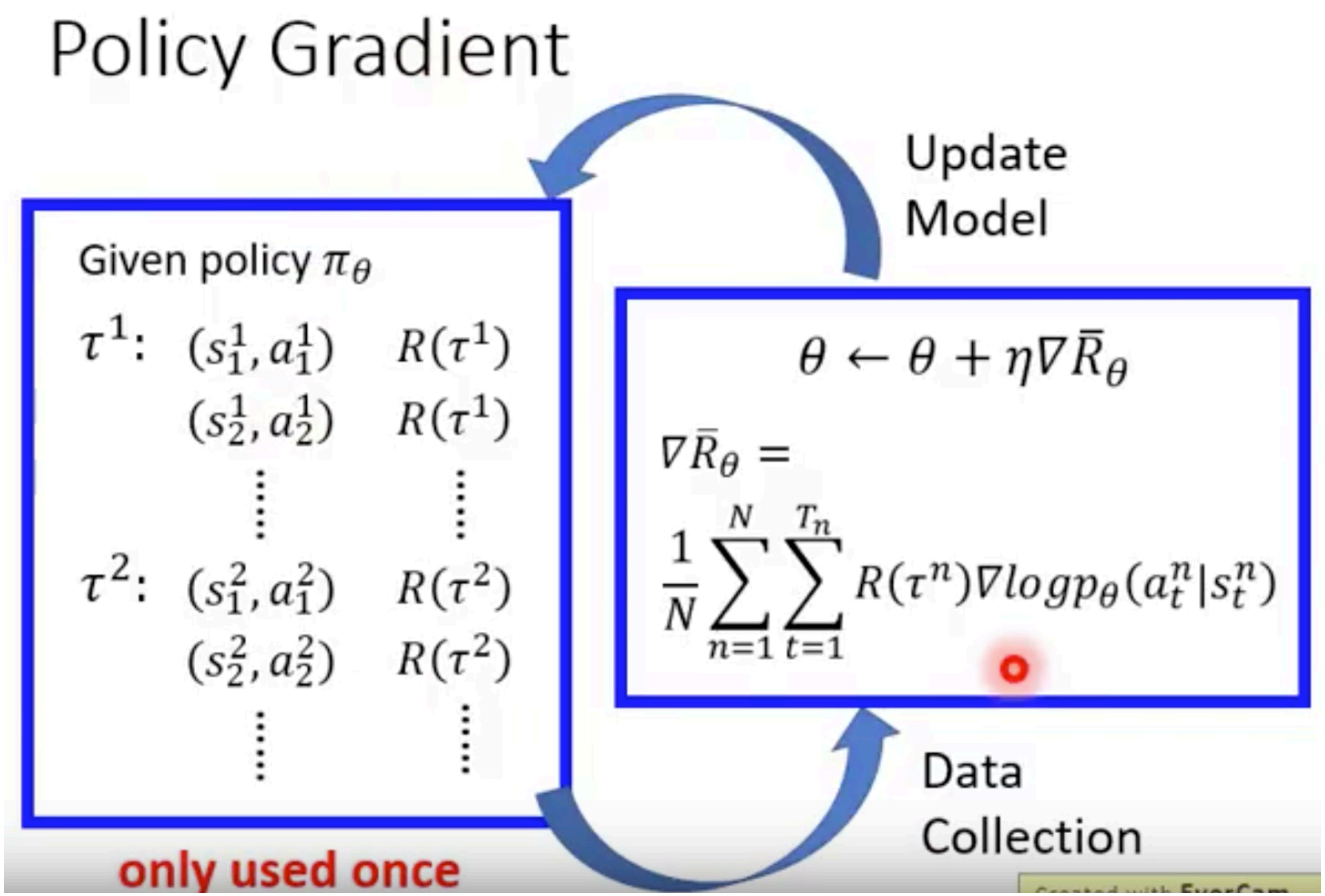
Rigorously speaking, we should use the above data only once (to update the NN's weights only once).

Then we need to collect data AGAIN!
Because when NN weights are updated, the probability of each episode/action has changed, so we need to sample again.

Policy Gradient



Rigorously speaking, we should use the above data only once (to update the NN's weights only once).



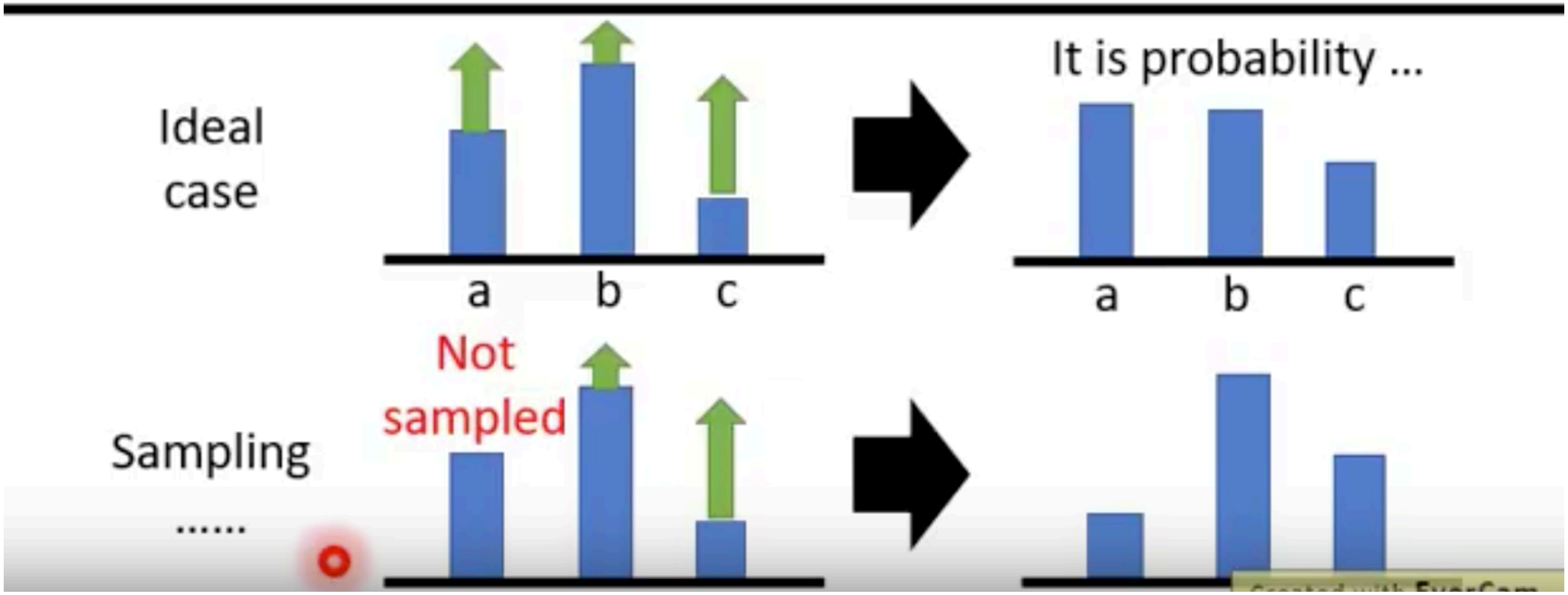
Very time consuming!

Tip 1: Add a Baseline

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

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

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

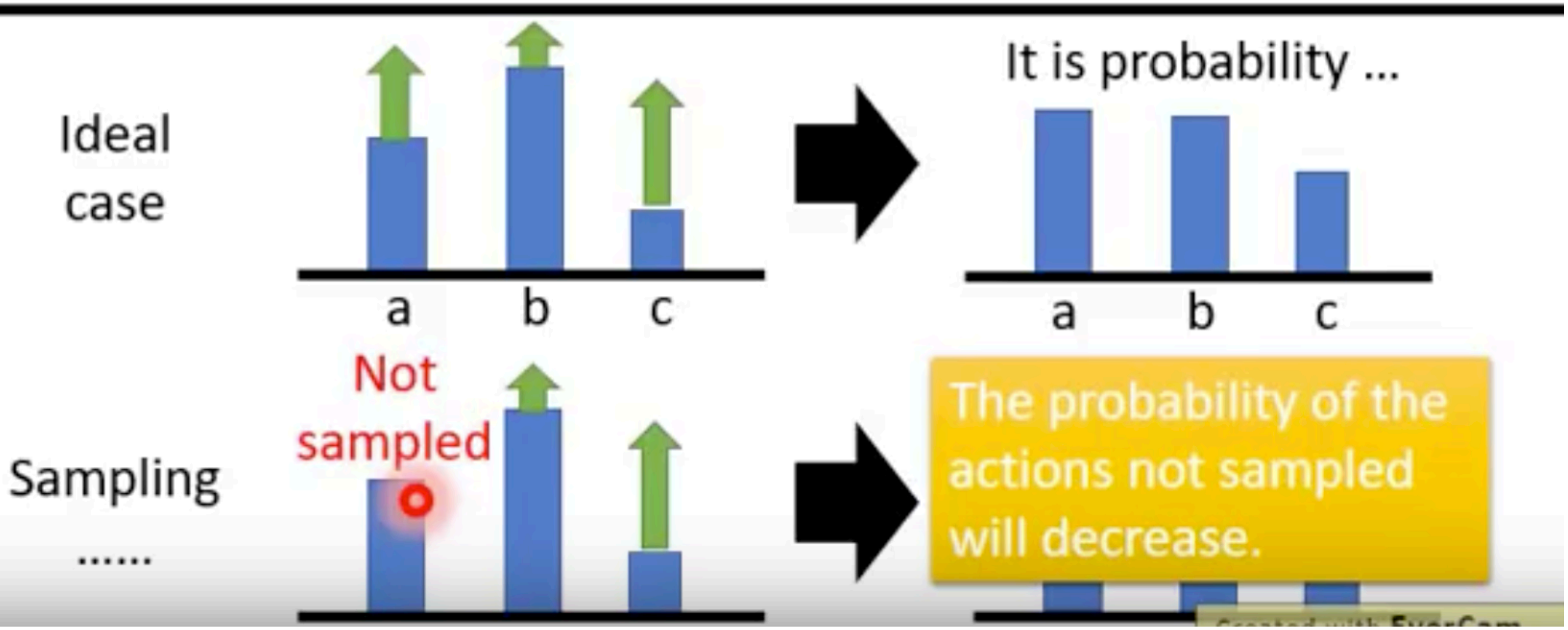


Tip 1: Add a Baseline

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

$\boxed{\text{It is possible that } R(\tau^n) \text{ is always positive.}}$

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

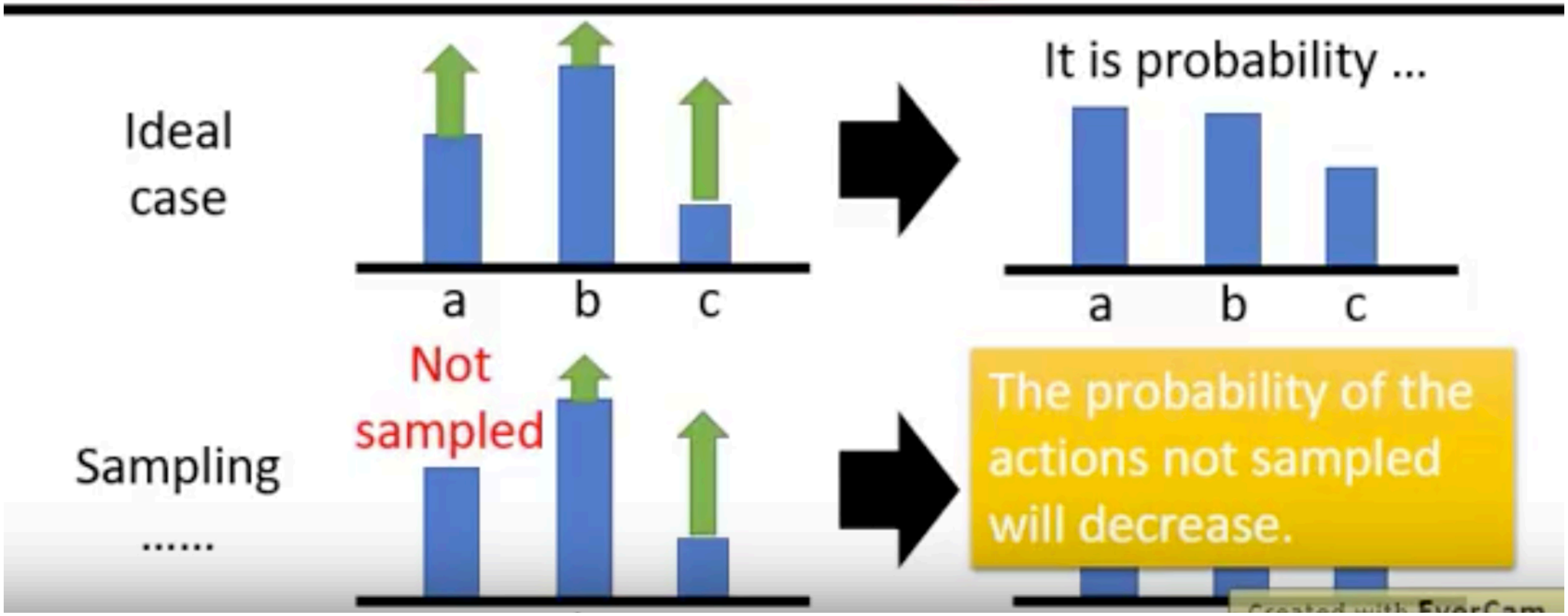


Tip 1: Add a Baseline

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

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

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



Tip 2: Assign Suitable Credit

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

$$R = +3$$

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

$$R = -7$$

Tip 2: Assign Suitable Credit

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

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

Let the weight for an action be the total reward since this action in the episode, instead of the total reward of this whole episode (before and after this action).

Tip 2: Assign Suitable Credit

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

$R = +3$

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

$R = -7$

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

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

Tip 2: Assign Suitable Credit

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

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

Add discount factor

Tip 2: Assign Suitable Credit

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

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

Add discount factor

Tip 2: Assign Suitable Credit

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

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

Add discount factor $\gamma < 1$

Tip 2: Assign Suitable Credit

Can be state-dependent

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

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

(k)

Add discount factor $\gamma < 1$

Tip 2: Assign Suitable Credit

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

Can be state-dependent

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

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

Add discount factor $\gamma < 1$

Tip 2: Assign Suitable Credit

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

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

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

Add discount factor $\gamma < 1$

From on-policy to off-policy

Using the experience more than once

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.

On-policy → Off-policy

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

- Use π_θ to collect data. When θ is updated, we have to sample training data again.

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.

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.

Old weights

New (updated) weights

On-policy → Off-policy

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

- 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

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

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.

Importance Sampling

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

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

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.

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 $q(x)$

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.

Importance Sampling

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

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

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

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

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. until the two distributions are quite different.

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. until the two distributions are quite different.

When the two distributions are quite different, we need to use the NN with updated weights to collect data again.

On-policy → Off-policy

Gradient for update

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

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

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

On-policy → Off-policy

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

Gradient for update

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

$$A^{\theta'}(s_t,a_t)$$

This term is from sampled data.

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

On-policy → Off-policy

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

Gradient for update

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

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

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

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

On-policy → Off-policy

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

Gradient for update

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

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

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

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

On-policy → Off-policy

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

Gradient for update

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

$$A^{\theta'}(s_t,a_t)$$

This term is from sampled data.

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

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

Objective function:

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

How to stop updating the NN's weights when the NN's updated weights are quite different from the old weights (with which the data were sampled)?

Add Constraint

PPO / TRPO

Proximal Policy Optimization (PPO)

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

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

PPO / TRPO

Proximal Policy Optimization (PPO)

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

KL divergence of the NN's two output probability vectors

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

KL divergence of two probability vectors

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

PPO / TRPO

Proximal Policy Optimization (PPO)

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

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

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

TRPO (Trust Region Policy Optimization)

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

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

PPO / TRPO

θ cannot be very different from θ'

Constraint on behavior not parameters

Proximal Policy Optimization (PPO)

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

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

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

PPO algorithm

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

- 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

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

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

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

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

Q-Learning

Outline

- Introduction of Q-Learning
- Tips of Q-Learning
- Q-Learning for Continuous Actions

Critic

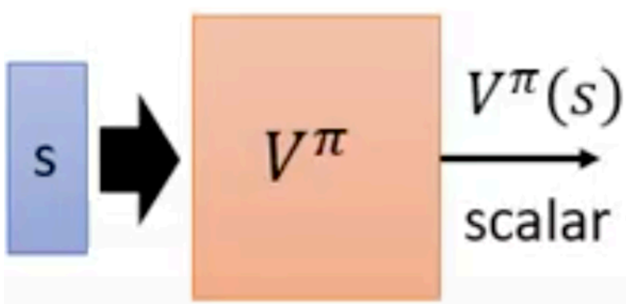
- A critic does not directly determine the action.
- Given an actor π, it evaluates how good the actor is

Critic

- A critic does not directly determine the action.
- Given an actor π, it evaluates how good the actor is
- State value function $V^\pi(s)$
    - When using actor π, the *cumulated* reward expects to be obtained after visiting state s

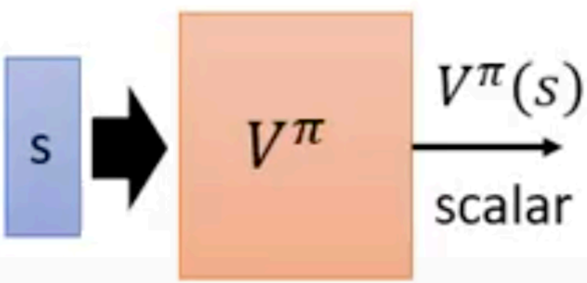
Critic

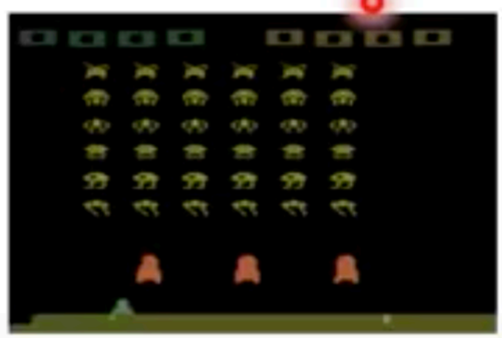
- A critic does not directly determine the action.
- Given an actor π, it evaluates how good the actor is
- State value function $V^{\pi}(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after visiting state s



Critic

- A critic does not directly determine the action.
- Given an actor π, it evaluates how good the actor is
- State value function $V^{\pi}(s)$
  - When using actor π, the *cumulated* reward expects to be obtained after visiting state s



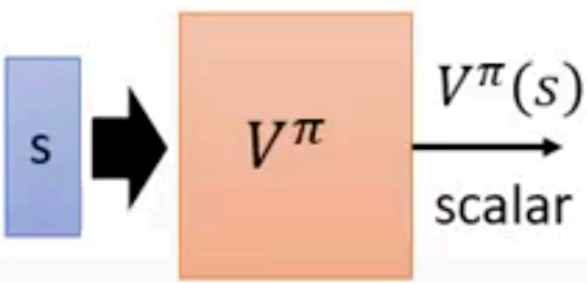


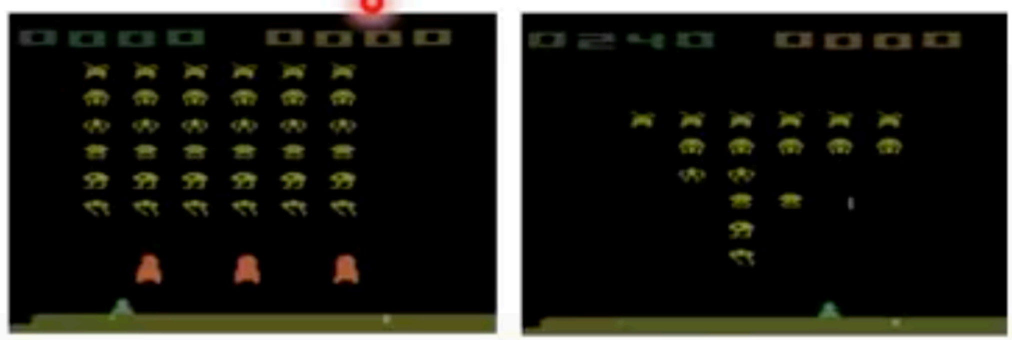
$V^{\pi}(s)$ is large

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

Critic

- A critic does not directly determine the action.
- Given an actor π, it evaluates how good the actor is
- State value function $V^{\pi}(s)$ It depends on the actor
  - When using actor π, the *cumulated* reward expects to be obtained after visiting state s





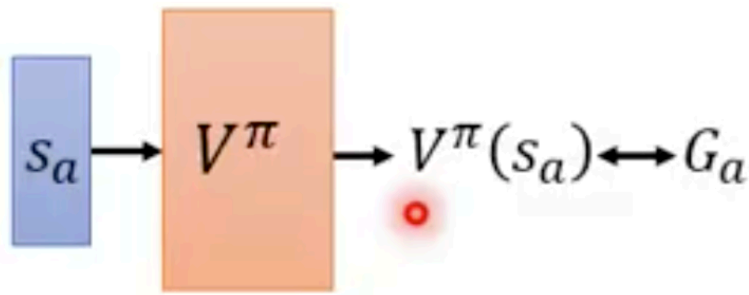
$V^{\pi}(s)$ is large $V^{\pi}(s)$ is smaller

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

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

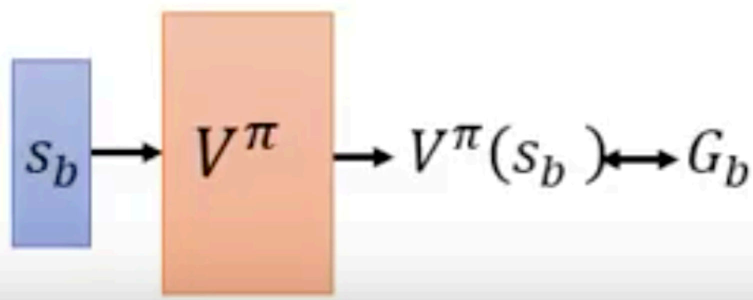
After seeing s_a,

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



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

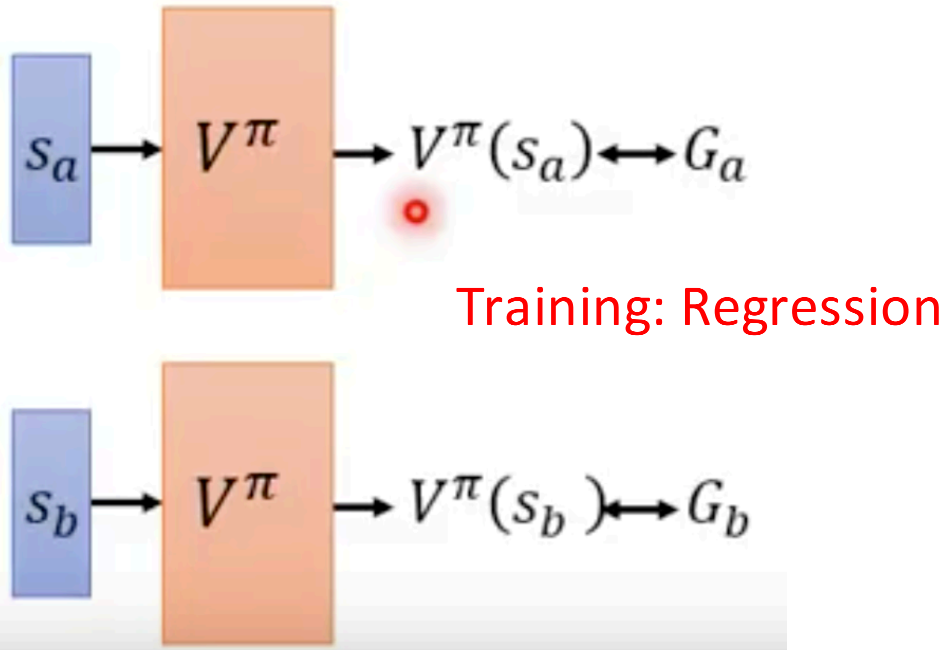
- **Monte-Carlo (MC) based approach**
  - The critic watches π playing the game

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



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

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

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

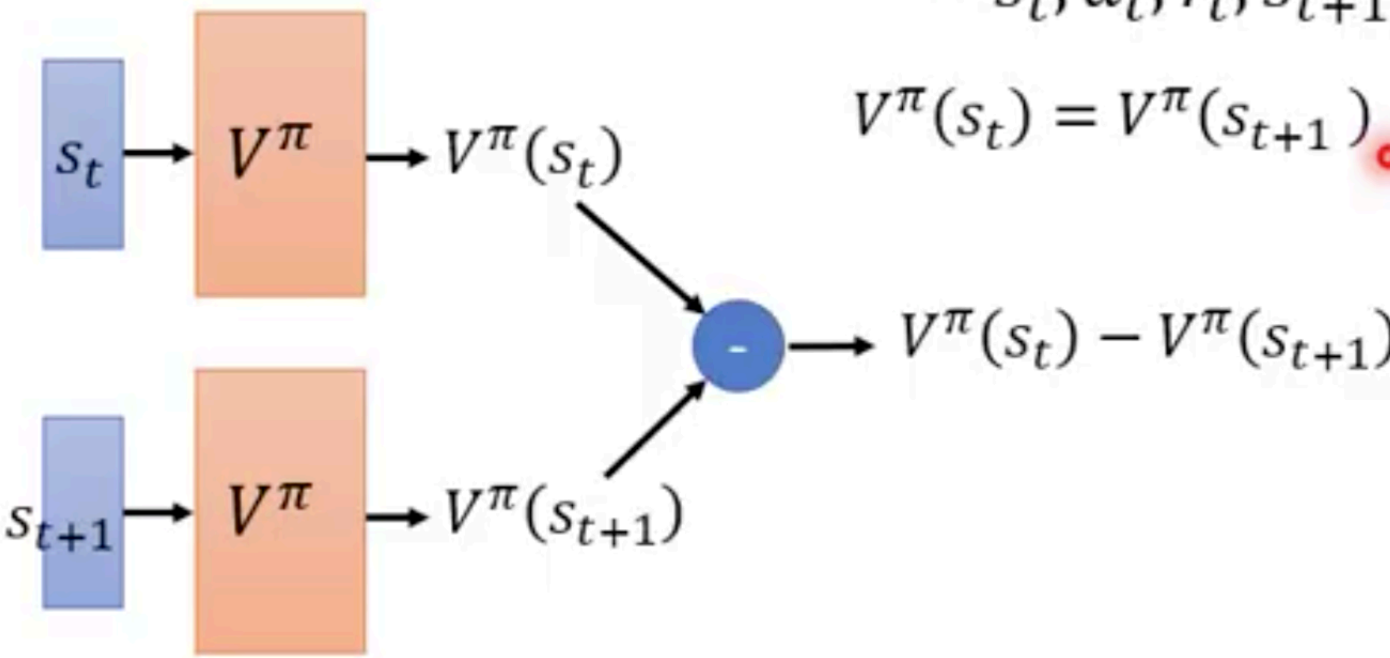
$$V^{\pi}(s_t) = V^{\pi}(s_{t+1}) + r_t$$

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

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

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

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

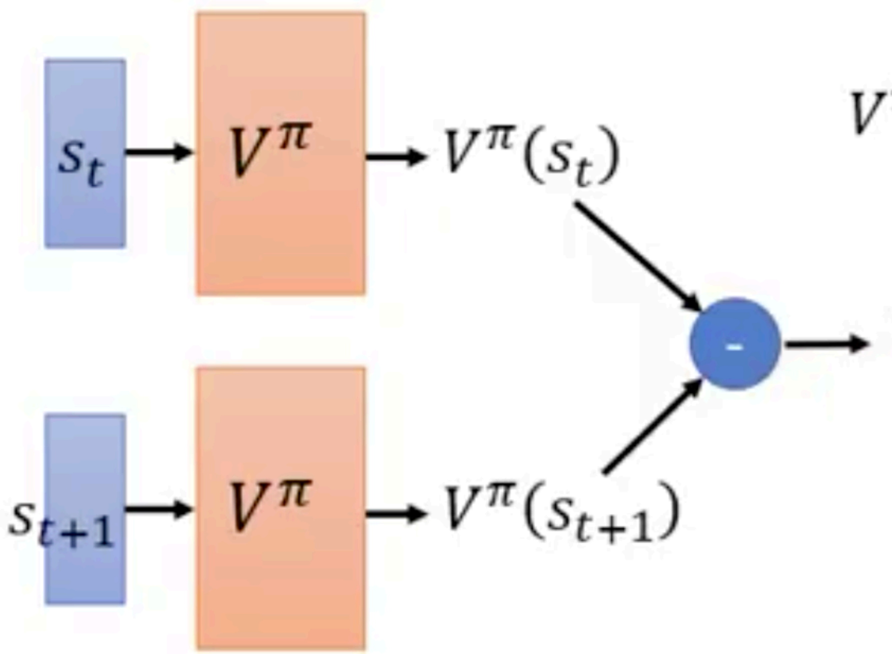


How to estimate $V^{\pi}(s)$

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

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

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

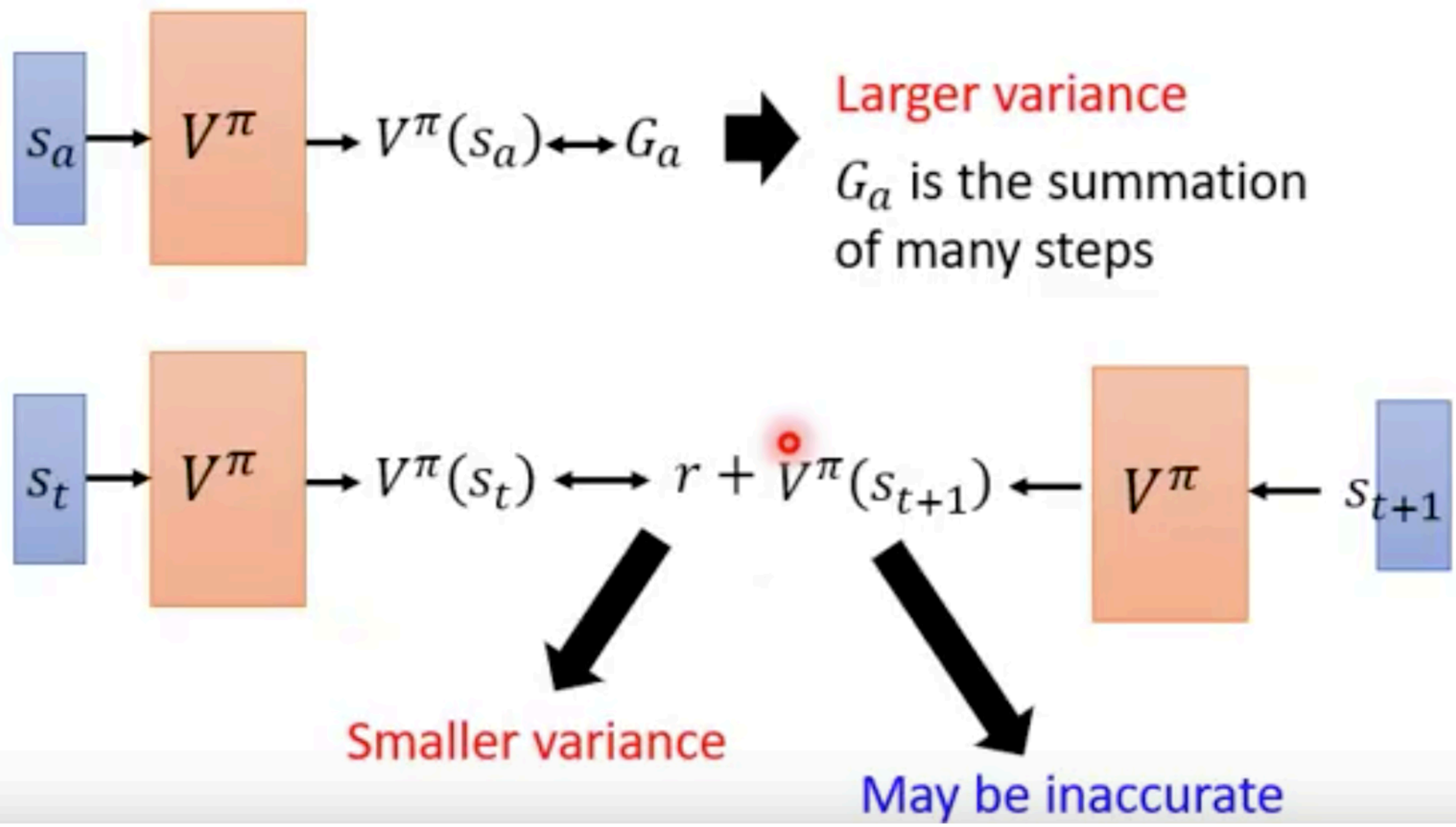


$$V^{\pi}(s_t) - V^{\pi}(s_{t+1}) \longleftrightarrow r_t$$

Training: Regression

$$Var[kX] = k^2 Var[X]$$

MC v.s. TD



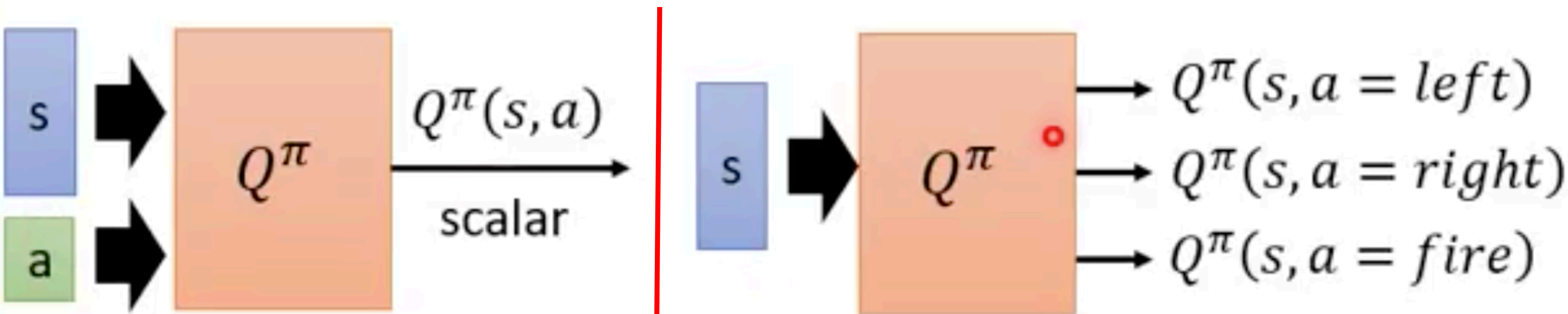
Another Critic

- 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

Another Critic

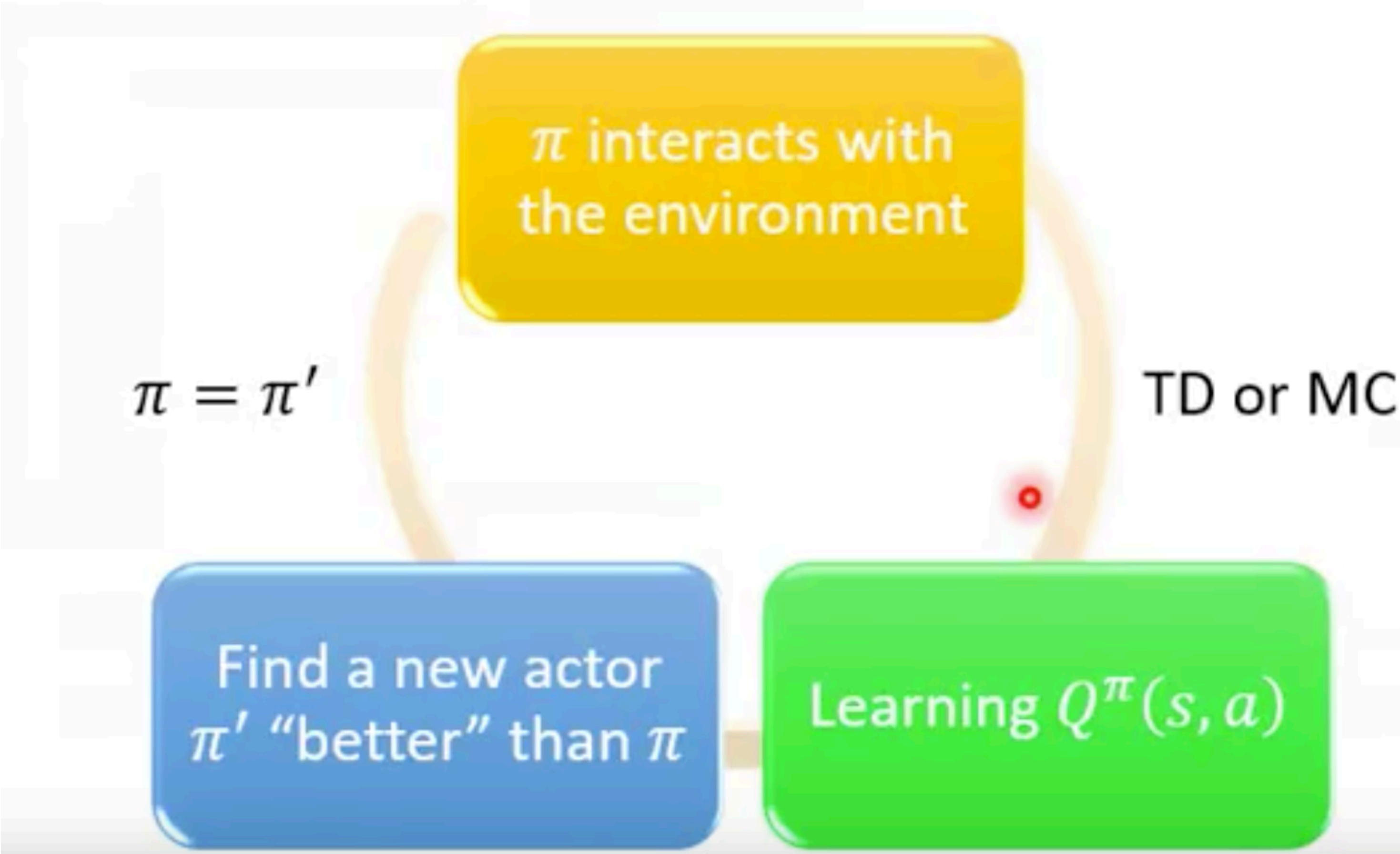
- State-action value function $Q^\pi(s, a)$
  - When using actor π, the *cumulated* reward expects to be obtained after taking a at state s

Two networks for Q-function

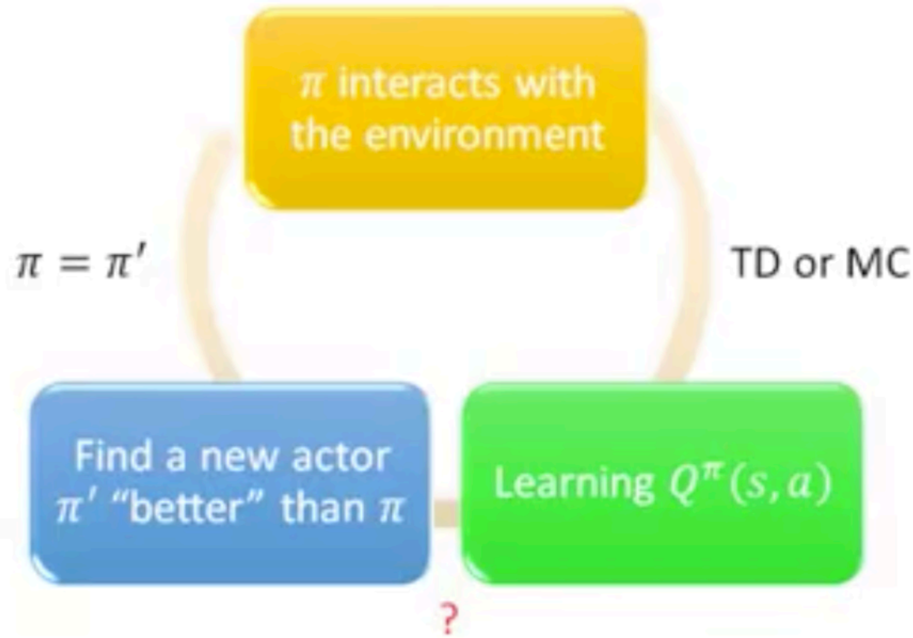


for discrete action only

Another Way to use Critic: Q-Learning



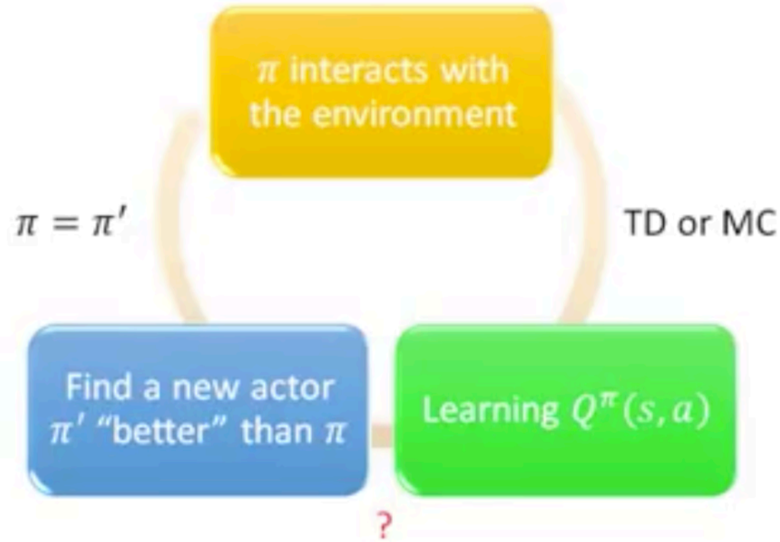
Q-Learning



- 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

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

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

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

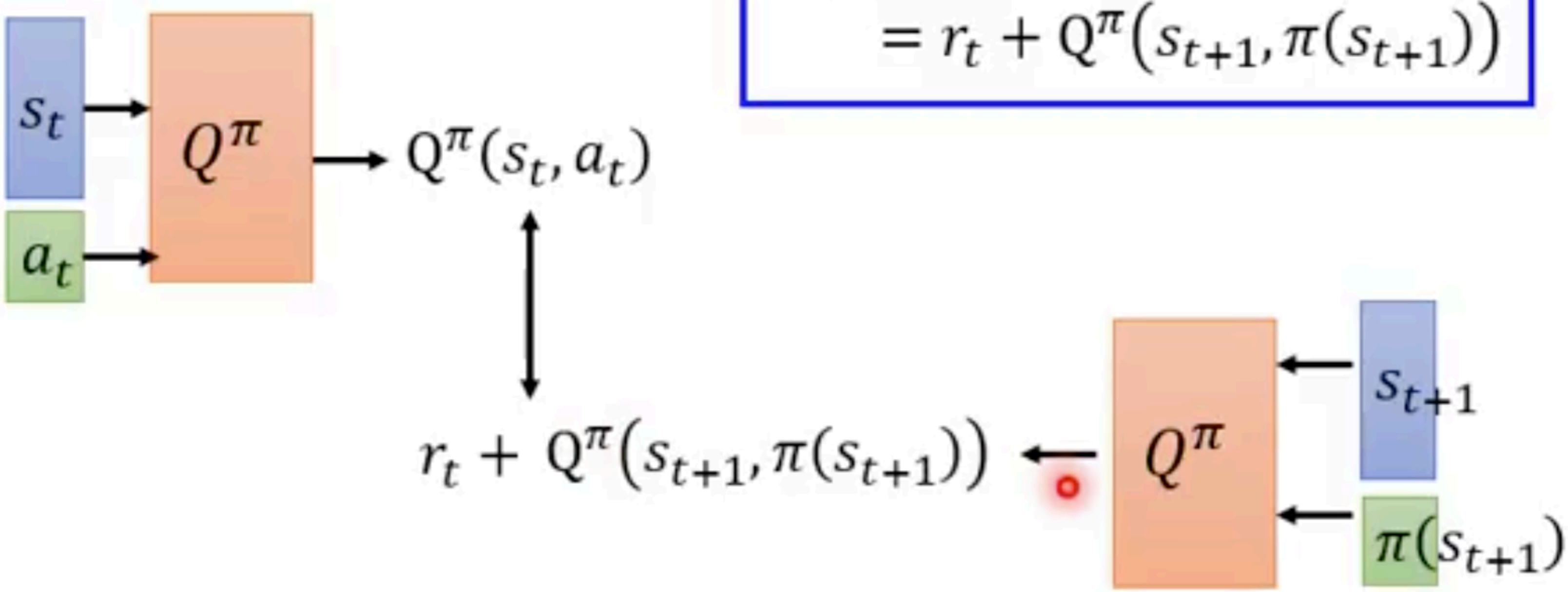
➢ π' does not have extra parameters. It depends on Q

➢ Not suitable for continuous action a (solve it later)

Target Network

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

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

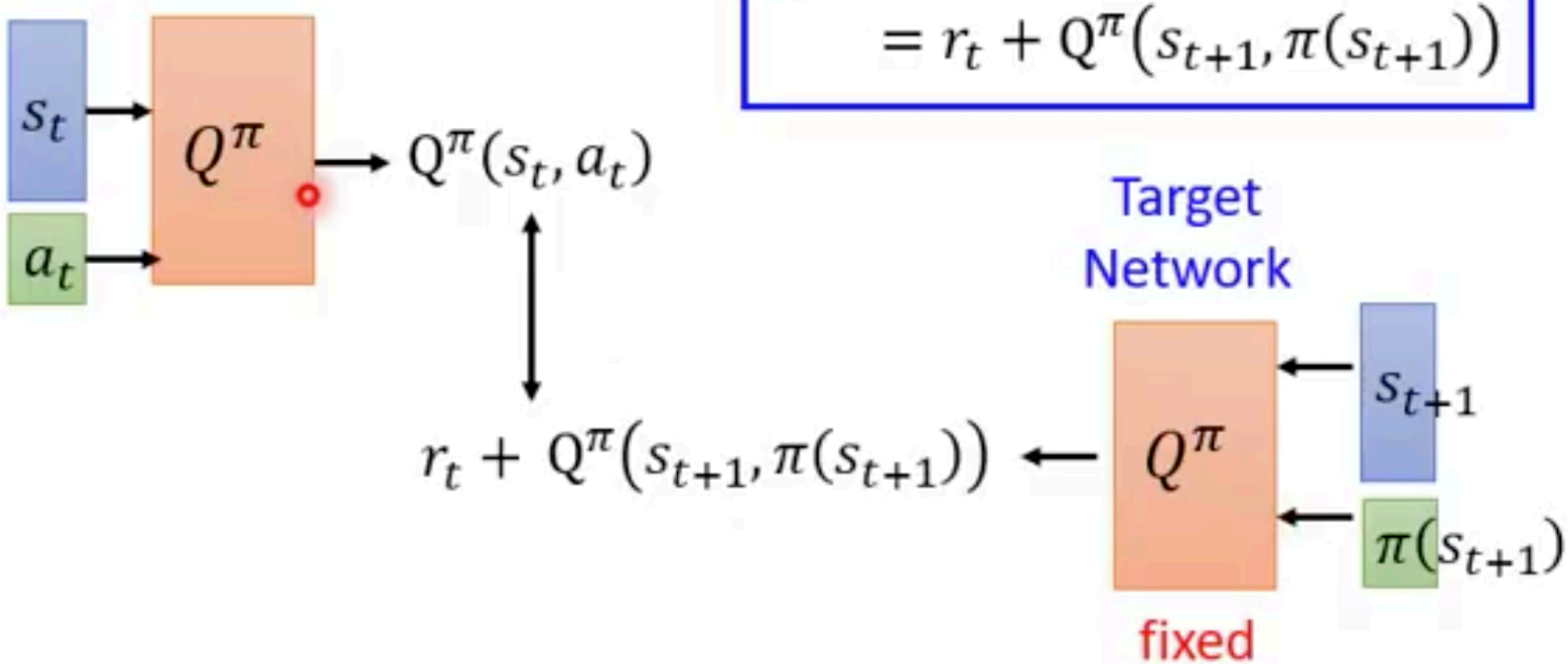


To make training stable: fixed

Target Network

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

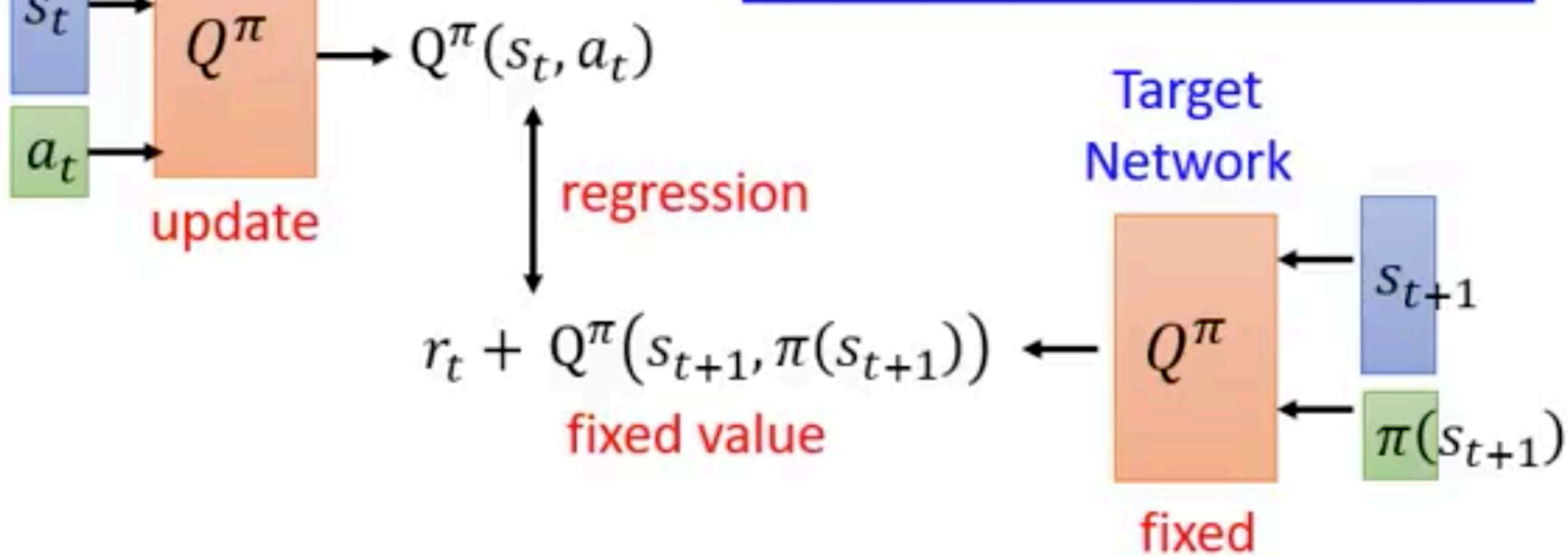
$$Q^\pi(s_t, a_t) = r_t + Q^\pi\big(s_{t+1}, \pi(s_{t+1})\big)$$



Target Network

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

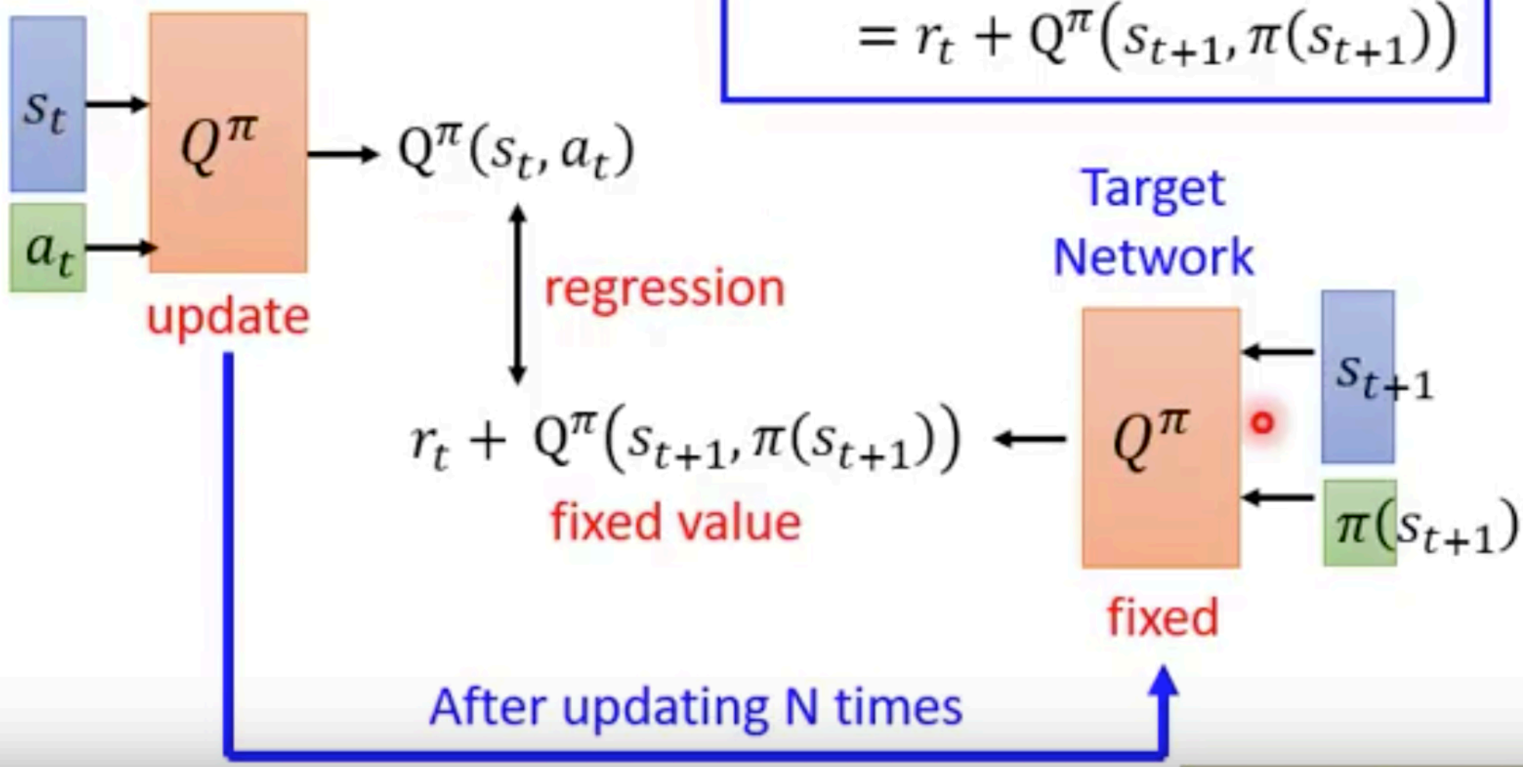
$$Q^\pi(s_t, a_t) = r_t + Q^\pi\big(s_{t+1}, \pi(s_{t+1})\big)$$



Target Network

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

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



Exploration

- The policy is based on Q-function

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

This is not a good way to collect data,
because if an action a is not sampled,
it will not be chosen by the neural network.

Exploration

$s \rightarrow a_1$ $\quad Q(s,a) = 0$ Never explore

$s \rightarrow a_2$ $\quad Q(s,a) = \boxed{1}$ Always sampled

$s \rightarrow a_3$ $\quad Q(s,a) = 0$ Never explore

- The policy is based on Q-function

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

This is not a good way for data collection.

Exploration

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

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

$s \to a_3$ $Q(s,a) = 0$ 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.

<u>Epsilon Greedy</u> ε would decay during learning

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

Exploration

$$s \to \begin{cases} a_1 & Q(s,a) = 0 \quad \text{Never explore} \\ a_2 & Q(s,a) = \boxed{1} \quad \text{Always sampled} \\ a_3 & Q(s,a) = 0 \quad \text{Never explore} \end{cases}$$

- The policy is based on Q-function

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

This is not a good way for data collection.

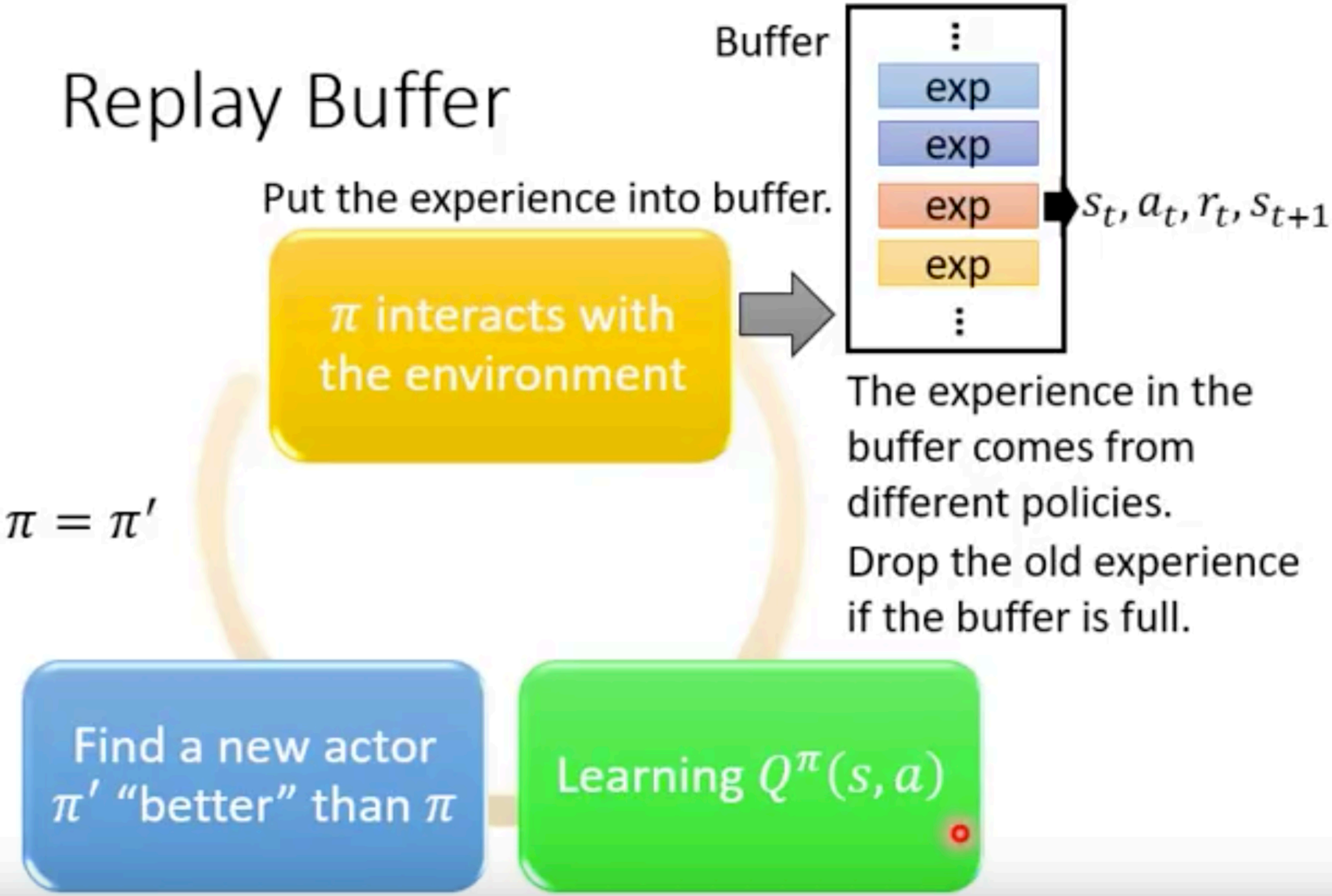
Epsilon Greedy ε would decay during learning

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

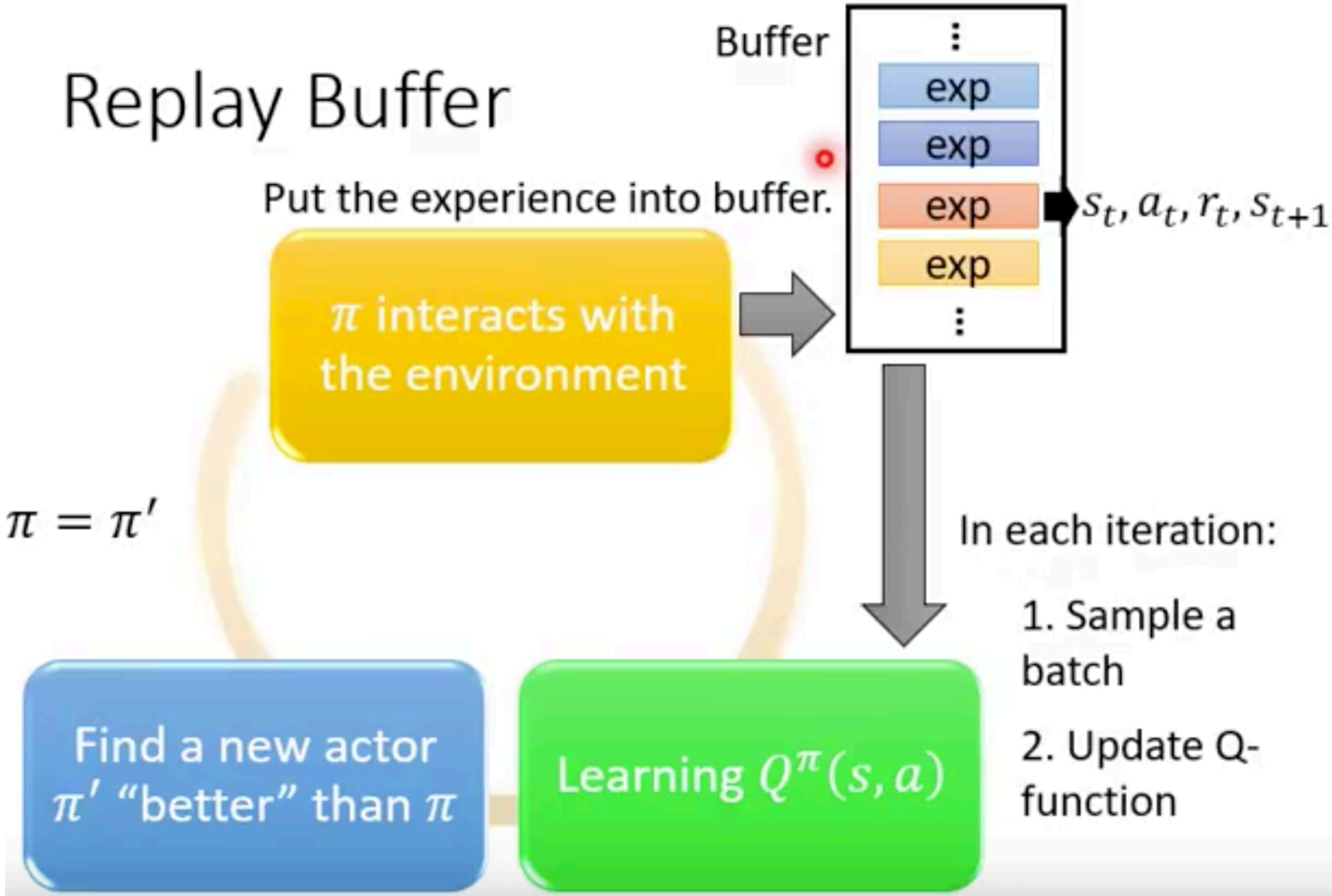
Boltzmann Exploration

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

Replay Buffer

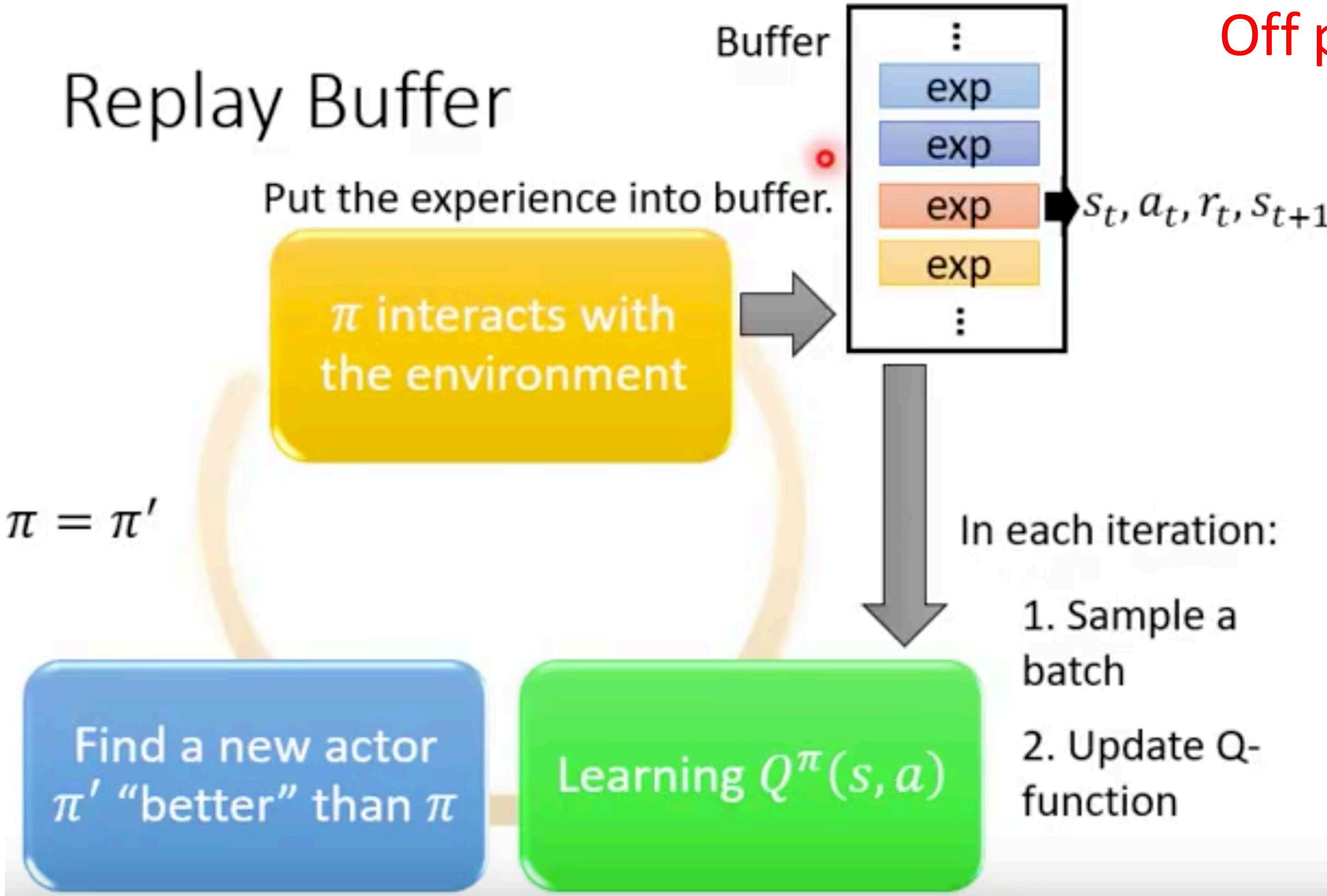


Replay Buffer



Replay Buffer

Off policy: OK



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 $\hat{Q} = Q$

Additional slide: some early demos of projects

- Music Genre Classification (Joshua Crockett), https://www.youtube.com/watch?v=OvO67VXRK7s&feature=youtu.be
- Stock Price Prediction (Tsao Yuan Chang), https://www.youtube.com/watch?v=T_9-PG96P7k&feature=youtu.be
- Traffic detection (Jackson Delametter), https://www.youtube.com/watch?v=bU_lE6iKNzQ&feature=youtu.be
- Hand Gesture Recognition (Syed Ali Hasnain), https://www.youtube.com/watch?v=KCxmaGc2U8Q&feature=youtu.be