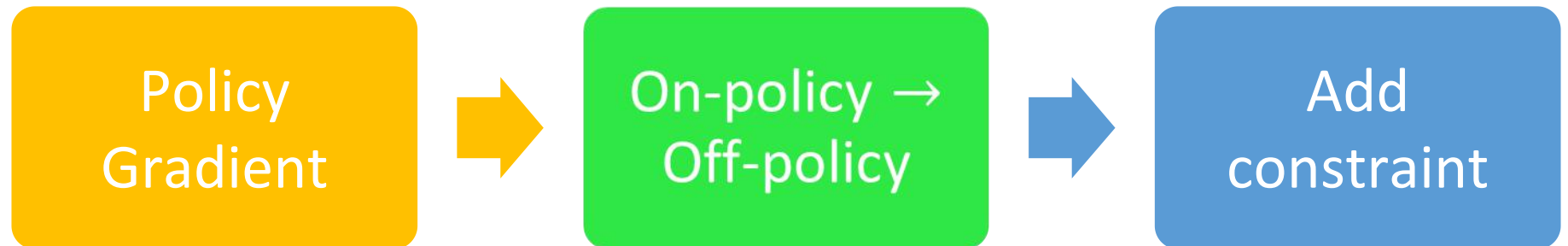


# Proximal Policy Optimization (PPO)

default reinforcement learning algorithm at OpenAI



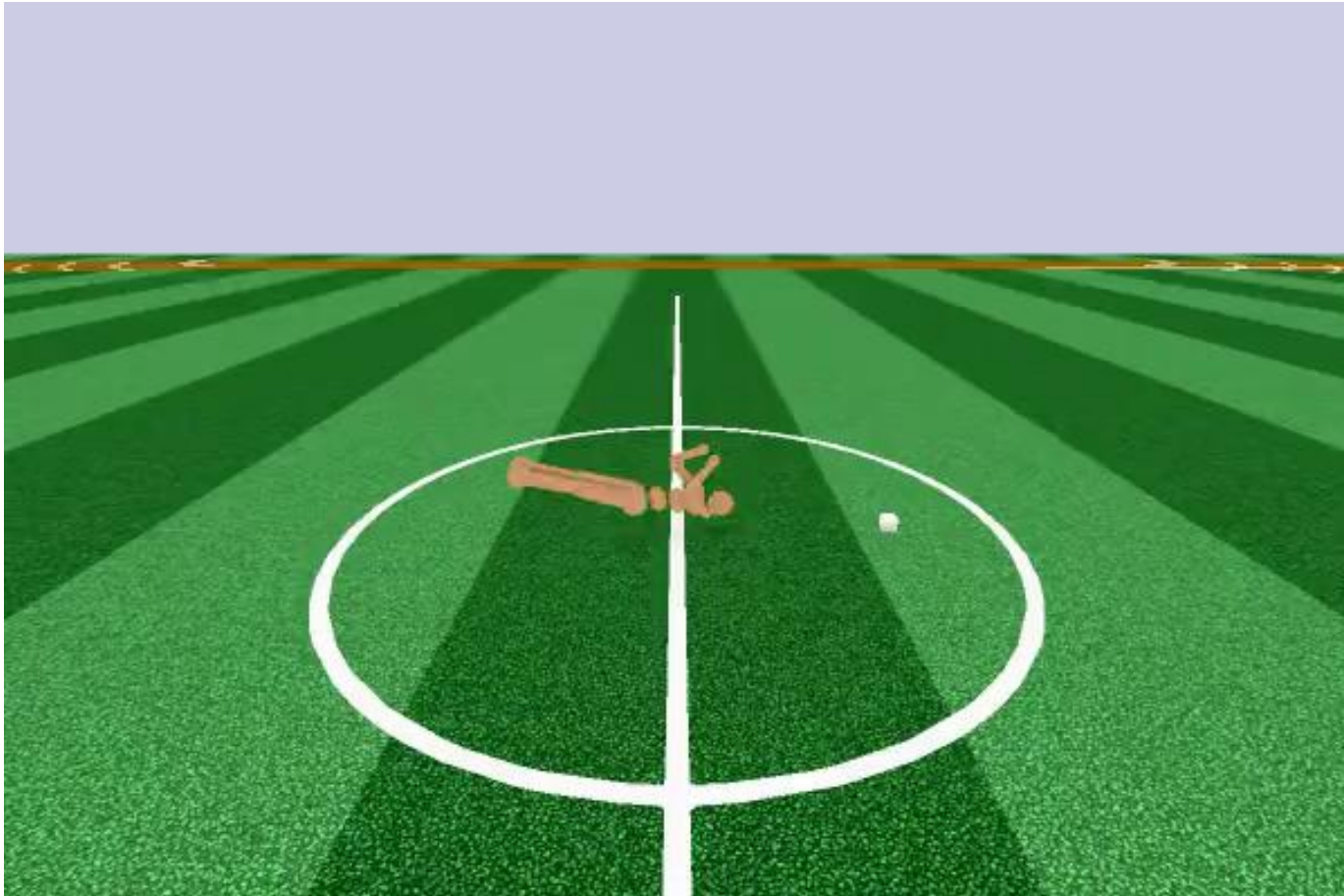
# DeepMind

<https://youtu.be/gn4nRCC9TwQ>



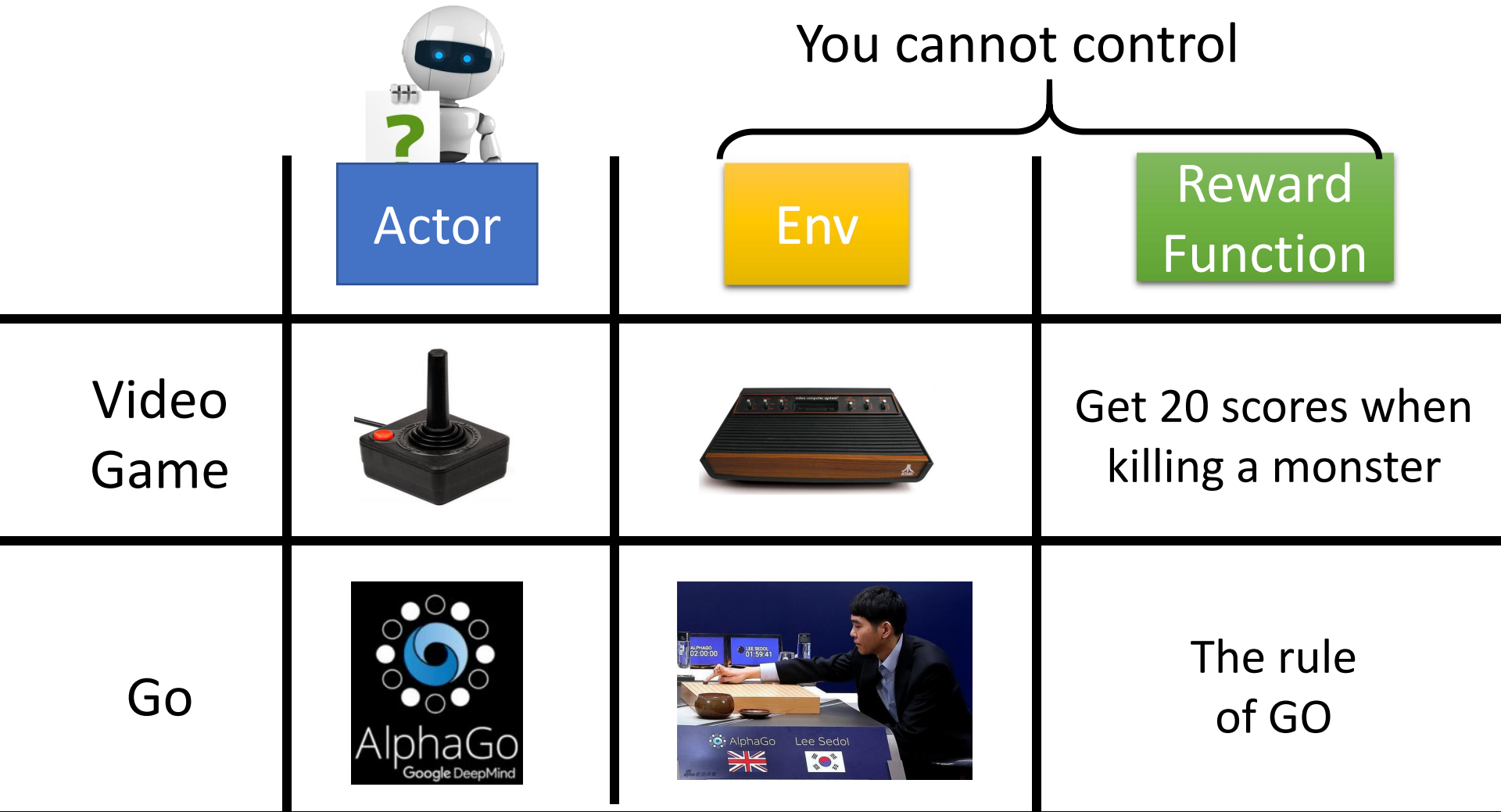
# OpenAI

<https://blog.openai.com/openai-baselines-ppo/>



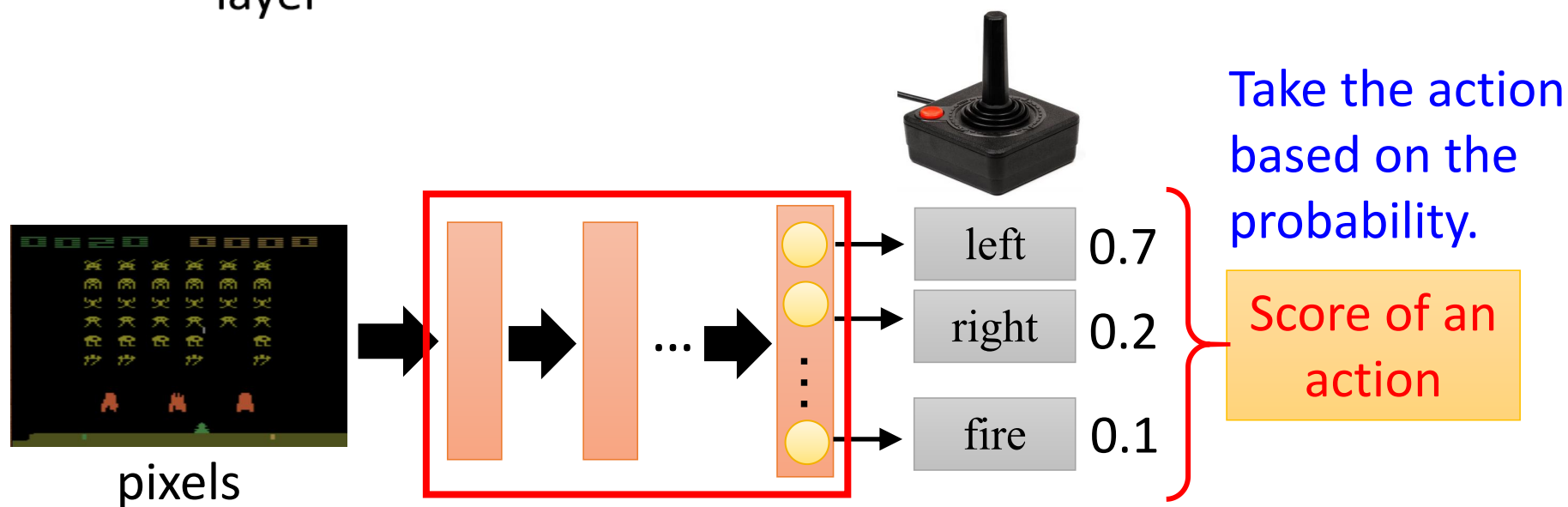
# Policy Gradient (Review)

# Basic Components

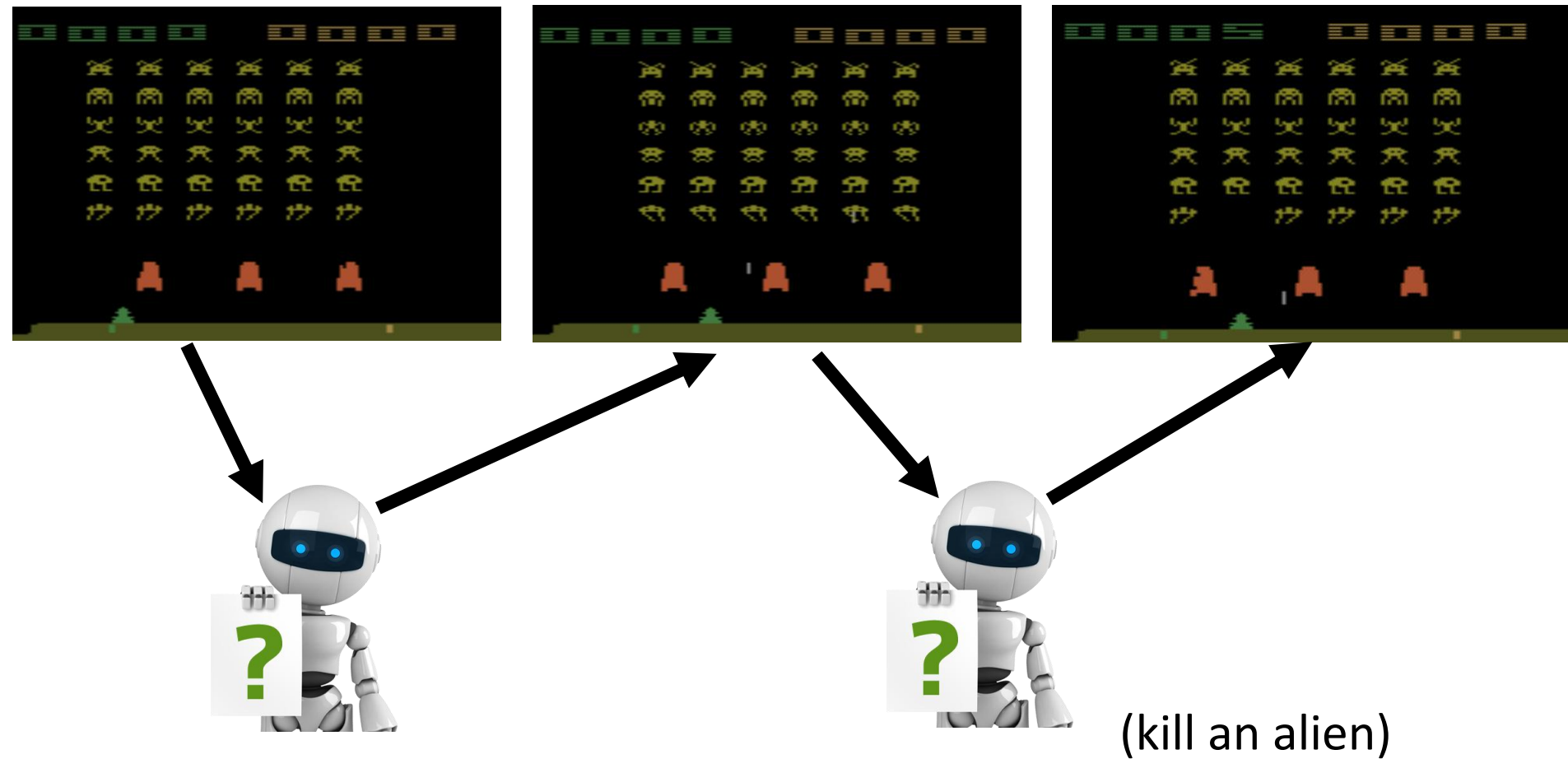


# Policy of Actor

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



# Example: Playing Video Game



# Example: Playing Video Game



This is an *episode*.

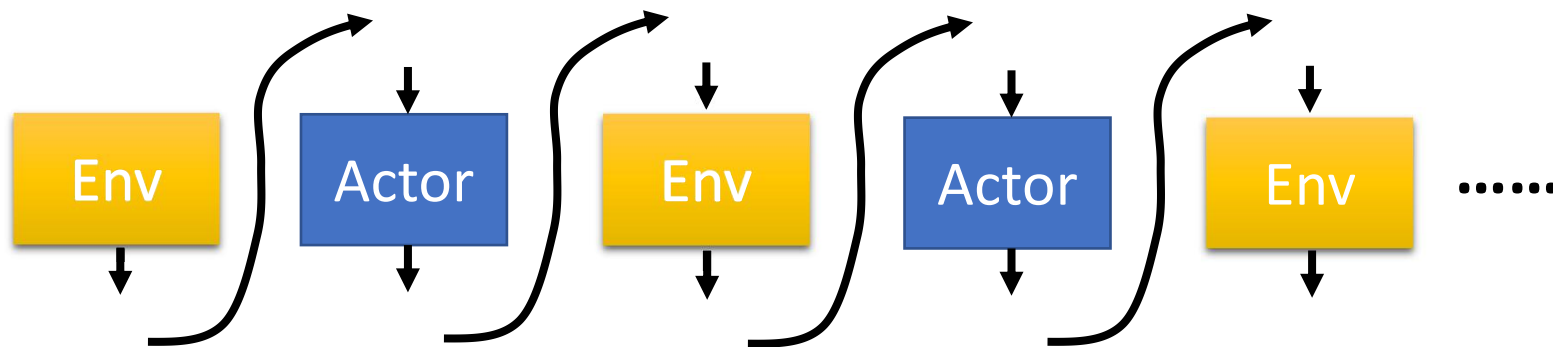
After many turns



We want the total  
reward be maximized.



# Actor, Environment, Reward



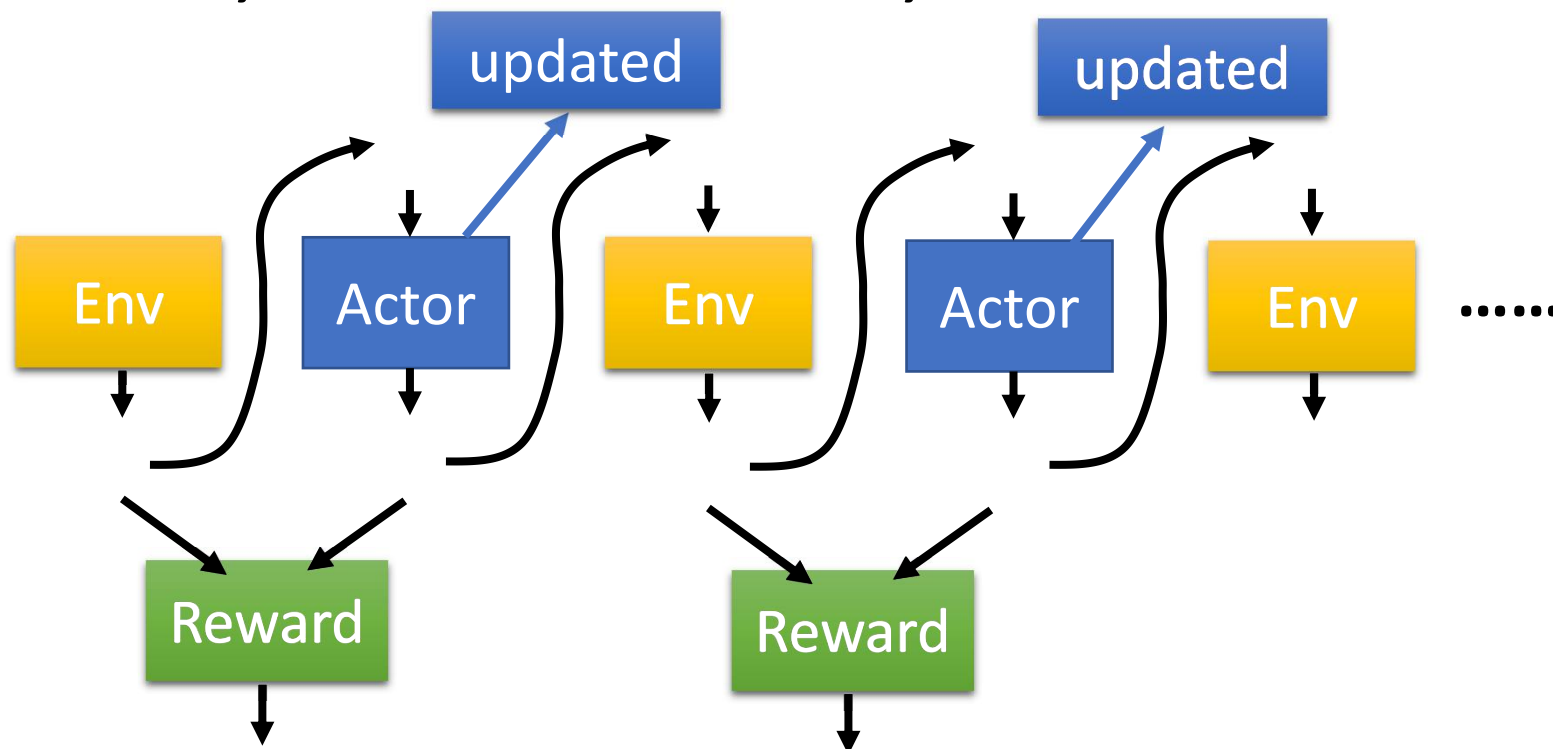
**Trajectory**  $\tau = \{s_1, a_1, s_2, a_2, \dots, s_T, a_T\}$

$p_{\theta}(\tau)$

$$= p(s_1)p_{\theta}(a_1|s_1)p(s_2|s_1, a_1)p_{\theta}(a_2|s_2)p(s_3|s_2, a_2) \dots$$

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

# 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

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

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

$R(\tau)$  do not have to be differentiable

It can even be a black box.

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

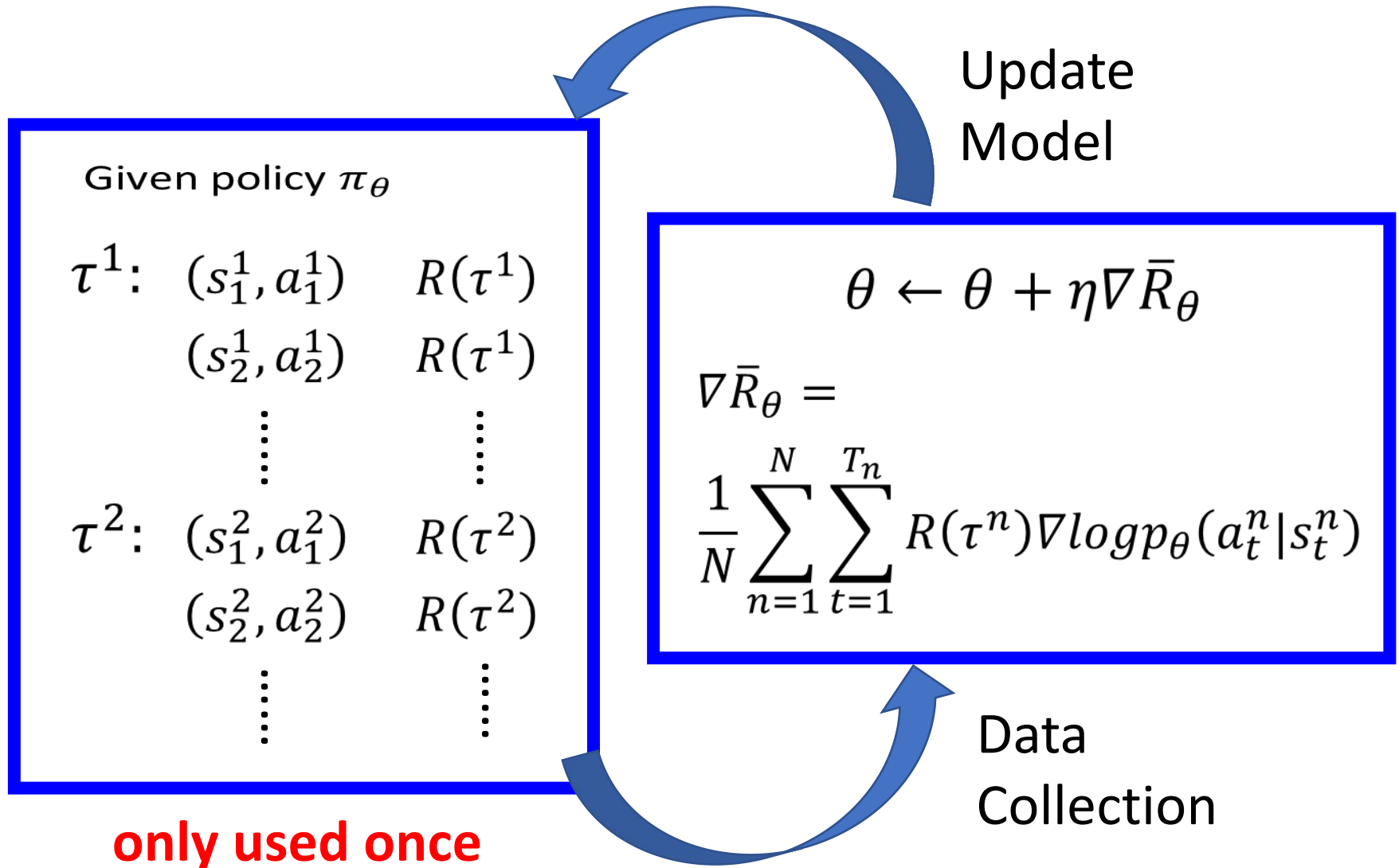
$$\nabla f(x) = f(x) \nabla \log f(x)$$

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

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

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

# Policy Gradient



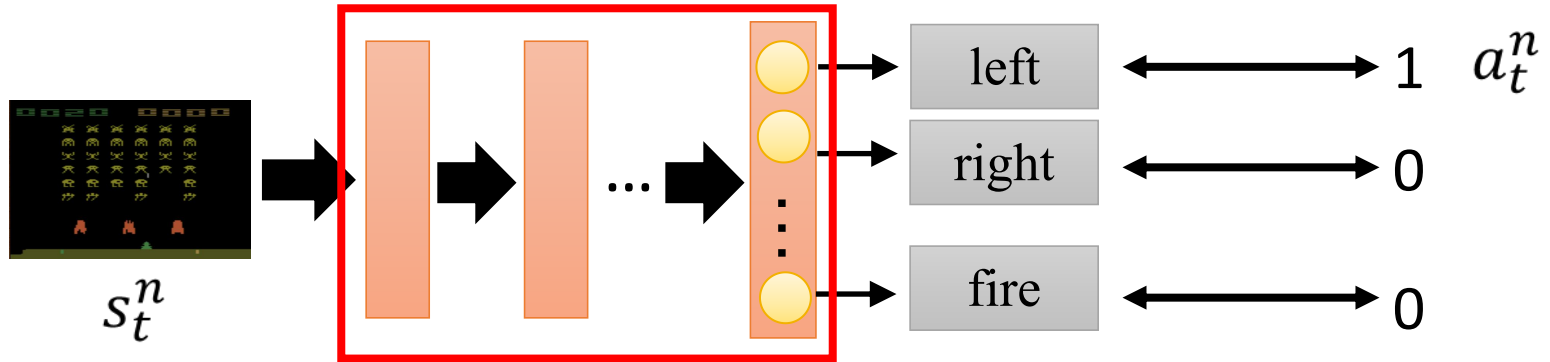
$$\theta \leftarrow \theta + \eta \nabla \bar{R}_\theta$$

## Implementation

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

Consider as classification problem

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



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

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

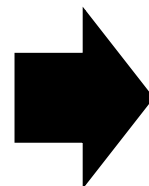
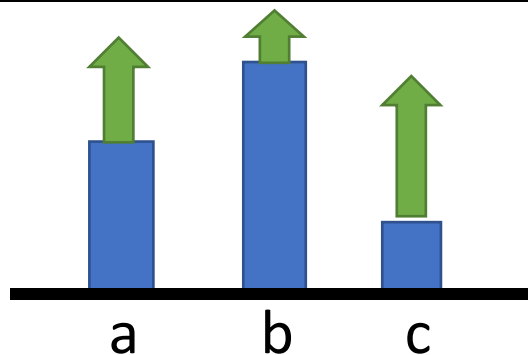
# Tip 1: Add a Baseline

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

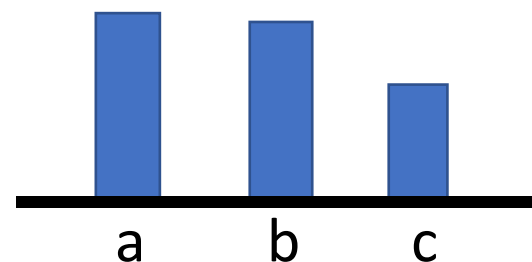
It is possible that  $R(\tau^n)$  is always positive.

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

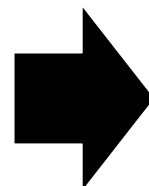
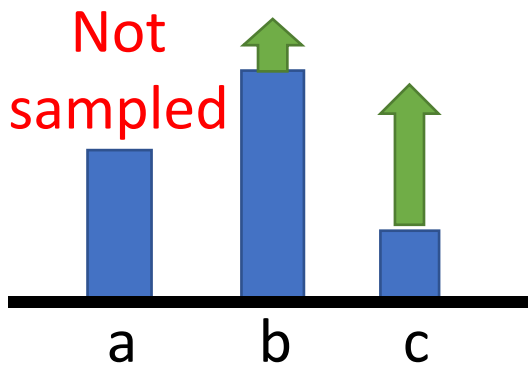
Ideal  
case



It is probability ...



Sampling  
.....



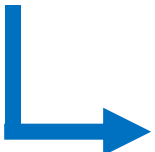
The probability of the  
actions not sampled  
will decrease.



## Tip 2: Assign Suitable Credit

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

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



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

# 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

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

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

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.



阿光下棋



佐為下棋、阿光在旁邊看

# On-policy $\rightarrow$ Off-policy

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

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

---

## Importance Sampling

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

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

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

# Issue of Importance Sampling

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

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

$$\begin{aligned} \text{VAR}[X] \\ = E[X^2] - (E[X])^2 \end{aligned}$$

$$\text{Var}_{x \sim p}[f(x)] = E_{x \sim p}[f(x)^2] - (E_{x \sim p}[f(x)])^2$$

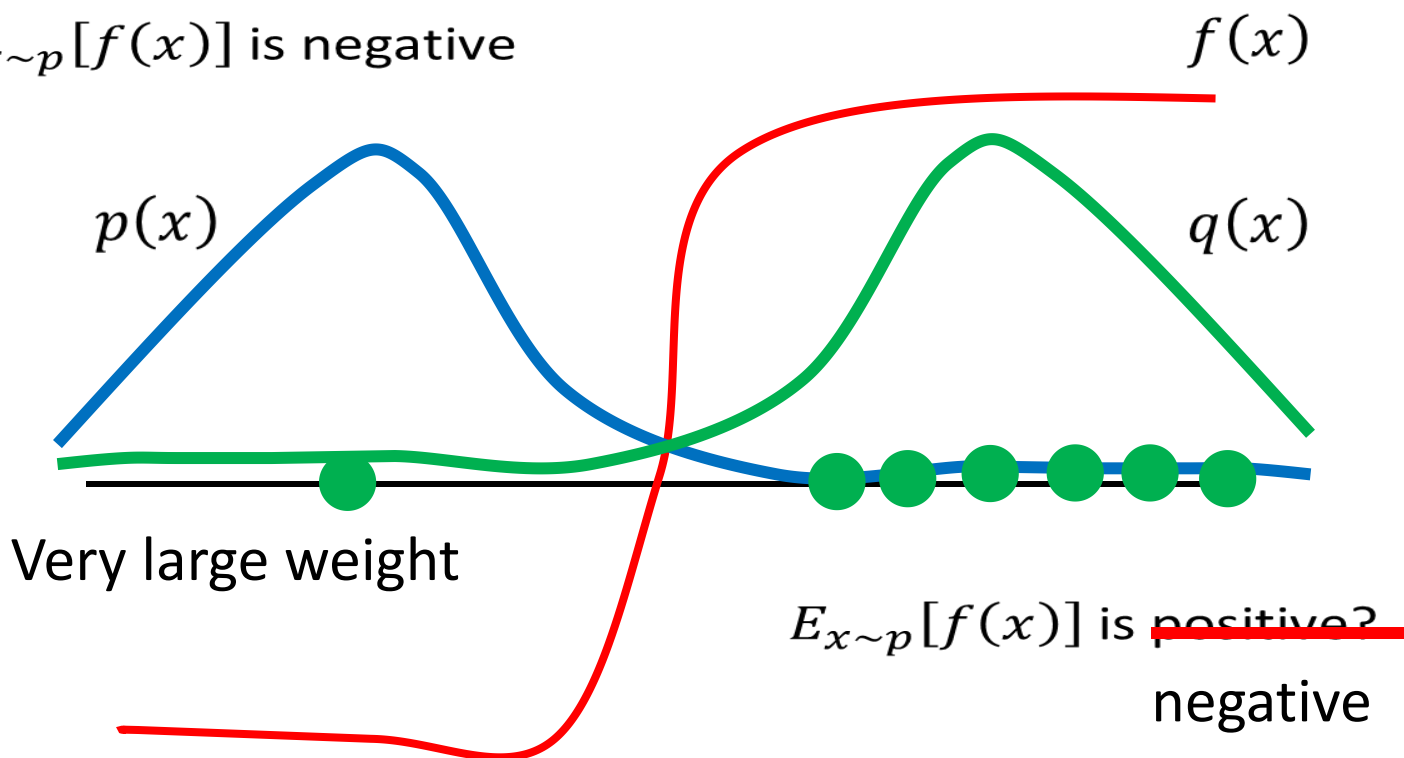
$$\text{Var}_{x \sim q}\left[f(x) \frac{p(x)}{q(x)}\right] = E_{x \sim q}\left[\left(f(x) \frac{p(x)}{q(x)}\right)^2\right] - \left(E_{x \sim q}\left[f(x) \frac{p(x)}{q(x)}\right]\right)^2$$

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

# Issue of Importance Sampling

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

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



# On-policy $\rightarrow$ Off-policy

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

- Use  $\pi_\theta$  to collect data. When  $\theta$  is updated, we have to sample training data again.
- Goal: Using the sample from  $\pi_{\theta'}$  to train  $\theta$ .  $\theta'$  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  $\theta'$ .
- Use the data to train  $\theta$  many times.

---

**Importance**  
**Sampling**

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

# On-policy $\rightarrow$ Off-policy

Gradient for update

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

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

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

This term is from  
sampled data.

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

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

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

# Add Constraint

穩紮穩打，步步為營



# PPO / TRPO

$\theta$  cannot be very different from  $\theta'$

Constraint on behavior not parameters

## Proximal Policy Optimization (PPO)

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

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

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

## 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 algorithm

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

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

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

Update parameters  
several times

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

Adaptive  
KL Penalty

# PPO algorithm

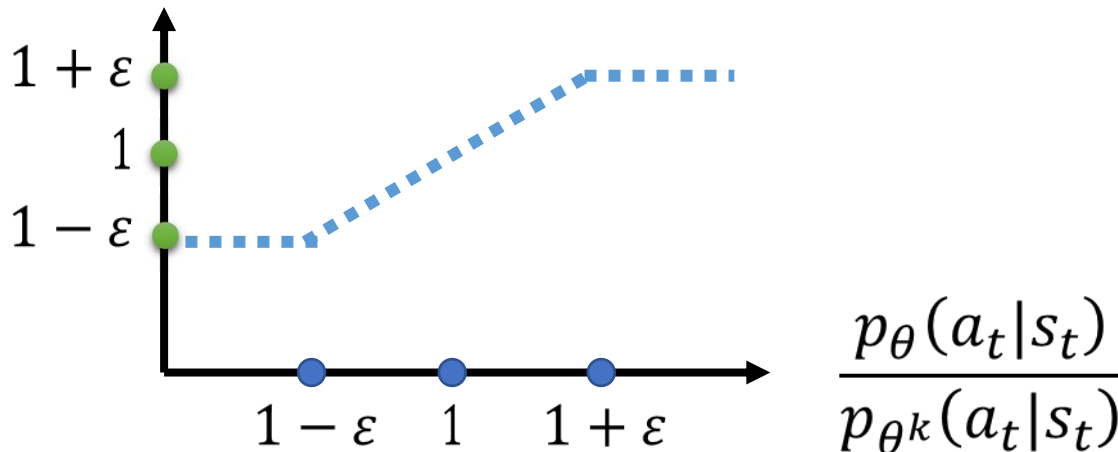
$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta K L(\theta, \theta^k)$$

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

# PPO2 algorithm

$$J_{PPO2}^{\theta^k}(\theta) \approx \sum_{(s_t, a_t)}$$

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



# PPO algorithm

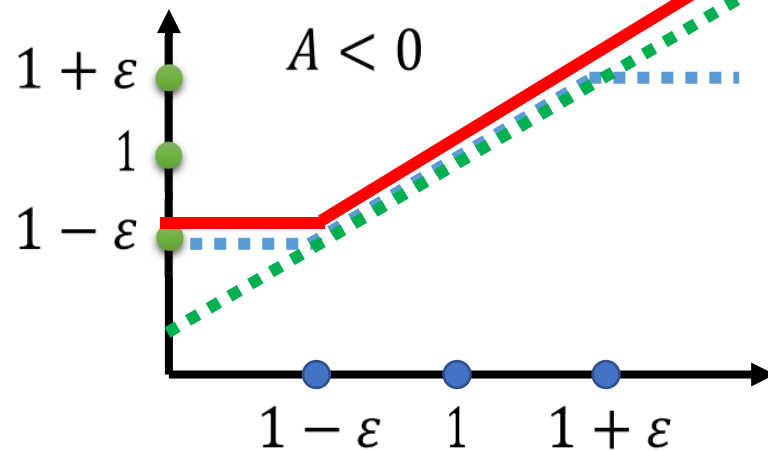
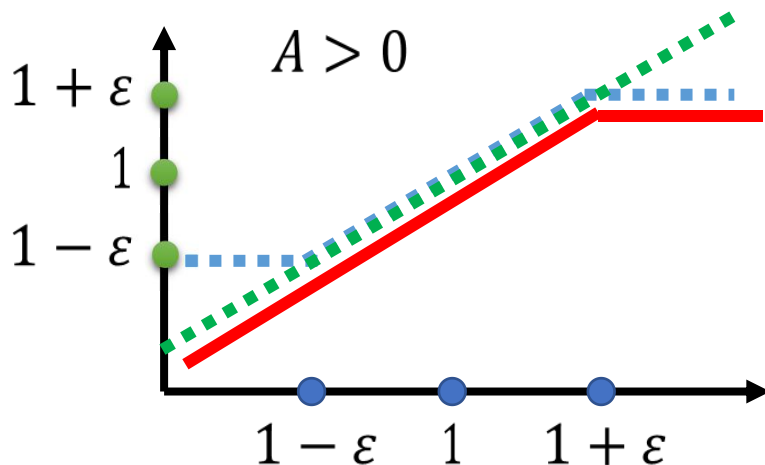
$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k)$$

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

# PPO2 algorithm

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

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



# Experimental Results

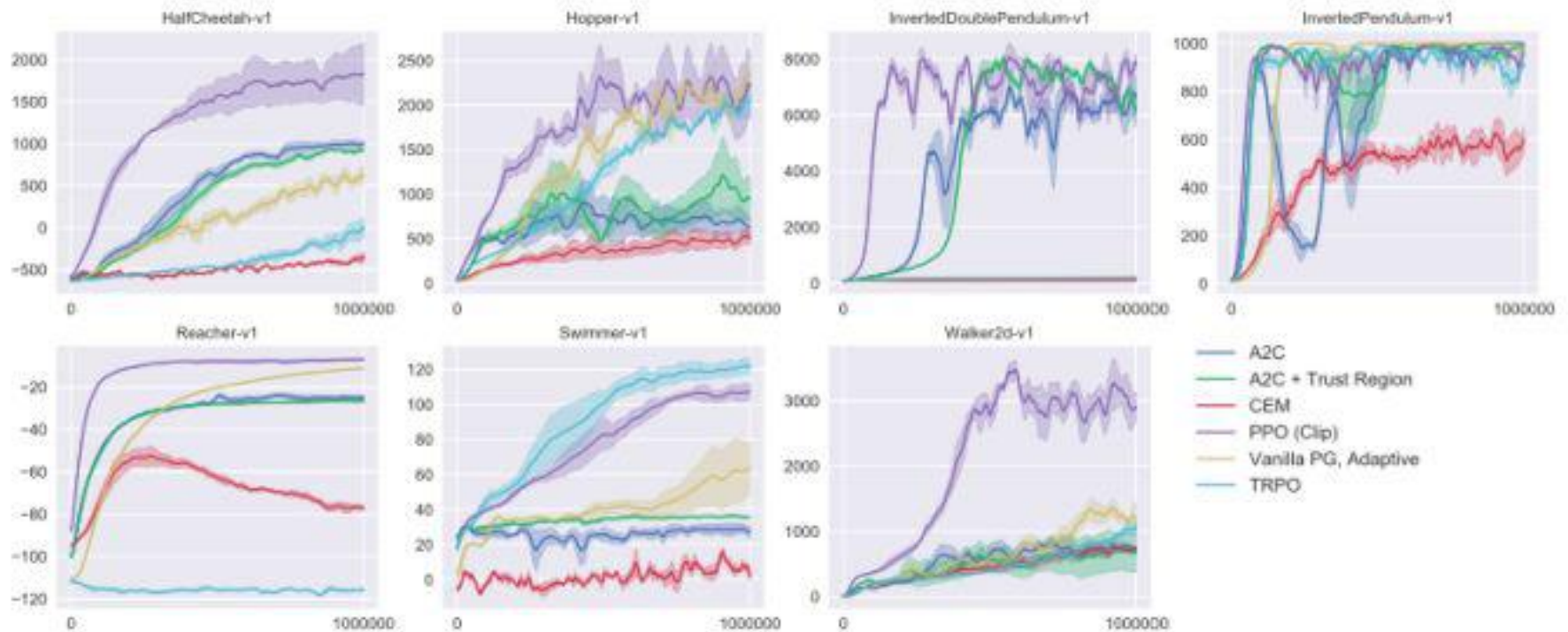


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.