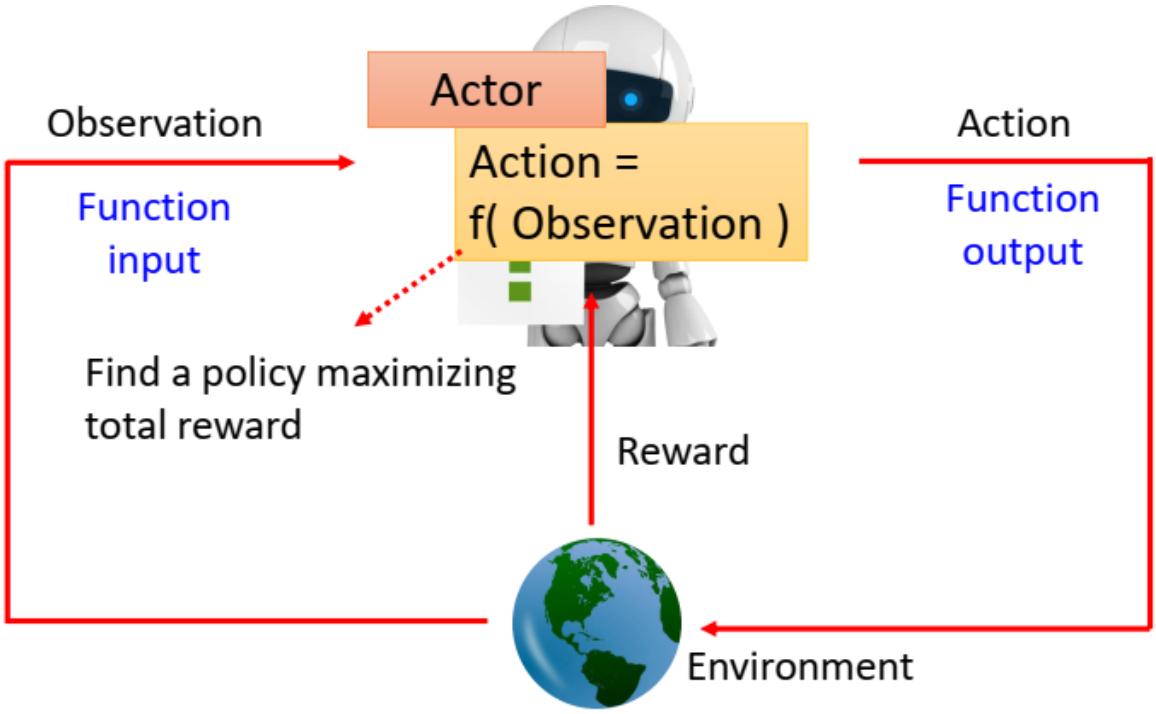
Reinforcement learning

Finding a function to find the best solution

Reference: Machine Learning 2021 by Dr. Hung-yi Lee

Observation
Actor
Action
Environment

Actor is a function which we are looking for
Environment will continuously give actor reward
Maximizing total reward

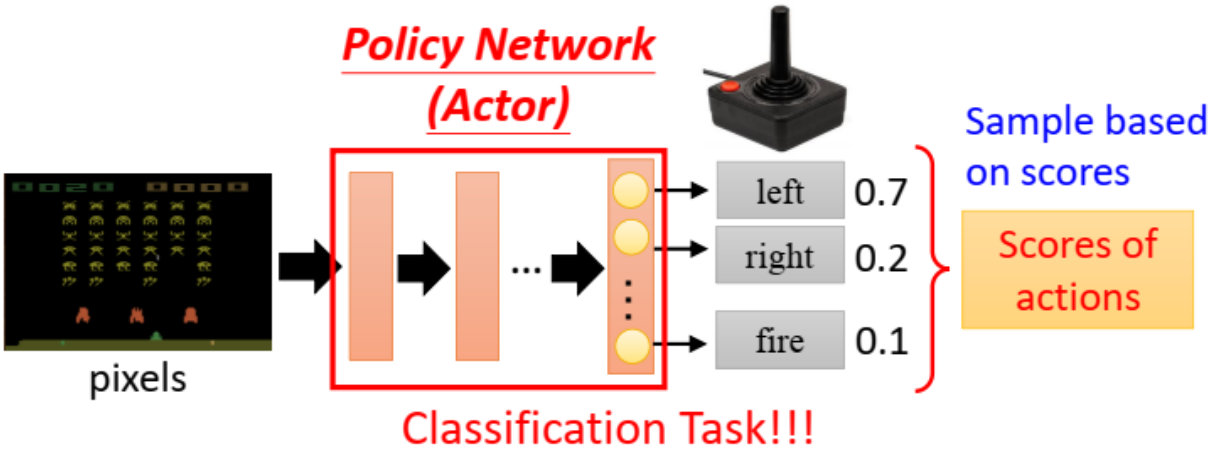


Machine learning 3 steps:

1. Function with unknown
2. Define loss from training data
3. Optimization

Function with unknown: Policy Network (Actor)

Policy Network (Actor)
Input: Observation (s_n)



Output: Action
See history: RNN
See pictures: CNN
Only see parameters

Final action we use sampling: we will add the random behavior to the output

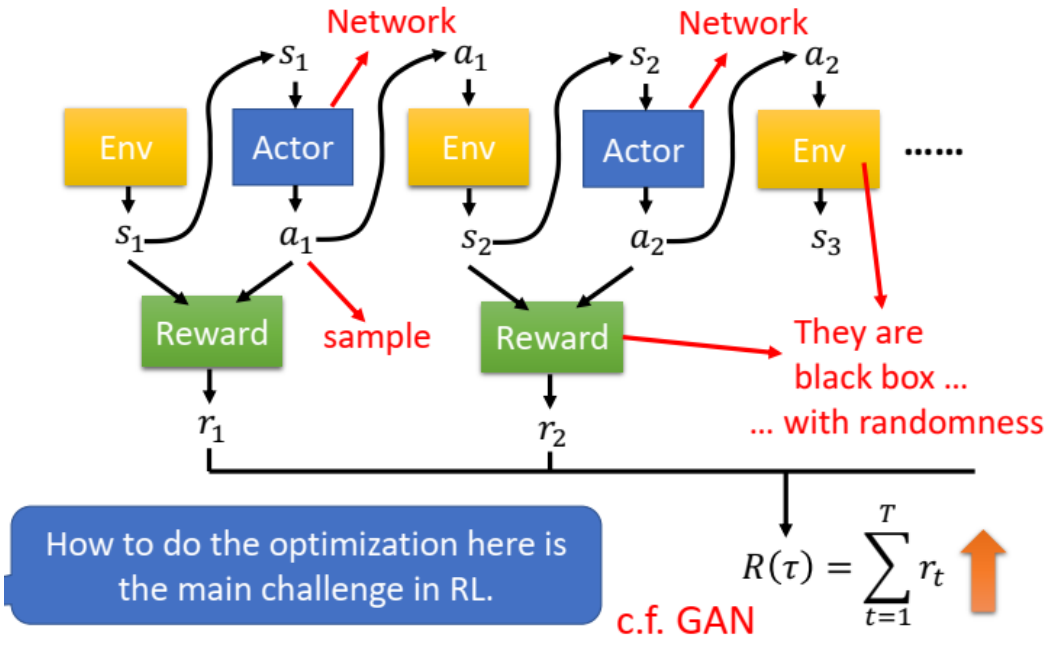
Define loss

Reward (r_T) or Return
Total Reward R for each episode

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

We need to maximize R or to minimize -R. Therefore -R is our loss.

$$loss = -R = -\sum_{t=1}^{T} r_t$$



Optimization

Environment and Reward are black box with randomness
Trajectory: The process about the interaction

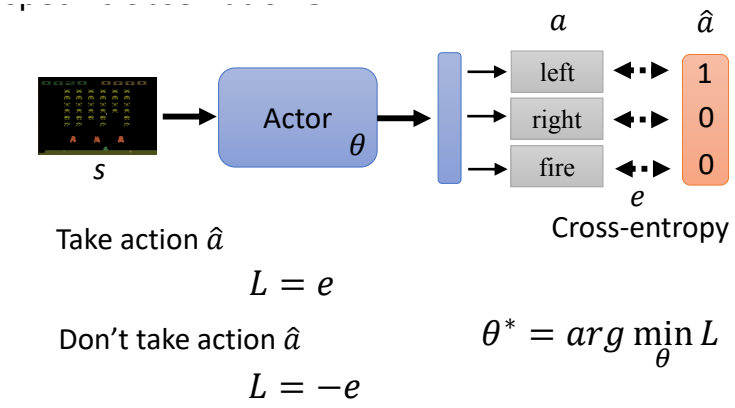
$Trajectory(\tau) = \{s_1, a_1, s_2, a_2, \ldots, s_n, a_n\}$

Reward can be seen as a function of observation and action. $r = f(s, a)$
Action (a) is generated by sampling. (With Random property)
Environment, Reward : a black box
隨機性 Randomness : Environment, Reward

The difficulty of reinforcement learning: How to optimization those randomness.
Actor is like Generator in GAN and environment, reward are like discriminator.
We adjust parameters in Generator to maximize discriminator.

Because of the randomness of Environment and Reward, we can not use Gradient Descend to optimize the parameters.

Policy Gradient



Just like a classification problem

$loss(L) = CrossEntropy(e)$

E1 : Crossentropy of doing a
E2 : Crossentropy of not doing a

$A^{\theta}(s_t, a_t) = R(\tau^n) - b$: Advantage function

$L = e_1 - e_2$

$\theta* = argmin(L)$

$$\{S_t, \hat{a}_t\} = score_t(A_t)$$

$$L = \sum A_n e_n$$

$$\theta * = argmin(L)$$

Here A means how much do you want to take the action!

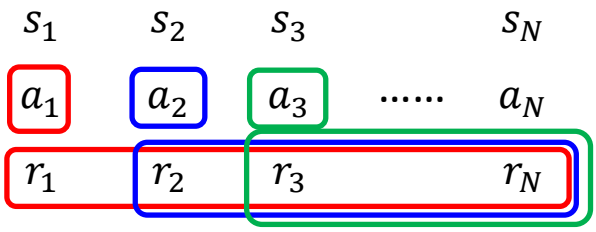
$$p_\theta(\tau) = p(s_1)p_\theta(a_1 \mid s_1)p(s_2 \mid s_1, a_1)p_\theta(a_2 \mid s_2)p(s_3 \mid s_2, a_2) \ldots$$

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

How to define a, action

- An action affects the subsequent observations and thus subsequent rewards.
- *Reward delay*: Actor has to sacrifice immediate reward to gain more long-term reward.
- In space invader, only “fire” yields positive reward, so vision 0 will learn an actor that always “fire”.

Version 0 is bad and you usually make this mistake when you implement reinforcement learning.



$$G_1 = r_1 + r_2 + r_3 + \ldots\ldots + r_N$$

$$G_2 = r_2 + r_3 + \ldots\ldots + r_N$$

$$G_3 = r_3 + \ldots\ldots + r_N$$

cumulated reward

Version1

Now we use cumulated reward (G)

$$G_1 = r_1 + r_2 + \ldots + r_n$$

Version 2

But rn might be less related to a1
We add discount factor gamma which is less than 1.

$G_1' = r_1 + \gamma r_2 + \gamma^2 r_3 + \dots$

Therefore

$$G_t' = \sum_{n=t}^{N} \gamma^{n-t} r_n$$

Version 3

* Different RL papers focus on A (which is preferable probability of RL model)

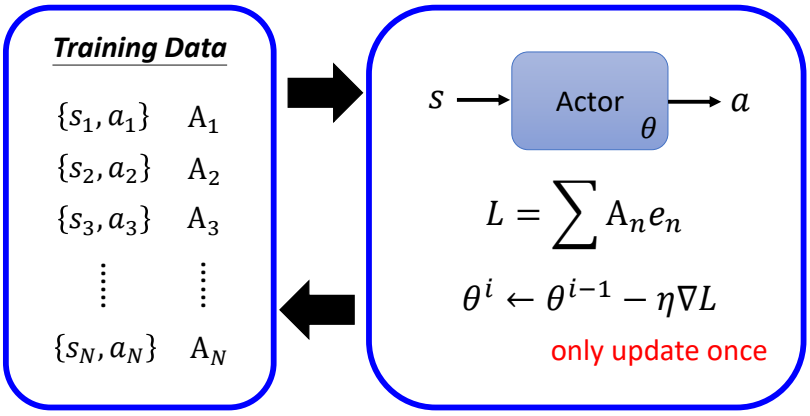
標準化 Standardization
Good or bad reward is "relative"
Minus by a baseline b to make G' have positive and negative values.

$A_n = G_N' - b$

HOW TO SET A GOOD BASELINE?

- Initialize actor network parameters θ^0
- For training iteration $i = 1$ to T
  - Using actor θ^{i-1} to interact
  - Obtain data $\{s_1, a_1\}, \{s_2, a_2\}, \dots, \{s_N, a_N\}$
  - Compute $A_1, A_2, \dots, A_N$
  - Compute loss L
  - $\theta^i \leftarrow \theta^{i-1} - \eta \nabla L$

Data collection is in the "for loop" of training iterations.



Each time you update the model parameters, you need to collect the whole training set again.

Policy gradient

When RL collects data, it is in the for loop. Very time consuming.

η: learning rate

θ: Actor

θ^{i-1}observations only can be used to train θ^{i-1}

On-policy

The actor to train and the actor for interacting is the same. → On-policy

Off-policy

Can the actor to train and the actor for interacting be different? → Off-policy

θ^i can learn by the observations by θ^{i-1}

E.g. PPO Proximal Policy Optimization

Expected Reward

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

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

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

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

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

The actor to train has to know its difference from the actor to interact.

Exploration

The actor needs to have randomness during data collection.
A major reason why we sample actions.

Actor-Critic - to evaluate actor θ

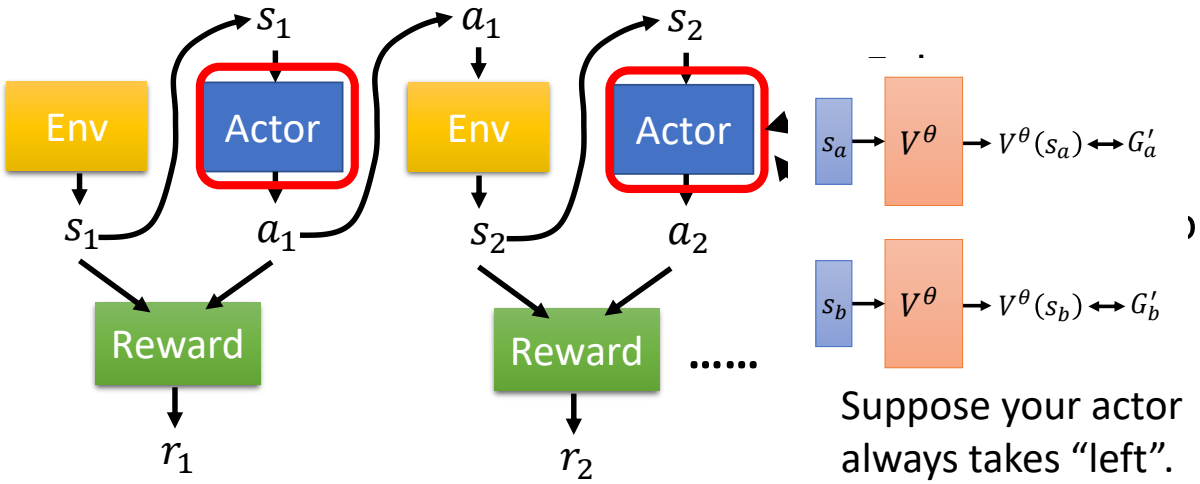
Critic: Given actor θ, how good it is when observing s (and taking action a)

Value function : When using actor θ, the **discounted *cumulated* reward** expects to be obtained after seeing s.

$V^\theta(s)$

Therefore $V^\theta(s)$ will guess the G' before finishing the episode.

How to train value function:



Suppose your actor always takes "left".

The actor needs to have randomness during data collection.

A major reason why we sample actions. ☺

We never know what would happen if taking "fire".

Monte-Carlo (MC) based approach

The critic watches actor θ to interact with the environment.

After seeing S_a,

Until the end of the episode, the cumulate reward is G'_a

After seeing S_b,

Until the end of the episode, the cumulate reward is G'_b

Temporal-difference (TD) approach $V^\pi(s)$

Good if the game is very long.

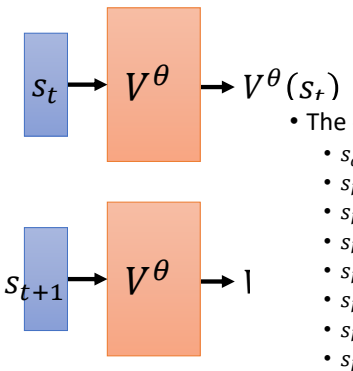
$\cdots s_t, a_t, r_t, s_{t+1} \cdots$

(ignore the expectation here)

$$V^\theta(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \cdots$$

$$V^\theta(s_{t+1}) = r_{t+1} + \gamma r_{t+2} + \cdots$$

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



- The critic has observed the following 8 episodes
  - $s_a, r = 0, s_b, r = 0$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 1$, END
  - $s_b, r = 0$, END

$V^\theta(s_b) = 3/4$

$V^\theta(s_a) =$? 0? 3/4?

Monte-Carlo: $V^\theta(s_a) = 0$

Temporal-difference:

$$V^\theta(s_a) = V^\theta(s_b) + r$$

3/4 3/4 0

(Assume $\gamma = 1$, and the actions are ignored here.)

Comparison of MC v.s.TD

The value function might be different when you apply MC v.s. TD.
Example:

Monte-Carlo: If we only see the end of the game.
Each s_t has relations.

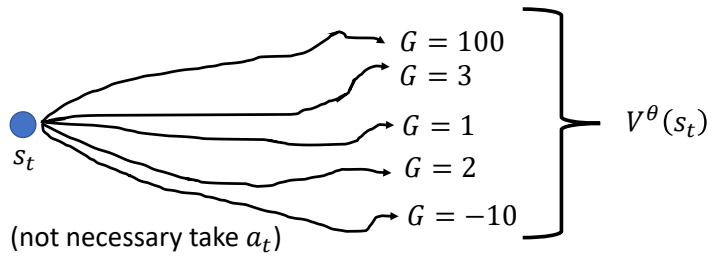
Temporal-difference (TD) approach
$V^{\theta}(S_a) = V^{\theta}(S_b) + r$
Each S_t can be seen independent.

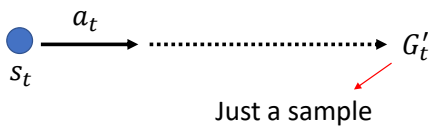
Application of Critic

Plan A: Use one sample to subtract the average.

The score from $V^{\theta}(s)$ can be b (normalization).
$\{s_t, a_t\}\ A_t = G'_t - V^{\theta}(st)$
Because we have randomness in actor, therefore $V^{\theta}(s_t)$ is an expect value.
You sample the actions based on a distribution.

s_t not necessary to take a_t
If $A_t > 0$ means a_t is better than average which means $a_t > V^{\theta}(s_t)$.
If $A_t < 0$ means a_t is worse than average which means $a_t < V^{\theta}(s_t)$.



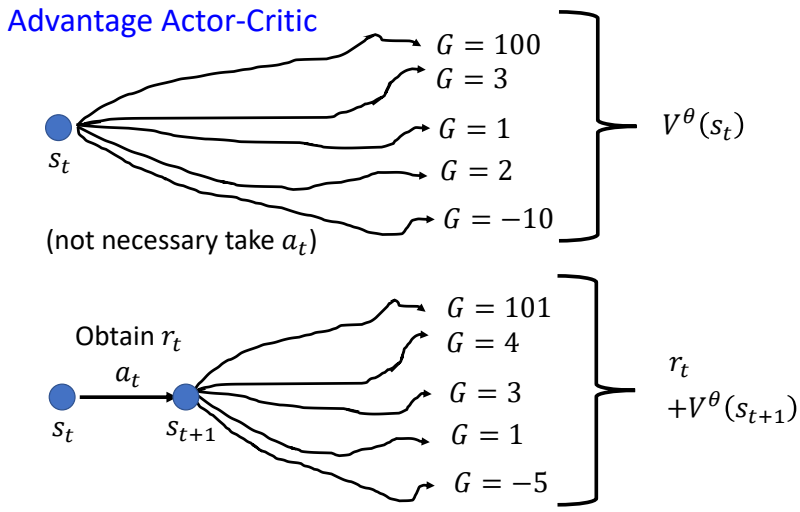


$A_t > 0$
a_t is better than average.

$A_t < 0$
a_t is worse than average.

Plan B: Use average to subtract average

Advantage Actor-Critic



$\{s_t, a_t\}$ $A_t = r_t + V^\theta(s_{t+1}) - V^\theta(s_t)$

Tip of Actor-Critic

The parameters of actor and critic can be shared.
Actor: network
Critic: network
Actor and critic can share the first few layers. -> just different outputs.

Question part:
If s_a does not follow s_b: That mentions the importance of sampling.
Depends on luck!

Deep Q Network (DQN)

Only use critic $V^\theta(s_t)$!
Rainbow

Reward Shaping

Once the Reward R is sparse (Sparse Reward)
If $r_t = 0$ in most cases, we won't know actions are good or bad.

We need to define extra rewards to guide agents θ.
The developers define extra rewards to guide agents.

Need **Domain knowledge**!

Sample: VizDoom

Curiosity

Obtaining extra reward when the agent sees something new (but meaningful)
We need to overcome the nonsense "NEW" like noise.

If there is no reward…

1. Even define reward can be challenging in some tasks.
2. Hand-crafted rewards can lead to uncontrolled behavior.

Imitation Learning

Actor can interact with the environment, but reward function is not available.
We have demonstration of the expert. Human expert show and demo to train.

Each $\hat{\tau}$ is a trajectory of the export.
$\{\hat{\tau}_1, \hat{\tau}_2, \ldots, \hat{\tau}_k\}$
Examples:
Self driving: record human drivers
Robert: grab the arm of robot

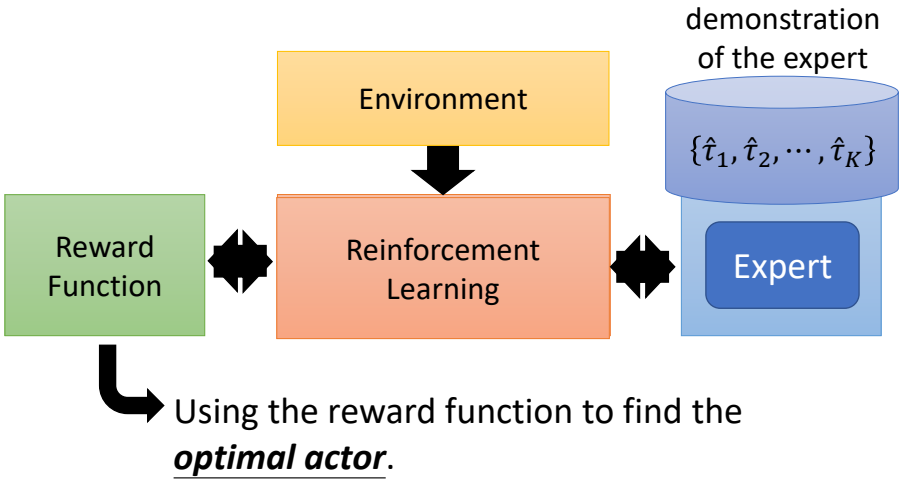
Supervised learning?

Behavior Cloning

Problem:
1. The expert only sample limited observation.
2. The agent will copy every behavior, **even irrelevant actions**.

When the ability of machine is limited.

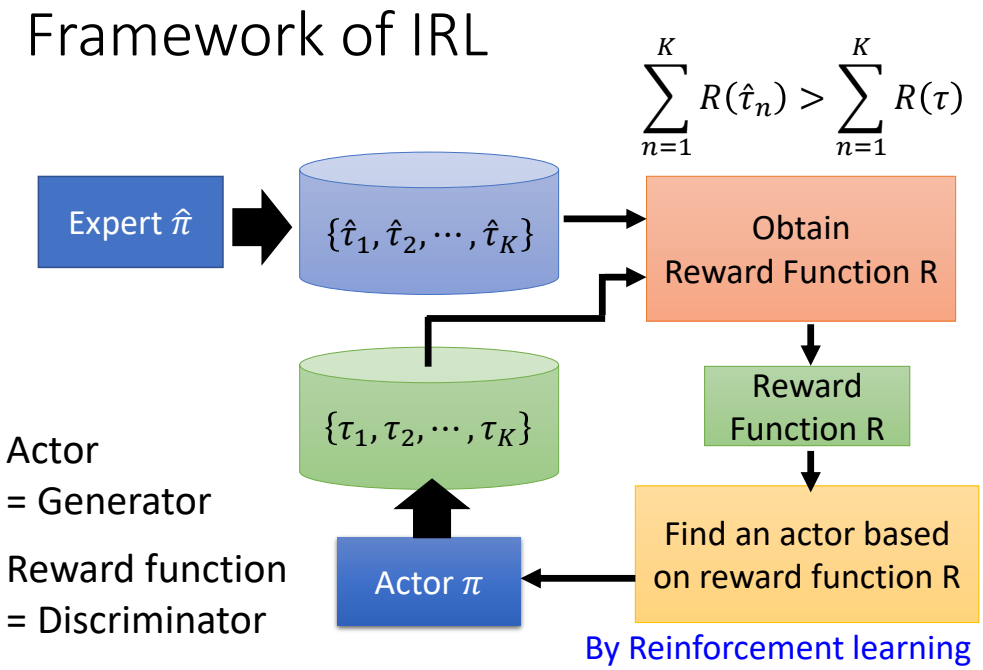
Inverse Reinforcement Learning



Use machine to define the reward.
To learn reward function!
A simple reward function doesn't mean the action function will be easy.
Principle: The teacher is always the best.

Basic idea:
Initialize an actor
In each iteration

1. The actor interacts with the environments to obtain some trajectories.
2. Define a reward function, which makes the trajectories of the teacher better than the actor.
3. The actor learns to maximize the reward based on the new reward function.
4. Output the reward function and the actor learned from the reward function.

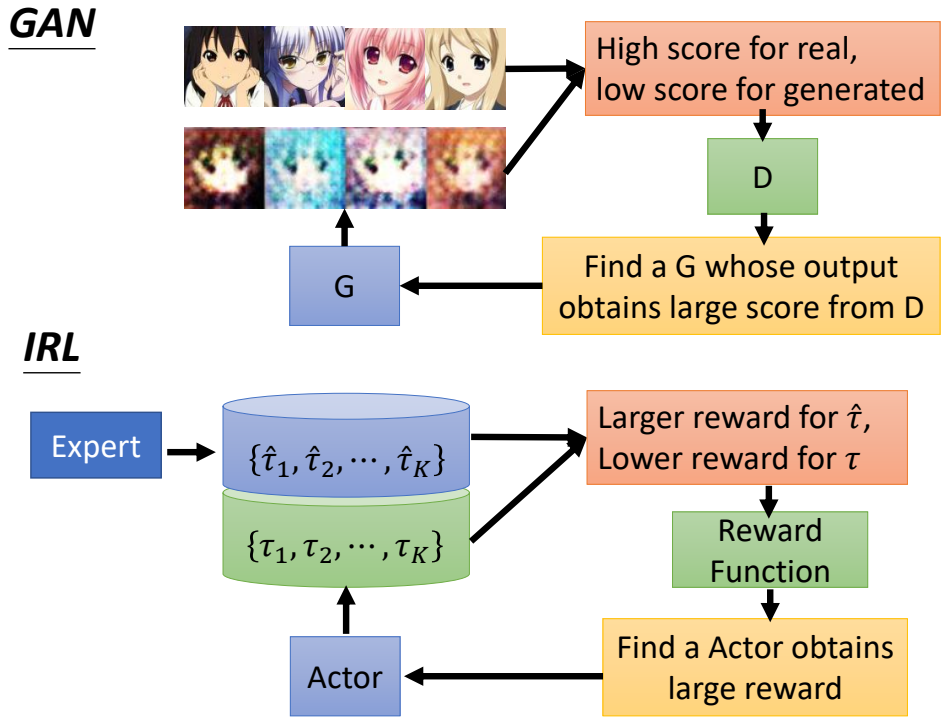


Expert: $\hat{\tau}$

Actor+Environment: τ

Just like **GAN**

Reward function: Result is from Expert: high score will be given otherwise low score will be assigned if the result is from computer.



Guided Cost Learning

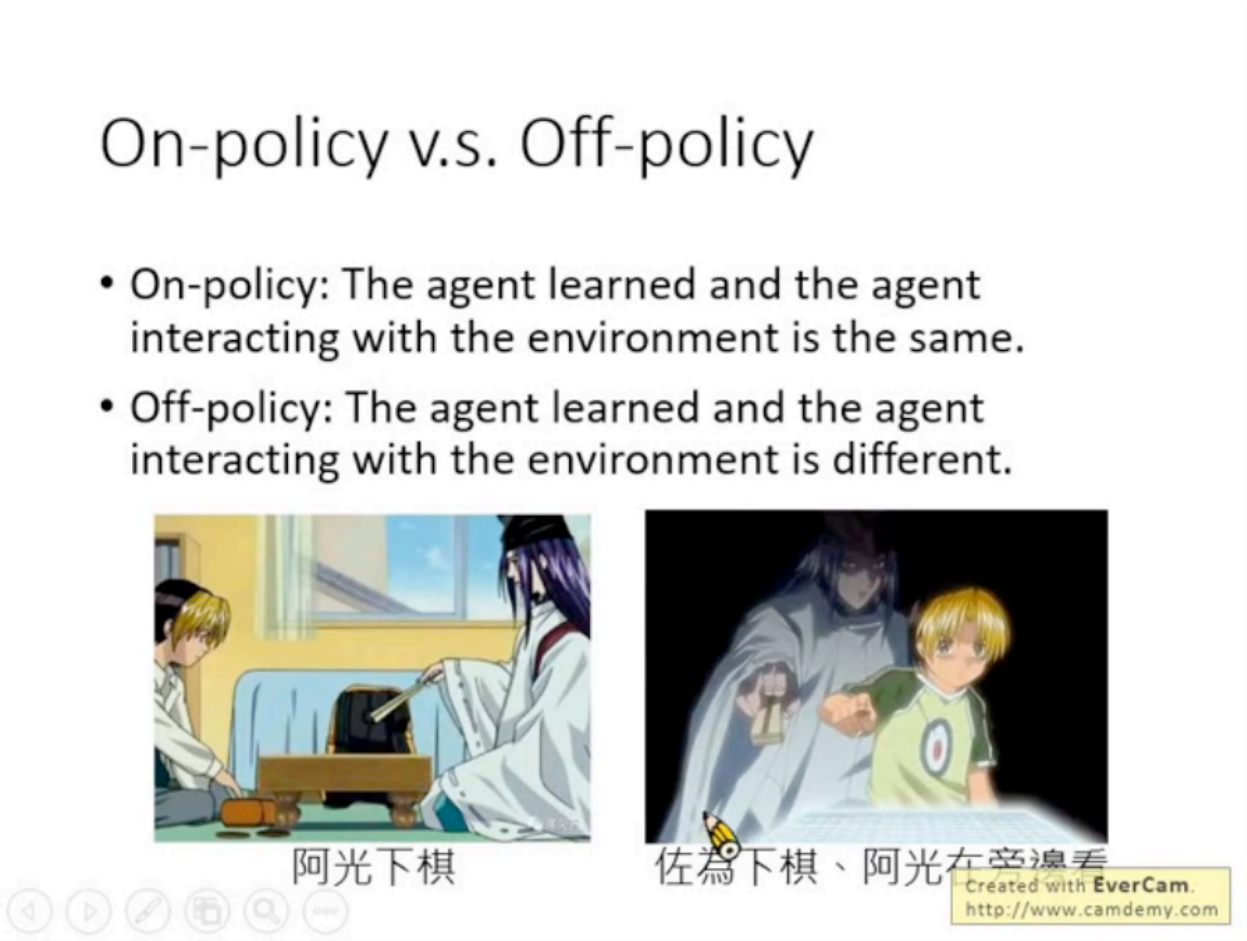
Skew-Fit

Proximal Policy Optimization (PPO)

Deep Q-learning

Optimization(PPO)，Default algorithm for OpenAI 。

1. Policy gradient
2. On-policy
3. Off-policy
4. Add constrain



Using the same agent to train and to interact with environment agent - On-policy，if not : Off-policy. In other words, Learning by doing is On-policy and learning by observation other people's behavior: Off-policy。

On-policy: Learn and interaction with environment with the same agent.
Off-policy: Learn and interaction with environment with different agents.

On-policy

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

$E_{\tau \sim p_\theta(\tau)}$ is to sample a trajectory-τ from policy-θ to calculate expected value. Therefore $p_\theta(\tau)$ changes after updating θ to θ'. At this time, resampling is required and you can not apply the previous sampled trajectory-τ .

That's why Policy Gradient spends a lot of time on sampling data because all the data can only update parameters once. After updating, we need to do sampling again.

Off-policy

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 therefore we can re-use sample data.

The advantage to shift from on On-policyto Off-policy is that we hope to use only one collected θ' to train θ. This means that we can repeatly use those data to update θ to increase training efficiency. There is a related solution, importance sampling and it is not only used on RL.

Importance Sampling

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

$$\int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx$$

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

x^i is sampled from p(x)

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

Weight:$\frac{p(x)}{q(x)}$ to cancel the difference of $p(x), q(x)$

Issue of importance Sampling

In implementation, the difference between p and q can not be huge because the variance will be different.

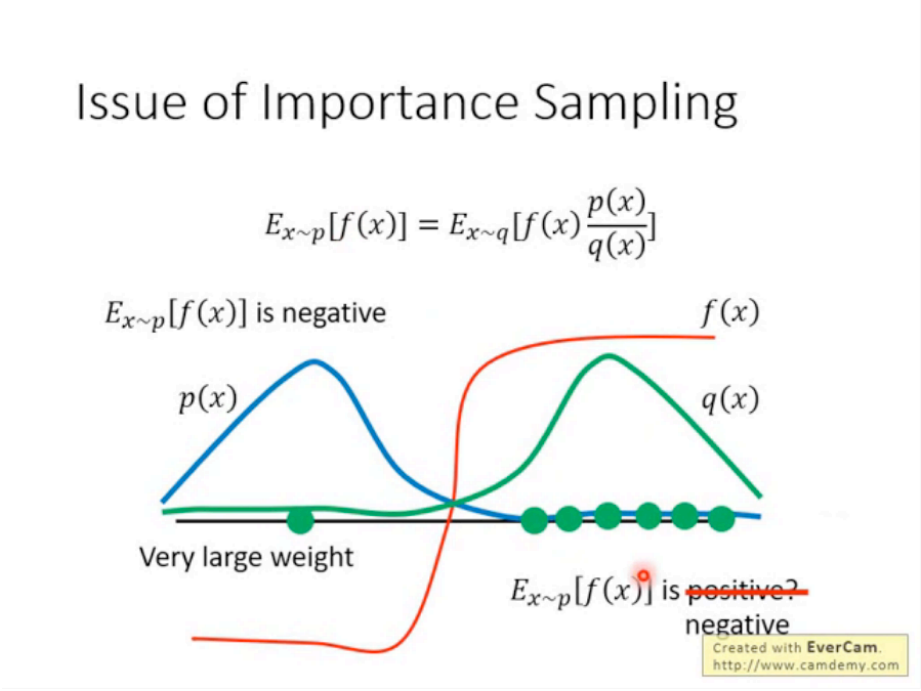
$Var_{x\sim p}[f(x)]$

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

Having the same mean value of two random variables does not mean they have the same variances. We can use equation to calculate:

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

The different of variance: $\frac{p(x)}{q(x)}$



If sampling is not enough, ….

Application to Off-policy

Use θ' to interact with environment.

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

Sample the data from θ'

Use the data to train θ many times.

Mentioned in the beginning of class, when we update parameters, we calculated the score for each state-action pair and we do not assign the same score to whole trajectory-**τ.**

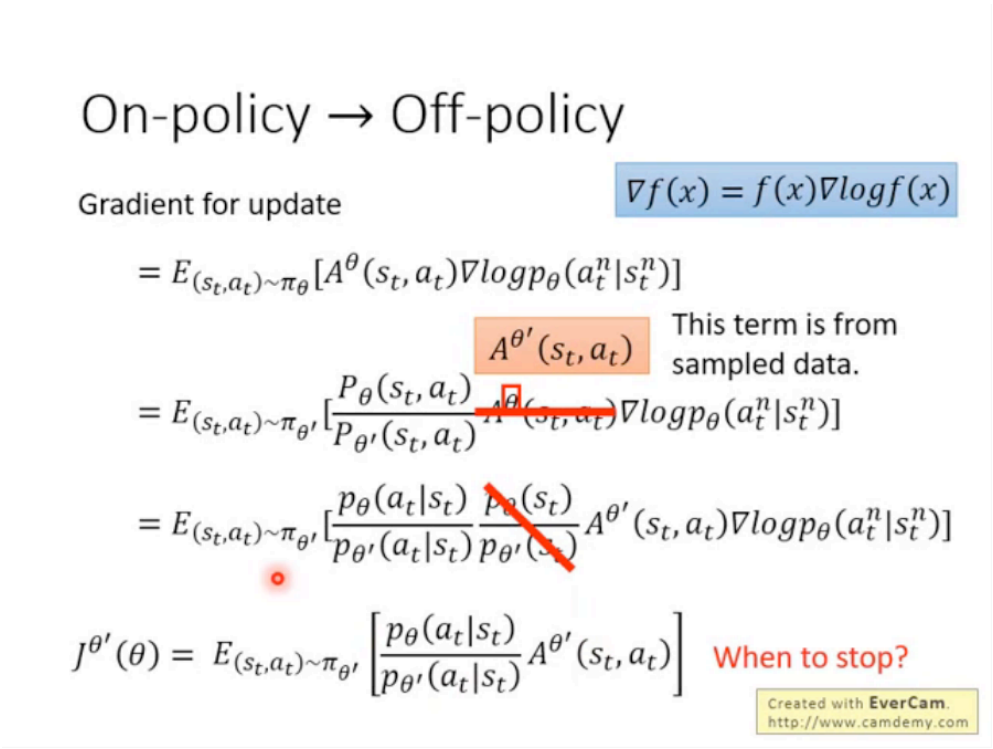
$E_{(st,at)\sim\pi\theta}[A^{\theta}(s_t,a_t)\nabla\log p_\theta(a_{nt}|s_{nt})$

Then we sample the state and action from θ and calculate the average. If the average is good, we will increase the probability for the action, and once the average is bad, we can lower the probability of the action.

On-policy

How to use importance sampling to shift from On-policy to Off-policy:

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



- Here we need to pay attention that advantage function calculates the estimated value from θ' 所not θ because currently it is θ' interacting with the environment.
- We assume that the probabilities are similar when the model sees s_t at θ and θ'. Therefore we can eliminate $\frac{p_\theta(s_t)}{p_{\theta'}(s_t)}$ because $\frac{p_\theta(s_t)}{p_{\theta'}(s_t)} \approx 1$.
- Another way to explain it is that it is difficult to estimate s_t, therefore we ignore it.

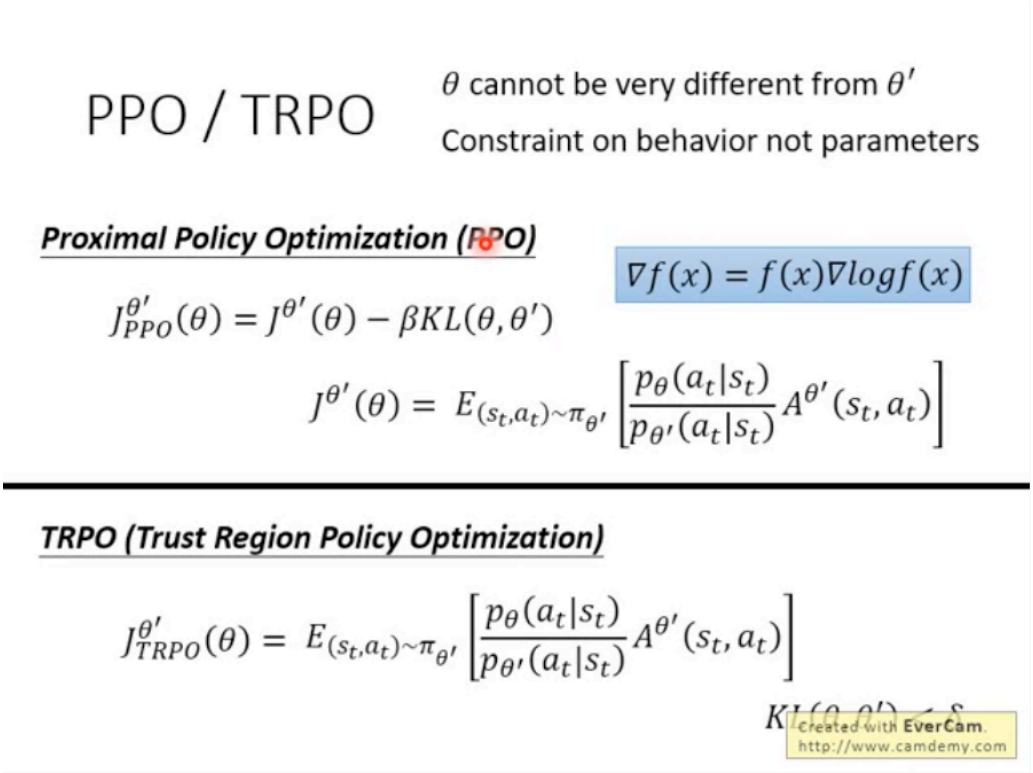
Gradient:

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

J : Objective function

Proximal Policy Optimization (PPO)

Add constrain $\beta KL(\theta, \theta')$ on behavior not parameter.



When we do importance sample, we can not let the difference of θ and θ' too big. Thus the solution of PPO is adding a constrain(約束項), $\beta KL(\theta, \theta')$.
$J_{\theta' PPO}(\theta) = J^{\theta'}(\theta) - \beta KL(\theta, \theta')$

KL-divergence $KL(\theta, \theta')$

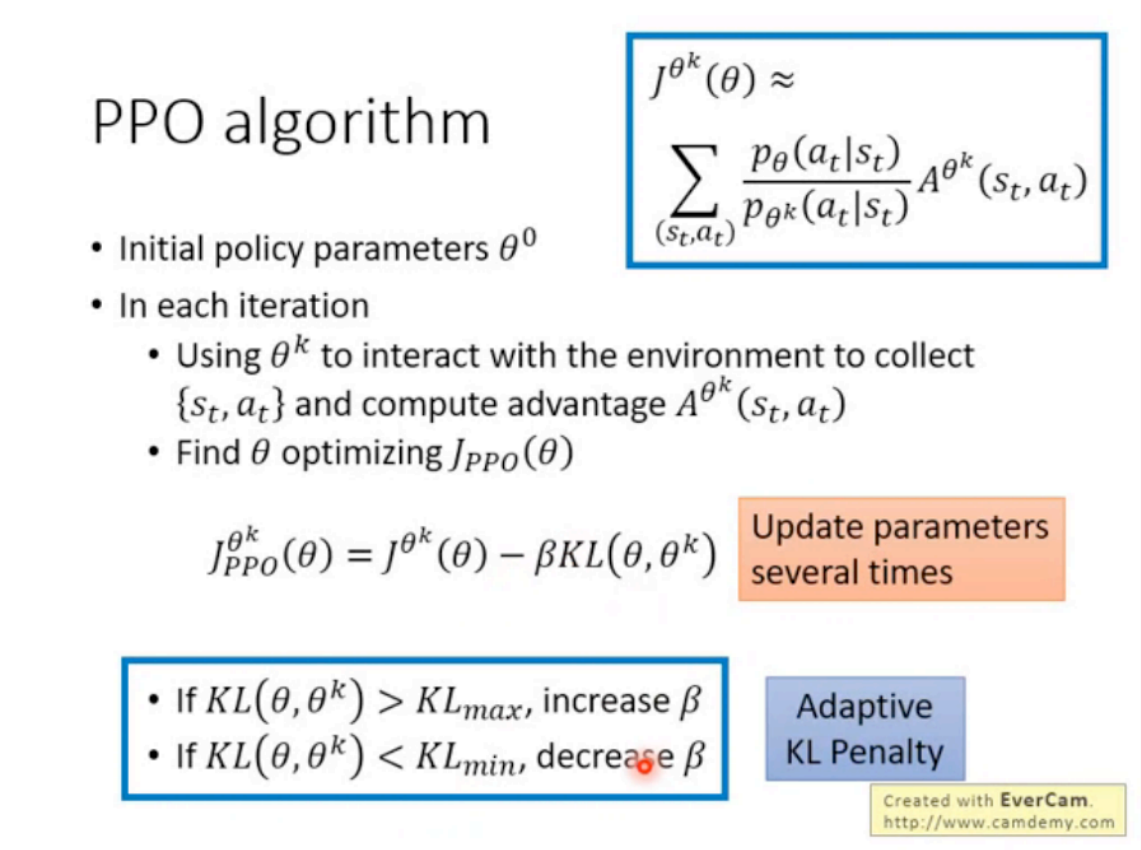
$KL(\theta, \theta')$ indicates the KL-divergence of two output actions from two models. It evaluates how similar they are and we hope they can be as similar as possible. Therefore we add the constrain, $\beta KL(\theta, \theta')$.

True Region Policy Optimization

In PPO, we directly put constrain $KL(\theta, \theta')$ into the equation to be optimized and then use gradient ascent to maximize it.
But in TRPO, $KL(\theta, \theta')$ is seen as an extra constrain and it needs to satisfy $KL(\theta, \theta') < \delta$. It is **difficult to handle by gradient-based optimization.**

Another thing, $KL(\theta, \theta')$ is the **distance between two behaviors**, not the distance between two parameters. It means **the difference of the model output actions** when the same state is given. Here the model outputted action is a distribution of probability. We can average the KL-divergence from those two output distributions in different state, and it is $KL(\theta, \theta')$.

The reason we do not consider the distance of parameters of θ and θ' is that a little change of parameters can cause uneven effect on actions. The effect can be huge or small. However we focus on the difference of behaviors not parameters. Thus we only consider the distance of actions of θ and θ'.
From literatures, PPO and TRPO have similar efficiency but **PPO is easier to implement.**



Algorithm:
1. Initialize the parameters of the policy θ^0,
2. In every iteration, use θ^k to interact with environment and get a bunch of $\{s_t, a_t\}$, then calculate the advantage $A^{\theta^k}(s_t, a_t)$
3. Use the parameters of actor which fetched from last training iteration of θ^k to optimize θ.不
Here we can use the data from θ^k to train our model multiple times and it is different from policy gradient.

The object function is:
$J_{\theta^k PPO}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k)$

Update β dynamically

Set a acceptable max and min $KL(\theta, \theta^k)$

- If the value calculated is over the maximum, which means the punishment is not effective, we increase β.
- If the value calculated is lower than the minimum, which means the punishment is too strong we decrease β.

PPO2

PPO algorithm

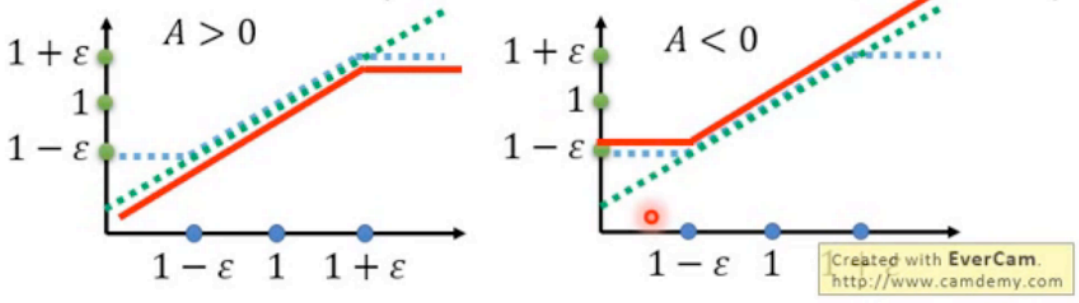
$$J_{PPO}^{\theta^k}(\theta) = J^{\theta^k}(\theta) - \beta KL(\theta, \theta^k)$$

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

PPO2 algorithm

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

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



Calculation of KL-divergence can be a little bit complicated for PPO. PPO2 is trying to solve this issue. Even the equation of PPO2 looks complicated, but it is easy in implementation.

$\sum s_t, a_t$: summation the pair over state, action

There are two parts in the equation. The function of min is to select smaller items.

$clip(\frac{p_\theta(a_t|s_t)}{p_\theta^k(a_t|s_t)}, 1-\epsilon, 1+\epsilon)$

There are three items. When $\frac{p_\theta(a_t|s_t)}{p_\theta^k(a_t|s_t)}$ is less than $1-\epsilon$, $1-\epsilon$ will be returned. Also if $\frac{p_\theta(a_t|s_t)}{p_\theta^k(a_t|s_t)}$ is bigger than $1+\epsilon$, $1+\epsilon$ will be returned.

This is just like the weight clipping in WGANs to restrict the value in a range.

ϵ is a hyperparameter.

Explanation of the figure:
Let's see the second item, clip
$clip(\frac{p_\theta(a_t|s_t)}{p_\theta^k(a_t|s_t)}, 1-\epsilon, 1+\epsilon)$
And the part it handles.: Blue dot line
X axis is $\frac{p_\theta(a_t|s_t)}{p_\theta^k(a_t|s_t)}$.

Y axis is the export of clip.
If the export of clip is greater than $1+\epsilon$, $1+\epsilon$ will be exported.
Once the export is less than $1-\epsilon$, $1-\epsilon$ will be exported.
If the export of clip is between $1+\epsilon$ and $1-\epsilon$, the value will be exported.
The first item is represented by the green dot line.
The export of advantage function-A affects the value range (red line).

- When $A > 0$, this pair is good, and we hope to increase its probility of existance (p_θ). Which means we want to make p_θ as big as possible. But the fraction of p_θ and p_θ^k can not exceed $1+\epsilon$.
- $A < 0$ indicates that the pair is not preferred and we hope to decrease its probility of existance (p_θ) but the fraction of p_θ and p_θ^k can not be less than 1−ϵ.