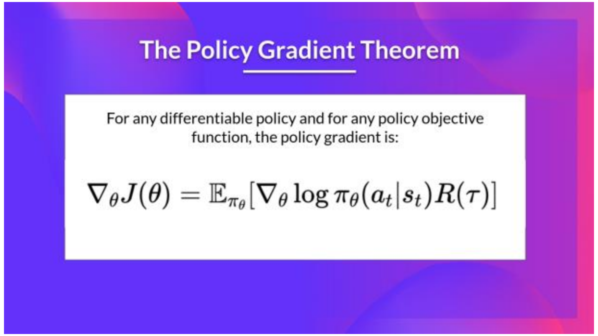
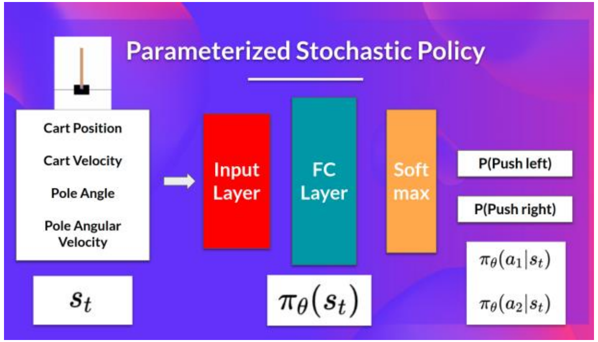
Proximal Policy Optimization (PPO) Algorithm
OpenAI

”PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance”

Brief Recap of Policy Gradient (REINFORCE)

What is Policy Gradient Methods ?





Notations and Definitions:

- s_t: the state at time step t within an episode
- a_t: the action taken at time step t within an episode
- $J(\theta)$: Expected return of the policy parameterized by θ.
- $R(\tau)$: Expected cumulative reward for taking action a in state s and following π.
- $\pi_\theta(a_t|s_t)$: Policy mapping states to actions using θ.
- $E_\pi[\cdot]$: Expectation under the policy π.
- ∇_θ: Gradient with respect to the policy parameters θ.

- **Policy Gradient Methods**:
  - The goal of Reinforcement Learning is optimizing the policy parameters to maximize the expected reward.
  - When optimizing the policy, we need to find the direction in which the expected reward increases the most.
  - Optimize the parameter θ directly by performing the gradient ascent $\theta \leftarrow \theta + \alpha * \nabla_\theta J(\theta)$ on the performance of the objective function.

Brief Recap of Policy Gradient (REINFORCE)

Weaknesses of Policy Gradient (REINFORCE)

- **Unstable update**: Step size is very important.
  - *Step size is too large* -> Generate bad policy -> Collect bad samples
  - *Step size is too small* -> The learning process is slow

- **Data Inefficiency**:
  - Learn a policy directly from the data generated by the curren policy -> sensitive to the current policy's performance -> new set of trajectories for every new policy
  - Set of trajectories is used only once for a single gradient update -> prevents it from leveraging the full potential of the collected experiences

Algorithm 1 REINFORCE Algorithm

Require: Policy $\pi_\theta(a|s)$, learning rate α

```
for episode = 1, 2, ..., M do
    Generate an episode (s_1, a_1, r_1, ..., s_T, a_T, r_T) by following policy π_θ(a|s)
    Initialize G ← 0
    for t = T, T − 1, ..., 1 do
        G ← γG + r_t
        θ ← θ + αG∇_θ ln π_θ(a_t|s_t)
    end for
end for
```

Notations and Definitions:

- $\pi_\theta(a|s)$: the policy, a function that maps states s to actions a with parameters θ
- α: the learning rate, a positive scalar controlling the size of the policy update
- M: the total number of episodes used for training
- s_t: the state at time step t within an episode
- a_t: the action taken at time step t within an episode
- r_t: the reward received at time step t within an episode
- T: the total number of time steps within an episode
- G: the return, a cumulative sum of rewards within an episode, discounted by the discount factor γ
- γ: the discount factor, a scalar between 0 and 1, used to weight the importance of immediate rewards over future rewards

Solving Data Inefficiency : Importance Sampling

What is Importance Sampling?

- **Importance Sampling**:
  - Eliminate the need to collect new trajectories for each update by using old policy to estimate the new rewards.
  - Do that by reweighting the rewards with the importance sampling ratio.

$$\nabla_\theta J(\theta) = E_{\tau \sim \bar{\pi}_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(\mathbf{a}_t|\mathbf{s}_t\right) \left(\prod_{t'=1}^{t} \frac{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\bar{\pi}_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

use old policy to sample data

old policy

- Estimate the expectation of a different distribution

$$\begin{aligned} \mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right] \end{aligned}$$

*Sample from q distribution to estimate p-distribution

Notations and Definitions:

- θ: Policy parameters to optimize.
- $\pi_\theta(a_t|s_t)$: Policy mapping states to actions using θ.
- $\pi_{\theta_{\text{old}}}(a_t|s_t)$: Old policy before optimization.
- $E_t[\cdot]$: Expectation over time steps in trajectories.
- $A(a_t|s_t)$: Estimated advantage of action a_t at state s_t.
- $D_{KL}[\cdot]$: Dissimilarity measure between old and updated policies.
- δ: Maximum allowed change in policy per optimization step.

Solving Unstable: Trust Region Policy Optimization

First look at the previous work!

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \le \delta$$

TRPO with objective function constrained by form of KL divergence

$$A(s,a) = Q(s,a) - V(s)$$

q value for action a in state s

average value of that state

$$V_\pi(s) = E_\pi[R_t | s_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s]$$

where $V_\pi(s)$ is the value of state s under policy π, E_π is the expectation under policy π, R_t is the return at time t, s_t is the state at time t, γ is the discount factor, and r_{t+k+1} is the reward at time t+k+1.

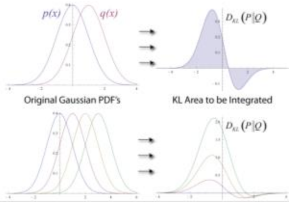
$$Q^\pi(s,a) = E_\pi[R_t | s_t = s, a_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a]$$

Measure the distance of two distributions

$$D_{KL}(P||Q) = \sum_x P(x) log \frac{P(x)}{Q(x)}$$

KL divergence of two policies

$$D_{KL}(\pi_1||\pi_2)[s] = \sum_{a \in A} \pi_1(a|s) log \frac{\pi_1(a|s)}{\pi_2(a|s)}$$



- **Trust Region Policy Optimization (TRPO) algorithm**:
  - *Key Idea*: Limit the size of each policy update -> new policy is not too far from the old one (using KL divergence) -> can maintain stability during learning!
  - Note :
    - Word “value" here refers to the expected cumulative return (total discounted reward that the agent accumulates over a trajectory).

Trust Region Policy Optimization (TRPO)

TRPO uses hard constraint

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$$

Hard constraint in form of KL divergence between old and new policy

Reason : Difficulty in choosing an appropriate penalty beta coefficient (soft constraint)

- If the coefficient too large -> The constraint will be too restrictive, hindering learning.
- If the coefficient too small -> The constraint will be violated too much, leading to unstable updates.

Problems with Trust Region Policy Optimization (TRPO)

Problem : Computationally Expensive

Algorithm 1 Trust Region Policy Optimization

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: Hyperparameters: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K

3: **for** $k = 0, 1, 2, ...$ **do**

4: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

5: Compute rewards-to-go $\hat{R}_t$.

6: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

7: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\big|_{\theta_k} \hat{A}_t.$$

8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k,$$

where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.

9: Update the policy by backtracking line search with

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k,$$

where $j \in \{0, 1, 2, ...K\}$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.

10: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

11: **end for**

A second-order optimization (conjugate gradient) is used to solve the constrained optimization problem!

#1 Key Idea of Proximal Policy Optimization (PPO)

PPO with Adaptive KL Penalty

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta \,\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

$$\text{Compute } d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

– If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$

– If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

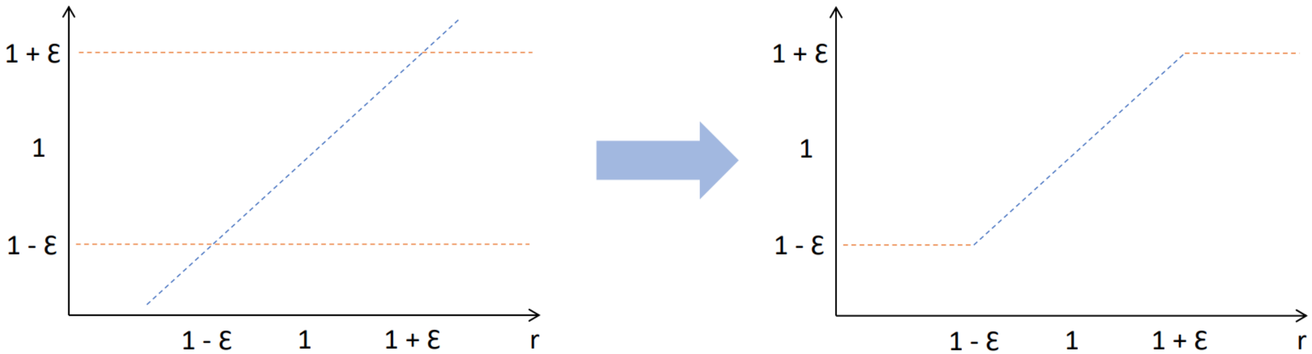
- **Adaptive KL Penalty**:
  - Hard to pick β value -> use adaptive penalty beta coefficient β
  - If the difference of two distribution (*d*) is too small - > soften the penalty
  - If the difference of two distribution (*d*) is too big - > add more penalty

#2 Key Idea of Proximal Policy Optimization (PPO)

PPO with Clipped Objective

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right]$$



$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

- **Clipped surrogate objective function**:
  - Unstable updates often happen when r changes too quickly -> limit r within a range of interval (1 −ϵ, 1+ϵ).

#2 Key Idea of Proximal Policy Optimization (PPO)

Clipped surrogate objective function

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Clipped Surrogate Objective function

- **Clipped surrogate objective function**:
  - *Key Idea*: If probability ratio is very big -> Clip it -> that value only lies within interval $(1-\epsilon, 1+\epsilon)$.
  - Take the minimum of the clipped and unclipped objective, so the final objective is a lower bound (i.e., a pessimistic bound) on the unclipped objective.
  - Eliminates the need to handle constraints -> simpler unconstrained optimization problem.
  - Can be solved using first-order methods like gradient ascent -> computationally less expensive compared to second-order methods.

PPO's Performance

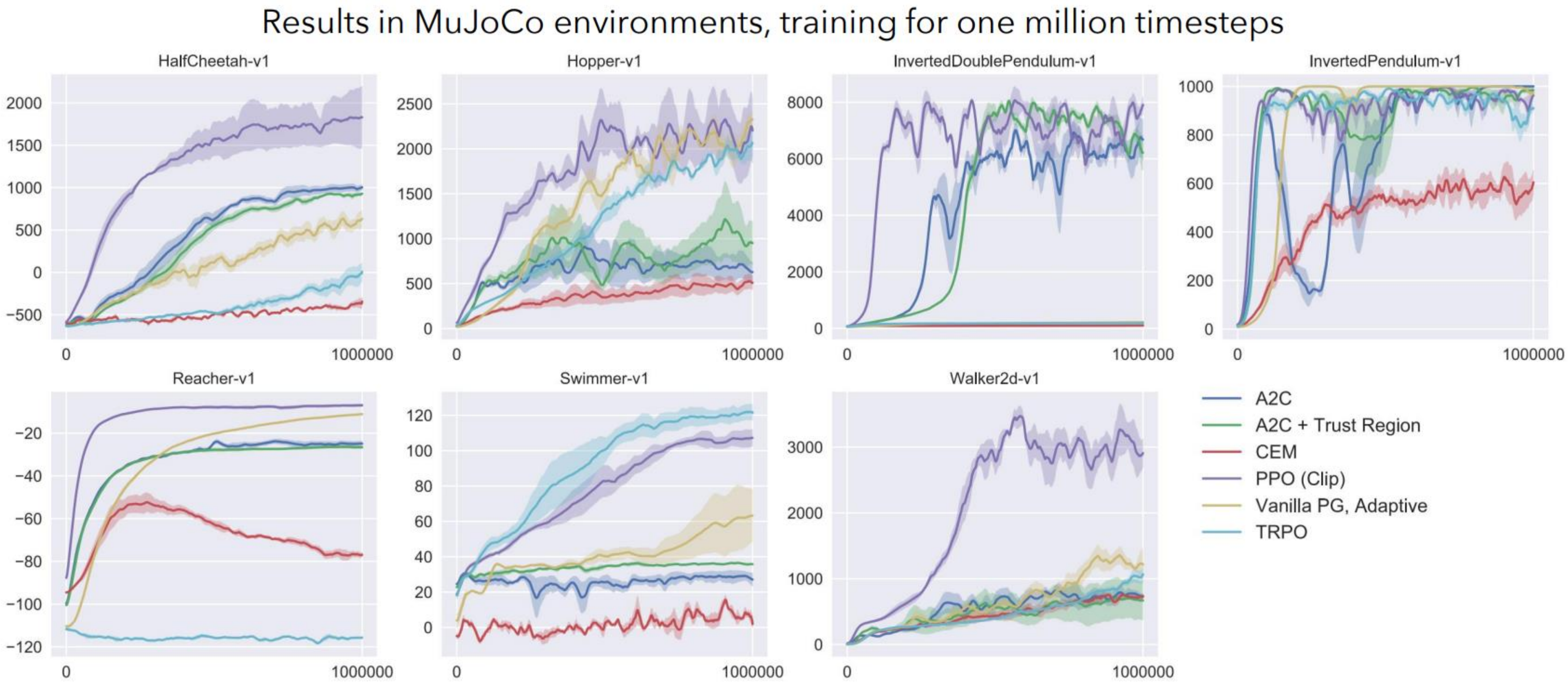
Proximal Policy Optimization (PPO) Performance

No clipping or penalty:	$L_t(\theta) = r_t(\theta)\hat{A}_t$
Clipping:	$L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$
KL penalty (fixed or adaptive)	$L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

PPO’s Performance

Proximal Policy Optimization (PPO) Performance



Proximal Policy Optimization (PPO) in Practice

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right]$$

Surrogate objective function

a squared-error loss for "critic"

$$(V_\theta(s_t) - V_t^{\text{targ}})^2$$

entropy bonus to ensure sufficient exploration

encourage "diversity"

* c1, c2: empirical values, in the paper, c1=1, c2=0.01

- **Breakdown of the function**:
  - Clipped surrogate objective -> optimize the policy while keeping the changes in the policy within a certain limit -> avoid large policy updates that could lead to instability.
  - A squared error loss -> make the predicted state-value function as close as possible to the target value function -> accurate approximation of the expected future returns for a given state.
  - Entropy bonus -> adding intensive for choosing actions with higher entropy -> can explore different parts of the state-action space.

Thank You