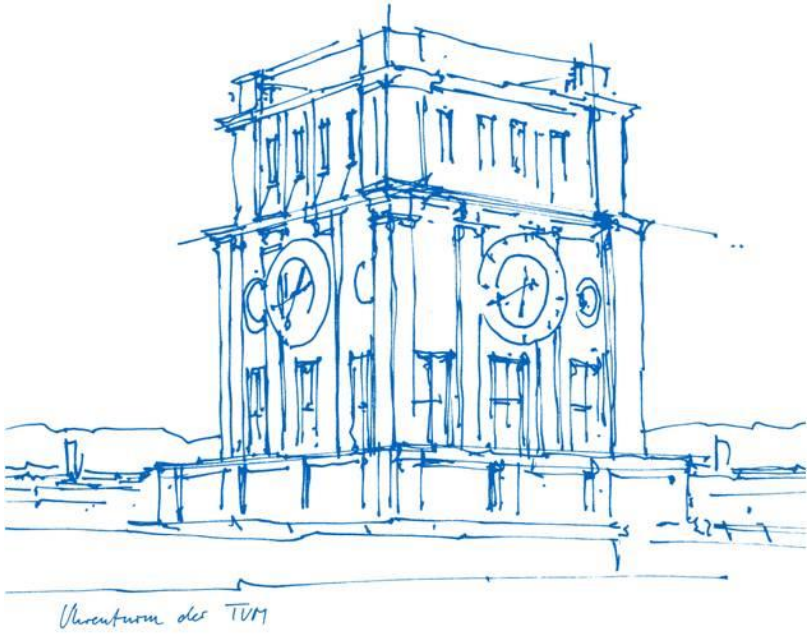
Policy Gradient and PPO

Recent Trends in Automated Machine Learning

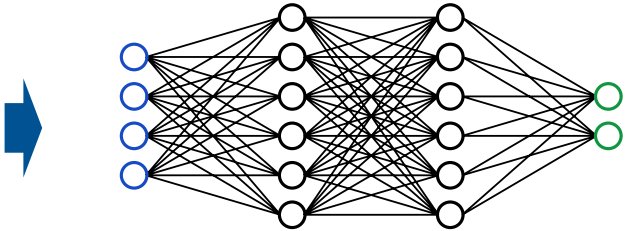
Technische Universität München

Yuanchun Shen

May 19th, 2021



Supervised Learning vs. Reinforcement Learning



o_t $-obervation$ $\pi_\theta(a_t|o_t)$ $-policy$ a_t $-action$

Basic components in RL:

Agent:

Environment:

An Atari Console

Policy:

Moves left when…

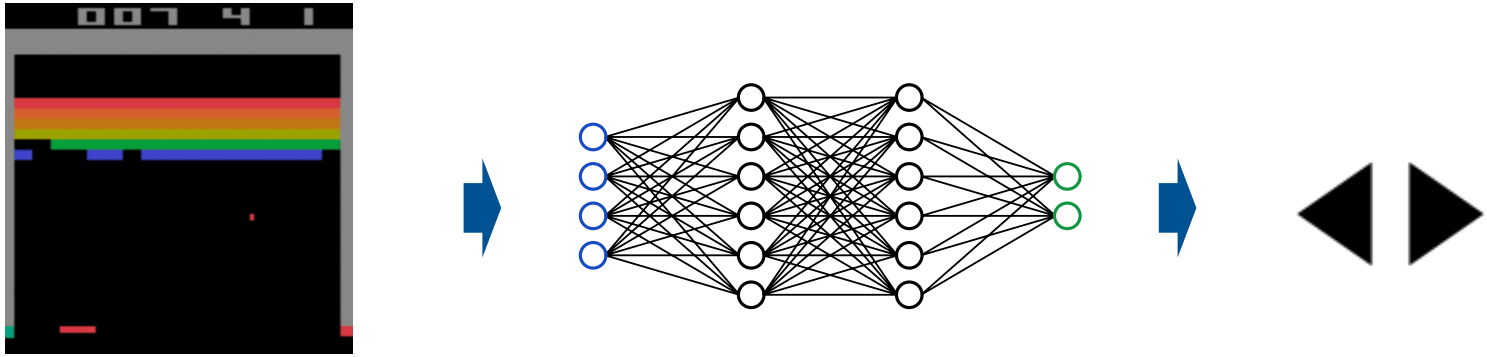
Moves right when…

Reward function:

3 points for breaking a green brick;

5 for pink…

Supervised Learning vs. Reinforcement Learning



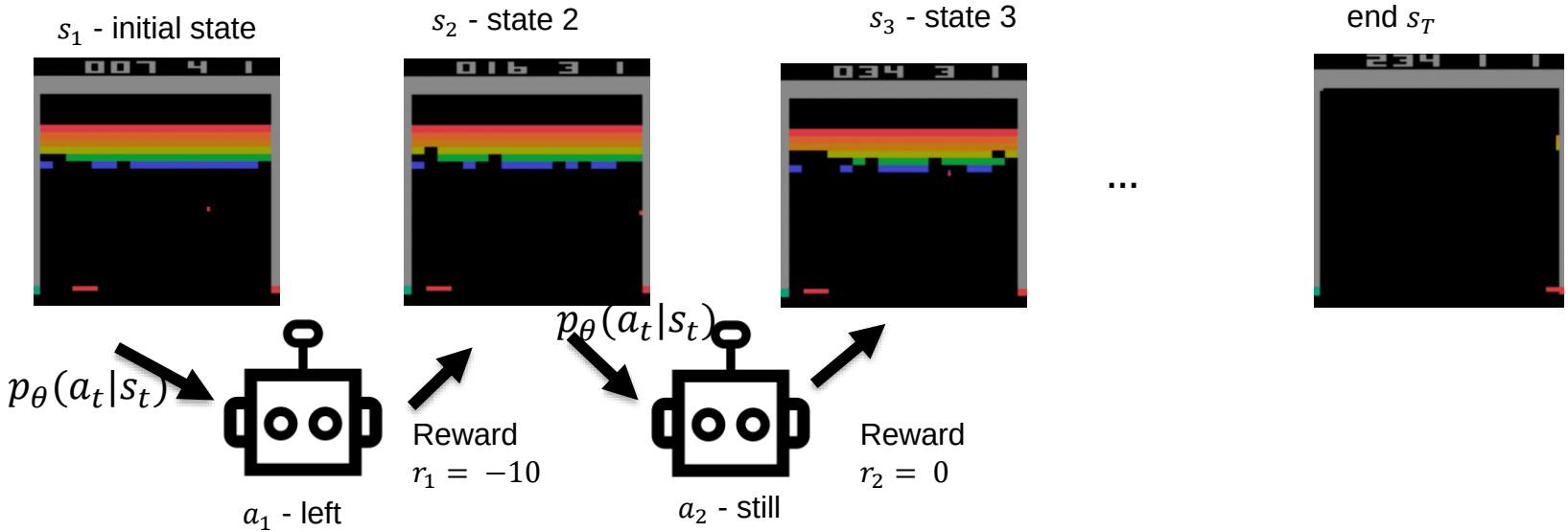
o_t $-obervation$ $\pi_\theta(a_t|o_t)$ $-policy$ a_t $-action$

	Supervised Learning	Reinforcement Learning
Dependency	Depend on initial input	Depend on previous states & actions
Data	Prepared trainning data	Trial and error in environment, get feedback from reward function
Goal	Minimize a loss	Maximize total reward

Reinforcement Learning

State: full representation of existing information

Observation partially reflects it.



Episode: from initial state s_1 to a final state s_T

τ: a trajectory $\tau = \{s_1, a_1, s_2, a_2, \dots, s_T, a_T\}$

$R(\tau)$: total reward $R(\tau) = \sum_{t=1}^{T} r_t$

$p_\theta(\tau)$: probability of τ on policy with parameters θ

Goal of Reinforcement Learning

Controlled by policy

Controlled by environment

Probability of a trajectory:

$$p_\theta(\tau) = p(s_1)p_\theta(a_1|s_1)p(s_2|s_1,a_1)p_\theta(a_2|s_2)p(a_3|s_2,a_2)\dots$$

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

Expected reward:

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

Objective:

$$\theta^\star = \arg\max_\theta E_{r\sim p_\theta(\tau)}[R(\tau)]$$

Evaluate the Objective

How can we **evaluate** a given policy?

- Sampling: Sample from the policy π_θ, compute rewards on samples,
take the average reward on these samples as the expected reward of the given policy.

Objective function:

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[R(\tau)] = \sum_\tau R(\tau) p_\theta(\tau) = \frac{1}{N} \sum_{i=1}^{N} R(\tau^i)$$

Number of samples

Gradient:

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta \left(R(\tau) p_\theta(\tau) \right)$$

Compute Gradient

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta \left(R(\tau) p_\theta(\tau)\right)$$

$$= \sum_\tau R(\tau) \nabla_\theta p_\theta(\tau)$$

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

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

$$\approx \sum_{i=1}^{N} R(\tau^i) \nabla_\theta \log p_\theta(\tau^i)$$

$$= \sum_{i=1}^{N} R(\tau^i) \left(\sum_{t=1}^{T} \nabla_\theta \log p_\theta(a_t^i | s_t^i) \right)$$

Environment uncontrollable

$$\nabla f(x) = f(x) \frac{\nabla f(x)}{f(x)}$$
$$= f(x) \nabla \log f(x)$$

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

Only this part relevant to θ

Gradient Update

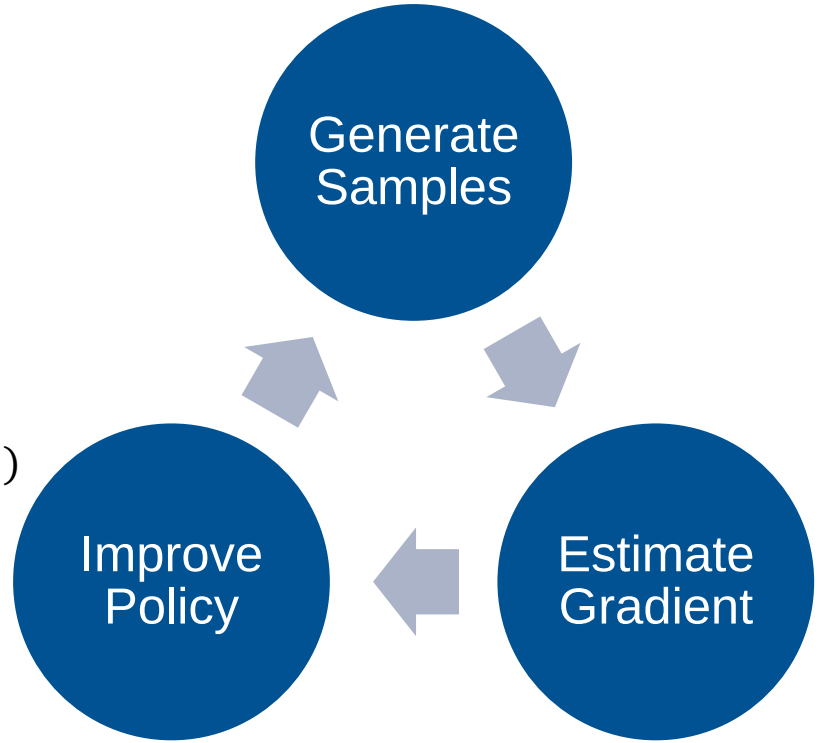
Gradient ascend:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

learning rate

Policy Gradient algorithms (REINFORCE):

1. Initialize the policy parameter θ randomly
2. Sample $\{\tau^i\}$ from π_θ
3. Gradient $\nabla_\theta J(\theta) = \sum_i R(\tau^i) \sum_t \nabla_\theta \log p_\theta(a_t^i | s_t^i)$
4. Update $\theta \leftarrow \theta + \alpha \nabla J(\theta)$



Problem 1: High Variance

High variance: gradients on difference batches of samples tend to be very different.

$Var(\nabla_\theta J(\theta))$ is big!

Intuitive Explanation:
Each trajectory can take very different steps depending on the states visited and actions sampled form policy π_θ

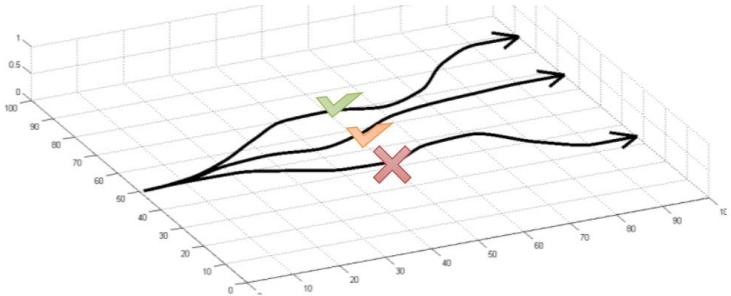
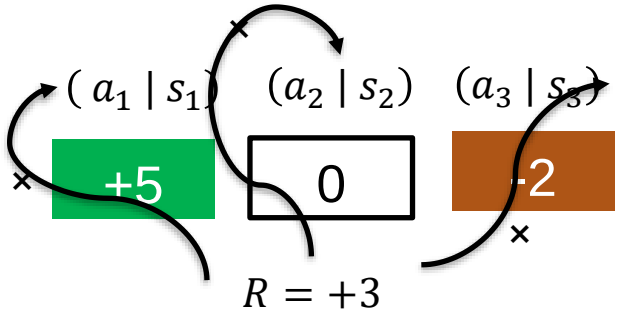
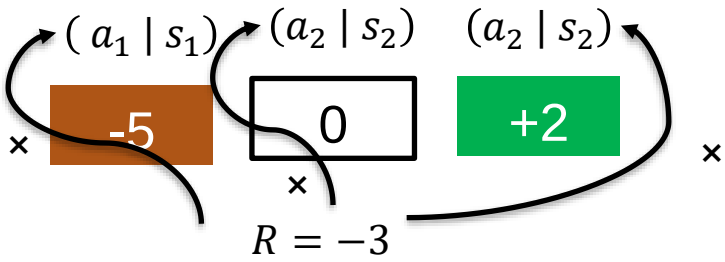


Image from CS294

Fix 1: Assign Suitable Weights





Causality: policy at time t' **cannot** affect rewards before time t'

Before: $\nabla_\theta J_\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} R(\tau^i) \nabla \log p_\theta(a_t^i | s_t^i)$ $\quad R(\tau^i) = \sum_{t'=1}^{T} r(s_{t'}, a_{t'})$

After: $\nabla_\theta J_\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} R_t(\tau^i) \nabla \log p_\theta(a_t^i | s_t^i)$ $\quad R_t(\tau^i) = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$

γ: a decaying factor smaller than 1 (further actions contributes less)

Fix 2: Add a Baseline

Example: $\nabla_\theta \log p_\theta(\tau) = [0.5, 0.2, 0.3]$

without baseline

$R(\tau) = [1000, 1001, 1002]$

$Var = Var(0.5 \times 1000, 0.2 \times 1001, 0.3 \times 1002)$
$= 23286.8$

with baseline

$R(\tau) = [-1, 0, 1]$

$Var = Var(0.5 \times -1, 0.2 \times 0, 0.3 \times 1)$
$= 0.1633$

$$\nabla_\theta J_\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} R_t(\tau^i) \nabla \log p_\theta(a_t^i | s_t^i) \qquad R_t(\tau^i) = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}) - b$$

We usually set b as running **average** of rewards

It also makes sense in that we **punish** actions below average, while **encourage** the rest.

Problem 2: Hard to Choose Step Sizes

Step too big:
Bad policy → data collected under bad policy → we cannot recover

e.g. A policy may update too much to increase its reward on a previous trajectory

Step too small:
Not efficient use of experience

Constraint: Proximal Policy Optimization (PPO)

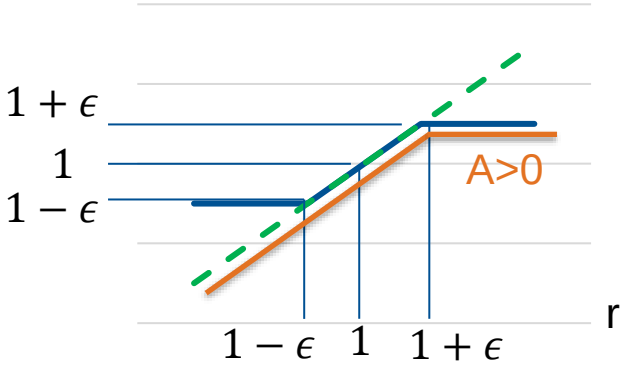
Clipped objective

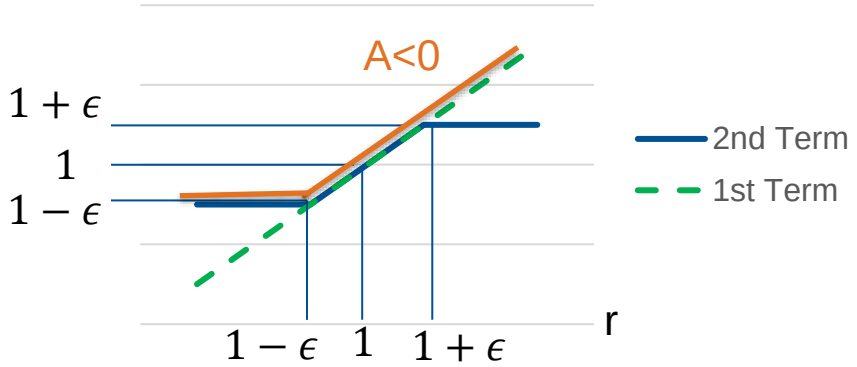
Similarity metric: $r(\theta) = p_\theta(a_t|s_t) \,/\, p_{\theta_{old}}(a_t|s_t)$

Actually has something to do with importance sampling

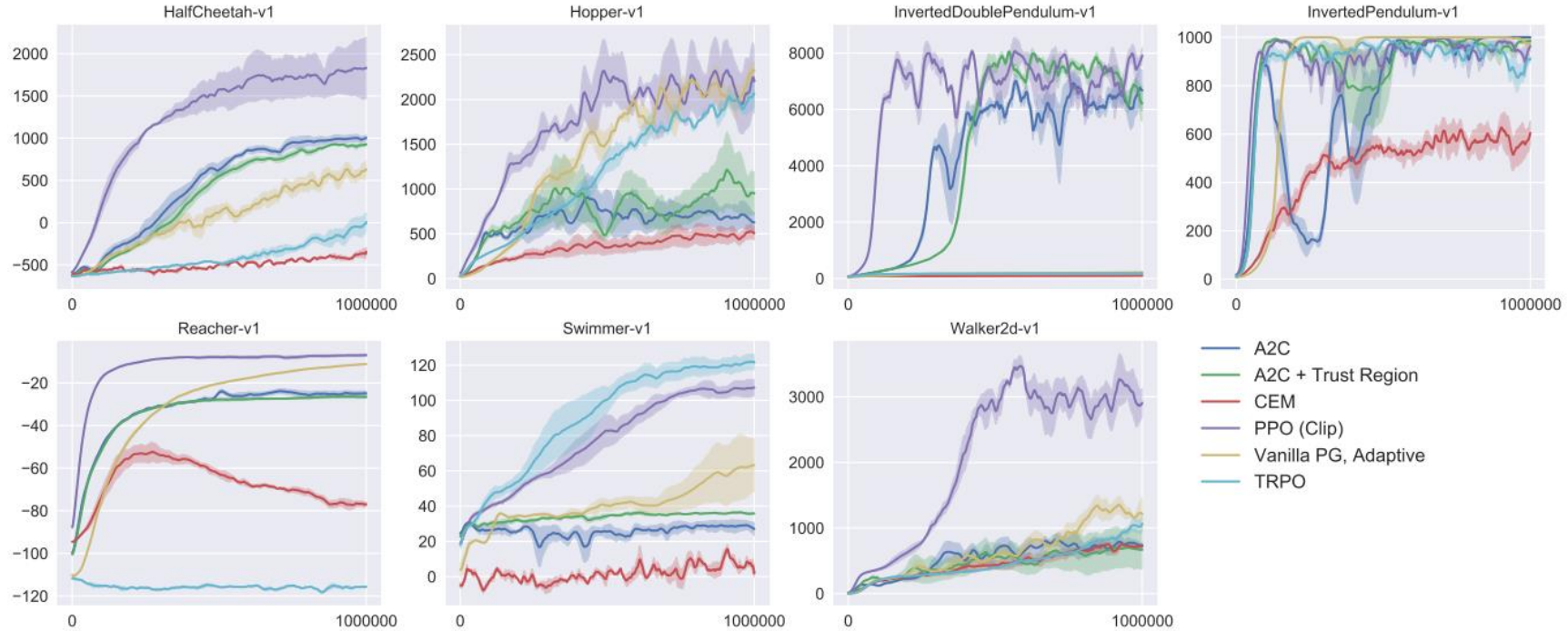
$$J^{CLIP}_{\theta_{old}}(\theta) = E_{\tau\sim\pi_{old}}\left[\sum_{t=1}^{T} \min\left(r_t(\theta)R_t^{\theta_{old}}, clip\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)R_t^{\theta_{old}}\right)\right]$$

ϵ: a hyperparameter controls how much a policy can update, usually [0.1, 0.2]





PPO results



Performance comparison between PPO with clipped objective and various other deep RL methods on several MuJoCo tasks.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. “Proximal Policy Optimization Algorithms”. (2017)

Thanks for your attention!

Q&A

References

- Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
- http://videolectures.net/DLRLsummerschool2018_levine_policy_search/
- DRL Lecture 2: Proximal Policy Optimization
- Policy Gradient Algorithms