Machine Learning

Lecture 16: Policy-Based Deep Reinforcement Learning

Nevin L. Zhang

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on the references listed at the end and internet resources.

Outline

1 Policy Gradients

2 Actor-Critic Algorithms

Value-Based RL vs Policy-Based RL

- **Value-based RL**:

$$\{(s, a, r, s')\} \Rightarrow Q(s, a : \theta) \Rightarrow \pi^*(s) = \arg\max_a Q(s, a : \theta)$$

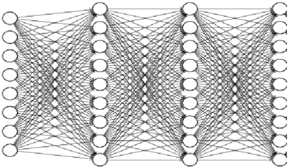
  where π^* is a deterministic policy.

- **Policy-based RL**

$$\{(s, a, r, s')\} \Rightarrow \pi(a|s)$$

  where $\pi(a|s)$ is a stochastic policy.

- For a given s, $\pi(a|s)$ is a distribution over actions, and is represented as a neural network:

$\mathbf{s}$  $\pi_\theta(\mathbf{a}|\mathbf{s})$

Acting according to Stochastic Policy

$$\tau: \quad \underset{\sim p(s)}{s_0} \xrightarrow[r_0]{a_0 \sim \pi_\theta(a|s_0)} \underset{\sim p(s'|s_0,a_0)}{s_1} \xrightarrow[r_1]{a_1 \sim \pi_\theta(a|s_1)} \underset{\sim p(s'|s_1,a_1)}{s_2} \longrightarrow \cdots$$

Trajectory

prob of τ: $\pi_\theta(\tau) = p(s_0)\,\pi_\theta(a_0|s_0)\,p(s_1|s_0,a_0)$
$\pi_\theta(a_1|s_1)\,p(s_2|s_1,a_1)$
$\cdots$
$= p(s_0)\prod_t \pi_\theta(a_t|s_t)\,p(s_{t+1}|s_t,a_t)$

Reward: $r(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

Expected cumulative reward for following $\pi_\theta(a|s)$

Learning: $\max_\theta J(\theta)$

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

Acting according to Stochastic Policy

- Suppose an agent interacts with its environment by following a stochastic policy $\pi_\theta(a|s)$ until an episode ends:
  - Experience trajectory τ: $s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
  - At each time point t, an action a_t is sampled from the distribution $\pi_\theta(a|s_t)$
  - The probability of an experience trajectory is:

$$\pi_\theta(\tau) = p(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{1}$$

Objective of the Policy Gradient Method

- Because of stochasticity in environment $p(s_{t+1}|s_t)$ and in action selection $\pi_\theta(a_t|s_t)$, the trajectory, and hence the total reward, will be different in different runs of the process.
- The expected total reward is:

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)] = E_{\tau\sim\pi_\theta(\tau)}[\sum_t \gamma^t r_t]$$

- The objective of policy gradient is to maximize $J(\theta)$:

$$\theta^* = \arg\max_\theta J(\theta)$$

- This is done via gradient ascent, and the output is (an approximation of) the optimal policy $\pi_{\theta^*}(a|s)$:

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

Policy Gradients

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta E_{\tau \sim \pi_\theta(\tau)}[r(\tau)] \\
&= \nabla_\theta \int r(\tau)\pi_\theta(\tau)d\tau \\
&= \int r(\tau)\nabla_\theta \pi_\theta(\tau)d\tau \\
&= \int r(\tau)\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)d\tau \qquad \text{(the log-gradient trick)} \\
&= E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]
\end{aligned}
$$

Policy Gradients

- Because of Equation (1), we have

$$\log \pi_\theta(\tau) = \log p(s_0) + \sum_t \log \pi_\theta(a_t|s_t) + \sum_t \log p(s_{t+1}|s_t)$$

- Hence,

$$\begin{aligned}
& E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\
= \quad & E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta (\log p(s_0) + \sum_t \log \pi_\theta(a_t|s_t) + \sum_t \log p(s_{t+1}|s_t)) r(\tau)] \\
= \quad & E_{\tau\sim\pi_\theta(\tau)}[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) r(\tau)] \\
\approx \quad & \frac{1}{N}\sum_{i=1}^{N}\sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) r(\tau^i)
\end{aligned}$$

where $\{\tau^i = \{s_0^i, a_0^i, r_0^i, s_1^i, a_1^i, r_1^i, \ldots\}_{i=1}^N$ is a collection of N sample trajectories.

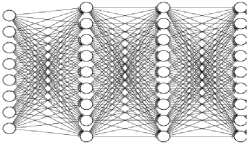
The REINFORCE Algorithm

- REINFORCE algorithm (Williams 1992):

  Repeat:

  1. sample $\{\tau^i\}_{i=1}^N$ from $\pi_\theta(a_t|s_t)$ (run the current policy)
  2. $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) r(\tau^i)$
  3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

- The term $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is calculated on the policy network:

$\mathbf{s}$  $\pi_\theta(\mathbf{a}|\mathbf{s})$

Supervised Learning, Imitation Learning, and RL (REINFORCE)

supervised learning $\{x_i, y_i\}_{i=1}^{N} \Rightarrow p(y|x,\theta)$

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(y_i|x_i, \theta)$$

max likelihood of model / prob of data

current setting : $\{(s_t^i, a_t^i) \mid i=1,\dots,N;\ t=1,\dots,T\}$

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_i \sum_t \nabla_\theta \log p(a_t^i | s_t^i)$$

max prob of data / prob of actions

Imitation learning : learn from expert demo

REINFORCE

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_i \sum_t \nabla_\theta \log p(a_t^i | s_t^i)\, r(\tau^i)$$

$r(\tau^i) > 0$, increase the prob of action in τ^i

$r(\tau^i) < 0$ decrease the prob of actions in τ^i

Interpretation of the REINFORCE Update Rule

$$\theta \leftarrow \theta + \alpha \sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) r(\tau^i)$$

- Changing θ in the direction of $\nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$ would increase the probability of the action a_t^i.
- If $r(\tau^i) < 0$, we change θ in the opposite direction and hence reduce the probability of a_t^i
  - So, **the update rule makes bad experiences less likely**.
- If $r(\tau^i) > 0$, we change θ so as to increase the probability of a_t^i
  - So, **the update rule makes good experiences more likely**
- So, the REINFORCE update formalizes the notion of "trial and error".

On-Policy vs Off-Policy

- REINFORCE is an **on-policy** algorithm because all the data used to improve the current policy are collected using the policy itself.

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

On-Policy vs Off-Policy

- DQN is an **off-policy** algorithm because some of the data used to improve the current policy are collected using other policies.

Repeat:

- Take action a in current state s, observe r and s'; add experience tuple (s, a, s', r) to a buffer D; $s \leftarrow s'$
- Sample a minibatch $B = \{s_j, a_j, s'_j, r_j\}$ from D.
- Update the parameters

$$\theta \quad \leftarrow \quad \theta - \alpha \nabla_\theta \sum_j ([r(s_j, a_j) + \gamma \max_{a'_j} Q(s'_j, a'_j; \theta^-)] - Q(s_j, a_j; \theta))^2$$

- $\theta^- \leftarrow \theta$ in every C steps.

On-Policy vs Off-Policy

- Q-learning is off-policy even without experience replay because the agent does not necessarily take the action $a' = \arg\max_{a'} Q(s', a')$ in the next step
  - The action a' used for update is chosen using the current Q, but
  - The next action is chosen using the updated Q.

- Initialize $Q(s, a)$ arbitrarily.
- Repeat (for each episode)
  - Pick initial state s.
  - **Repeat**
    - Choose a for the state s (ϵ-greedy with $\arg\max_a Q(s, a)$)
    - Take action a, observe r and s'
    - Update:

$$
\begin{aligned}
Q(s, a) &\leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \\
s &\leftarrow s'
\end{aligned}
$$

  - **until** s is terminal

Policy Gradient has High Variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) r(\tau^i)$$

- The policy gradient is estimated using N trajectory samples.
- N cannot be large because running a policy is costly.
- Because we can use only a small number of trajectory samples, the variance is high.

Reducing Variance using Causality

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t^i|s_t^i) r(\tau^i) \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=0}^{T}\gamma^{t'} r_{t'}^i]\end{aligned}$$

- An action a_t^i taken at time point t does not affect only rewards at earlier time points.
- Hence, rewards before time t should not be considered when optimizing a_t.
- So, we use the following gradient instead:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\nabla_\theta \log \pi_\theta(a_t^i|s_t^i)\sum_{t'=t}^{T}\gamma^{t'} r_{t'}^i]$$

- The variance of $\sum_{t'=t}^{T}\gamma^{t'} r_{t'}^i$ is smaller than that of $\sum_{t'=0}^{T}\gamma^{t'} r_{t'}^i$ because it is influenced by less stochasticity.

Reducing Variance using Baselines

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) r(\tau^i)$$

- Another way to reduce the variance is to use

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) (r(\tau^i) - b)$$

  where the **baseline** b is given by:

$$b = \frac{1}{N} \sum_{i=1}^{N} r(\tau^i).$$

Reducing Variance using Baselines: Analogy

- Let x_1, x_2, ..., x_n be i.i.d random variables and b be another random variable.

$$
\begin{aligned}
V(\sum_{i=1}^{n}(x_i - b)) &\approx \sum_{i=1}^{n} V(x_i - b) \quad \text{(strictly true if independent)} \\
&= \sum_{i=1}^{n}(E[(x_i - b)^2] - (E[x_i - b])^2) \\
&= \sum_{i=1}^{n} E[(x_i - b)^2] - \sum_{i=1}^{n}(E[x_i - b])^2
\end{aligned}
$$

- The first term is minimized when $b = \frac{1}{n}\sum_{i=1}^{n} x_i$
- The second term is also minimized when $b = \frac{1}{n}\sum_{i=1}^{n} x_i$.

Baseline does not Make the Estimation Unbiased

- This is the policy gradient

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

- Subtracting a baseline $b = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$, we get

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]$$

- This does not introduce bias because

$$\begin{aligned} E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) b] &= \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) b d\tau \\ &= \int \nabla_\theta \pi_\theta(\tau) b d\tau \\ &= b\nabla_\theta \int \pi_\theta(\tau) d\tau \\ &= 0 \end{aligned}$$

Advanced Policy Gradient Methods

- Problems with policy gradient:
  - The parameters θ are changed only a little bit at each gradient step because it does not make efficient use of the sampled trajectories $\{\tau^i\}$.
  - Large learning rate can lead to performance collapse, while small learning rate implies slow learning.
- Advanced policy gradient methods:
  - Purpose: Make efficient use of data and find an update rule that is just right.
  - Methods: Natural policy gradient (Peters and Schall 2008); Trusted region policy optimization (Schulman et al. 2015); Proximal policy optimization (Schulman et al. 2017).

Outline

1 Policy Gradients

2 Actor-Critic Algorithms

Optimal Value Functions and Value Functions of Policy

- Optimal value functions:
  - $V^*(s)$: Total reward for acting optimally from state s.
  - $Q^*(s,a)$: total reward for, starting from s, taking action a and acting optimally after that.

$$V^*(s) = \max_a Q^*(s,a).$$

- Value functions of a policy π:
  - $V^\pi(s)$: Total reward for following π from state s.
  - $Q^\pi(s,a)$: total reward for, starting from s, taking action a and then following π.

$$V^\pi(s) = E_{a\sim\pi(a|s)}[Q^\pi(s,a)].$$

- Actor-Critic Algorithms: Policy gradient with policy evaluation, i.e., $Q^\pi(s,a)$.

Key Idea of Actor-Critic Algorithms

- The **advantage function**:

$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

tells us how good the action a_t is relative to the average.

- Using the advantage function, we can write our policy gradient estimate as follows:

$$\nabla_\theta J(\theta) \quad \approx \quad \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\gamma^t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) A^{\pi}(s_t^i, a_t^i)],$$

  - Increase probabilities of actions that are better than average,
  - Decrease probabilities of actions worse than average.

Key Idea of Actor-Critic Algorithms

REINFORCE

Actor: $s \to \boxed{\theta} \to \pi_\theta(a|s)$

$$\theta \leftarrow \theta + \alpha \sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\, \underline{r(\tau^i)}$$

Actor-Critic

$$\theta \leftarrow \theta + \alpha \sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\, \underline{A^\pi(s_t^i, a_t^i)}$$

$A^\pi(s_t^i, a_t^i) > 0$, a_t^i better than average,
prob of a_t^i increased

$A^\pi(s_t^i, a_t^i) < 0$, a_t^i worse than average,
prob of a_t^i decreased

$$A^\pi(s_t^i, a_t^i) \approx r_t^i + \gamma V^\pi(s_{t+1}^i) - V^\pi(s_t^i)$$

How to Estimate the Advantage Function?

- In general, we have

$$Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} V^{\pi}(s')P(s'|s,a).$$

- Estimating $Q^{\pi}(s_t, a_t)$ using one sample (s_{t+1}) of s',

$$Q^{\pi}(s_t, a_t) \approx r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}).$$

- Hence we can estimate $A^{\pi}(s_t, a_t)$ using

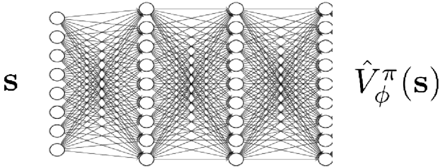
$$A^{\pi}(s_t, a_t) \approx r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$$

  where $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$, which can be obtained by letting agent act for one step.

- If we can estimate $V^{\pi}(s_t)$, then we can get $A^{\pi}(s_t, a_t)$ and perform gradient asent.

How to Estimate the Value Function $V^\pi(s_t)$?

- This is called the **policy evaluation** problem.
- To learn the function from data, we use a neural network with parameters ϕ to represent it. The output for input s is denoted by $\hat{V}^\pi_\phi(s)$.



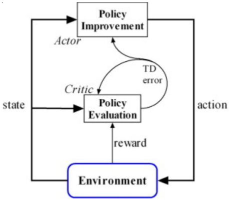
- After obtaining a collection of experience tuples $\{(s_i, a_i, s'_i, r_i)\}$, we update ϕ via backprop based on the following training set:

$$\{(s_i, r_i + \gamma \hat{V}^\pi_\phi(s'_i))\}$$

Online Actor-Critic algorithm

Repeat:

1. Take action $a \sim \pi_\theta(a|s)$, get (s, a, s', r).
2. Update Critic parameters ϕ using L2 loss and training data:
$$\{(s, r + \gamma \hat{V}^\pi_\phi(s')\}$$
3. $\hat{A}^\pi(s, a) \leftarrow r + \gamma \hat{V}^\pi_\phi(s') - \hat{V}^\pi_\phi(s)$
4. Update actor parameters:
$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) \hat{A}^\pi(s, a)$$

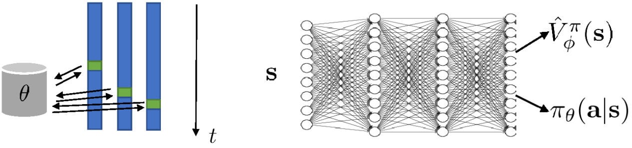


Batch Actor-Critic algorithm

Repeat:

1. Sample experiences $\{(s_i, a_i, s'_i, r_i)\}$ by following π_θ for multiple steps.
2. Update Critic parameters ϕ using L2 loss and training data
$$\{(s_i, r_i + \gamma \hat{V}^\pi_\phi(s'_i))\}$$
3. $\hat{A}^\pi(s_i, a_i) \leftarrow r_i + \gamma \hat{V}^\pi_\phi(s'_i) - \hat{V}^\pi_\phi(s_i) \qquad \forall i$
4. Update actor parameters:
$$\theta \leftarrow \theta + \alpha \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i)\hat{A}^\pi(s_i, a_i)$$

Asynchronous Advantage Actor-Critic (A3C)

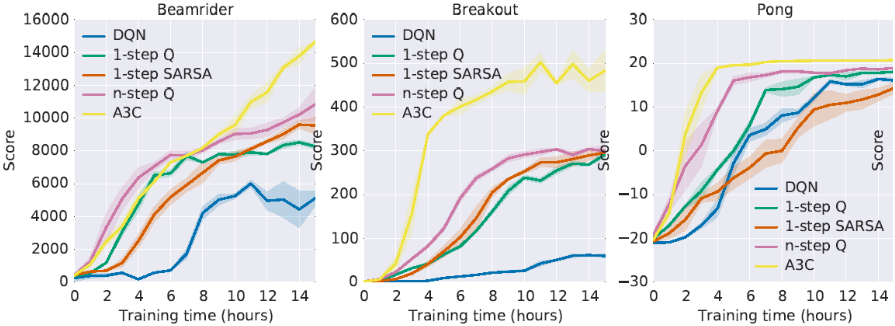


- Run multiple learners in parallel and let them take turns to update parameters .
- Decorrelates data used in learning. (Alternative to experience relay).
- Different learners explore different parts of environment.
- Actor and Critic share the same network

`https://www.youtube.com/watch?v=Ajjc08-iPx8&sns=tw&`

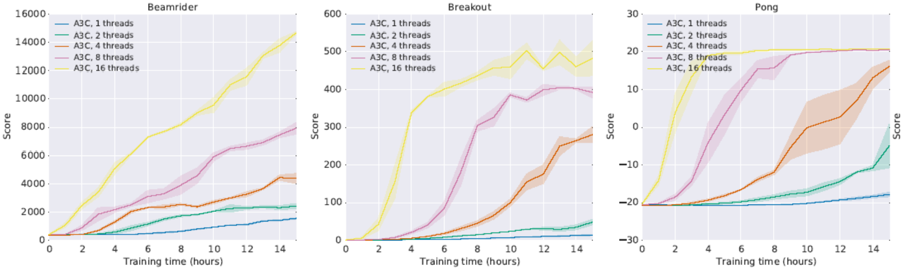
Comparisons on Atari Games: Learning Speed

- DQN on single GPU
- Asynchronous methods use 16 CPU cores



Comparisons on Atari Games: Learning Speed

- More cores mean faster learning



Comparisons on Atari Games: Performance [1]

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric.

[1] `github.com/vmayoral/basic_reinforcement_learning/blob/master/tutorial12/README.md`

Proximal Policy Optimization Algorithms (PPO) [2]

- Recall Actor-Critic Algorithm:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\gamma^t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) A^\pi(s_t^i, a_t^i)] \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\gamma^t \frac{\nabla_\theta \pi_\theta(a_t^i|s_t^i)}{\pi_\theta(a_t^i|s_t^i)} A^\pi(s_t^i, a_t^i)]\end{aligned}$$

- To get PPO, replace π_θ in the denominator with an old policy

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\gamma^t \nabla_\theta \frac{\pi_\theta(a_t^i|s_t^i)}{\pi_{old}(a_t^i|s_t^i)} A^\pi(s_t^i, a_t^i)]$$

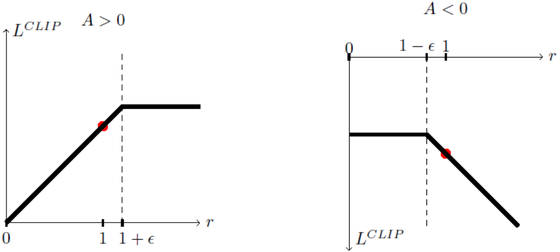
- The idea of PPO is keep $r_t^i = \frac{\pi_\theta(a_t^i|s_t^i)}{\pi_{old}(a_t^i|s_t^i)}$ close to 1 so that the gradient update step does not introduce too much change in the policy, and hence make the training more **stable**. Often used.

[2] Schulman et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

Proximal Policy Optimization Algorithms (PPO)

- PPO uses a lower bound of $r_t^i A^\pi(s_t^i, a_t^i)$, i.e.,
  $L_t^{CLIP}(\theta) = min(r_t^i A^\pi(s_t^i, a_t^i), clip(r_t^i, 1-\epsilon, 1+\epsilon)A^\pi(s_t^i, a_t^i))$:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T}[\gamma^t \nabla_\theta L_t^{CLIP}(\theta)]$$



The clipped surrogate objective function limits the deviation from the previous policy. This avoids large policy updates that may harm the performance.

Soft Actor-Critic [3]

- Policy gradient and Actor-Critic:

$$\pi^* = \arg\max_{\pi} E_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1})]$$

- **Soft Actor-Critic (SAC)**:

$$\pi^* = \arg\max_{\pi} E_{\tau\sim\pi}[\sum_{t=0}^{\infty} \gamma^t (r(s_t, a_t, s_{t+1}) + \alpha H(\pi(\cdot|s_t)))]$$

  where $H(\pi(\cdot|s_t))$ is the entropy of the policy, and α is the temperature parameter.

  - SAC prefers policies with high entropy, which results in better exploration performance.
  - Consequently, SAC is more stable and more sample efficient than others algorithms such as DDPG and A3C.

[3] https://spinningup.openai.com/en/latest/algorithms/sac.html
Haarnoja *et al*, 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Sample Complexity (Sergey Levine)

gradient-free methods
(e.g. NES, CMA, etc.)

10x

fully online methods
(e.g. A3C)

10x

policy gradient methods
(e.g. TRPO)

10x

replay buffer value estimation methods
(Q-learning, DDPG, NAF, SAC, etc.)

10x

model-based deep RL
(e.g. PETS, guided policy search)

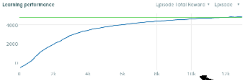
10x

model-based "shallow" RL
(e.g. PILCO)

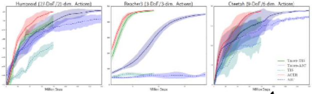
Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Tim Salimans[1] Jonathan Ho[1] Xi Chen[1] Ilya Sutskever[1]

half-cheetah (slightly different version)



TRPO+GAE (Schulman et al. '16)

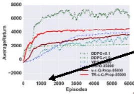


Wang et al. '17

100,000,000 steps
(100,000 episodes)
(~ 15 days real time)

10,000,000 steps
(10,000 episodes)
(~ 1.5 days real time)

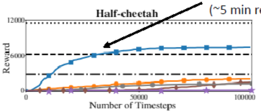
half-cheetah



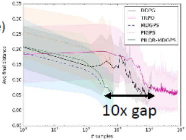
Gu et al. '16

1,000,000 steps
(1,000 episodes)
(~3 hours real time)

30,000 steps
(30 episodes)
(~5 min real time)



Chua et a. '18: Deep Reinforcement Learning in a Handful of Trials



Chebotar et al. '17 (note log scale)

about 20 minutes of experience on a real robot

Value-Based RLvs Policy-Based RL

- Value-based RL:
  - Pros: Can be off-policy, can use experience reply, more sample efficient
  - Cons: Indirect, accuracy estimation of Q is difficulty, might lead to unstable policy.
- Policy-based RL:
  - Pros: Directly optimize policy, more stable.
  - Cons: It is on-policy, not sample efficient.
- Recent Trends: Combine the best of two worlds.