8. Policy Gradients

2019 Fall

Yusung Kim
yskim525@skku.edu

Review of Last Class (Value-based)

- To solve large-scale problems with Q-learning, we need a value function approximation.
- DQN : Experience Replay, Fixed Target
- Double DQN : Reducing Overestimations
- Prioritized Experience Replay: selecting experience with a priority
- Dueling DQN : New Neural Network Architecture
- Multi-Steps, Distributional RL, Noisy-nets, … RAINBOW!

Policy-based Reinforcement Learning

- In the last lecture we approximated the action-value function using parameters θ

$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

- A policy was generated from the action-value function (Q-function)
- In this lecture we will directly parameterize the policy

$$\pi_\theta(s, a) \approx \mathbb{P}[\, a \mid s \,]$$

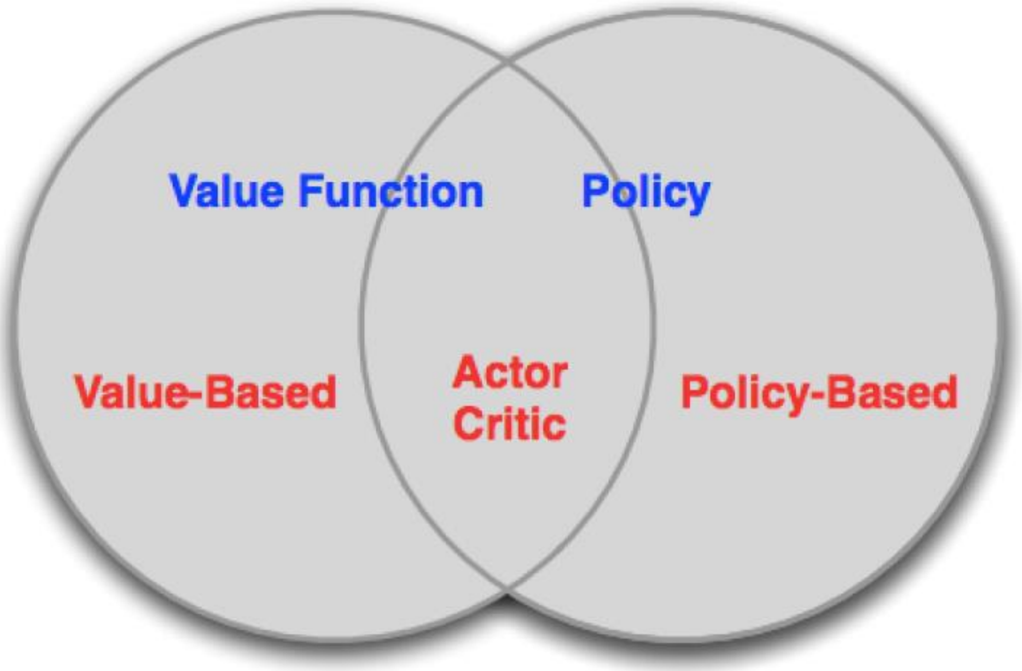
- We will focus again on **model-free** reinforcement learning.

Value-based vs. Policy-based RL

- Value-based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

- Policy-based
  - No Value Function
  - Learned Policy

- Actor-Critic
  - Learned Value Function
  - Learned Policy



Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies

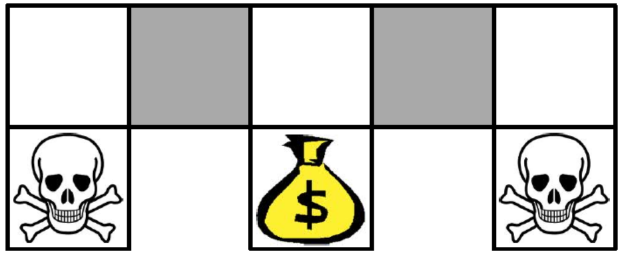
- Disadvantages:
  - Less stable during training process due to high variance
  - Sample inefficient (need more sample data)

Example: Rock-Paper-Scissors

- Two-player game of rock-paper-scissors
  - Scissors beats paper
  - Rock beats scissors
  - Paper beats rock

- Consider policies for iterated rock-paper-scissors
  - A deterministic policy is easily exploited
  - A uniform random policy is optimal (i.e. Nash equilibrium)

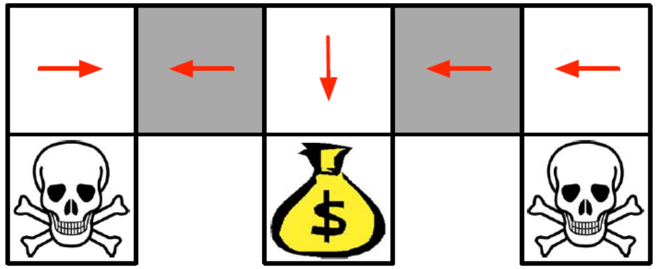
Example: Aliased Gridworld (1/3)



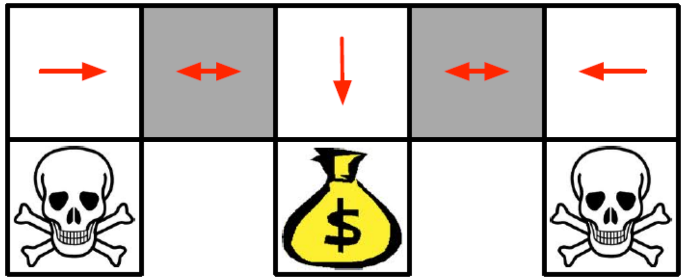
- The agent cannot differentiate the grey states
- Consider features of the following form (for all N, E, S, W) : $\emptyset(s)$ = wall to N or wall to S
- Value-based RL, using an approximate value function : $Q_\theta(s, a) = f(\emptyset(s), a, \theta)$
- Policy-based RL, using a parameterized policy : $\pi_\theta(s, a) = g(\emptyset(s), a, \theta)$

Example: Aliased Gridworld (2/3)

- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - or move E in both grey states
- Either way, it can get stuck and never reach the money
- Value-based RL learns a near-deterministic policy
  - e.g. greedy or ε-greedy
- So it will traverse the corridor for a long time



Example: Aliased Gridworld (3/3)



- An optimal stochastic policy will randomly move E or W in grey states

$$\pi_\theta(\text{wall to N and S, move E}) = 0.5$$

$$\pi_\theta(\text{wall to N and S, move W}) = 0.5$$

- It will reach the goal state in a few steps with high probability
- Policy-based RL can learn the optimal stochastic policy

Policy Search

- Let's find the optimal policy π_θ , that has a parameter θ, outputs a probability distribution of optimal actions.

$$\pi_\theta(s, a) = \mathrm{P}\,[a|s]$$

- But how do we improve/optimize a policy π_θ?
- We must find the best parameters θ to maximize a score function, $J(\theta)$

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum \gamma \cdot R\right]$$

- There are two steps:
  - Measure the quality of a policy π_θ with a policy score function $J(\theta)$
  - Use policy gradient ascent to find the best parameter θ that improves our π_θ .

Policy Score (Objective) Functions

- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[V(s_1)]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s, a) \mathcal{R}_s^a$$

- where $d^{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ

Policy Optimization

- Policy based reinforcement learning is an optimization problem
- Find θ that maximizes $\mathrm{J}(\theta)$
- We focus on gradient approaches

Policy Gradient

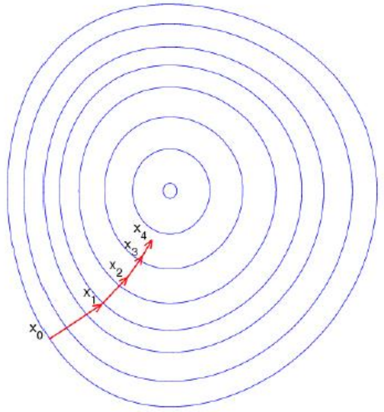
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

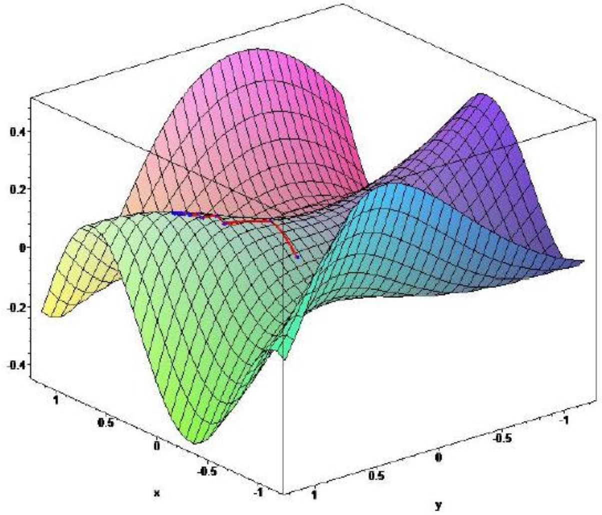
$$\Delta\theta = \alpha \nabla_\theta J(\theta)$$

- where $\nabla_\theta J(\theta)$ is the policy gradient

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix}$$

- and α is a step-size parameter





Policy Gradient Ascent

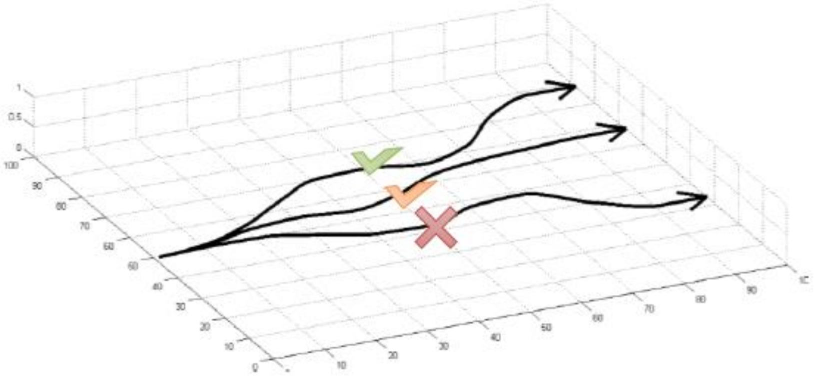
- Maximizing the score function means finding the optimal policy.
- To maximize the score function $J(\theta)$, we need to do gradient ascent on policy parameters.

$$J_1(\theta) = V_{\pi\theta}(s_1) = E_{\pi\theta}[v_1] = \underbrace{\sum_{s \in S} d(s)}_{\text{State distribution}} \underbrace{\sum_{a \in A} \pi_\theta(s,a) R_s^a}_{\text{Action distribution}}$$

- How do we determine the effect of policy on the state distribution?
- How do we estimate the (gradient) with unknown state distribution?

Policy Differentiation

$$\theta^\star = \arg\max_\theta \underbrace{E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]}_{J(\theta)}$$



$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

sum over samples from π_θ

Policy Differentiation

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta \pi_\theta(\tau)}\, r(\tau)\, d\tau$$

$$\nabla_\theta \pi_\theta(\tau) = \frac{\pi_\theta(\tau)}{\pi_\theta(\tau)} \nabla_\theta \pi_\theta(\tau)$$

$$= \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)}$$

$$= \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)$$

using likelihood ratio trick

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

$$\nabla_\theta J(\theta) = \int \underline{\nabla_\theta \pi_\theta(\tau)}\, r(\tau)\, d\tau = \int \pi_\theta(\tau)\, \nabla_\theta \log \pi_\theta(\tau)\, r(\tau)\, d\tau = E_\theta[\nabla_\theta \log \pi_\theta(\tau)\, r(\tau)]$$

Policy Differentiation

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

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

log of both sides

$$\Downarrow$$

$$\log \pi_\theta(\tau) = \log p(\mathbf{s}_1) + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

Policy Differentiation

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

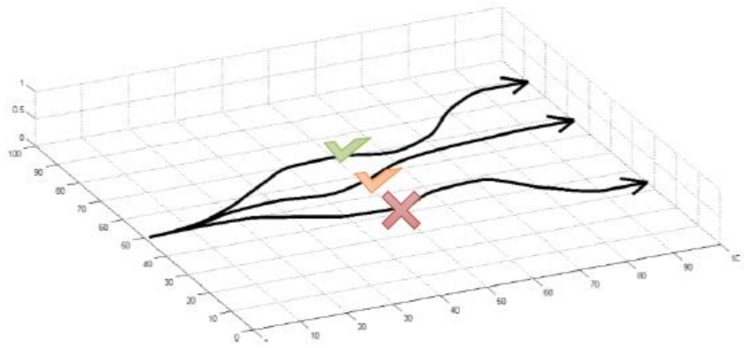
$$\nabla_\theta \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \right]$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

Do not need to know dynamics model

Evaluating the Policy Gradient

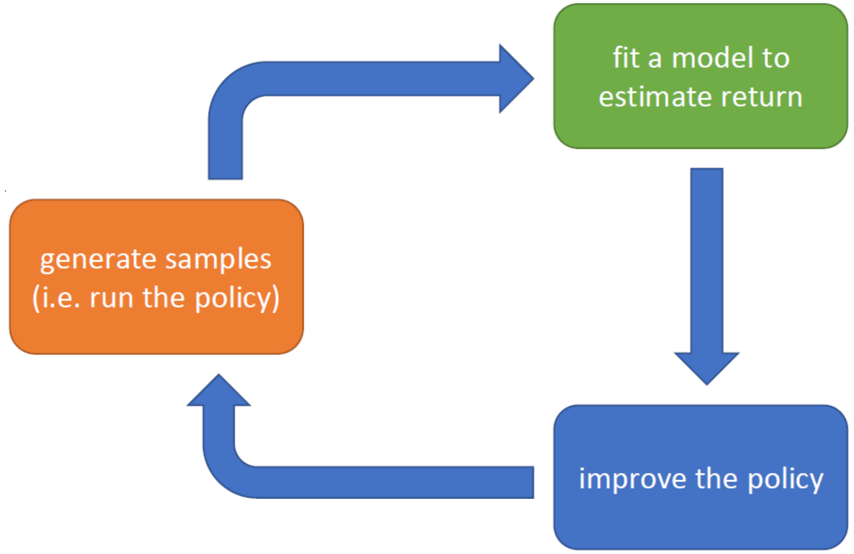
$$J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$



$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

$$\nabla_\theta J(\theta) \approx \underline{\frac{1}{N}\sum_{i=1}^{N}}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\underline{\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)}$$

$$\underline{\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)}$$



In practice, we replace expectation by sampling multiple trajectories.

Differentiable Policy Classes

- Discrete action space
  - Softmax

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{\exp(h(s,a,\boldsymbol{\theta}))}{\sum_b \exp(h(s,b,\boldsymbol{\theta}))}$$

- Continuous action space
  - Gaussian policy

$$N(x\,|\,\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

Reducing Variance

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

$$\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t) \underline{\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})}\right]$$

In this form, actions are only reinforced based on rewards obtained after they are taken.

https://spinningup.openai.com/en/latest/spinningup/extra_pg_proof1.html

Monte-Carlo Policy Gradient (REINFORCE)

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

"Baselines" in Policy Gradients

- We subtract a baseline function $B(s)$ from the policy gradient
- This can reduce variance, without changing expectation

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t) \right) \right]$$

- A good baseline is the state value function $B(s) = V^\pi(s)$
- This results in faster and more stable policy learning.

Baseline does NOT introduce bias-derivation

$$
\begin{aligned}
&\mathbb{E}_\tau[\nabla_\theta \log \pi(a_t|s_t,\theta)b(s_t)] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t,\theta)b(s_t)]\right] \text{ (break up expectation)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi(a_t|s_t,\theta)]\right] \text{ (pull baseline term out)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[b(s_t)\mathbb{E}_{a_t}[\nabla_\theta \log \pi(a_t|s_t,\theta)]] \text{ (remove irrelevant variables)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\sum_a \pi_\theta(a_t|s_t)\frac{\nabla_\theta \pi(a_t|s_t,\theta)}{\pi_\theta(a_t|s_t)}\right] \text{ (likelihood ratio)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\sum_a \nabla_\theta \pi(a_t|s_t,\theta)\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\nabla_\theta \sum_a \pi(a_t|s_t,\theta)\right] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}[b(s_t)\nabla_\theta 1] \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}[b(s_t)\cdot 0] = 0
\end{aligned}
$$

$B(s) = V^{\pi}(s)$

- $V^{\pi}(s)$ cannot be computed exactly, so it has to be approximated.

- With a neural network, updated concurrently with the policy
  - Value network always approximates the value function of the most recent policy

- Minimize a mean-squared-error objective where π_k is the policy at epoch k.

$$\varnothing_k = \arg\min_{\varnothing} E_{s_t, \hat{R}_t \sim \pi_k} \left[\left(\hat{R}_t - V_{\emptyset}(s_t) \right)^2 \right]$$

Vanilla Policy Gradient Algorithm [Sutton 2000]

Algorithm 1 Vanilla Policy Gradient Algorithm

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

for $k = 0, 1, 2, \ldots$ **do**

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

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

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

  Estimate policy gradient as

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

  Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

  or via another gradient ascent algorithm like Adam.

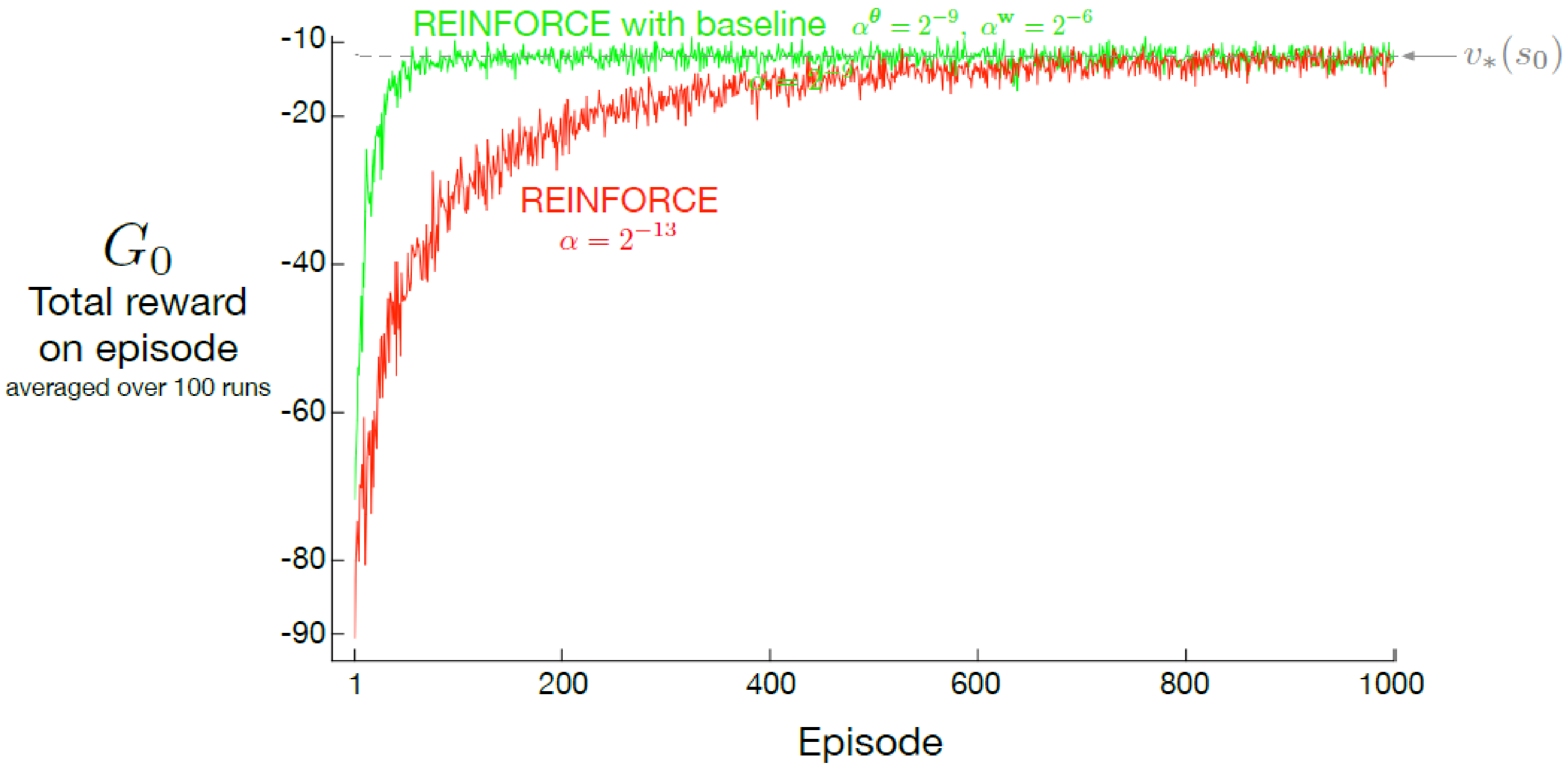
  Fit value function by regression on mean-squared error:

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

  typically via some gradient descent algorithm.

end for

Performance Impact of Baseline

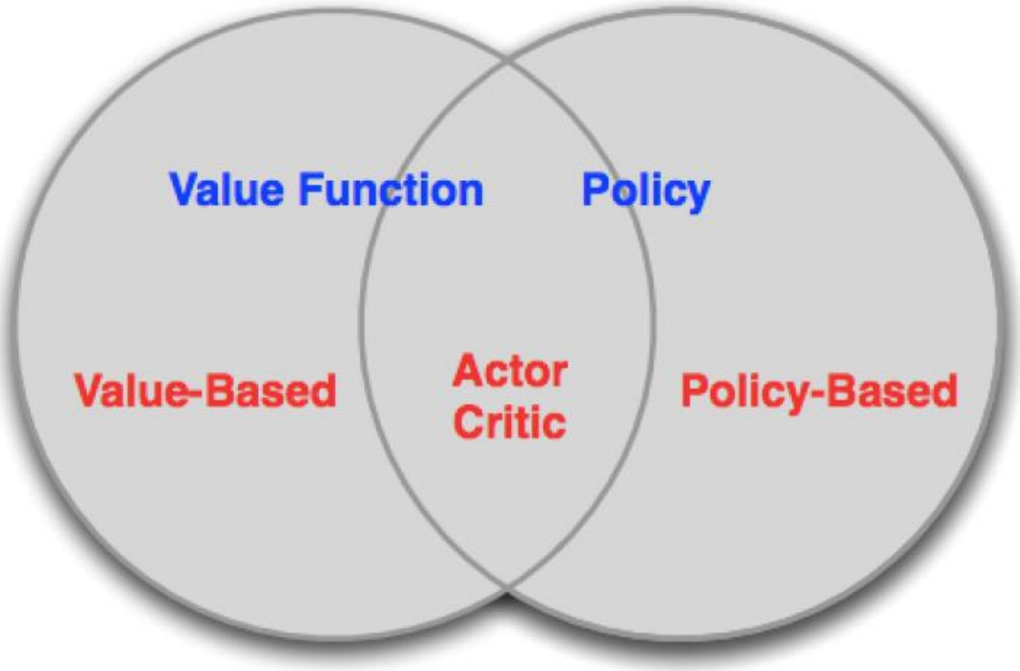


Recall: Policy-based Reinforcement Learning

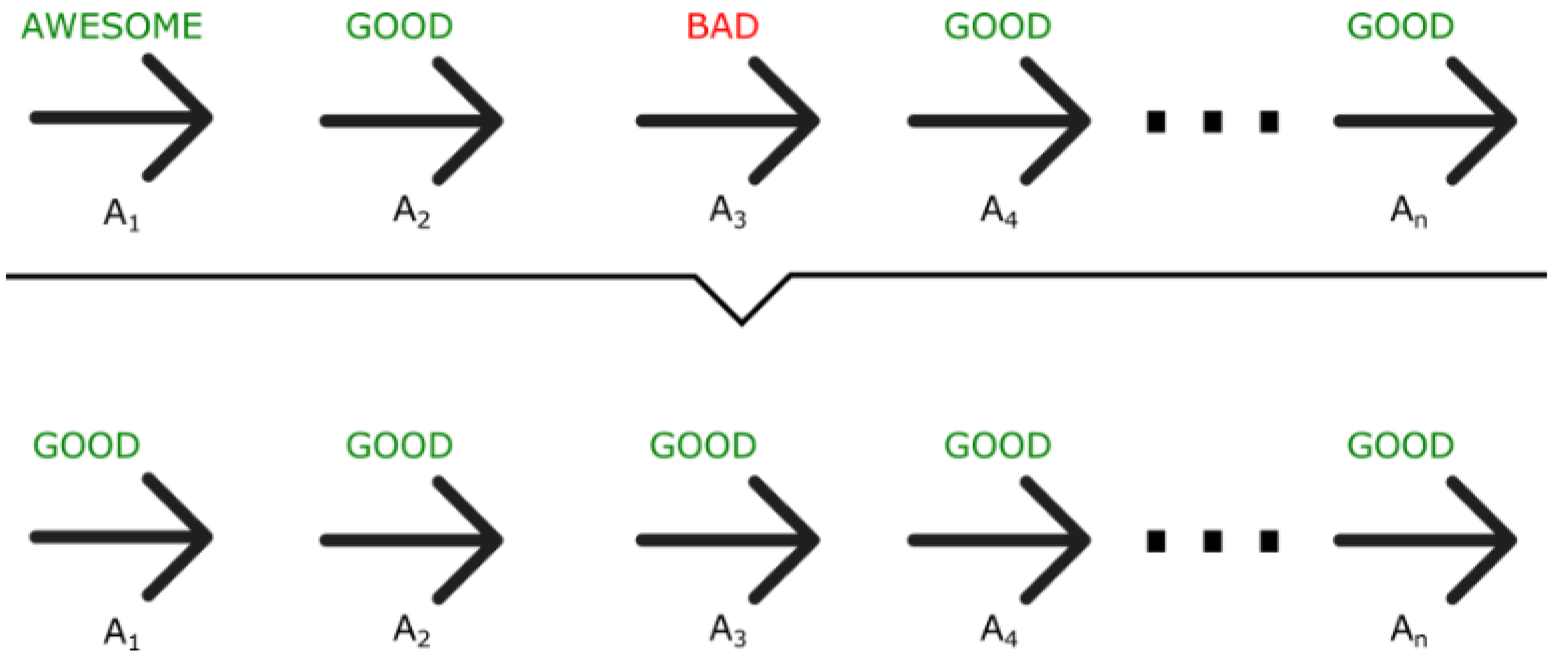
- Value-based
  - Learned Value Function
  - Implicit policy (e.g. ε-greedy)

- Policy-based
  - No Value Function
  - Learned Policy

- Actor-Critic
  - Learned Value Function
  - Learned Policy



Problem Example of Policy Gradient Method



About Choosing the Target

- Monte-Carlo policy gradient still has high variance
  - $R(\tau^i)$ is an estimation of the value function from a single roll out
  - Unbiased but high variance

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underline{\left(\sum_{t'=1}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) - b \right)}$$

- Let's use the value estimate (critic) & **bootstrapping**

  (just like in we saw for MC vs. TD)
  - the better estimate, the lower the variance

Reducing Variance Using a Critic

- We use a critic to estimate the action-value function,

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain two sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic

- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q_w(s,a)\right]$$

Estimating the Advantage Function

- The advantage function can significantly reduce variance of policy gradient.
- So the critic should really estimate the advantage function.
- For example, by estimating both V(s) and Q(s, a)
- Using two function approximates and two parameter vectors,

$$V_v(s) \approx V^{\pi_\theta}(s)$$

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

$$A(s, a) = Q_w(s, a) - V_v(s)$$

$$A(s_t, a_t) = (r_{t+1} + \gamma V_v(s_{t+1})) - V_v(s_t)$$

Various Forms of the Policy Gradient

$$\nabla_\theta J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\Phi_t\right]$$

$$\Phi_t = R(\tau)$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

$$\Phi_t = Q^{\pi_\theta}(s_t, a_t)$$

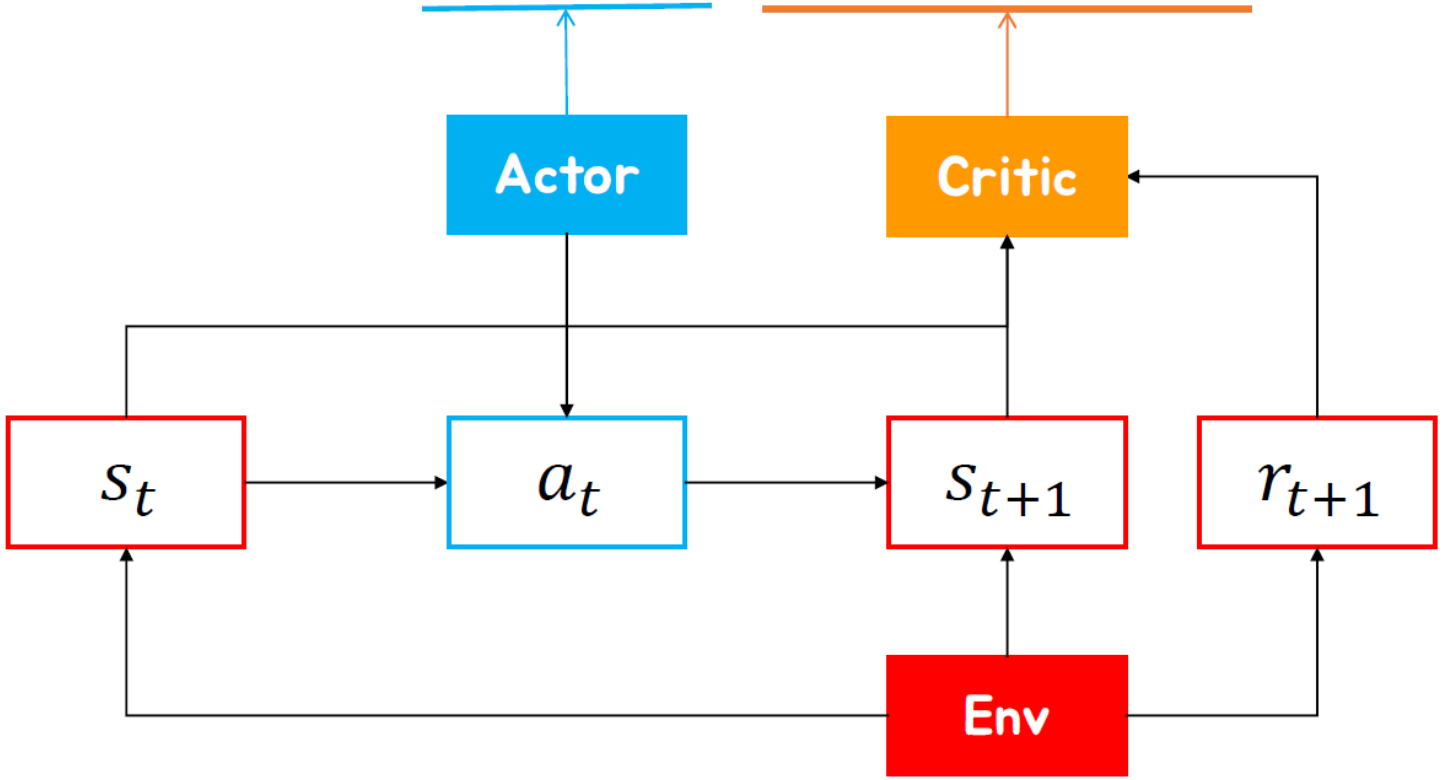
$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t)$$

$$\Phi_t = A^{\pi_\theta}(s_t, a_t)$$

https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html#deriving-the-simplest-policy-gradient

Actor-Critic Overview

$$-\underbrace{\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{Actor}}\underbrace{(r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t)\ )}_{\text{Critic}}$$



Actor-Critic Algorithm

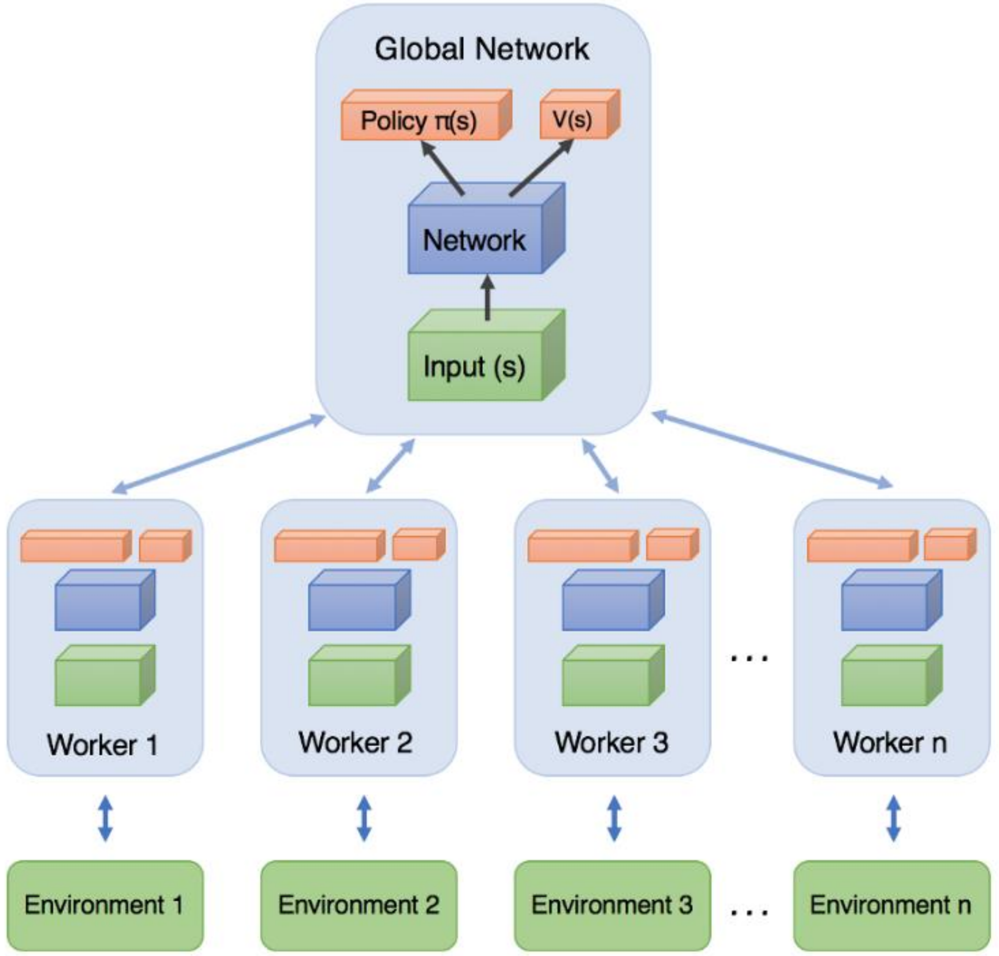
One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

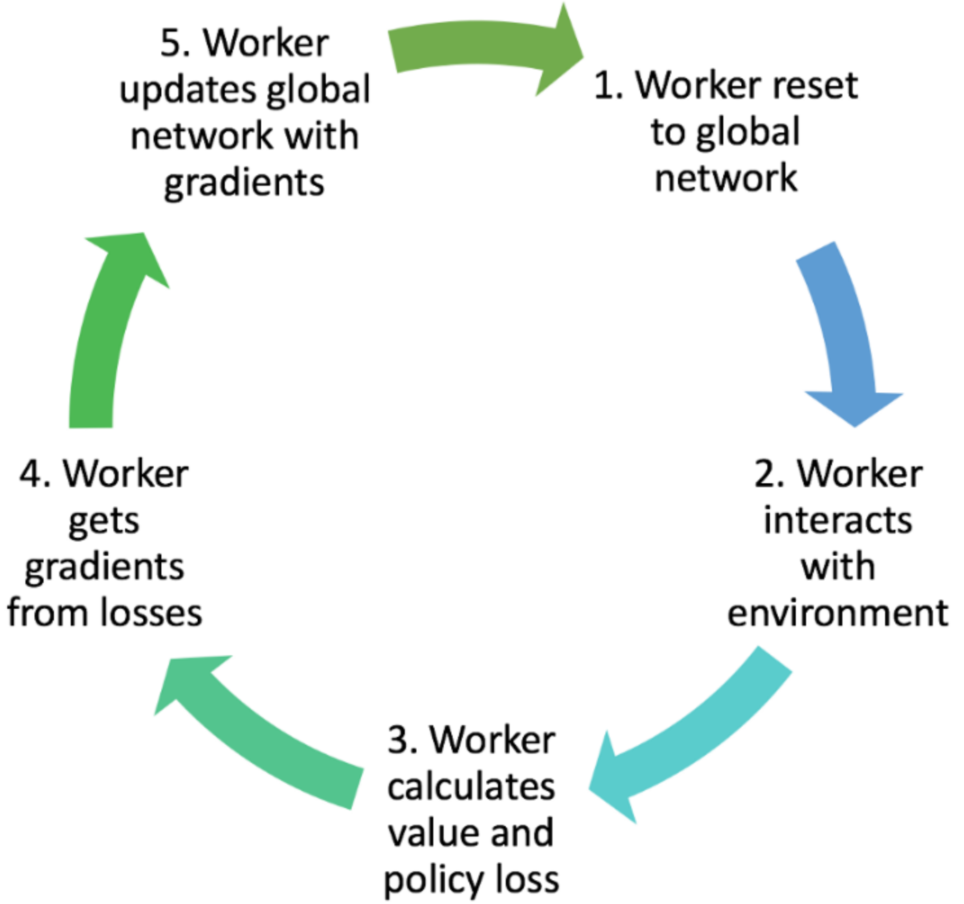
A3C : Asynchronous Advantage Actor-Critic [ICML 2016]

- Parallel actor-learners
  - Asynchronous gradient descent using multi-threads
  - A single multi-core CPU instead of a GPU
- N-step returns to update both the policy and the value-function
- Could work in continuous as well as discrete action space

A3C Architecture



Individual Agent's Training Workflow in A3C



Asynchronous one-step Q-learning

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
    Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
    Receive new state s' and reward r
    $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$
    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s = s'$
    $T \leftarrow T + 1$ and $t \leftarrow t + 1$
    if $T \mod I_{target} == 0$ **then**
        Update the target network $\theta^- \leftarrow \theta$
    end if
    if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
        Perform asynchronous update of θ using $d\theta$.
        Clear gradients $d\theta \leftarrow 0$.
    end if
until $T > T_{max}$

One-step Q-learning

One-step SARSA

Asynchronous N-step Q-learning

Algorithm S2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

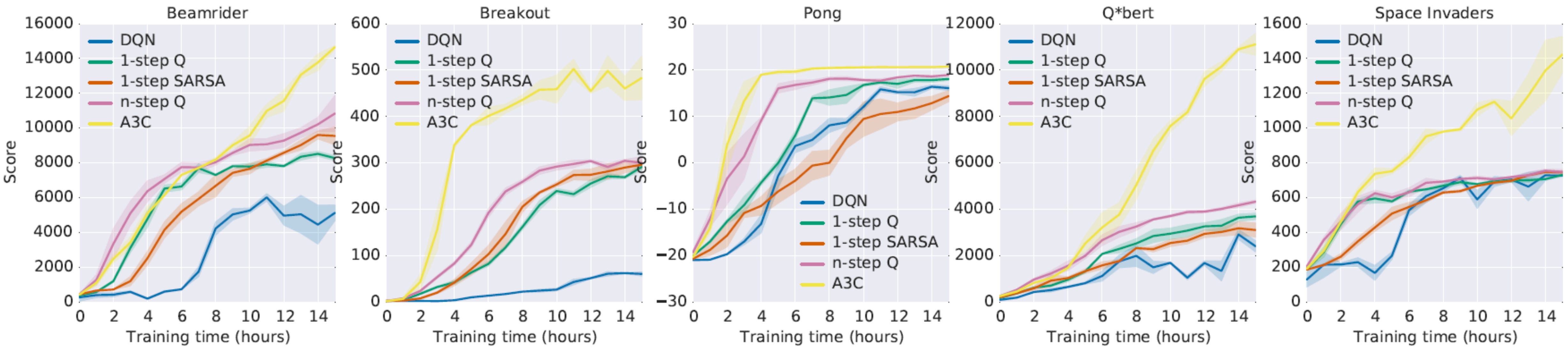
// Assume global shared parameter vector θ.
// Assume global shared target parameter vector θ^-.
// Assume global shared counter $T = 0$.
Initialize thread step counter $t \leftarrow 1$
Initialize target network parameters $\theta^- \leftarrow \theta$
Initialize thread-specific parameters $\theta' = \theta$
Initialize network gradients $d\theta \leftarrow 0$
repeat
    Clear gradients $d\theta \leftarrow 0$
    Synchronize thread-specific parameters $\theta' = \theta$
    $t_{start} = t$
    Get state s_t
    repeat
        Take action a_t according to the ϵ-greedy policy based on $Q(s_t, a; \theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ \max_a Q(s_t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \frac{\partial \left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$
    end for
    Perform asynchronous update of θ using $d\theta$.
    if $T \mod I_{target} == 0$ **then**
        $\theta^- \leftarrow \theta$
    end if
until $T > T_{max}$

Asynchronous Advantage Actor-Critic

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

```
// Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter t ← 1
repeat
    Reset gradients: dθ ← 0 and dθ_v ← 0.
    Synchronize thread-specific parameters θ' = θ and θ'_v = θ_v
    t_start = t
    Get state s_t
    repeat
        Perform a_t according to policy π(a_t|s_t; θ')
        Receive reward r_t and new state s_{t+1}
        t ← t + 1
        T ← T + 1
    until terminal s_t or t − t_start == t_max
    R = { 0            for terminal s_t
        { V(s_t, θ'_v)  for non-terminal s_t// Bootstrap from last state
    for i ∈ {t − 1, ..., t_start} do
        R ← r_i + γR
        Accumulate gradients wrt θ': dθ ← dθ + ∇_θ' log π(a_i|s_i; θ')(R − V(s_i; θ'_v))
        Accumulate gradients wrt θ'_v: dθ_v ← dθ_v + ∂(R − V(s_i; θ'_v))^2/∂θ'_v
    end for
    Perform asynchronous update of θ using dθ and of θ_v using dθ_v.
until T > T_max
```

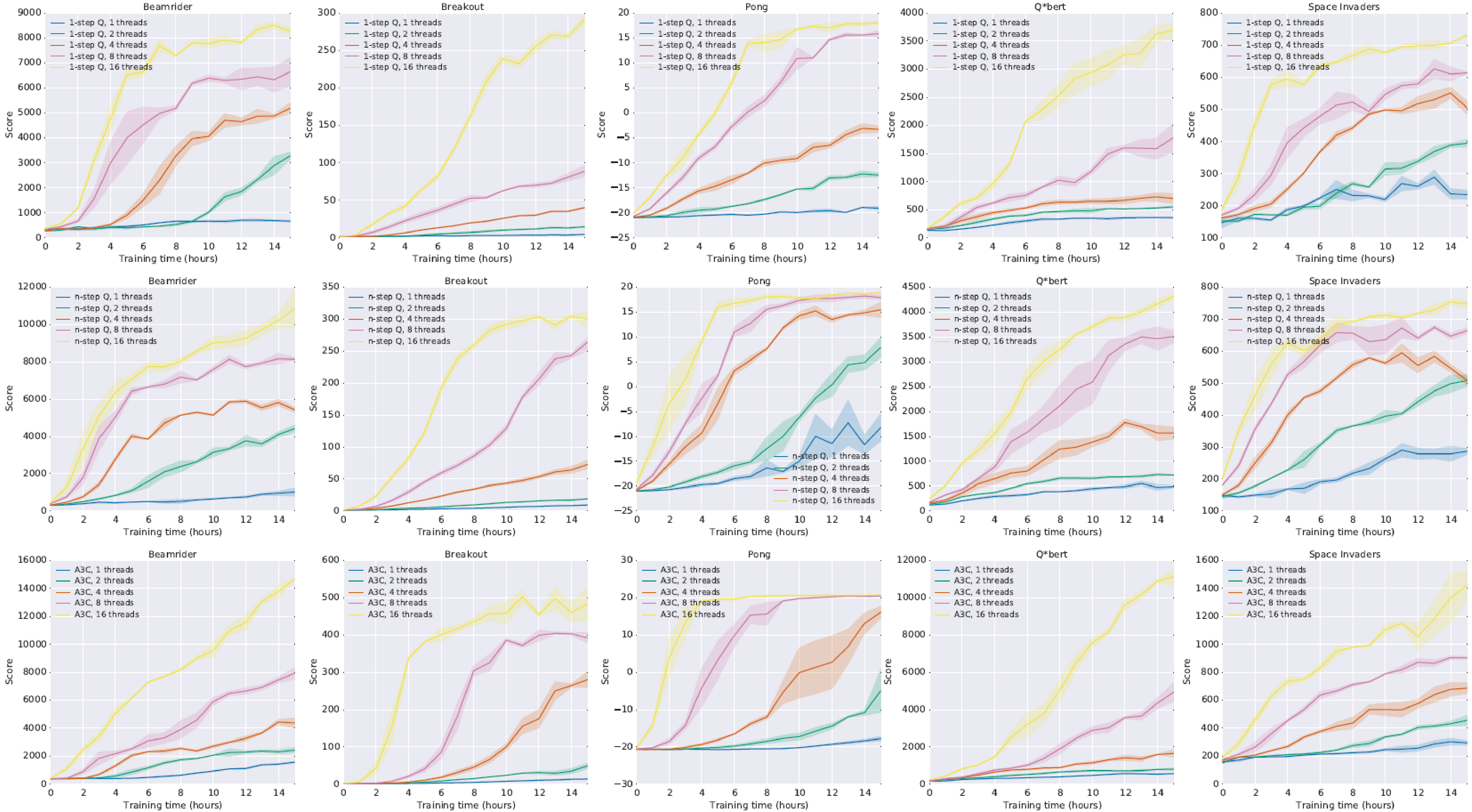
Learning Speed Comparison



DQN : Nature 2015 DQN (Experience Replay + Fixed Target)

Performance Comparison on 57 Atari games

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%


Beamrider
Breakout
Pong
Q*bert
Space Invaders
1-step Q, 1 threads
1-step Q, 2 threads
1-step Q, 4 threads
1-step Q, 8 threads
1-step Q, 16 threads
n-step Q, 1 threads
n-step Q, 2 threads
n-step Q, 4 threads
n-step Q, 8 threads
n-step Q, 16 threads
A3C, 1 threads
A3C, 2 threads
A3C, 4 threads
A3C, 8 threads
A3C, 16 threads
Score
Training time (hours)

Training speedup for number of threads

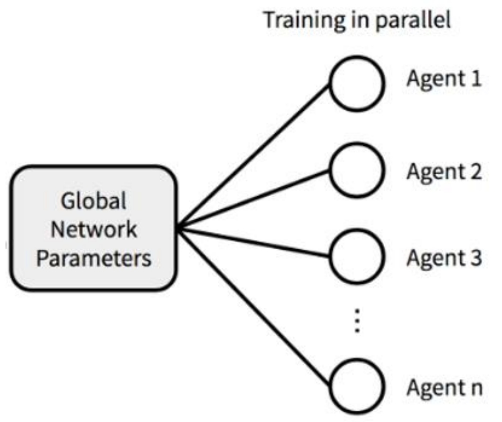
	Number of threads				
Method	1	2	4	8	16
1-step Q	1.0	**3.0**	**6.3**	**13.3**	**24.1**
1-step SARSA	1.0	**2.8**	**5.9**	**13.1**	**22.1**
n-step Q	1.0	**2.7**	**5.9**	**10.7**	**17.2**
A3C	1.0	2.1	3.7	6.9	12.5

Super-linear speedups

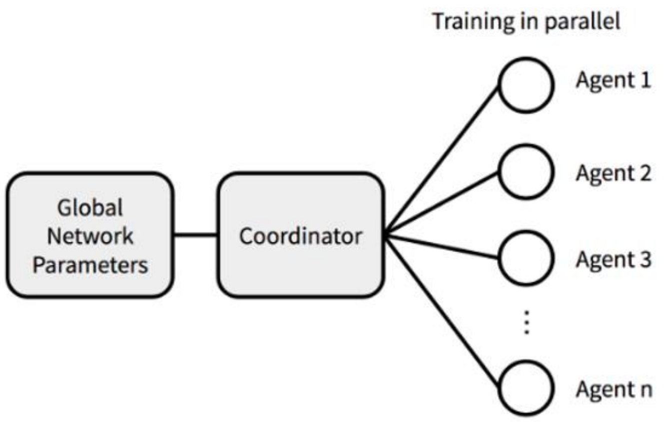
Synchronous version of A3C

- Agents in A3C work with the global parameters independently, so they would play with policies of different versions.

- Synchronized gradient update keeps the training more cohesive and potentially to make convergence faster.



A3C (Async)



A2C (Sync)

https://blog.openai.com/baselines-acktr-a2c/

Interim Check !

- REINFOCE : Policy-gradient + Monte Carlo
- Vanilla Policy-gradient : REINFOCE + Reward To Go + Baseline
- Actor-Critic : Policy-gradient + Critic (Bootstrapping)
- A3C : Actor-Critic + Asynchronous + Advantage + N-step
- Synchronous Version of A3C

Policy Gradient and Step Sizes

- Gradient descent approaches update the weights a step in direction of gradient

- Is it possible that each step of policy gradient yields an updated policy π' whose value is greater than or equal to the prior policy $\pi : v^{\pi'} \geq v^{\pi}$

Trust Region Policy Optimization

[ICML 2015]

Why are step sizes a big deal in RL?

- Step size is important in any problem involving finding the optima of a function
- Supervised learning: Step too far → next updates will fix it
- Reinforcement learning
- Step too far → bad policy
  - Policy is determining data collect!
  - In next batch, data is collected under bad policy
  - May not be able to recover from a bad choice, collapse in performance!

Policy Gradient Methods with Auto-Step-Size Selection

- Can we automatically ensure the updated policy π' has value greater than or equal to the prior policy $\pi : v^{\pi'} \geq v^{\pi}$

- Consider this for the policy gradient setting, and hope to address this by modifying step size

Surrogate Objective

- Let $\eta(\pi)$ denote the expected return of π
- We collect data with π_{old} . Want to optimize some objective to get a new policy π
- Define $L_{\pi_{old}}(\pi)$ to be the "surrogate objective"

$$L(\pi) = \mathbb{E}_{\pi_{\mathrm{old}}}\left[\frac{\pi(a \mid s)}{\pi_{\mathrm{old}}(a \mid s)} A^{\pi_{\mathrm{old}}}(s, a)\right]$$

Find the Lower-Bound in General Stochastic Policies

- Bound the difference between and $L_{\pi_{old}}(\pi)$ and $\eta(\pi)$
- Monotonic improvement guaranteed

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - C D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}), \quad \text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$$

$$\max_{\pi} L(\pi), \text{ subject to } \overline{\mathrm{KL}}[\pi_{\mathrm{old}}, \pi] \leq \delta$$

$$\text{where } L(\pi) = \mathbb{E}_{\pi_{\mathrm{old}}}\left[\frac{\pi(a \mid s)}{\pi_{\mathrm{old}}(a \mid s)} A^{\pi_{\mathrm{old}}}(s, a)\right]$$

Trust Region Policy Optimization Algorithm

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.
for $i = 0, 1, 2, \ldots$ until convergence **do**
Compute all advantage values $A_{\pi_i}(s, a)$.
Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

where $C = 4\epsilon\gamma/(1-\gamma)^2$

and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

Guaranteed Improvement

$$\pi_{i+1} = \arg\max \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

$$\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq \ldots.$$

$$M_i(\pi) = L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi)$$

$$\eta(\pi_{i+1}) \geq M_i(\pi_{i+1})$$

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - C D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi})$$

$$\eta(\pi_i) = M_i(\pi_i)$$

$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M(\pi_i)$$

TRPO Performance Results

	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
Random	354	1.2	0	−20.4	157	110	179
Human (Mnih et al., 2013)	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Learning (Mnih et al., 2013)	4092	168.0	470	20.0	1952	1705	581
UCC-I (Guo et al., 2014)	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

500 iterations about 30 hours on a 16-core computer

Proximal Policy Optimization [OpenAI 2017]

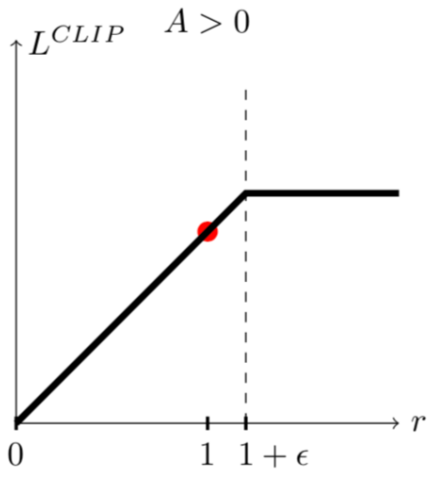
- PPO is motivated by TRPO, and is significantly simpler to implement
  - some of the benefits of TRPO
  - simpler to implement
  - more general
  - better sample efficiency

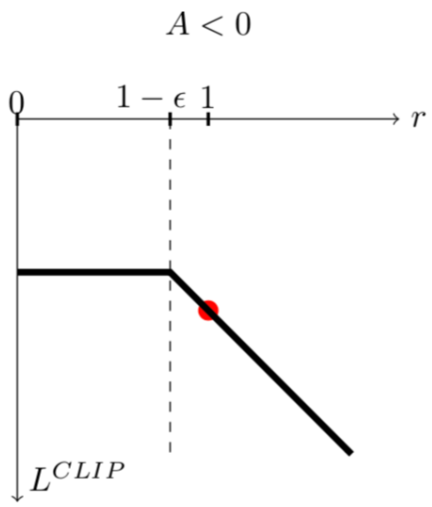
$$\begin{aligned}\underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.\end{aligned}$$

Clipped Surrogate Objective

- Lower bound of unclipped objective

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





$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$$

PPO Algorithm

for iteration$=1, 2, \ldots$ **do**

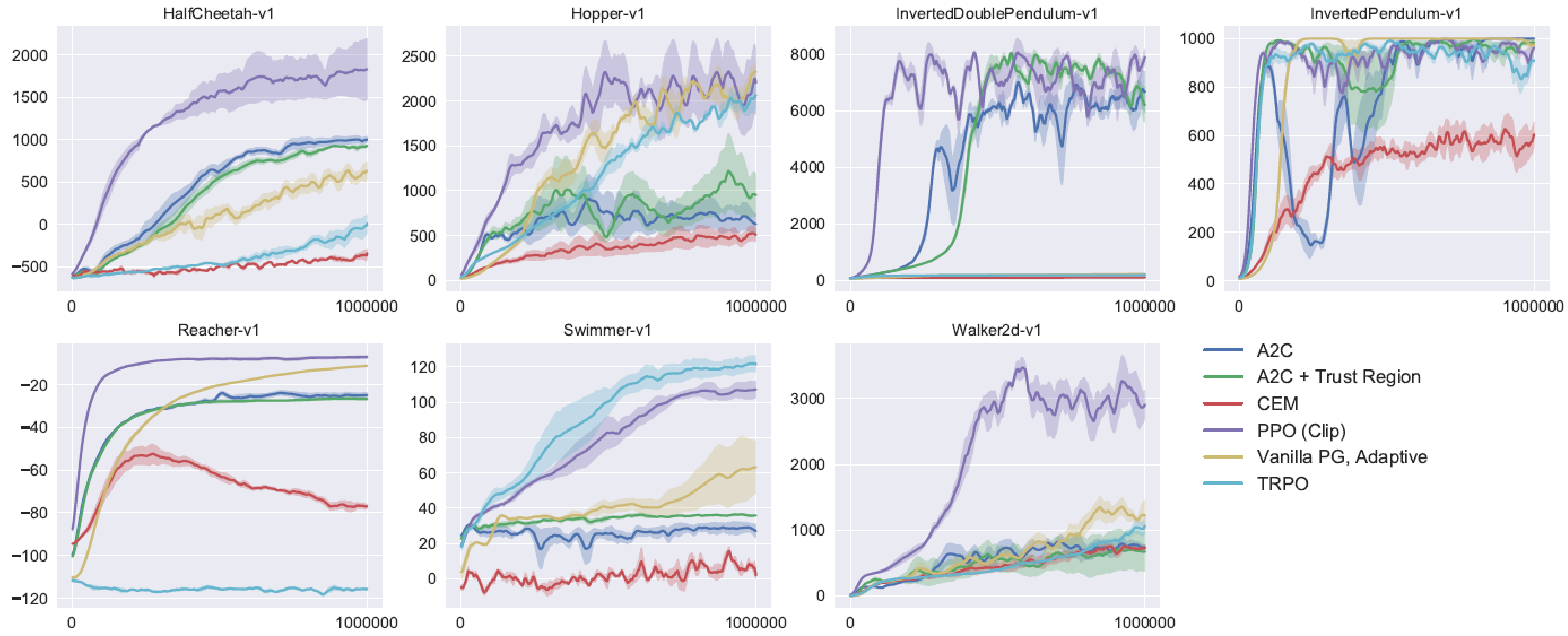
  Run policy for T timesteps or N trajectories

  Estimate advantage function at all timesteps

  Do SGD on $L^{CLIP}(\theta)$ objective for some number of epochs

end for

Comparison on MoJoCo Environments



Comparison on the Atari Domain

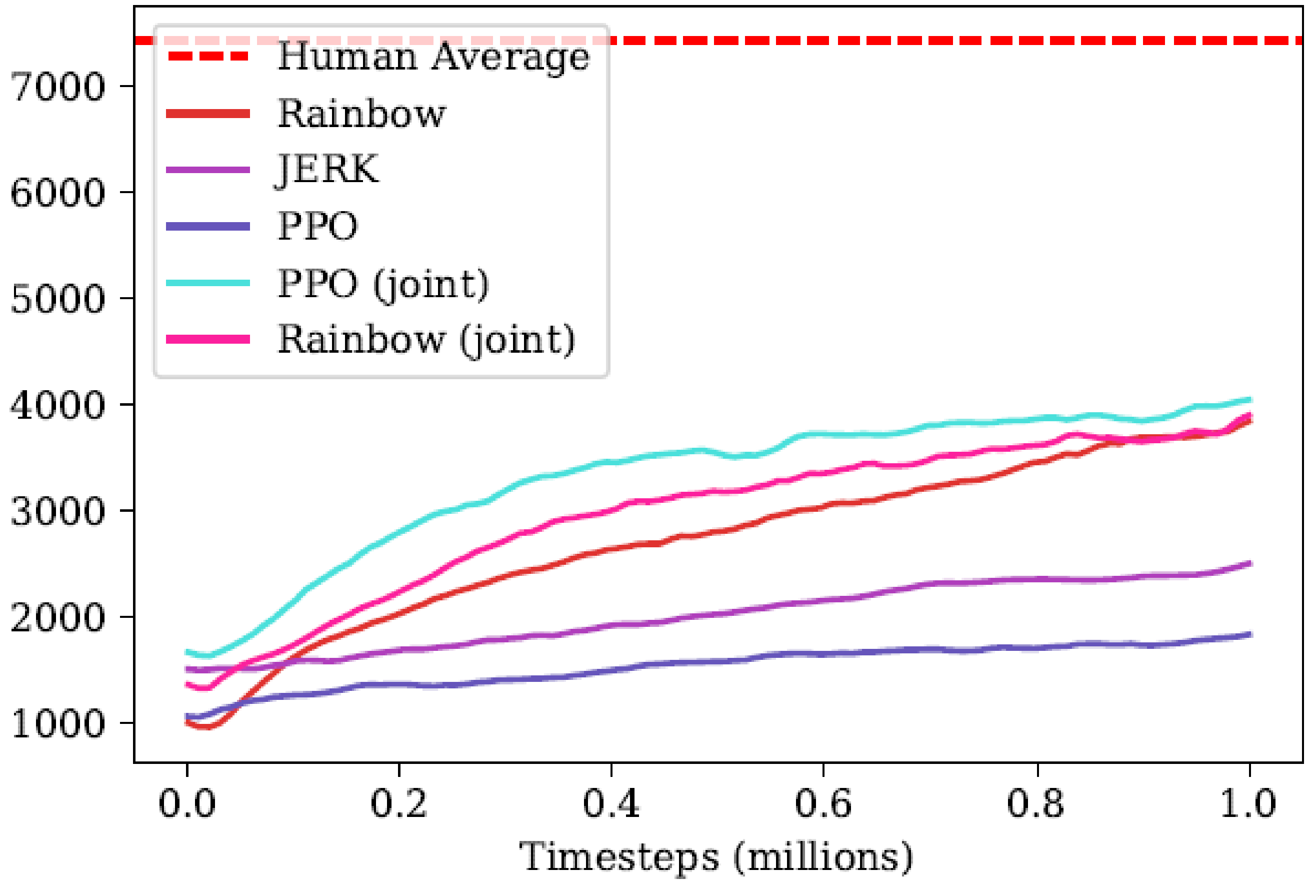
	A2C	ACER	PPO	Tie
(1) avg. episode reward over all of training	1	18	**30**	0
(2) avg. episode reward over last 100 episodes	1	**28**	19	1

Table 2: Number of games "won" by each algorithm, where the scoring metric is averaged across three trials.

ACER [ICLR 2017] : A3C + Experience Replay + Trust Region Policy Optimization

Open AI The Sonic Benchmark: Train & Test set

Performance Comparison



"joint" means that it trains a policy on all training sets and then use it as an initialization on test sets.

Unity Obstacle Tower Challenge

https://youtu.be/owKdLnCjy3o

Challenge Awards:

https://blogs.unity3d.com/kr/2019/08/07/announcing-the-obstacle-tower-challenge-winners-and-open-source-release/