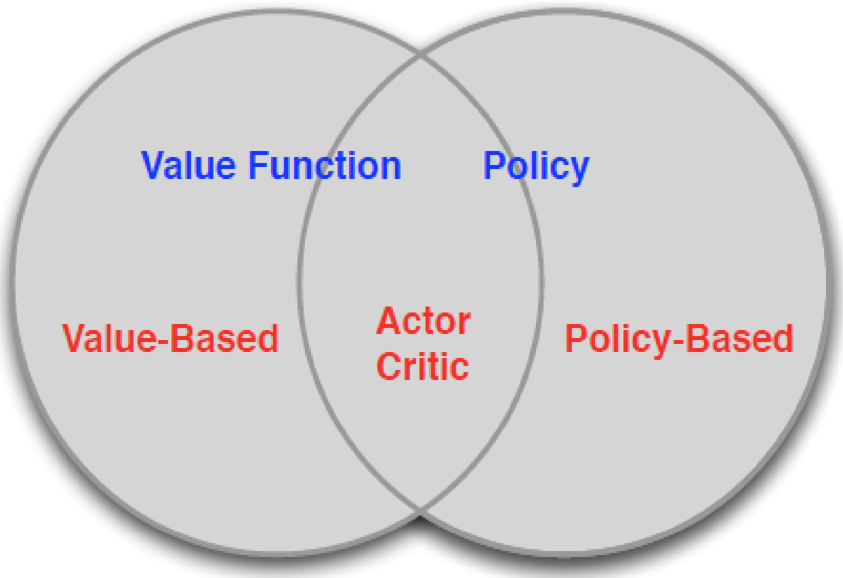
Outline of This Course

- RL1: Introduction to Reinforcement Learning
- RL2: Reinforcement Learning for Lightweight Model
  - Applications
  - Fundamentals of RL
- RL3: Value Based Reinforcement Learning
  - Fundamentals of Value Based RL
  - Algorithms
- RL4: Policy-based Reinforcement Learning
  - Fundamentals of Policy Based RL
  - Algorithms

Value-Based and Policy-Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ε-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



References

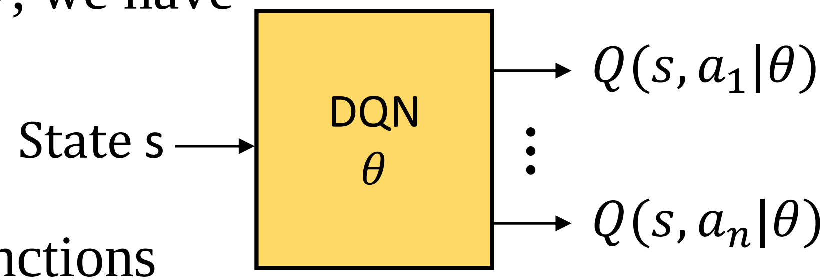
- A3C
  - Asynchronous Methods for Deep Reinforcement Learning
    - *Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu*
    - Google DeepMind, Montreal Institute for Learning Algorithms (MILA), University of Montreal
- TRPO
  - Trust Region Policy Optimization
    - *Schulman, J., et al. Trust region policy optimization. In: International Conference on Machine Learning. 2015. p. 1889-1897.*
- PPO
  - Proximal Policy Optimization Algorithms
    - *Schulman, J., et al. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.*
- Contributors for the slides include: 蔡承倫, 林九州, 何國豪, etc.

Policy-Based Reinforcement Learning

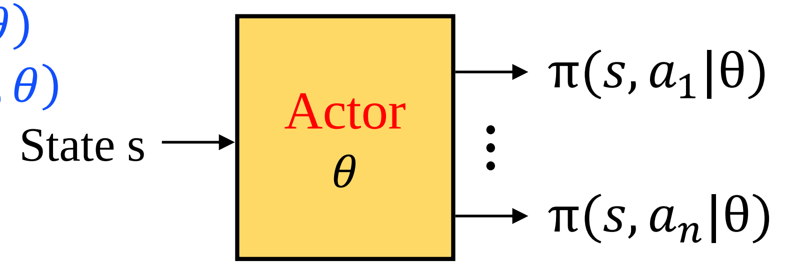
- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Policy-Based Reinforcement Learning

- By approximation with parameters θ, we have
  $V_\theta(s) \approx V^\pi(s)$
  $Q_\theta(s,a) \approx Q^\pi(s,a)$



- A policy for value-based was generated directly from the value functions
  - e.g. using greedy or ε-greedy
  - This implies: the policy is also parametrized by θ.
- For policy-based, we directly parametrize the policy in actor
  - Deterministic: $a = \pi_\theta(s)$, or $a = \pi(s,\theta)$
  - Stochastic: $\pi_\theta(s,a)$, $\pi_\theta(a|s)$, or $\pi(a|s,\theta)$



- We will focus again on model-free reinforcement learning

An Example

- DQN outputs the values of actions. (Up/Down/Left/Right)
- Actor outputs the policy, probability of selecting actions.

State s → DQN θ

$Q(s, \Uparrow \mid \theta) = 500$

$Q(s, \Downarrow \mid \theta) = 300$

$Q(s, \Leftarrow \mid \theta) = 600$

$Q(s, \Rightarrow \mid \theta) = 700$

State s → Actor θ

$\pi(s, \Uparrow \mid \theta) = 0.1$

$\pi(s, \Downarrow \mid \theta) = 0.1$

$\pi(s, \Leftarrow \mid \theta) = 0.3$

$\pi(s, \Rightarrow \mid \theta) = 0.5$

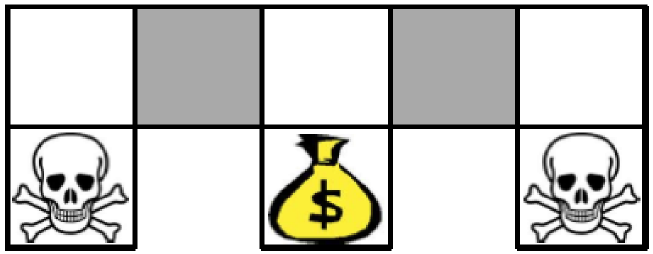
Advantages of Policy-Based RL

- Advantages:
  - Better convergence properties
    - Recall grid world with equal policy for left/up/right/down operations.
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
- Disadvantages:
  - Typically converge to a local rather than global optimum
  - Evaluating a policy is typically inefficient and high variance

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)
- Hard for deterministic policy

Example: Aliased Gridworld (1)



- The agent cannot differentiate the grey states, when functional approximation is used.
- Consider features of the following form (for all N, E, S, W)

$$\phi(s,a) = 1(\text{wall to N}, a = \text{move E})$$

- Compare value-based RL, using an approximate value function

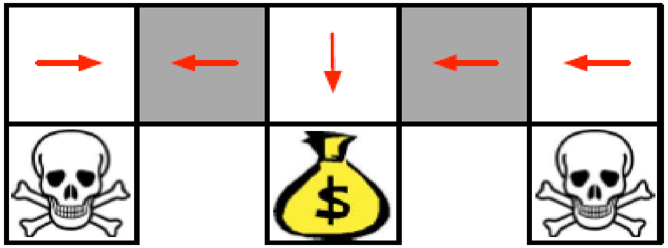
$$Q_\theta(s,a) = f(\phi(s,a),\theta)$$

- To policy-based RL, using a parametrized policy

$$\pi_\theta(s,a) = g(\phi(s,a),\theta)$$

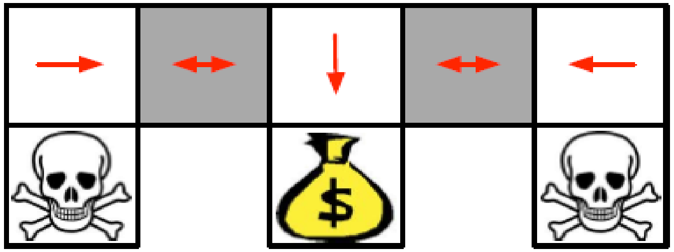
- Difficult for deterministic policy with approximator

Example: Aliased Gridworld (2)



- Under aliasing, an optimal deterministic policy will either
  - move W in both grey states (shown by red arrows)
  - 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)



- 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 Objective Functions

- Goal:
  - given policy $\pi_\theta(s, a)$ with parameters θ, find best θ
    - What does the best mean?
    - How do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_0(\theta) = V^{\pi\theta}(s_0) = \mathbb{E}_{\pi_\theta}[v_0]$$

- 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

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

  - 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 $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Simplex / amoeba / Nelder Mead
  - Genetic algorithms
- Greater efficiency often possible using gradient
  - Gradient descent
  - Conjugate gradient
  - Quasi-newton
- We focus
  - on gradient descent, many extensions possible
  - And on methods that exploit sequential structure

Policy Gradient

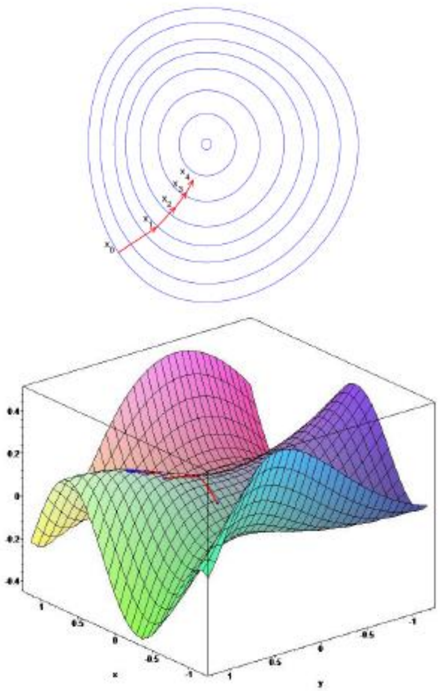
- Let $J(\theta)$ be any policy objective function
- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters θ

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

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

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

  - and α is a step-size parameter



Computing Gradients By Finite Differences

- To evaluate policy gradient of $\pi_\theta(s,a)$
- For each dimension $k \in [1,n]$
  - Estimate kth partial derivative of objective function w.r.t. θ
  - By perturbing θ by small amount ϵ in kth dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

    - where u_k is unit vector with 1 in kth component, 0 elsewhere
  - Uses n evaluations to compute policy gradient in n dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable

Policy Gradient (One Step)

- Consider a simple class of one-step MDPs
- Starting in state $s_0 \sim d(s)$
- Terminating after one time-step with reward $r = R_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$J_0(\theta) = V^{\pi_\theta}(s_0) = \mathbb{E}_{\pi_\theta}[r] = \sum_{a \epsilon A} \pi_\theta(s_0, a)\, R_{s_0,a}$$

$$\nabla_\theta J_0(\theta) = \sum_{a \epsilon A} \nabla_\theta \pi_\theta(s_0, a)\, R_{s_0,a}$$

$$= \sum_{a \epsilon A} \pi_\theta(s_0, a) \nabla_\theta \log \pi_\theta(s_0, a) R_{s_0,a}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \cdot r]$$

Score function

Let $s_0 \sim d(s)$

$$J(\theta) = \mathbb{E}_{d(s),\pi_\theta}[r]$$

$$= \sum_{s \epsilon S} d(s) \sum_{a \epsilon A} \pi_\theta(s, a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{d(s),\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \cdot r]$$

Score Function

- We now compute the policy gradient analytically
- Assume
  - policy π_θ is differentiable whenever it is non-zero
  - we know the gradient $\nabla_\theta \pi_\theta(s,a)$
- Likelihood ratios exploit the following identity

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}$$

$$= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)$$

  - $\nabla_\theta \log \pi_\theta(s,a)$ is called the score function.

Softmax Policy

- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^T\theta}$$

  - Weight actions using linear combination of features $\phi(s,a)^T\theta$
- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$$

- Example:
  - In Computer Go, Silver used this to solve a problem
    - Simulation Balancing

Gaussian Policy

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features $\mu_\theta(s) = \phi(s)^T\theta$
- Variance may be fixed σ^2 or can also parametrized
- Policy is Gaussian, $a \sim \mathcal{N}(\mu_\theta(s), \sigma^2)$
- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{(a - \mu_\theta(s))\phi(s)}{\sigma^2}$$

Score Function Gradient Estimator

- Consider an expectation $\mathbb{E}_{x\sim p(x|\theta)}[f(x)]$.
- The gradient w.r.t. θ is:

$$\nabla_\theta \mathbb{E}_x[f(x)] = \mathbb{E}_x[f(x)\nabla_\theta \log p(x|\theta)]$$

  - Just sample $x_i \sim p(x|\theta)$, and compute

$$\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$$

  - Need to be able to compute and differentiate density $p(x|\theta)$ w.r.t. θ
  - This gives us an unbiased gradient estimator.
  - Note: $\pi_\theta(s,a)$ can be viewed as $p(x|\theta)$.

One-Step MDPs

- Consider a simple class of one-step MDPs
- Starting in state $s \sim d(s)$
- Terminating after one time-step with reward $r = R_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r]$$

$$= \sum_{s \epsilon S} d(s) \sum_{a \epsilon A} \pi_\theta(s,a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \epsilon S} d(s) \sum_{a \epsilon A} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) R_{s,a}$$

$$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \cdot r]$$

Policy Gradient Theorem

- Comments:
  - The policy gradient theorem generalizes the likelihood ratio approach to multi-step MDPs
  - Replaces instantaneous reward r with long-term value $Q^{\pi}(s,a)$
  - Policy gradient theorem applies to start state objective, average reward and average value objective
- Theorem
  - For any differentiable policy $\pi_\theta(s,a)$,
  - for any of the policy objective functions $J = J_1, J_{avR}, or\ \frac{1}{1-\gamma} J_{avV}$
  - the policy gradient is

$$\nabla_\theta\ J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \cdot Q^{\pi_\theta}(s,a)]$$

Monte-Carlo Policy Gradient (REINFORCE)

- Using policy gradient theorem
  - Update parameters by stochastic gradient ascent
  - Using return G_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

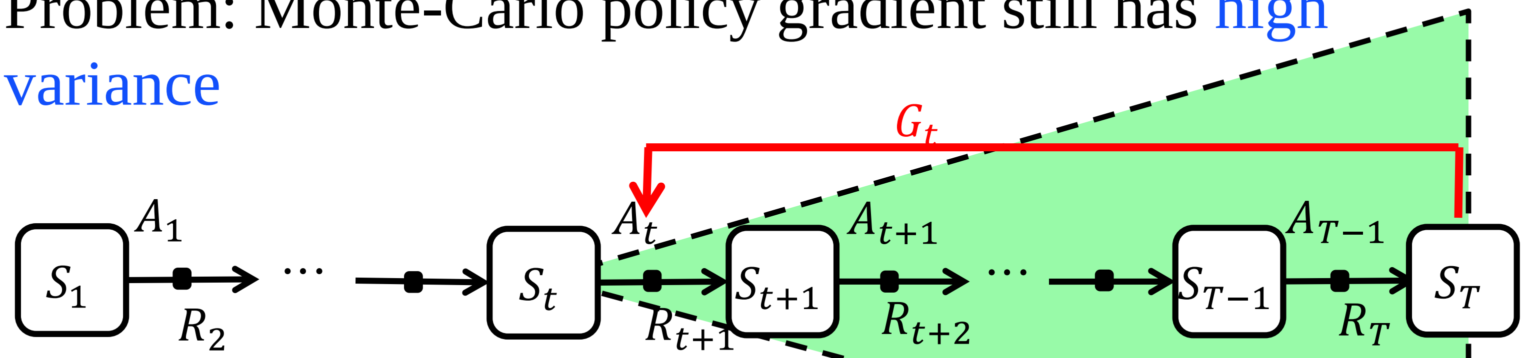
$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot G_t$$

  - If G_t is large, $\Delta\theta_t$ moves towards the score function more.
- Applications: Go, job-shop scheduling (hard to calculate value anyway)

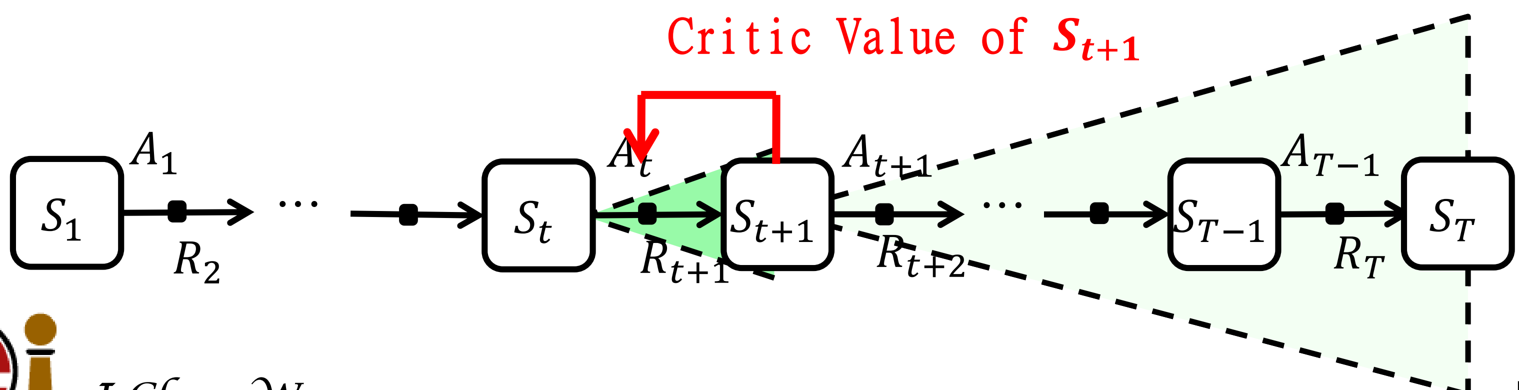
function REINFORCE
  Initialize θ arbitrarily
  for each episode $\{s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_t\} \sim \pi_\theta$ **do**
    for $t = 1$ to $T - 1$ **do**
      $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot G_t$
    end for
  end for
  return θ
end function

Problem of REINFORCE

- Problem: Monte-Carlo policy gradient still has high variance



- Solution: Actor Critic
  - Policy gradient based on the Critic value of S_{t+1}



Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Reducing Variance Using a Critic

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

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

- 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

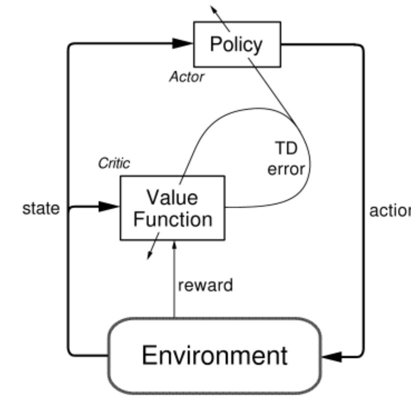
$$\nabla_\theta \ J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)]$$

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) \cdot Q_w(s, a)$$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
- But, how good is policy π_θ for current parameters θ?
- This problem was explored in previous two chapters, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - TD(λ)
- Could also use e.g. least-squares policy evaluation

Actor-Critic (Discrete Action Space)



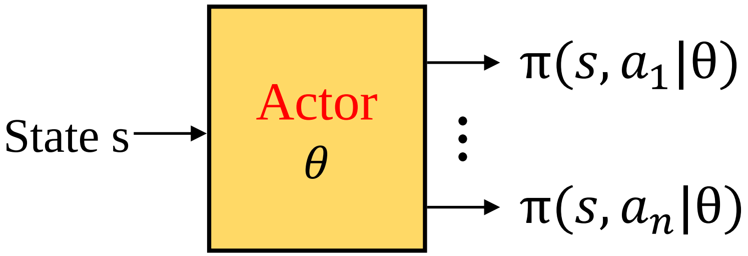
- Use two networks: an actor and a critic
  - Critic estimates the action-value function
    - Gradient:

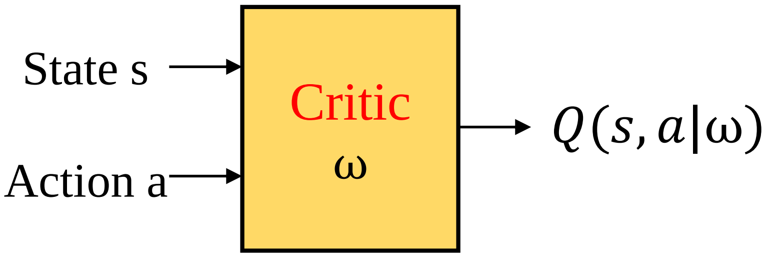
$$\nabla_\omega L_Q(s_t, a_t|\omega) = \left(\left(r_{t+1} + \gamma Q(s_{t+1}, a'|\omega)\right) - Q(s_t, a_t|\omega)\right)\nabla_\omega Q(s_t, a_t|\omega)$$

  - Actor updates policy in direction suggested by critic
    - Gradient (approximate policy gradient):

$$J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[Q(s, a|\omega)]$$

$$\nabla_\theta J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[\boldsymbol{\nabla_\theta \log \pi(s_t, a_t|\theta)}\, Q(s_t, a_t|\omega)]$$





Actor-Critic (Discrete Action Space)

- Using linear value function approx. $Q_w(s,a) = \varphi(s,a)^T w$
  - Critic: Updates w by linear TD(0)
  - Actor: Updates θ by policy gradient

function QAC
  Initialise s, θ
  Sample $a \sim \pi_\theta$
  for each step **do**
    Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$
    Sample action $a' \sim \pi_\theta(s', a')$
    $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
    $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
    $w \leftarrow w + \beta \delta \phi(s, a)$
    $a \leftarrow a', s \leftarrow s'$
  end for
end function

Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Reducing Variance Using a Baseline

- Recall: $\nabla_\theta J(\theta) = \mathrm{E}_{s,a}^{\pi_\theta}[\boldsymbol{\nabla_\theta \log \pi(s_t, a_t|\theta)}\, Q(s_t, a_t|\omega)]$
- Problem: Can we further reduce variance?
- Solution:
  - This can reduce variance, without changing expectation

$$\mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)B(s)] = \sum_{s\epsilon S} d^{\pi\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a)B(s)$$
$$= \sum_{s\epsilon S} d^{\pi_\theta} B(s) \nabla_\theta \left(\underbrace{\boxed{\sum_{a\epsilon A} \pi_\theta(s,a)}}_{=1\text{ (constant)}}\right)$$
$$= 0$$

  - Subtract a baseline function $B(s)$ from the policy gradient
    - A good baseline is the state value function $B(s) = V^{\pi_\theta}(s)$
- So we can rewrite the policy gradient using the advantage function $A^{\pi_\theta}(s,a)$

$$A^{\pi_\theta}(s) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)$$
$$\nabla_\theta\ J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

Estimating the Advantage Function (1)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
  - For example, by estimating both $V^{\pi_\theta}(s)$ and $Q^{\pi_\theta}(s,a)$
  - Using two function approximators and two parameter vectors,

$$V_v(s) \approx V^{\pi_\theta}(s)$$
$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a)$$
$$A(s,a) = Q_w(s,a) - V_v(s)$$

- And updating both value functions by e.g. TD learning

Estimating the Advantage Function (2)

- For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

  - is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_\theta}[\delta^{\pi_\theta}|\mathrm{s},\mathrm{a}] &= \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s,a] - V^{\pi_\theta}(s)\\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s)\\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

- So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\delta^{\pi_\theta}]$$

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

- This approach only requires one set of critic parameters v

Critics at Different Time-Scales

- Critic can estimate value function $V_\theta(s)$ from many targets at different time-scales From last lecture...
  - For MC, the target is the return v_t
    $$\Delta\theta = \alpha(v_t - V_\theta(s))\phi(s)$$
  - For TD(0), the target is the TD target $\mathrm{r} + \gamma V(s')$
    $$\Delta\theta = \alpha(\mathrm{r} + \gamma V(s') - V_\theta(s))\phi(s)$$
  - For forward-view TD(λ), the target is the λ-return v_t^λ
    $$\Delta\theta = \alpha(v_t^\lambda - V_\theta(s))\phi(s)$$
  - For backward-view TD(λ), we use eligibility traces
    $$\delta_v = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
    $$e_t = \gamma\lambda\, e_{t-1} + \phi(s_t)$$
    $$\Delta\theta = \alpha\delta_t e_t$$

Actors at Different Time-Scales

- The policy gradient can also be estimated at many time-scales

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

- Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha(v_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha(r + \gamma V_v(s_{t+1}) - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Advantage Actor-critic (A2C or A3C) policy gradient uses the (k+1)-step TD error

$$\Delta\theta = \alpha(v_t^{(k)} - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Some policy gradient algorithms (like PPO) uses TD(λ) error

$$\Delta\theta = \alpha(v_t^{\lambda} - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

Actors at Different Time-Scales

- The policy gradient can also be estimated at many time-scales

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

- Monte-Carlo policy gradient uses error from complete return

$$\Delta\theta = \alpha(v_t - V_v(s_t))\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Actor-critic policy gradient uses the one-step TD error

$$\Delta\theta = \alpha(\delta_t)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Advantage Actor-critic (A2C or A3C) policy gradient uses the (k+1)-step TD error $=A^{(k+1)}$

$$\Delta\theta = \alpha(\delta_t + \gamma\delta_{t+1} + \cdots + \gamma^k \delta_{t+k})\nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Some policy gradient algorithms (like PPO) uses TD(λ) error

$$\Delta\theta = \alpha(\delta_t + \lambda\gamma\delta_{t+1} + \cdots + (\lambda\gamma)^k \delta_{t+k} + \cdots)\nabla_\theta \log \pi_\theta(s_t, a_t)$$

$=A_t^{GAE}$: Also called GAE (Generalized Advantage Estimator)

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ v_t]$	REINFORCE
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)]$	Q Actor-Critic
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a)\ \delta]$	TD Actor-Critic
$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^{(k)}\right]$	Advantage Actor-Critic
$= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^{GAE}\right]$	TD(λ) Actor-Critic

- Each leads a stochastic gradient ascent algorithm

Appendix for Advantages and TD(λ) Errors

- TD errors
- n-step TD errors
- GAE
- Eligibility Trace

Appendix: TD Errors

- TD errors:

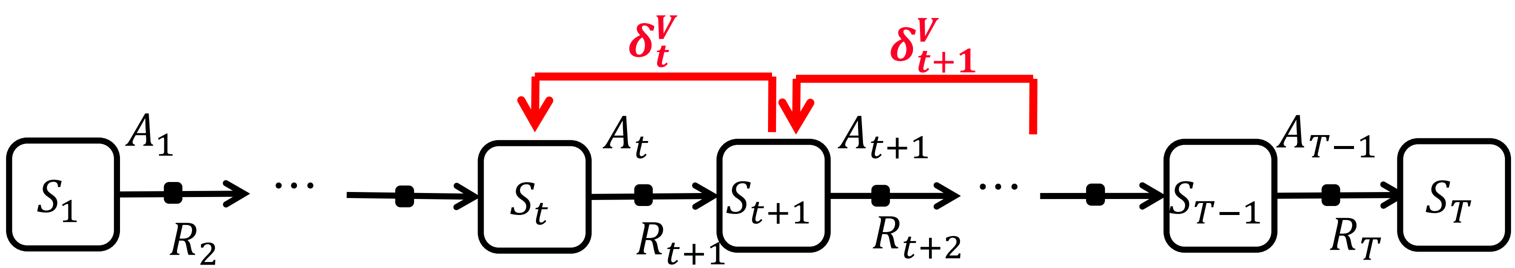
$$\delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1})$$

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

$$\delta_{t+2}^V = -V(s_{t+2}) + r_{t+2} + \gamma V(s_{t+3})$$

$$\vdots$$

$$\delta_{t+n}^V = -V(s_{t+n}) + r_{t+n} + \gamma V(s_{t+n+1})$$



Appendix: TD Errors

- Weighted TD errors:

$$\begin{aligned}
\delta_t^V &= -V(s_t) + r_t + \gamma V(s_{t+1}) \\
\gamma * \delta_{t+1}^V &= \gamma * (-V(s_{t+1}) + r_{t+1} + \gamma V(s_{t+2})) \\
\gamma^2 * \delta_{t+2}^V &= \gamma^2 * (-V(s_{t+2}) + r_{t+2} + \gamma V(s_{t+3})) \\
&\vdots \\
\gamma^n * \delta_{t+n}^V &= \gamma^n * (-V(s_{t+n}) + r_{t+n} + \gamma V(s_{t+n+1}))
\end{aligned}$$

Appendix: n-Step TD Errors

- Sum them up, becoming n-step TD errors.

$$
\begin{aligned}
\delta_t^V &= -V(s_t) + r_t + \cancel{\gamma V(s_{t+1})} \\
\gamma * \delta_{t+1}^V &= \gamma * (\cancel{-V(s_{t+1})} + r_{t+1} + \cancel{\gamma V(s_{t+2})}) \\
\gamma^2 * \delta_{t+2}^V &= \gamma^2 * (\cancel{-V(s_{t+2})} + r_{t+2} + \cancel{\gamma V(s_{t+3})}) \\
&\vdots \\
+ \quad \gamma^n * \delta_{t+n}^V &= \gamma^n * (\cancel{-V(s_{t+n})} + r_{t+n} + \gamma V(s_{t+n+1}))
\end{aligned}
$$

$$
\begin{aligned}
&\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V + \cdots \gamma^n\delta_{t+n}^V \\
&= -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n+1} V(s_{t+n+1}) \\
&= -V(s_t) + G_t^{(n+1)} \\
&= \hat{A}_t^{(n+1)}
\end{aligned}
$$

If $n \rightarrow \infty$, it becomes MC learning. Why?

Appendix: n-Step TD Errors

- n-step TD errors:

$$
\begin{aligned}
\hat{A}_t^{(1)} &:= \delta_t^V \\
\hat{A}_t^{(2)} &:= (\delta_t^V + \gamma \delta_{t+1}^V) \\
\hat{A}_t^{(3)} &:= (\delta_t^V + \gamma \delta_{t+1}^V + \gamma^2 \delta_{t+2}^V) \\
&\vdots \\
\hat{A}_t^{(n)} &:= \sum_{k=1}^{n} \gamma^{k-1} * \delta_{t+k-1}^V
\end{aligned}
$$

Appendix: n-Step TD Errors and GAE

- Weighted n-step TD errors:
  - The same trick as TD(λ)
- Then, sum them up.

$$(1-\lambda) * \hat{A}_t^{(1)} := (1-\lambda) \quad * \delta_t^V$$

$$(1-\lambda)\lambda * \hat{A}_t^{(2)} := (1-\lambda)\lambda \quad * (\delta_t^V + \gamma\delta_{t+1}^V)$$

$$(1-\lambda)\lambda^2 * \hat{A}_t^{(3)} := (1-\lambda)\lambda^2 \quad * (\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V)$$

$$\vdots$$

$$+ \quad (1-\lambda)\lambda^{n-1} * \hat{A}_t^{(n)} := (1-\lambda)\sum_{k=1}^{n}\gamma^{k-1} * \delta_{t+k-1}^V$$

$$\boxed{\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots + \lambda^{n-1}\hat{A}_t^{(n)} + \cdots)}$$

Appendix: n-Step TD Errors and GAE

- The sum of exponentially-weighted TD residuals denoted as $\hat{A}_t^{GAE(\gamma,\lambda)}$ (actually equals to $G_t^\lambda - V(S_t)$ for $TD(\lambda)$)

$$
\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + \cdots + \lambda^{n-1}\hat{A}_t^{(n)} + \cdots\right) \\
&= (1-\lambda)\left((\delta_t^V) + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma\delta_{t+2}^V) + \cdots\right) \\
&= (1-\lambda)\begin{pmatrix} \delta_t^V(1+\lambda+\lambda^2+\cdots) + \\ \gamma\lambda\delta_{t+1}^V(1+\lambda+\lambda^2+\cdots) + \\ (\gamma\lambda)^2\delta_{t+2}^V(1+\lambda+\lambda^2+\cdots) + \\ \cdots \end{pmatrix} \\
&= (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\lambda\delta_{t+1}^V\left(\frac{1}{1-\lambda}\right) + (\gamma\lambda)^2\delta_{t+2}^V\left(\frac{1}{1-\lambda}\right) + \cdots\right) \\
&= \sum_{n=0}^{\infty}(\gamma\lambda)^n\delta_{t+n}^V = \delta_t^V + \lambda\gamma\delta_{t+1}^V + \cdots + (\lambda\gamma)^k\delta_{t+k}^V + \cdots
\end{aligned}
$$

Appendix: Recall $TD(\lambda)$

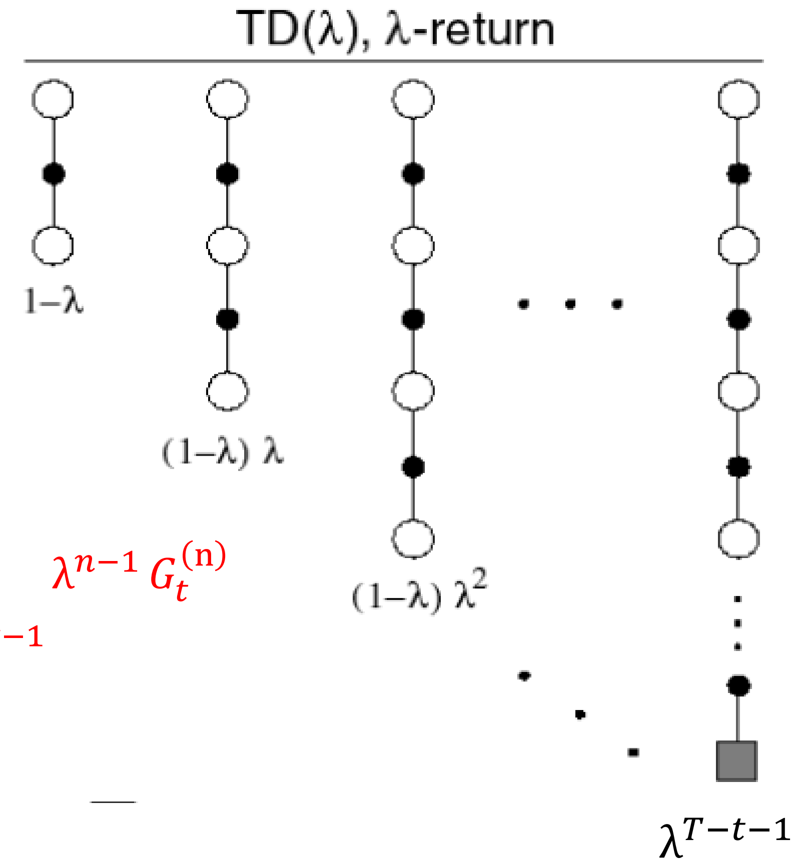
- λ-return G_t^λ:
  - combines all n-step returns $G_t^{(n)}$
- Using weight $(1-\lambda)\,\lambda^{n-1}$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\,G_t^{(n)}$$

or (in the case of termination)

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}\,G_t^{(n)} + (1-\lambda)\sum_{n=T-t-1}^{\infty}\lambda^{n-1}\,G_t^{(n)}$$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}\,G_t^{(n)} + \lambda^{T-t-1}G_t$$



- Forward-view TD(λ)

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$$

Appendix: GAE and Eligibility Trace

- Eligibility trace:

$$E_0(s) = 0$$

$$E_t(s) = (\gamma\ \lambda)\ E_{t-1}(s) + 1(S_t = s)$$

$$\hat{A}_t^{GAE(\gamma,\lambda)} = 1\ \delta_t^V + \lambda\gamma\delta_{t+1}^V + (\lambda\gamma)^2\delta_{t+2}^V + \cdots + (\lambda\gamma)^k\delta_{t+k}^V + \cdots$$

$$\hat{A}_{t+1}^{GAE(\gamma,\lambda)} = 1\ \delta_{t+1}^V + \lambda\gamma\delta_{t+2}^V + \cdots + (\lambda\gamma)^{k-1}\delta_{t+k}^V + \cdots$$

$$\hat{A}_{t+2}^{GAE(\gamma,\lambda)} = 1\ \delta_{t+2}^V + \cdots + (\lambda\gamma)^{k-2}\delta_{t+k}^V + \cdots$$

$\dots$

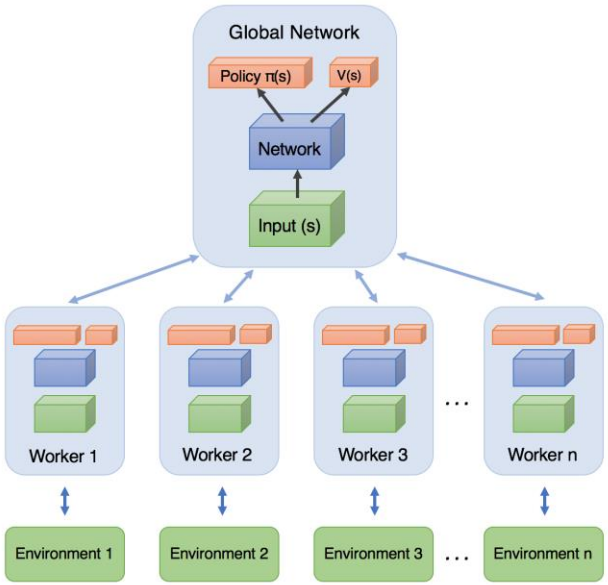
$E_t(s_t)$ $E_t(s_{t+1})$ $E_t(s_{t+2})$

Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Asynchronous Advantage Actor-critic (A3C)

- Asynchronous Lock-Free Reinforcement Learning
  - Use two main ideas to make the algorithm practical:
    - Multiple threads on a single machine
    - Multiple actor-learners applying online updates in parallel (no experience replay)



Asynchronous Advantage Actor-critic (A3C)

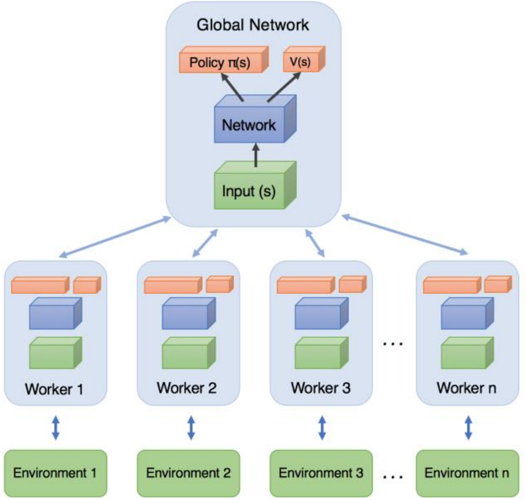
- Instead of experience replay, we asynchronously execute multiple agents in parallel.
  - Decorrelate the agents' data into a more stationary process
  - Enable a much larger spectrum of fundamental on-policy RL algorithms
- For each worker (asynchronous part):

  Copy all parameters from the global network.

  keep playing and computing gradients.

  Every N iterations:

  1. Update all gradients to the global network.
  2. Copy all new parameters from the global network

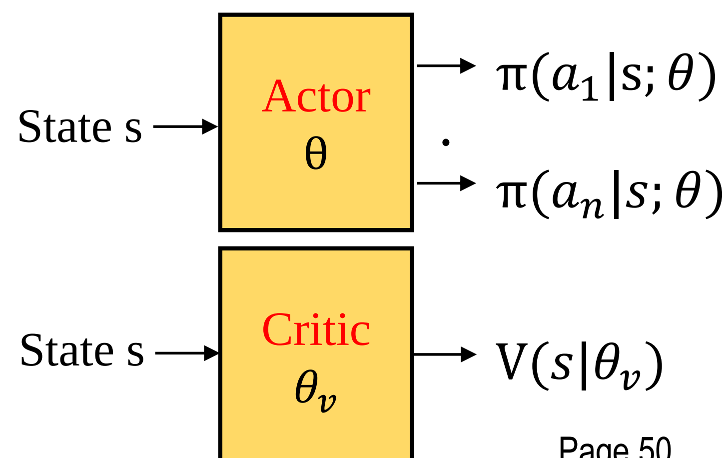


Asynchronous Advantage Actor-critic (A3C)

- Asynchronous advantage actor-critic(A3C) maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t,\theta_v)$.
- The update performed by the algorithm can be seen as $\nabla_\theta \log \pi(a_t|s_t;\theta)A(s_t,a_t;\theta,\theta_v)$

$$A(s_t,a_t;\theta,\theta_v) \rightarrow \sum_{i=0}^{k-1}\gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$$

  - Make k-step operations, and then calculate advantages backwards.
- Intuitively, the network should be

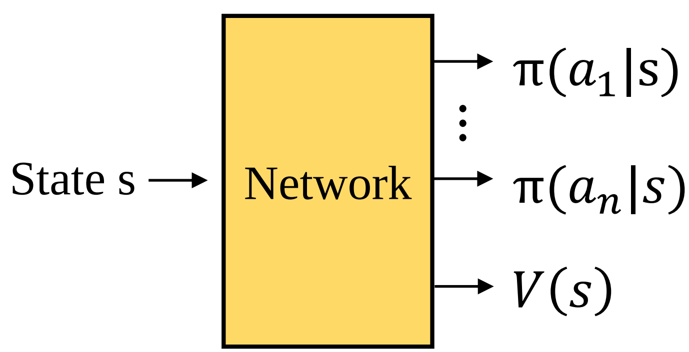


Asynchronous Advantage Actor-critic (A3C)

- Asynchronous advantage actor-critic(A3C) maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of the value function $V(s_t, \theta_v)$.
- The update performed by the algorithm can be seen as $\nabla_\theta \log \pi(a_t|s_t;\theta) A(s_t, a_t; \theta, \theta_v)$

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v)$$

  - Make k-step operations, and then calculate advantages backwards.
- Typically use a convolutional neural network that has two heads:
  - one softmax output for the policy $\pi(a_t|s_t;\theta)$
  - one output for the value function $V(s_t;\theta_v\,)$
  - all non-output layers are shared



Asynchronous Advantage Actor-critic (A3C)

θ, θ_v: global shared parameters
T: global shared counter

θ', θ'_v: thread specific parameters
t: thread step counter

repeat
    Sync $\theta' = \theta, \theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat   (note: $t = t + 1$)
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive s' and reward r
    until terminal s_t or $t - t_{start} == t_{max}$ (note: $t = t_{start} + t_{max}$, if not terminal)

$$R = \begin{cases} 0 & \text{for terminal } s' \\ V(s_t, \theta'_v) & \text{for non-terminal } s' \end{cases}$$

    for $i \in \{t-1, \cdots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} log\pi(a_i|s_i;\theta')\big(R - V(s_i;\theta'_v)\big)$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\big(R - V(s_i;\ \theta'_v)\big)^2/\partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T \rightarrow T_{max}$

Experiments – A3C

Method	Training Time	Mean	Median
DQN (from [Nair et al., 2015])	8 days on GPU	121.9%	47.5%
Gorila [Nair et al., 2015]	4 days, 100 machines	215.2%	71.3%
Double DQN [Van Hasselt et al., 2015]	8 days on GPU	332.9%	110.9%
Dueling Double DQN [Wang et al., 2015]	8 days on GPU	343.8%	117.1%
Prioritized DQN [Schaul et al., 2015]	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.

Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Trust Region Policy Optimization (TRPO)

- TRPO is a policy optimization algorithm
  - can replace gradient descent
- There are many gradient descent methods
  - Original gradient descent method
  - Natural gradient descent method
  - Stochastic gradient descent method
- TRPO is similar to natural gradient descent method
- TRPO can be combined with A2C, called ACKTR

TRPO

- Consider a Markov decision process (MDP), defined by the tuple
$$(S, A, P, r, \rho_0, \gamma)$$
  - S is a finite set of states, A is finite set of actions
  - $P: S \times A \times S \to \mathbb{R}$ is the transition probability distribution
  - r is reward function
  - $\rho_0: S \to \mathbb{R}$ is the distribution of initial state (implicitly, $s_0 \sim \rho_0$)
  - $\gamma \in (0, 1)$ is discounted factor
- Let π be a stochastic policy $\pi: S \times A \to [0, 1]$
- The return function of reinforcement learning is

$$\eta(\pi) \coloneqq E_{s_0 \sim \rho_0}[V_\pi(s_0)] = \mathbb{E}_{s_0, a_0, \ldots \sim \sim \rho_0, \pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right]$$

TRPO

- Starting point:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0,a_0,\ldots\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right]$$

  - Proposed in 2002 by Kakade & Langford
  - Note: for simplicity, $\sim\rho_0$ is omitted later.
- This implies that we can derive "return of new policy" from "advantage of old policy"
  - Advantage $A_\pi(s_t,a_t) \coloneqq Q_\pi(s_t,a_t) - V_\pi(s_t)$

Appendix (Proof of the previous equation)

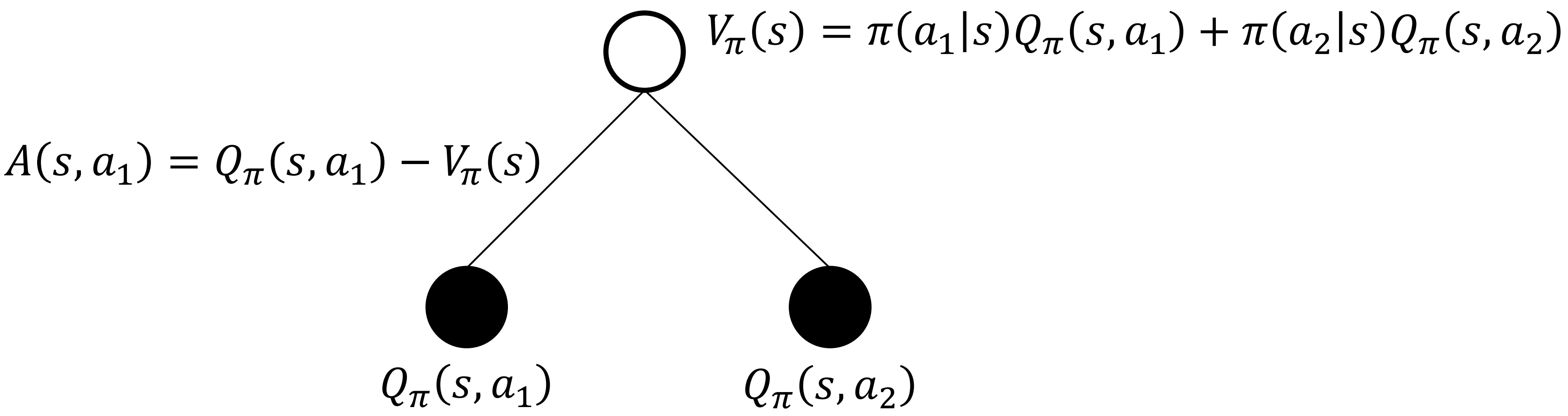
Since $A_\pi(s,a) = E_{s'\sim P(s'|s,a)}[r(s) + \gamma V_\pi(s') - V_\pi(s)]$,

we have

$$
\begin{aligned}
&E_{s_0,a_0,\ldots\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right] \\
&= E_{s_0,a_0,\ldots\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t\big(r(s_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)\big)\right] \\
&= E_{s_0,a_0,\ldots\sim\tilde{\pi}}\left[-V_\pi(s_0) + \sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \\
&= -E_{s_0}[V_\pi(s_0)] + E_{s_0,a_0,\ldots\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t)\right] \\
&= -\eta(\pi) + \eta(\tilde{\pi}) \qquad \square
\end{aligned}
$$

TRPO

- Advantage $A_\pi(s_t, a_t) \coloneqq Q_\pi(s_t, a_t) - V_\pi(s_t)$
- Can evaluate the current action compared to average value



TRPO

- Expanding $\eta(\tilde{\pi})$, we get

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0,a_0,\ldots\sim\tilde{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A_\pi(s_t,a_t)\right]$$

$$= \eta(\pi) + \sum_{t=0}^{\infty}\left(\sum_s\left(P(s_t = s|\tilde{\pi})\sum_a \tilde{\pi}(a|s)\gamma^t A_\pi(s,a)\right)\right)$$

$$= \eta(\pi) + \sum_s\left(\underbrace{\left(\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\tilde{\pi})\right)}_{\text{Called density of } s\text{, denoted } \rho_{\tilde{\pi}}(s)}\left(\sum_a \tilde{\pi}(a|s)A_\pi(s,a)\right)\right)$$

$$= \eta(\pi) + \sum_s\left(\rho_{\tilde{\pi}}(s)\left(\sum_a \tilde{\pi}(a|s)A_\pi(s,a)\right)\right)$$

  – Convert the view from each time point t to each state s

TRPO

$$\rho_{\tilde{\pi}}(s) \coloneqq \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) = \underbrace{P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots}_{\text{unnormalized discounted visitation frequencies}}$$

- Denote the un-normalized discounted visitation frequencies by $\rho_{\tilde{\pi}}(s)$, then the return of $\tilde{\pi}$ become

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_{\pi}(s,a)$$

- This implies
  - any policy update $\pi \rightarrow \tilde{\pi}$ that has a nonnegative expected advantage at every state s, is guaranteed to increase the policy performance η
  - *or*: If all $\sum_a \tilde{\pi}(a|s) A_{\pi}(s,a)$ are non-negative for the new policy $\tilde{\pi}$, the policy performance η must be improved.

TRPO

$$\rho_{\tilde{\pi}}(s) \coloneqq \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) = \underbrace{P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \cdots}_{\text{unnormalized discounted visitation frequencies}}$$

- Denote the un-normalized discounted visitation frequencies by $\rho_{\tilde{\pi}}(s)$, then the return of $\tilde{\pi}$ become

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

- This implies
  - any policy update $\pi \rightarrow \tilde{\pi}$ that has a nonnegative expected advantage at every state s, is guaranteed to increase the policy performance η
  - *or*: If all $\sum_a \tilde{\pi}(a|s) A_\pi(s,a)$ are non-negative for the new policy $\tilde{\pi}$, the policy performance η must be improved.

TRPO

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

- However, due to the complex dependency of $\rho_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ makes above equation difficult to optimize directly
- Instead, introducing local approximation to η:

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

  - L ignores changes in state visitation density due to changes in the policy
- Key: try to maximize $L_\pi(\tilde{\pi})$ instead of $\eta(\tilde{\pi})$.
  - Question: why is it fine to replace $\rho_{\tilde{\pi}}(s)$ by $\rho_\pi(s)$?

TRPO

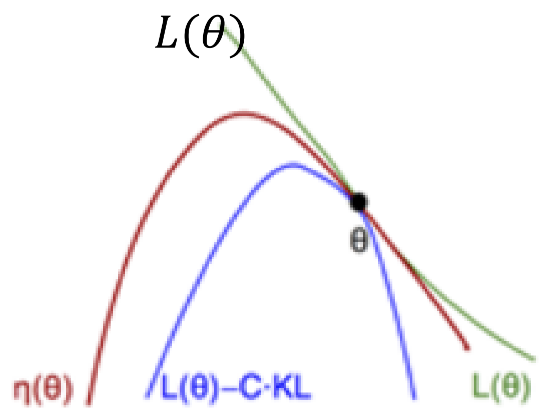
- If we have a policy π_θ, which is differentiable w.r.t. θ, then L_π matches η to first order. i.e., for any parameter θ_{old}

$$L_{\pi_{\theta_{old}}}(\pi_{\theta_{old}}) = \eta(\pi_{\theta_{old}}),$$

$$\nabla_\theta L_{\pi_{\theta_{old}}}(\pi_\theta)\Big|_{\theta=\theta_{old}} = \nabla_\theta\, \eta(\pi_\theta)\Big|_{\theta=\theta_{old}}$$

Proved in next page

- This implies that a step small enough that improves $L_{\pi_{old}}$ will also improve η.



- Sutton's proof by induction for

$$\frac{\partial\eta(\pi_\theta)}{\partial\theta} = \sum_s \rho^\pi(s) \sum_a \frac{\partial\pi_\theta(a|s)}{\partial\theta} Q^\pi(s,a)$$

For the start-state formulation:

$$\begin{aligned}
\frac{\partial V^\pi(s)}{\partial\theta} &\stackrel{\text{def}}{=} \frac{\partial}{\partial\theta}\sum_a \pi(s,a)Q^\pi(s,a) \qquad \forall s \in \mathcal{S} \\
&= \sum_a \left[\frac{\partial\pi(s,a)}{\partial\theta}Q^\pi(s,a) + \pi(s,a)\frac{\partial}{\partial\theta}Q^\pi(s,a)\right] \\
&= \sum_a \left[\frac{\partial\pi(s,a)}{\partial\theta}Q^\pi(s,a) + \pi(s,a)\frac{\partial}{\partial\theta}\left[\mathcal{R}_s^a + \sum_{s'}\gamma\mathcal{P}_{ss'}^a V^\pi(s')\right]\right] \\
&= \sum_a \left[\frac{\partial\pi(s,a)}{\partial\theta}Q^\pi(s,a) + \pi(s,a)\sum_{s'}\gamma\mathcal{P}_{ss'}^a\frac{\partial}{\partial\theta}V^\pi(s')\right] \qquad (7) \\
&= \sum_x \sum_{k=0}^{\infty}\gamma^k Pr(s \rightarrow x, k, \pi)\sum_a \frac{\partial\pi(x,a)}{\partial\theta}Q^\pi(x,a),
\end{aligned}$$

- Sutton's proof by induction for

$$\frac{\partial \eta(\pi_\theta)}{\partial \theta} = \sum_s \rho^\pi(s) \sum_a \frac{\partial \pi_\theta(a|s)}{\partial \theta} Q^\pi(s,a)$$

$$= \sum_s \rho^\pi(s) \sum_a \frac{\partial \pi_\theta(a|s)}{\partial \theta} A^\pi(s,a)$$

(Why? $\sum_a \pi_\theta(a|s) V^\pi(s) = 1$)

$$= \frac{\partial L(\pi_\theta)}{\partial \theta}$$

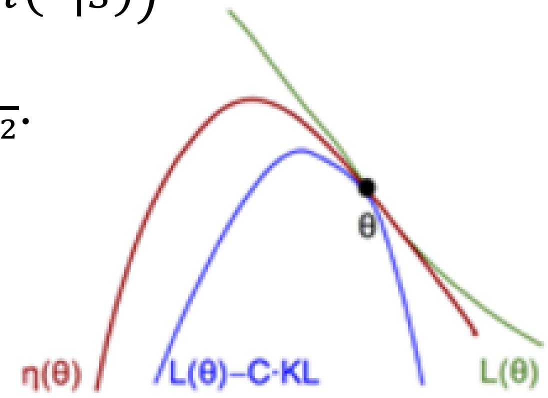
TRPO (next five pages can be skipped)

- The main result in this paper is the following theorem:
- Let $\alpha = D_{TV}^{max}(\pi, \tilde{\pi})$, then the following bound holds:

$$\eta(\tilde{\pi}) \geq L_{\pi_{old}}(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

$$\text{where } \epsilon = \max_{s,a}|A_\pi(s,a)|$$

  - $D_{TV}(p,q) = \frac{1}{2}\sum_i |p_i - q_i|$ for discrete probability distribution p, q
  - $D_{TV}^{max}(\pi, \tilde{\pi}) = \max_s D_{TV}\big(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)\big)$
  - Note: we will use C to denote $\frac{4\epsilon\gamma}{(1-\gamma)^2}$.



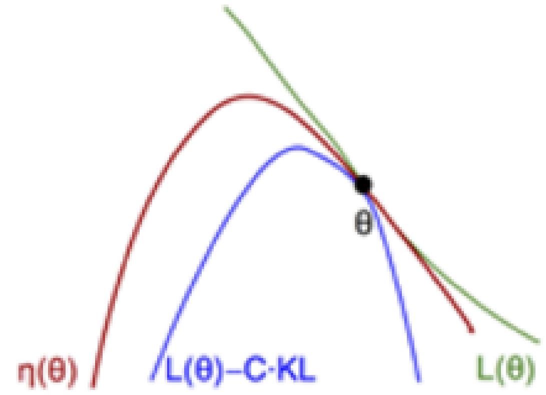
TRPO

- And $D_{TV}(p \parallel q)^2 \leq D_{KL}(p \parallel q)$.
- Let $D_{KL}^{max}(\pi, \tilde{\pi}) = \max_s D_{KL}\big(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)\big)$, then

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

$$\text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$$

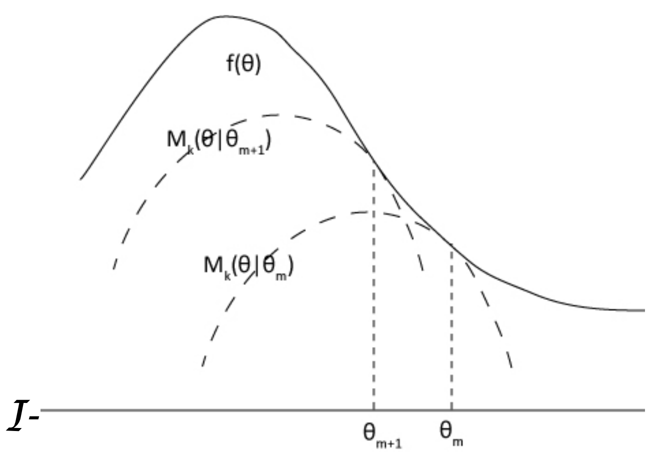
  - When $\pi \to \tilde{\pi}$, $D_{KL}^{max}(\pi, \tilde{\pi}) \to 0$, so the lower bound is tight. How much we improve on $L_\pi(\tilde{\pi})$, how much the return $\eta(\tilde{\pi})$ also improve
  - When π is not close to $\tilde{\pi}$, the penalty is large since constant C is large, and the lower bound is meaningless.
- A kind of MM algorithm
  - Minorize-Maximization or
  - Majorize-Minimization

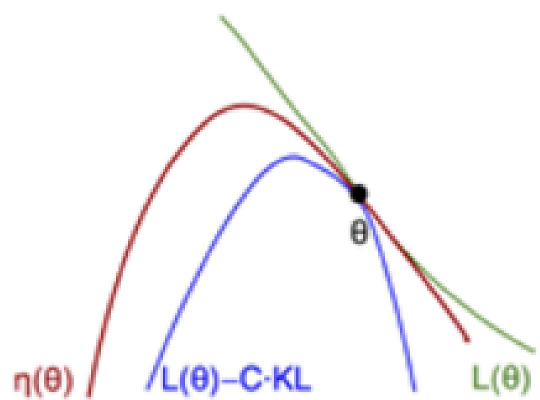


TRPO

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

- We show that the improvement must be monotonically increasing (MM algorithm)
- Let $M_i(\pi) = L_{\pi_i}(\pi) - C \cdot D_{KL}^{max}(\pi_i, \pi)$:
$$\eta(\pi) \geq M_i(\pi)$$
$$\eta(\pi_i) = M_i(\pi_i)$$
$$\eta(\pi) - \eta(\pi_i) \geq M_i(\pi) - M_i(\pi_i)$$
- Let $\pi_{i+1} = \text{argmax}_{\pi} M_i(\pi)$, then
$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M_i(\pi_i) \geq 0$$
and thus the return of next iteration is not worse than current one.





M =

TRPO

- Algorithm

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

Initialize π_0.
for $i = 0, 1, 2, \dots$ 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

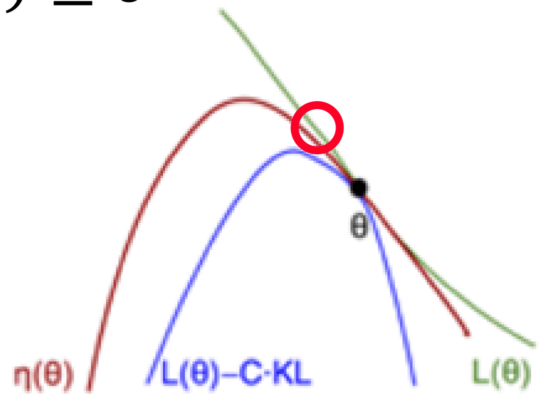
TRPO

- Problems:
  - In practice, the step size is very small
  - D_{KL}^{max} is hard to compute
  - How do we approximate the objective function and constraint?

TRPO

- In practice, if using the penalty coefficient C recommended by the theory above, the step size would be very small.
- One way to take larger steps in a robust way is to use a constraint on the KL divergence between the new policy and the old policy, i.e., a trust region constraint:

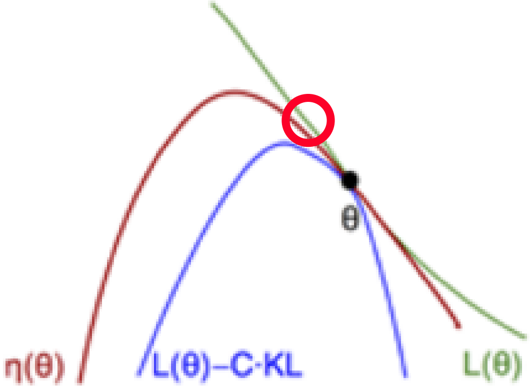
$$\max_{\theta} L_{\theta_{old}}(\theta)$$
$$\text{subject to } D_{KL}^{max}(\theta_{old}, \theta) \leq \delta$$



TRPO (can be skipped)

- Since the D_{KL}^{max} is hard to compute, we can use a heuristic approximation which considers the average KL divergence $\bar{D}_{KL}^{\rho}(\theta_{old}, \theta) \coloneqq \mathbb{E}_{s\sim\rho}\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot\,|s) \parallel \pi_{\theta}(\cdot\,|s)\right)\right]$
- Thus, the problem becomes

$$\max_{\theta} L_{\theta_{old}}(\theta)$$
$$\text{subject to } \bar{D}_{KL}^{\rho}(\theta_{old}, \theta) \leq \delta$$



TRPO

- Transform the problem: $\max_{\theta} L_{\theta_{old}}(\theta)$

$$\max_{\theta} \sum_{s} \rho_{\theta_{old}}(s) \sum_{a} \pi_{\theta}(a|s) A_{\theta_{old}}(s,a)$$
$$\text{subject to } \overline{D}_{KL}^{\rho}(\theta_{old}, \theta) \leq \delta$$

1. Replace $\sum_s \rho_{\theta_{old}}(s)[\cdots]$ by expectation $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{old}}}[\cdots]$
2. Replace the sum over the actions by an importance sampling estimator. Using $\pi_{\theta_{old}}(a|s)$ to denote the sampling distribution, then the contribution of a single s_n to the loss function is:
$$\sum_{a} \pi_{\theta}(a|s_n) A_{\theta_{old}}(s_n,a) = \mathbb{E}_{a\sim\pi_{\theta_{old}}(a|s_n)}\left[\frac{\pi_{\theta}(a|s_n)}{\pi_{\theta_{old}}(a|s_n)} A_{\theta_{old}}(s_n,a)\right]$$

TRPO

- The problem at the beginning:

$$\max_\theta L(\pi_{\theta_{old}}) \ \ or$$
$$\max_\theta \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{old}}(s,a)$$
$$\text{subject to } \bar{D}_{KL}^{\rho}(\theta_{old}, \theta) \le \delta$$

- And currently, we solve:

$$\max_\theta \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim\pi_{\theta_{old}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\theta_{old}}(s,a)\right]$$
$$\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{old}}}\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot|s) \parallel \pi_\theta(\cdot|s)\right)\right] \le \delta$$

- In another form, maximize a surrogate objective:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t\right]$$

  - CPI: conservative policy iteration
  - $\hat{A}_t$: can be any form of advantage, like GAE.

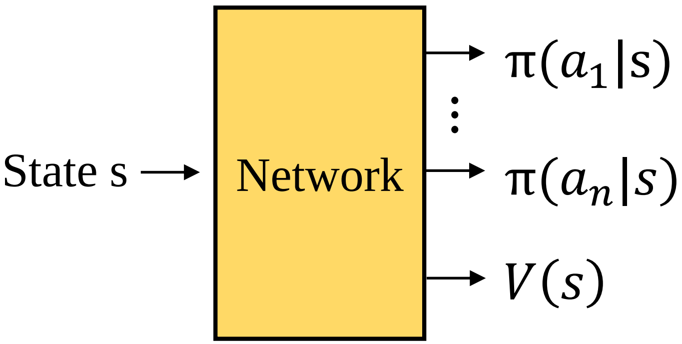
Proximal Policy Optimization (PPO)

- Problems of TRPO:
  - still relatively complicated, and
  - not compatible with architectures that include noise (such as dropout) or parameter sharing
- Background:
  - In 2017, OpenAI release a new reinforcement learning algorithms, PPO.
  - PPO has some of the benefits of TRPO, but much simpler to implement, more general, and has better sample complexity.
  - attains the data efficiency and reliable performance of TRPO, while using only first-order optimization
- The experiments show that PPO outperforms other online policy gradient methods, like A2C or TRPO.
  - Although PPO is a little worse than ACER (Actor-Critic with Experience Replay), the implementation of PPO is much easier than ACER.

Generalized Advantage Estimation (GAE)

- Use the learned state-value function $V(s)$ to compute variance-reduced advantage-function estimators.
- PPO uses a truncated version of generalized advantage estimation

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$
$$\text{where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

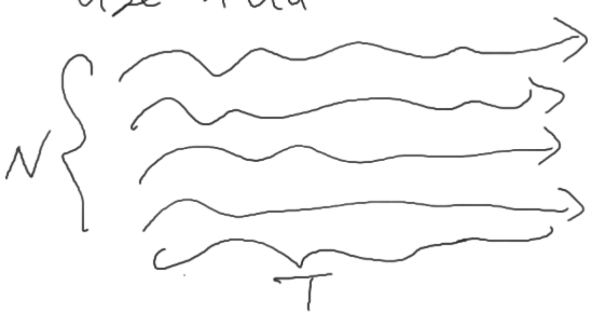


PPO Algorithm

Algorithm 1 PPO, Actor-Critic Style

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \le NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Use π_{old}



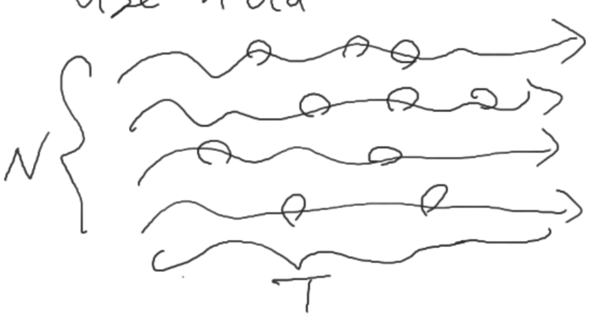
PPO Algorithm

Algorithm 1 PPO, Actor-Critic Style

---

for iteration$=1, 2, \ldots$ **do**
  for actor$=1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Use π_{old}



Let $\pi_0 = \pi_{old}$

1. pick a batch with M
2. optimize θ. from $\pi_i \to \pi_{i+1}$
3. repeat 1.

$$\pi_0 \to \pi_1 \to \pi_2 \to \cdots \to \pi_K$$

Recall TRPO

- Recall: TRPO maximizes a surrogate objective: $\max_{\theta} L^{CPI}(\theta)$ (with small change on $\pi_\theta(a|s)$)

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

  - CPI: conservative policy iteration
  - $\hat{A}_t$: can be any form of advantage, like GAE.
- Let $r_t(\theta)$ denote the probability ratio (not reward)

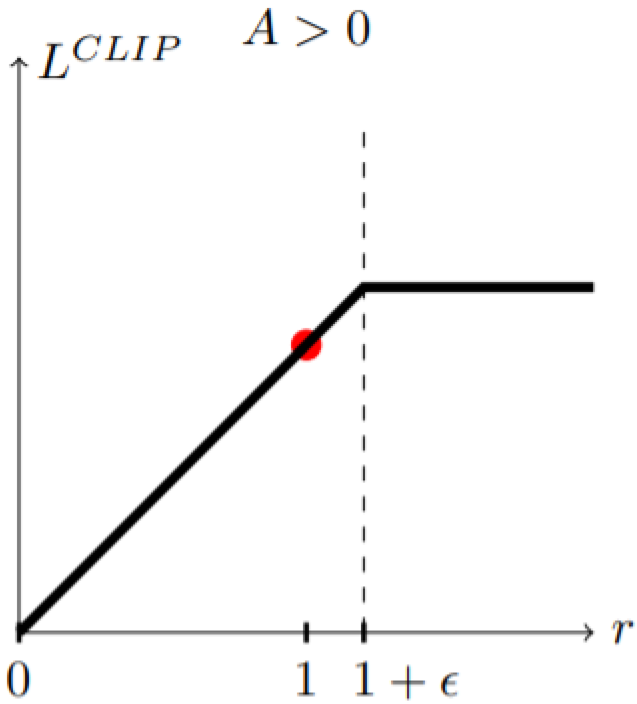
$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

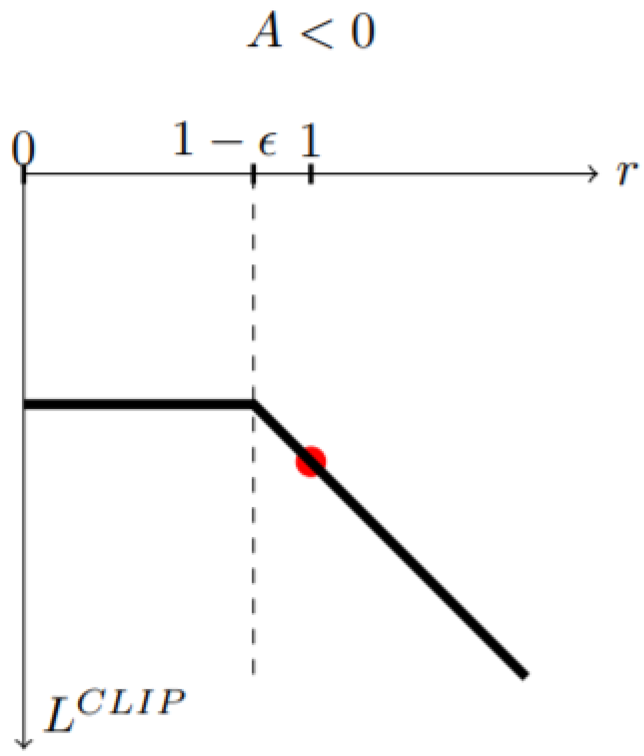
  - $r(\theta_{old}) = 1$
  - Note: π_θ can be any of π_i in PPO

PPO Clip

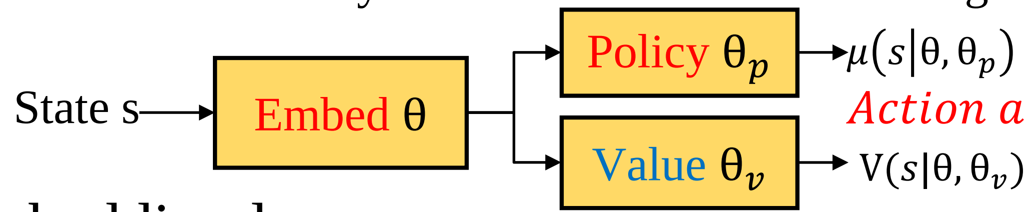
- Without constraint, the step size of L^{CPI} would be large
- Hence, we consider modifying the objective, to penalize changes to the policy that move $r_t(\theta)$ away from 1
- The main objective proposed in PPO is:
  $L^{CLIP} = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$
  - ϵ is a hyper-parameter
  - First term implies that the min is L^{CPI}
  - Second term modifies the surrogate objective by clipping the probability ratio
  - The final objective is a lower bound on L^{CPI}

- If clipped, $\nabla_\theta L^{CLIP}$ becomes 0, and then drop the gradient





PPO



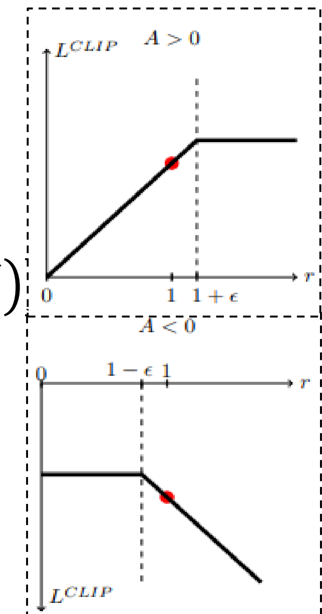
- Use one network with same embedding layer: policy and value
  - Value: estimates value of current state by TD-like learning
    - Value loss: $L_t^{VF}(\theta) = \left(V_\theta(s_t) - V_t^{target}\right)^2$
  - Policy: output probability of actions
    - Policy obj.: $L_t^{CLIP}(\theta) = \widehat{E_t}\left[\min\left(r_t(\theta)\widehat{A_t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_t}\right)\right]$

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

      $\widehat{A_t}$ is generalized advantage estimation (GAE)

      $\widehat{A_t} = \sum_{n=0}^{\infty}(\gamma\lambda)^n \delta_{t+n}^V$,

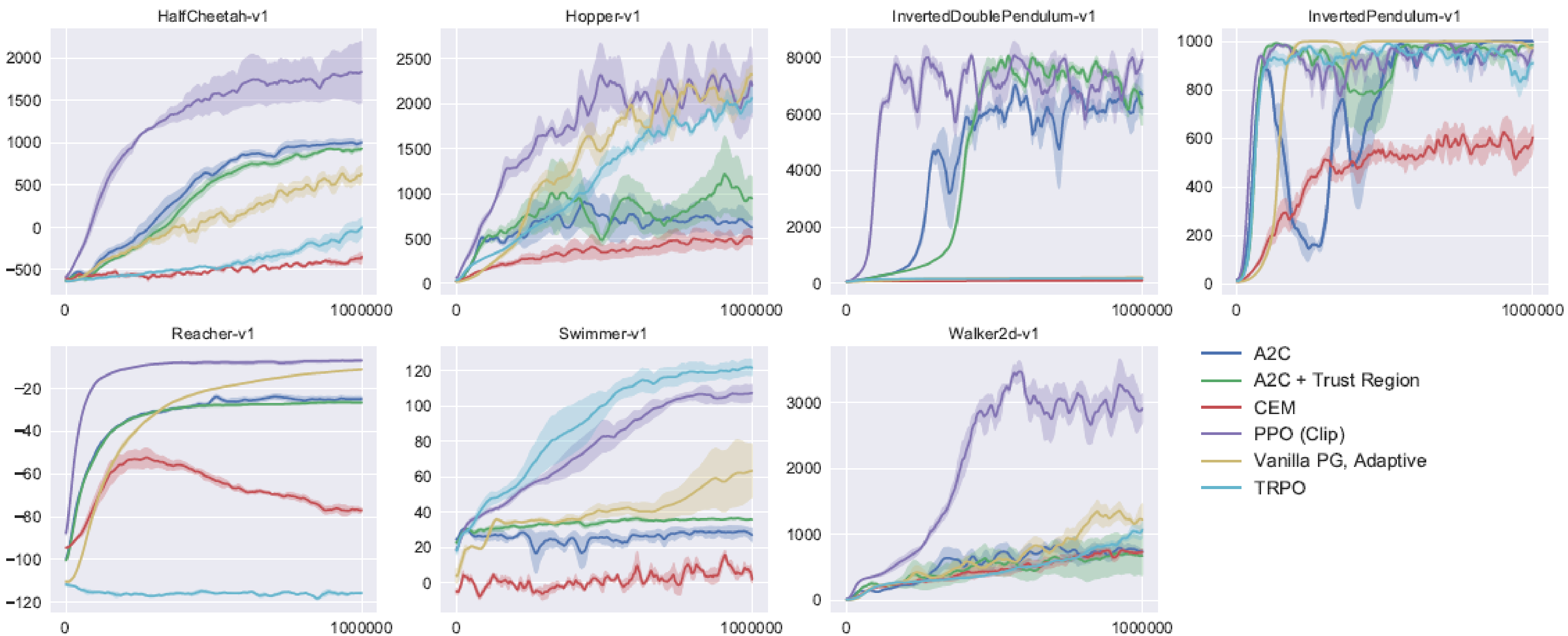
      where $\delta_t^V = r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$ [TD error]



  - Total objective (usually version): maximize

    $L_t^{CLIP+VF+S}(\theta) = \widehat{E_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$
    - Augment with an entropy bonus (S) to ensure sufficient exploration

Experiments - PPO



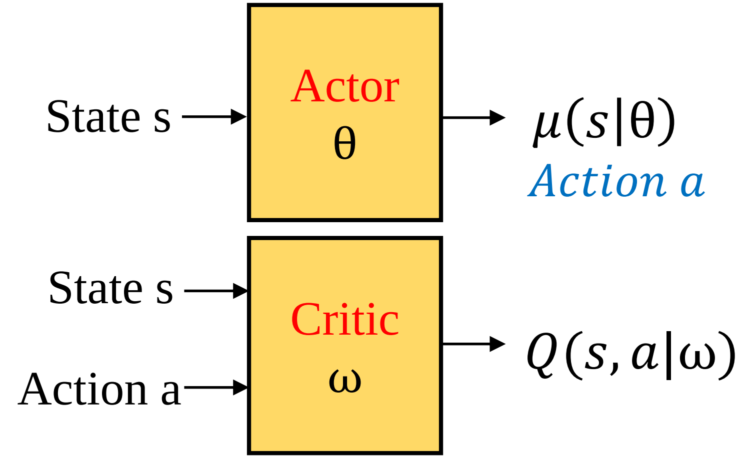
Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Deterministic Policy Gradient

- Deterministic policy gradient can be estimated more efficiently, especially in high-dimensional continuous action spaces
  - Deterministic policy integrates over only states space
  - Use off-policy learning to ensure adequate exploration

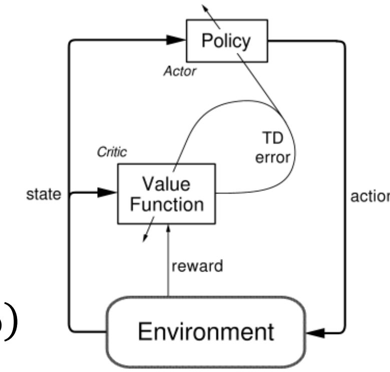
[Lillicrap, et al., 2016] "Continuous control with deep reinforcement learning," in 4th International Conference on Learning Representations (ICLR 2016).



Deep Deterministic Policy Gradient (DDPG) (A Kind of Actor-Critic For Continuous Actions)

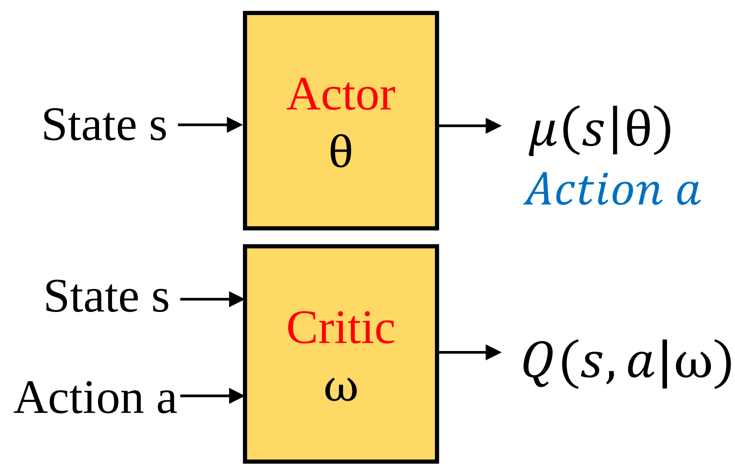
- Use two networks: an actor and a critic
  - Critic estimates value of current action by Q-learning

$$\nabla_\omega L_Q(s_t, a_t|\omega)$$
$$= \left(\left(r_{t+1} + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta)|\omega)\right) - Q(s_t, a_t|\omega)\right) \nabla_\omega Q(s_t, a_t|\omega)$$



  - Actor updates policy in direction suggested by critic (DDPG):

$$\nabla_\theta J(\mu_\theta) \approx \mathbb{E}_\mu[\nabla_\theta Q(s_t, \mu(s_t|\theta)|\omega)]$$
$$= \mathbb{E}_\mu\left[\nabla_a Q(s_t, a|\omega)\big|_{a=\mu(s_t|\theta)} \nabla_\theta \mu(s_t|\theta)\right]$$



DDPG(1/2)

Behavior and target network

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$. Initialize replay buffer R

for $t = 1, T$ **do**

Select action $a_t = \mu(s_t|\theta^\mu) + \boxed{N_t}$ A noise process

Execute action a_t and observe reward r_t and observe new state s_{t+1}

Experience replay

Store transition (s_t, a_t, r_t, s_{t+1}) in R

Sample random minibatch of M transitions (s_j, a_j, r_j, s_{j+1}) from R

Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{M}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$

Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \qquad \theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$$

DDPG(2/2)

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$. Initialize replay buffer R
for $t = 1, T$ **do**

Select action $a_t = \mu(s_t|\theta^\mu) + N_t$
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample random minibatch of M transitions (s_j, a_j, r_j, s_{j+1}) from R
Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{M}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
Update the actor policy using the sampled gradient:

$$\nabla_{\theta^\mu}\mu|_{s_i} \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$$

Update the behavior networks (both actor and critic)

Update the target networks:

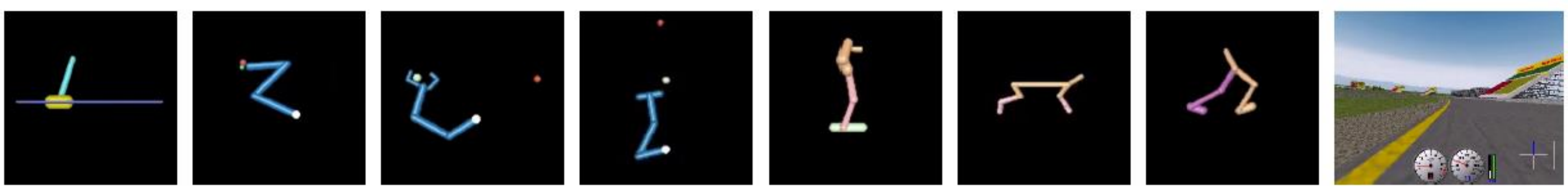
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \qquad \theta^{\mu'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{\mu'}$$

Apply “soft” target updates
$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tau \ll 1$

(0.001 in practice.)
(Note in DQN, θ is copied periodically. Later, some DQN also used this way)

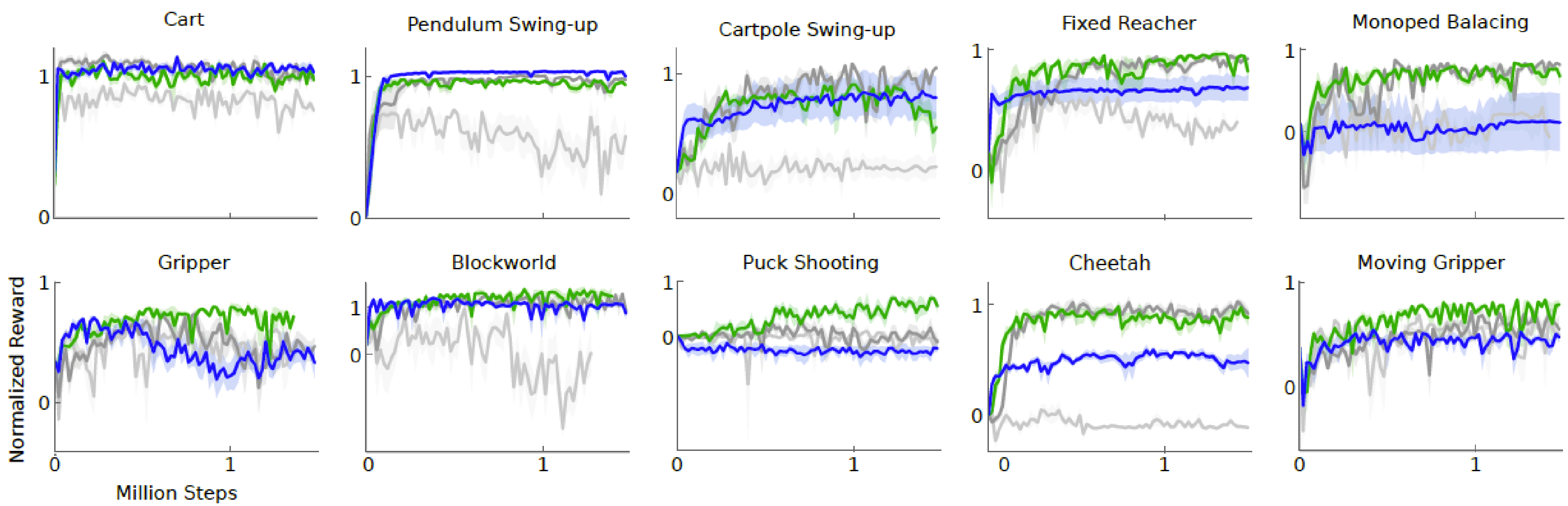
Experiment Settings

- Run experiments using both a low-dimensional state description and high-dimensional renderings of the environment
- The frames were downsampled to 64x64 pixels and the 8-bit RGB values were converted to floating point scaled to [0, 1]



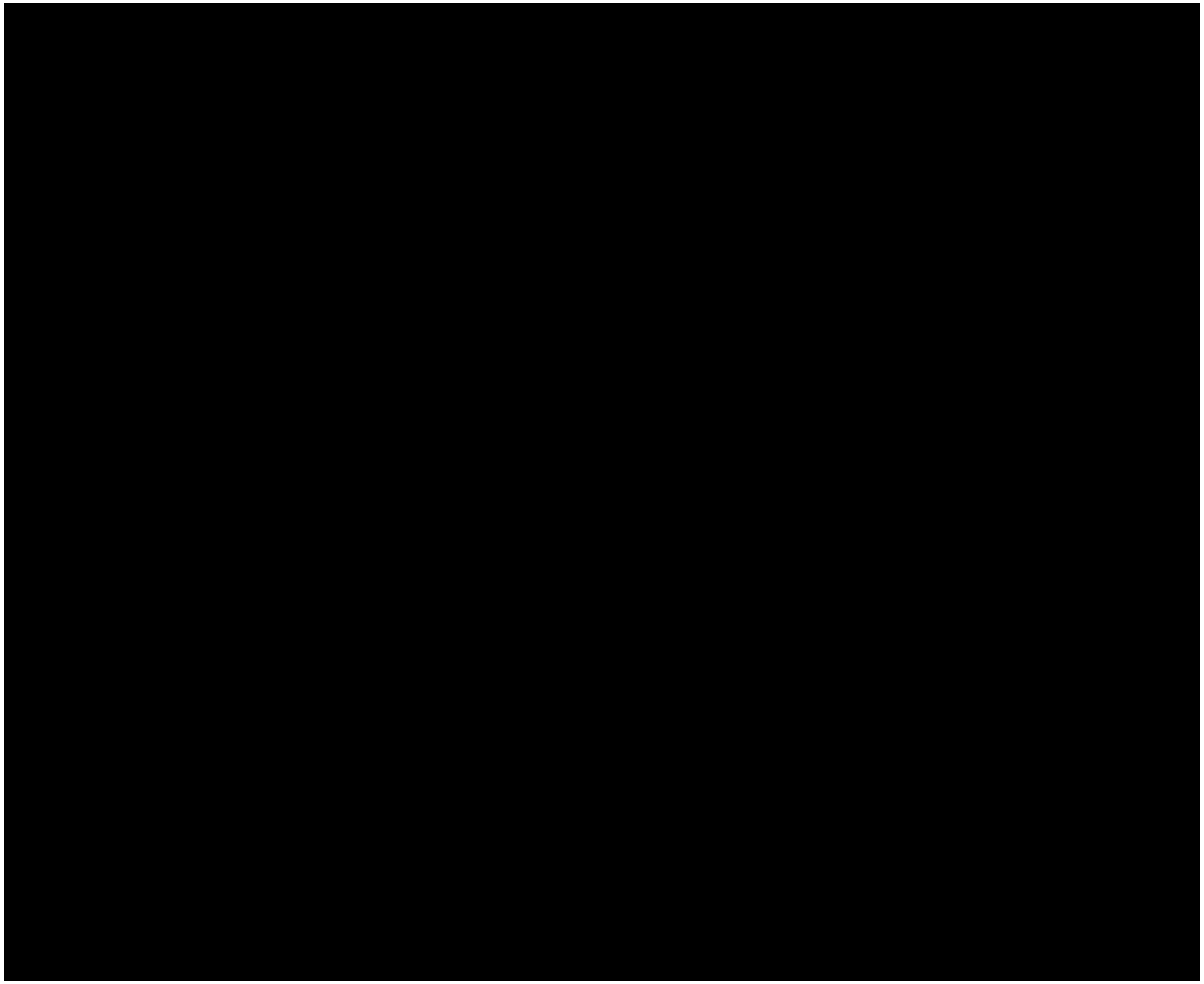
Example screenshots of a sample of environments to solve with DDPG.

Performance Curves for Those Using Variants of DPG



Light Gray: State Description + Batch Normalization
Dark Gray: State Description + Target Network
Green: State Description + Batch Normalization + Target Network
Blue: Pixels + Target Network

Demo



Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC

Twin Delayed DDPG (TD3)

Addressing Function Approximation Error in Actor-Critic Methods

Scott Fujimoto, Herke van Hoof and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods." ICML (2018).

DDPG Overview

initial $\theta, \theta', \phi, \phi'$, replay buffer B
for episode = 1~M **do**
  for t = 1~T **do**
    Select action using π_ϕ
    Play and store transition in B
    Sample a batch from B

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s'))$$

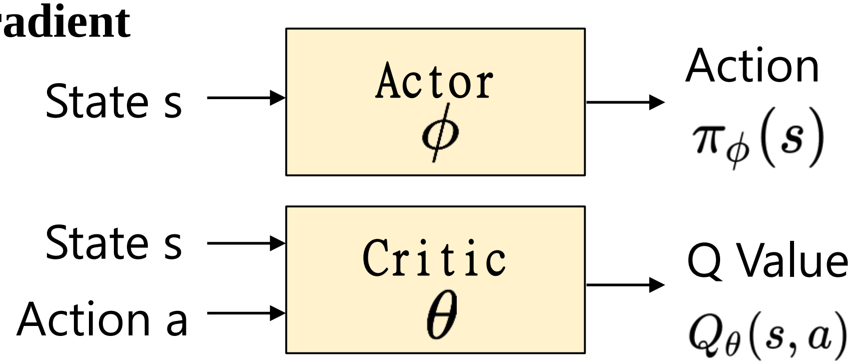
    Update Behavior Critic θ using y
    Update Behavior Actor ϕ using **policy gradient**
    Update Target

$$\theta' \to \tau\theta + (1-\tau)\theta'$$

$$\phi' \to \tau\phi + (1-\tau)\phi'$$

	Actor	Critic
Behavior	ϕ	θ
Target	ϕ'	θ'

Network Weight Notation



Method

- Twin Delayed DDPG (TD3)

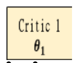
- TD3 = DDPG + 3 Tricks
  - Clipped Double Q-Learning
  - Delayed Policy Updates
  - Target Policy Smoothing

TD3 Overview

initial $\theta, \theta', \phi, \phi'$, replay buffer B

for episode = 1~M **do**

for t = 1~T **do**

Select action using 

Play and store transition in B

Sample a batch from B

Trick 1

$$y = r + \gamma \boxed{\min_{i=1,2} Q_{\theta_i'}}(s', \pi_{\phi'}(s') \boxed{+\,\epsilon})$$

Trick 3

Update Behavior Critic θ_1, θ_2 using y

Trick 2 $\boxed{\textbf{if } \text{t mod d } \textbf{then}}$

Update Behavior Actor ϕ using **policy gradient**

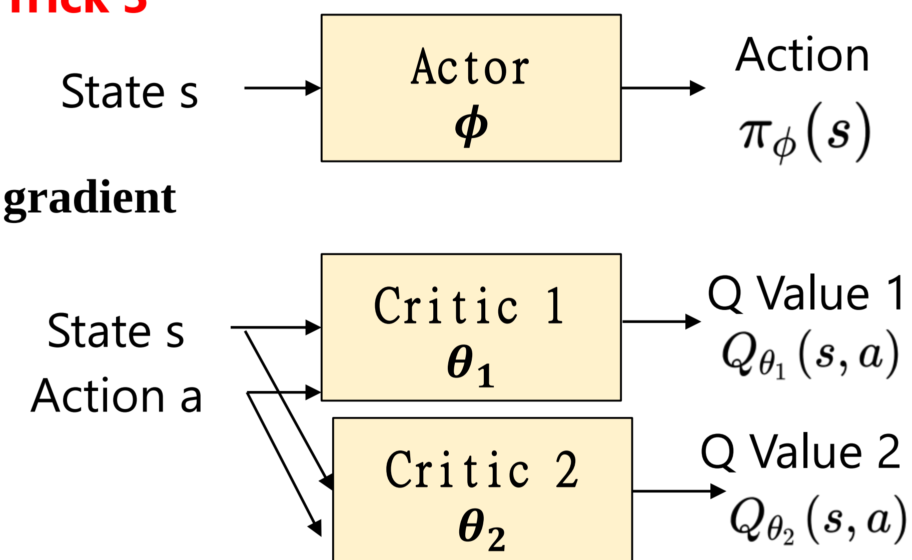
Update Target

$\theta_i' \rightarrow \tau\theta_i + (1-\tau)\theta_i'$

$\phi' \rightarrow \tau\phi + (1-\tau)\phi'$

	Actor	Critic
Behavior	ϕ	θ_1, θ_2
Target	ϕ'	θ_1', θ_2'

Network Weight Notation



I-Chen Wu

Trick 1 : Clipped Double-Q Learning

- Origin DDPG (Not Good)

$$y = r + \gamma Q_{\theta'}(s', \pi_{\boxed{\phi'}}(s'))$$

- Methods to solve overestimation problem
  - Double DQN (Not Good Enough)

$$y = r + \gamma Q_{\theta'}(s', \pi_{\boxed{\phi}}(s'))$$

  - Double-Q Learning (Not Good Enough)

$$y_1 = r + \gamma Q_{\theta'_2}(s', \pi_{\phi_1}(s'))$$
$$y_2 = r + \gamma Q_{\theta'_1}(s', \pi_{\phi_2}(s'))$$

	Actor	Critic
Behavior	ϕ	θ
Target	ϕ'	θ'

Network Weight Notation

(Recall) Overestimation Problem

- Q-Learning update

$$Q(s,a) = r + \gamma \max_{a'} Q(s', a')$$

Trick 1 : Clipped Double-Q Learning

- Methods to solve overestimation problem
  - Double DQN (Not Good Enough)

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi}(s'))$$

  - Double-Q Learning (Not Good Enough)

$$y_1 = r + \gamma Q_{\theta'_2}(s', \pi_{\phi_1}(s'))$$
$$y_2 = r + \gamma Q_{\theta'_1}(s', \pi_{\phi_2}(s'))$$

	Actor	Critic
Behavior	ϕ	θ_1, θ_2
Target	ϕ'	θ'_1, θ'_2

Network Weight Notation

- Clipped Double-Q Learning (Better)

$$\boxed{y} = r + \gamma \min[Q_{\theta'_1}(s', \boxed{\pi_{\phi}}(s')), Q_{\theta'_2}(s', \boxed{\pi_{\phi}}(s'))]$$

Only one Q target

Only one actor

Trick 2 : Delayed Policy Updates

- Use lower frequency to update behavior actor and target networks.

initial
for episode = 1~M **do**
  for t = 1~T **do**
    ...
    Update Behavior Critic
    Update Behavior Actor
    Update Targets Networks

initial
for episode = 1~M **do**
  for t = 1~T **do**
    ...
    Update Behavior Critic
    if t mod **d** **then**
      Update Behavior Actor
      Update Targets Networks

Hyperparameter

Trick 3 : Target Policy Smoothing

- Assumption
  - Similar actions have similar values
- Add noise to action value

$$y = r + \gamma Q(s', \pi(s') + \epsilon), \epsilon \sim clip(\mathcal{N}(0, \underbrace{\sigma), -c, c}_{\text{Hyperparameters}})$$

Hyperparameters

- Regularization

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

2. Delayed Policy Updates

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

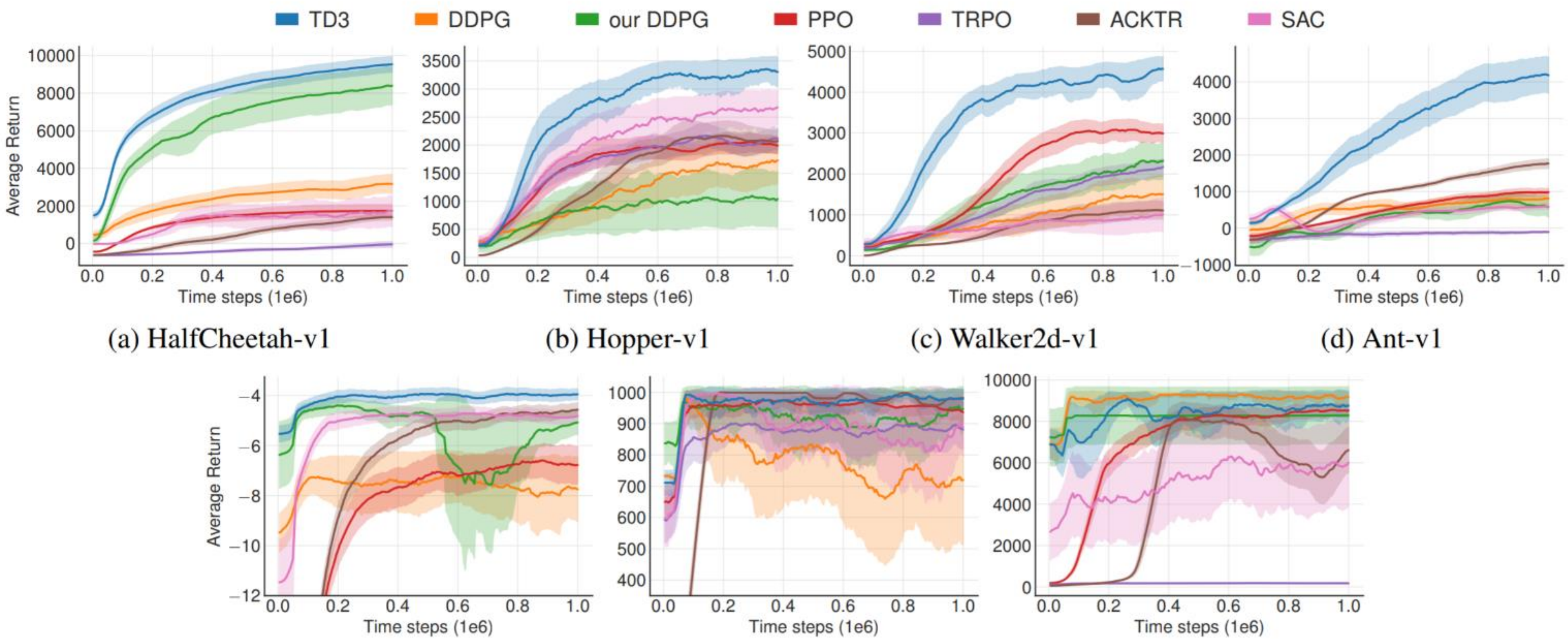
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

2. Delayed Policy Updates

3. Target Policy Smoothing Regularization

Experiment



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

(e) Reacher-v1 (f) InvertedPendulum-v1 (g) InvertedDoublePendulum-v1

Experiments: Compared to Others

Environment	TD3	DDPG	Our DDPG	PPO	TRPO	ACKTR	SAC
HalfCheetah	**9636.95 ± 859.065**	3305.60	8577.29	1795.43	-15.57	1450.46	2347.19
Hopper	**3564.07 ± 114.74**	2020.46	1860.02	2164.70	2471.30	2428.39	2996.66
Walker2d	**4682.82 ± 539.64**	1843.85	3098.11	3317.69	2321.47	1216.70	1283.67
Ant	**4372.44 ± 1000.33**	1005.30	888.77	1083.20	-75.85	1821.94	655.35
Reacher	**-3.60 ± 0.56**	-6.51	**-4.01**	-6.18	-111.43	-4.26	-4.44
InvPendulum	**1000.00 ± 0.00**	**1000.00**	**1000.00**	**1000.00**	985.40	**1000.00**	**1000.00**
InvDoublePendulum	**9337.47 ± 14.96**	**9355.52**	8369.95	8977.94	205.85	9081.92	8487.15

Policy-Based Reinforcement Learning

- Policy Gradient
- Actor-Critic (Discrete actions)
- A3C (Asynchronous Advantage Actor-Critic)
- TRPO & PPO
- DDPG (Deep Deterministic Policy Gradient)
  - TD3
  - SAC (Soft Actor Critic)

Reference

- Haarnoja, T., Tang, H., Abbeel, P., & Levine, S. (2017). Reinforcement Learning with Deep Energy-Based Policies. ICML.
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. ArXiv, abs/1801.01290.
- Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. ArXiv, abs/1812.05905.
- Open source:
  - https://github.com/haarnoja/sac (original author)
    https://github.com/rail-berkeley/softlearning

- Credit goes to Guo-Hao Ho for most of the slides.

Introduction

- SAC is
  - Open-source (by original authors)
    - https://sites.google.com/view/sac-and-applications
  - Perform well (as in realistic environment)
  - Key idea is easy to understand
    - Maximum entropy reinforcement learning

Introduction

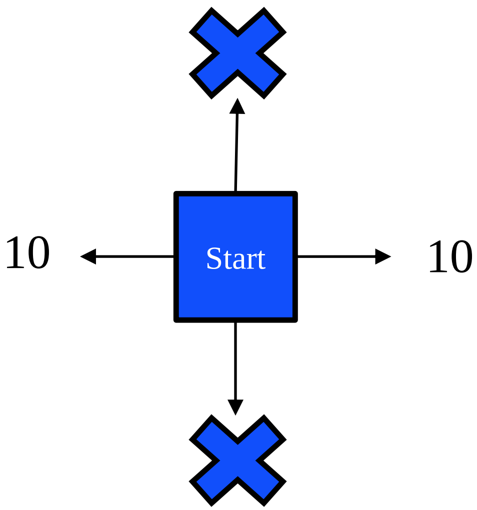
- Soft actor critic (SAC) train a policy that maximizes a trade-off between expected return and entropy
  - Still getting high performance while acting as random as possible
  - Augment the objective function with entropy term
- Evolution of SAC
  - Soft Q-learning (SQL)
    → Soft Actor-Critic (SAC)
    → Soft Actor-Critic with automating entropy adjustment(SAC)

Problem

- The above methods (PPO, DDPG) focus more on exploitation
  - The objective function is mainly based on the return
    - May be trapped in local optimum without exploration

Extremely simple case

Return	Up	Left	Down	Right
	0	10	0	10



Policy	Up	Left	Down	Right
T=0	0.25	0.25	0.25	0.25
T=1	0.2	0.4	0.2	0.2
…				
T=n	0	1	0	0

If we sampled “left” first

Without any exploration, the chance to sample the “right” is harder, resulting in the policy converges to “left” gradually

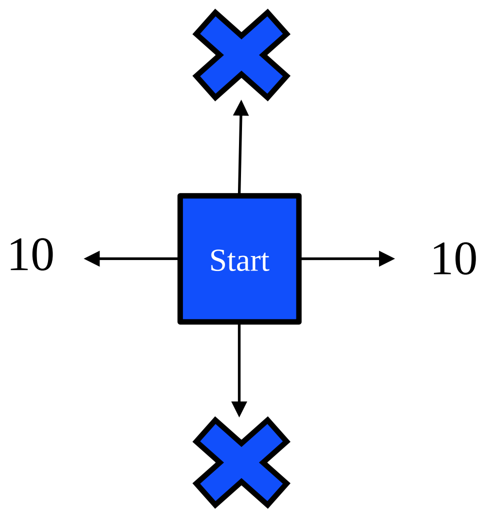
The agent will be
- either right or left with100%
- not right and left with 50%

Problem

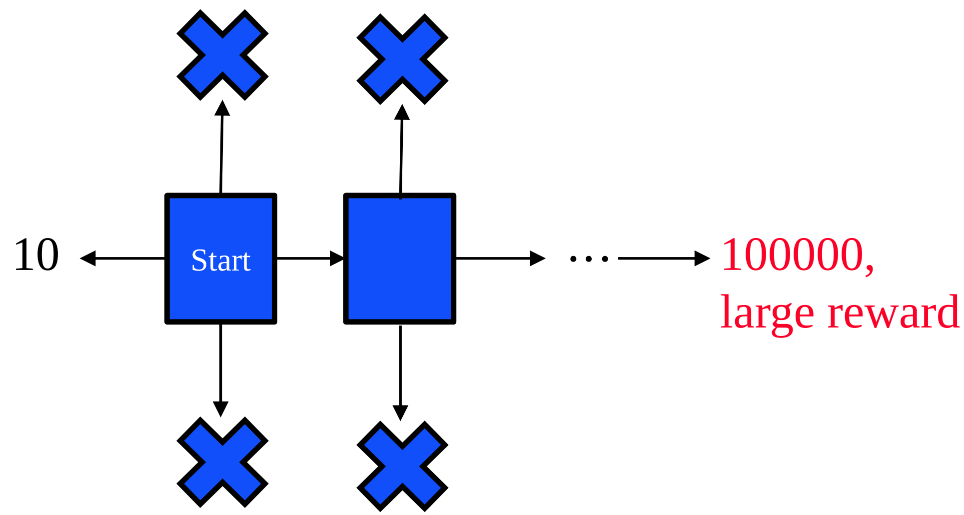
- Hard exploration case
  - Extend previous “extremely simple case”

The agent will be either right or left for 100%
But not right and left for 50%

Hard for agent to discover policy of “right”
May trap in policy of “left”



Extremely simple case



Hard exploration case

Problem-Solution

- The exploration ability relies on
  - Random noise in selected action
    E.g. DDPG
    - During training, the action is disturbed with the random noise

Algorithmus 4 : Deep Deterministic Policy-Gradient

Result : policy parameter $\boldsymbol{\theta}$ and action-value weights $\mathbf{w}$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and action-value weights $\mathbf{w} \in \mathbb{R}^{d}$;
Initialize target policy parameter $\boldsymbol{\theta}' \in \mathbb{R}^{d'}$ and target action-value weights $\mathbf{w}' \in \mathbb{R}^{d}$;
Initialize experience replay memory $\mathcal{D}$;
for *episode* $= 1, M$ **do**
  Observe initial state s_0 from environment ;
  for $t = 1, T$ **do**
    $\boxed{\text{Select action } a_t = \tau(s, \boldsymbol{\theta}_t) + \mathcal{N}_t}$;
    Observe reward r_t and next state s_{t+1} from environment ;
    Store (s_t, a_t, r_t, s_{t+1}) tupel in $\mathcal{D}$;
    Sample random batch (s_i, a_i, r_i, s_{i+1}) of size B from $\mathcal{D}$;
    $\delta_i \leftarrow r_i + \gamma \hat{q}(s_{i+1}, \tau(s_{i+1}, \boldsymbol{\theta}_t'), \mathbf{w}_t') - \hat{q}(s_i, a_i, \mathbf{w}_t)$;
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta \frac{1}{B} \sum_i^B \delta_i \nabla_{\mathbf{w}} \hat{q}(s_i, a_i, \mathbf{w}_t)$;
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \frac{1}{B} \sum_i^B \nabla_{\boldsymbol{\theta}} \hat{q}(s_i, \tau(s_i, \boldsymbol{\theta}_t), \mathbf{w}_t) \nabla_{\boldsymbol{\theta}} \tau(s_i, \boldsymbol{\theta}_t)$;
    Update target networks by

$$\boldsymbol{\theta}_{t+1}' \leftarrow \upsilon \boldsymbol{\theta}_t + (1 - \upsilon) \boldsymbol{\theta}_t'$$
$$\mathbf{w}_{t+1}' \leftarrow \upsilon \mathbf{w}_t + (1 - \upsilon) \mathbf{w}_t'$$

  end
end

Problem-Solution

- The exploration ability relies on
  - Random noise in selected action
    E.g. DDPG
  - Entropy regularization in objective
    E.g. PPO
    $L_t^{CLIP+VF+S}(\theta) = \widehat{E_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 \boxed{S[\pi_\theta](s_t)]}$
    - To maximum the objective, policy π_θ gets less entropy bonus $S[\pi_\theta]$ if π_θ is deterministic

Maximum Entropy Reinforcement Learning

- Standard reinforcement learning (RL) objective function:
  - Total expected rewards:

$$J(\pi_\theta) = \sum_t E_{(s_t, a_t) \sim \rho_{\pi_\theta}}[r(s_t, a_t)]$$

where ρ_{π_θ} is data distribution for policy π_θ

- Maximum entropy RL objective function:
  - Augment with entropy term:

$$J(\pi_\theta) = \sum_t E_{(s_t, a_t) \sim \rho_{\pi_\theta}}[r(s_t, a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

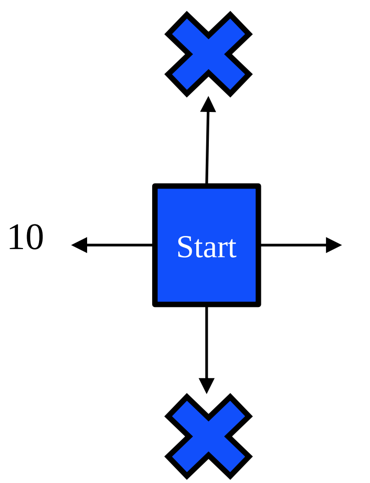
where α is temperature for importance of the entropy term

Maximum Entropy Reinforcement Learning

$$J(\pi_\theta) = \sum_t E_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

Return	Up	Left	Down	Right
	0	10	0	10

- Example:
  $J(\pi_\theta) = r(s_t,a_t) - log(\pi_\theta(s_t,a_t))$,
  Assume α=1



Extremely simple case

Policy	Up	Left	Down	Right
T=0	0.25	0.25	0.25	0.25
$J(\pi_\theta)$=	0-log0.25	10-log0.25	0-log0.25	10-log0.25
T=1	0.2	0.4	0.2	0.2
$J(\pi_\theta)$=	0-log0.2	10-log0.4	0-log0.2	10-log0.2
…				
T=k	10^{-10}	≈1	10^{-10}	10^{-10}
$J(\pi_\theta)$=	0-log10^{-10}	10-log1	0-log10^{-10}	10+10
…				
T=n	0	0.5	0	0.5

Ideal convergence

If we sampled "left" first

- Encourage take this action ("right") with entropy term
- The exploration bonus is vanish when the policy become deterministic

I-Chen Wu

Maximum Entropy Reinforcement Learning

- Encourage exploration with entropy term
  - Entropy in loss function: Consider entropy as regularized term
    - E.g.: PPO

$$L_t^{CLIP+VF+S}(\theta) = \widehat{E_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

The entropy term only cares the current state

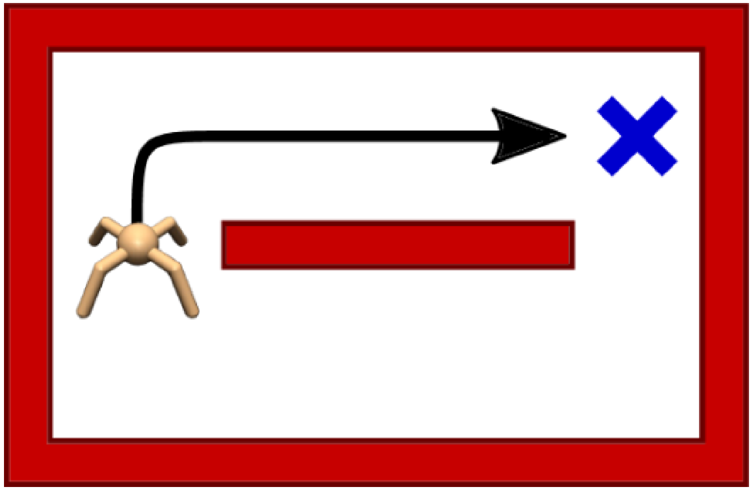
  - Entropy in objective function: Consider entropy as incentivized exploration reward
    - E.g.: SAC

$$J(\pi_\theta) = \sum_t E_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

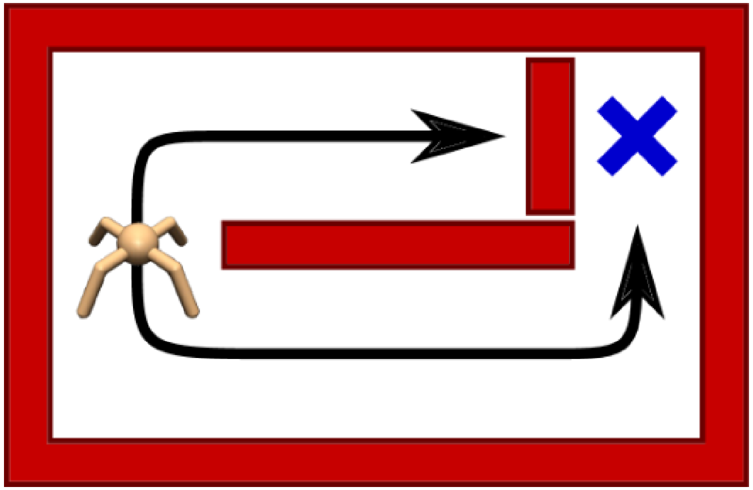
The entropy term affects following future states by accumulated return

Soft Actor Critic

- Soft Q-learning
- Soft actor critic
- Soft actor critic with automating entropy adjustment



2a



2b

Soft Q-Learning

- Objective function: Maximum entropy RL

$$J(\pi_\theta) = \sum_t E_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

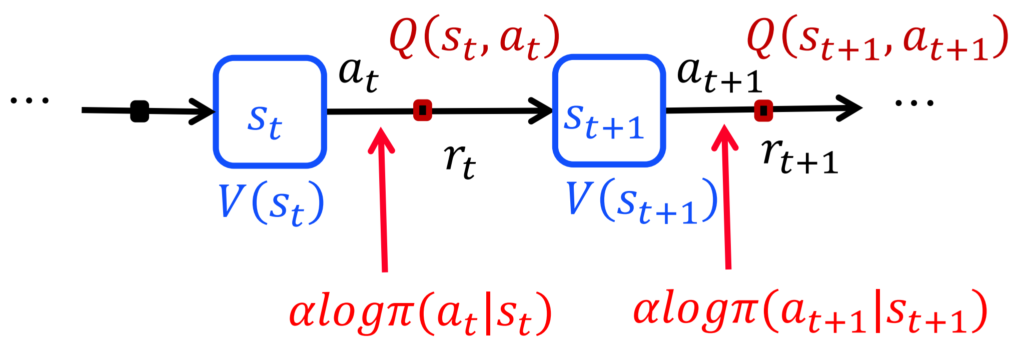
- Soft V-function:

$$V_{soft}(s_t) = E_{a_t\sim\pi_\theta}[Q_{soft}(s_t,a_t) - \alpha log\pi(a_t|s_t)]$$

- Soft Q-function:

$$Q_{soft}(s_t,a_t) = r_t + \gamma E_{s_{t+1}\sim\rho_{\pi_\theta}}[V_{soft}(s_{t+1})]$$

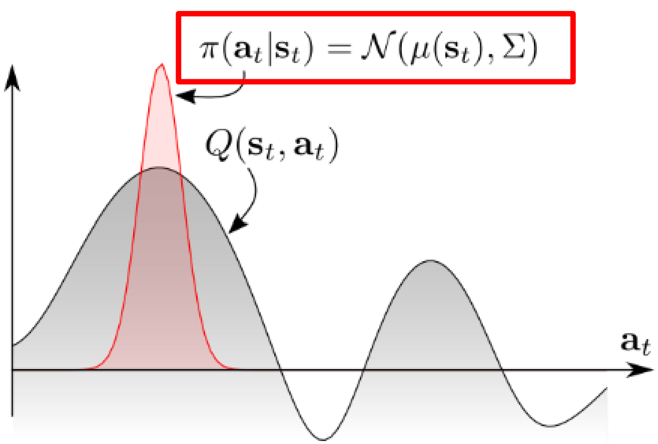
- Authors prove augment the entropy term still follow Bellman equation property
  - Policy evaluation
  - Policy improvement
  - Policy iteration



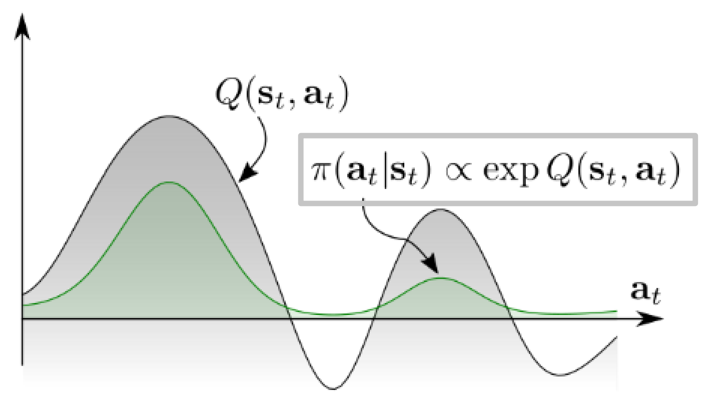
Soft Q-Learning

- Gaussian policy:
  - For convenient, usually assume the policy distribution is Gaussian distribution
  - Problem: Not suitable for multimodal case
- Energy-based policy:
  - Use Q value distribution to indicate the policy distribution
  - Assumption: $\pi(a_t|s_t) \propto \exp\left(Q(s_t, a_t)\right)$

Gaussian policy



Energy-based policy:
Stochastic policy with multimodal

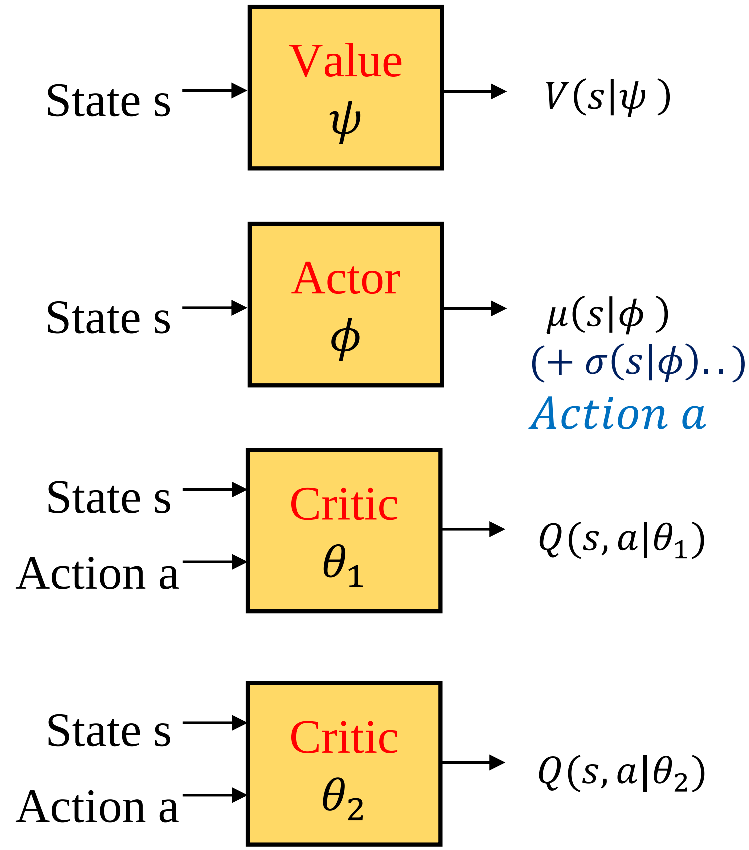


Soft Actor Critic

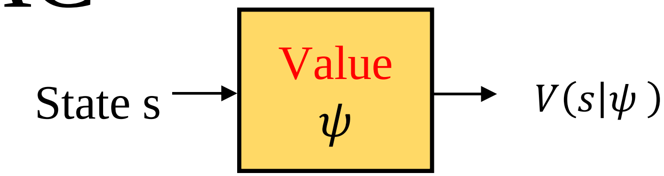
- Policy: (ideal)

  $$\pi(a_t|s_t) = \exp\left(\frac{1}{\alpha}\left(Q_{soft}(s_t, a_t) - V_{soft}(s_t)\right)\right)$$

- Architecture
  - 1 state value (V_{ψ}) network
  - 1 policy network (π_{ϕ})
  - 2 action-state value (Q-value) network (Q_{θ})
    - Double Q trick: Prevent overestimated in Q
    - Like TD3



Training of SAC



- D is the distribution of sampled states and actions

- Value network (V_ψ):

$$J_V(\psi) = E_{s_t \sim D}[\frac{1}{2}\left(V_\psi(s_t) - \widehat{V_\psi}(s_t)\right)^2]$$

  where $\widehat{V_\psi}(s_t) = E_{a_t \sim \pi_\phi}[Q_\theta(s_t, a_t) - \alpha log\pi_\phi(a_t|s_t)]$

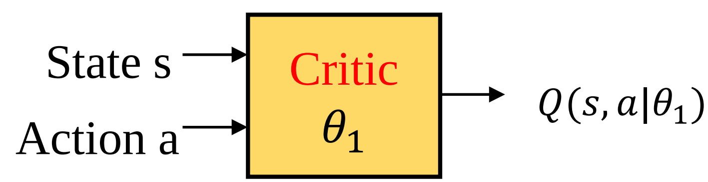
  Trained by minimizing the squared residual error (TD error)

- Q-Value network (Q_θ):

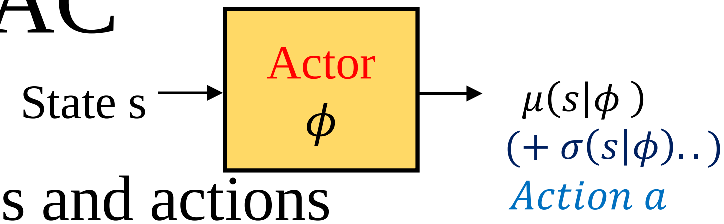
$$J_Q(\theta) = E_{(s_t,a_t) \sim D}[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \widehat{Q_\theta}(s_t, a_t)\right)^2]$$

  where $\widehat{Q_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1} \sim p}[V_\psi(s_{t+1})]$

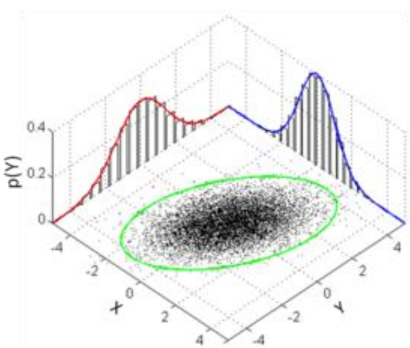
  Trained by minimizing the soft Bellman residual error (TD error)



Training of SAC



- D is the distribution of sampled states and actions
- Policy network (π_ϕ)
  - Train by minimizing the KL-divergence
  - Use reparameterization trick, sample action from fixed distribution
    $$J_\pi(\phi) = E_{s_t \sim D, \epsilon_t \sim N}\left[log\pi_\phi\left(f_\phi(\epsilon_t; s_t)\middle|s_t\right) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))\right]$$
    - $a_t = f_\phi(\epsilon_t; s_t)$,
    - ϵ_t is a noise vector
    - E.g.: $f_\phi(\epsilon_t; s_t)$ as spherical Gaussian distribution
    - Take gradient $\nabla_\phi J_\pi(\phi)$



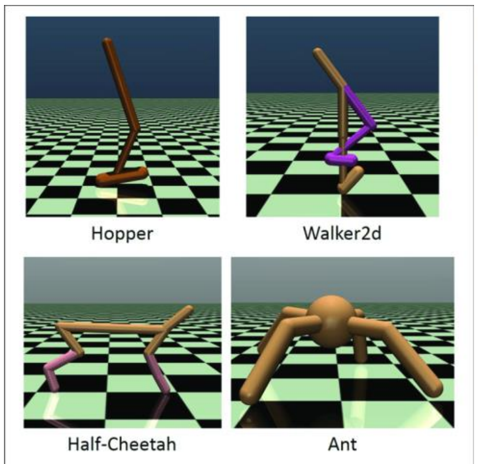
SAC Algorithm

Algorithm 1 Soft Actor-Critic

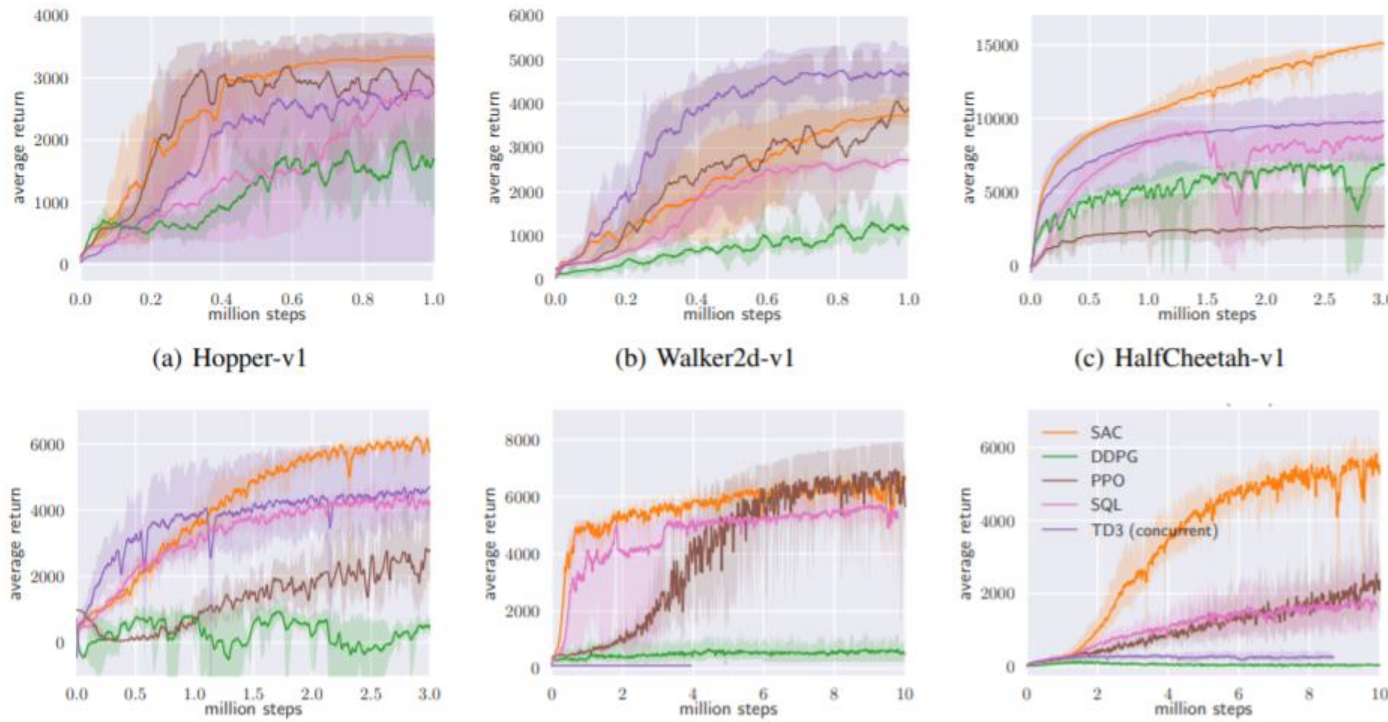
Initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ Double Q trick
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Result

- OpenAI gym v1



Humanoid



(a) Hopper-v1 (b) Walker2d-v1 (c) HalfCheetah-v1

(d) Ant-v1 (e) Humanoid-v1 (f) Humanoid (rllab)

Conclusion

- Soft actor critic (SAC) train a policy that maximize a trade-off between expected return and entropy
  - Still getting high performance while acting as random as possible
- Evolution of SAC
  - Soft Q-learning (SQL)
    - Soft: $\pi \propto Q(s, a)$
  - Soft Actor-Critic (SAC)
    - Argument the objective function with entropy term
  - Soft Actor-Critic with auto-adjusted temperature (SAC)
    - Argument the objective function with entropy term
    - Auto-adjust temperature
      - By constrained policy optimization