Lecture 13: Policy Optimization

Ziyu Shao

School of Information Science and Technology
ShanghaiTech University

June 01 & 03, 2020

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Value-based RL versus Policy-based RL

- In previous lectures, we focused on value-based RL and had value function approximation with parameter θ

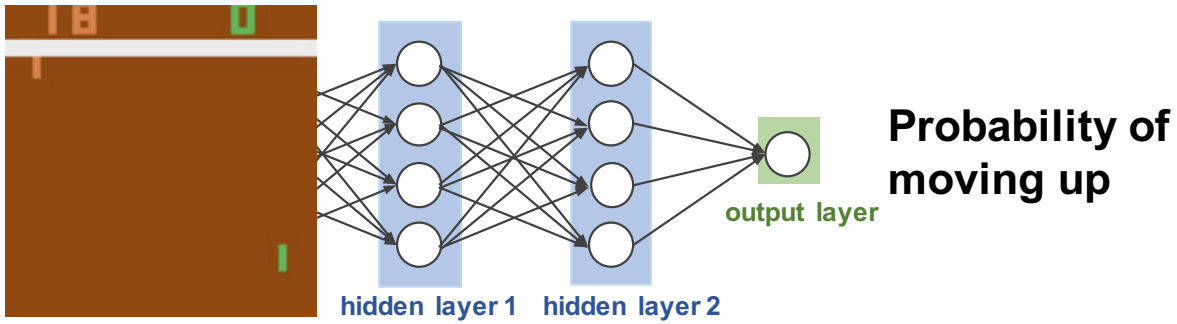
$$V_\theta(s) \approx v^\pi(s)$$
$$q_\theta(s,a) \approx q^\pi(s,a)$$

- A policy is generated directly from the value function:
  - using ϵ-greedy or greedy
- Instead, we can also parameterize the policy function:

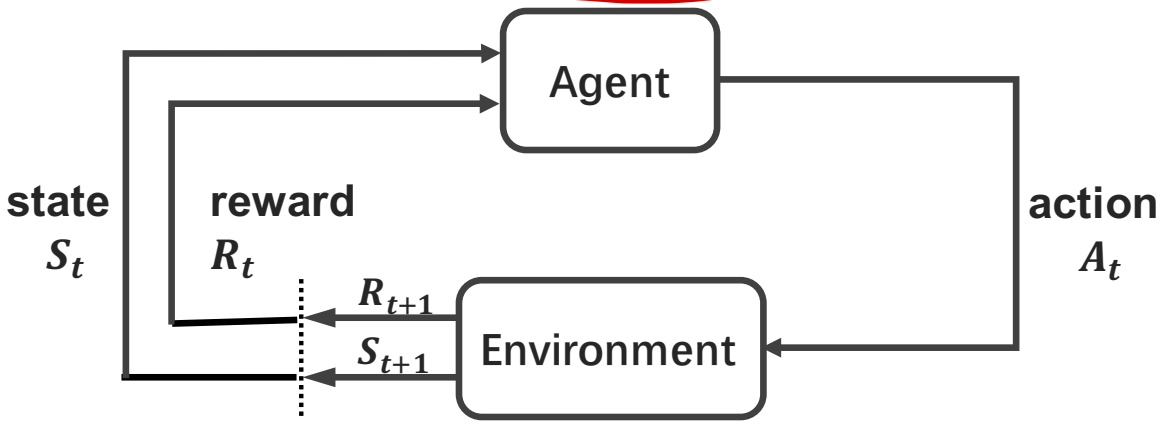
$$\pi_\theta(s,a) \approx P(a|s)$$

RL Diagram in the View of Policy Optimization

- Making the action is all we care, then let's optimize the policy directly!



$$\pi_\theta(A_t|S_t)$$



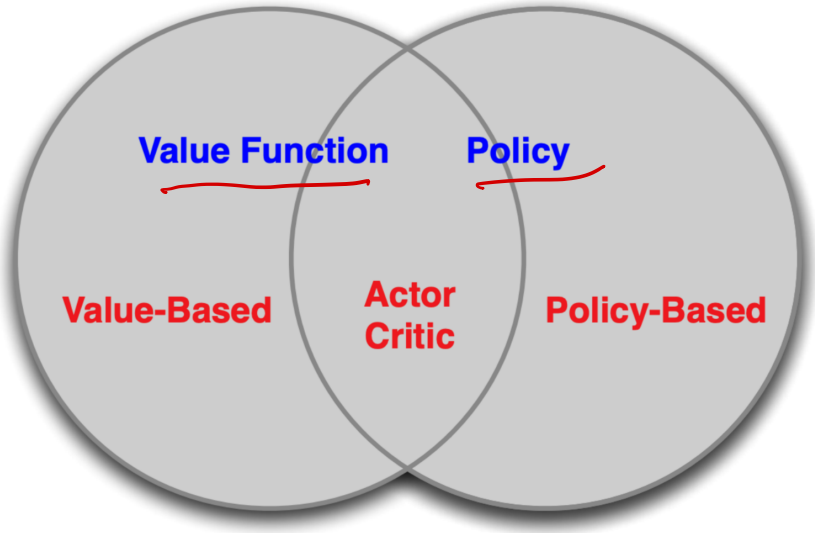
Value-based RL versus Policy-based RL

Primal
Dual
primal-dual
} convex optimization

- Value-based RL
  - to learn value function
  - implicit policy based on the value function
- Policy-based RL
  - No value function
  - to learn policy directly
- Actor-critic
  - to learn both policy and value function

Actor – policy

Critic – value function



Actor-Critic Framework

GPI

- Actor: a decision function that chooses a decision given the state
- Critic: the process that determines the contribution (cost or reward) from a decision, from which we can compute a value function
- Actor–critic framework: the interaction of making decisions and updating the value function
- An updating process with two time scales
  - one for the inner iteration to evaluate a policy (critic)
  - and one for the outer iteration where the policy is updated (actor)

Advantages of Policy-based RL

- **Advantages**:
  - Better convergence properties: we are guaranteed to converge on a local maximum (worst case) or global maximum (best case)
  - Policy gradients are more effective in high-dimensional action space
  - Policy gradients can learn stochastic policies, while value function can't
- **Disadvantages**:
  - typically converges to a local optimum
  - evaluating a policy is inefficient and high variance

Different Schools of Reinforcement Learning

- Value-based RL: solve RL through dynamic programming
  - Classic RL and control theory
  - Representative algorithms: Deep Q-learning and its variant
  - Representative person: Richard Sutton (no more than 20 pages on PG out of the 500-page textbook), David Silver, from DeepMind
- Policy-based RL: solve RL mainly through learning
  - Machine learning and deep learning
  - Representative algorithms: PG, and its variants TRPO, PPO, and others
  - Representative person: Pieter Abbeel, Sergey Levine, John Schulman, from OpenAI, Berkelely

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Two Types of Policies

- Deterministic: given a state, the policy returns the action to take
- Stochastic: given a state, the policy returns a probability distribution of the actions (e.g., 40% chance to turn left, 60% chance to turn right)

Example: Rock-Paper-Scissors

- Two-player game
- What is the best policy?
  - A deterministic policy is easily beaten
  - A uniform random policy is optimal (Nash equilibrium)

Optimizing Policy Value

- Objective: Given a policy approximator $\pi_\theta(s, a)$ with parameter θ, find the best θ
- How do we measure the quality of a policy π_θ?
- In episodic environments we can use the start value

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- 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

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

where d^{π_θ} is stationary distribution of Markov chain for π_θ

Optimizing Policy Value

- Policy value

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\sum_t r(s_t, a_t)] \approx \frac{1}{m} \sum_m \sum_t r(s_{t,m}, a_{t,m})$$

Sum over samples

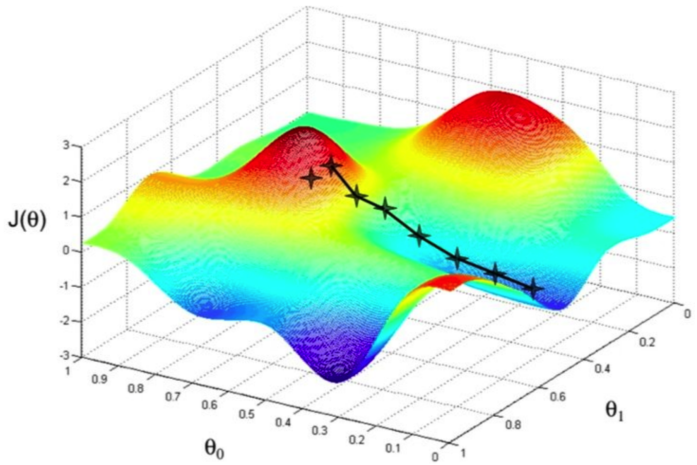
  - τ is a trajectory sampled from the policy function π_θ
- The goal of policy-based RL

$$\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t r(s_t, a_t) \right]$$

Optimizing Policy Value

- Policy-based RL is an optimization problem that find θ to maximize $J(\theta)$
- If $J(\theta)$ is differentiable, we can use gradient-based methods:
  - gradient ascend
  - conjugate gradient
  - quasi-newton
- Some derivative-free black-box optimization methods:
  - Cross-entropy method (CEM)
  - Hill climbing
  - Evolution algorithm

Policy Optimization using Gradient (Policy Gradient)



- Consider a function $J(\theta)$ to be any policy objective function
- Goal is to find parameter θ^* that maximizes $J(\theta)$ by ascending the gradient of the policy,w.r.t parameter θ

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

- Adjust θ in the direction of the gradient, where α is step-size
- Define the gradient of $J(\theta)$ to be

$$\nabla_\theta J(\theta) = (\frac{\partial J(\theta)}{\partial\theta_1}, \frac{\partial J(\theta)}{\partial\theta_2}, \ldots, \frac{\partial J(\theta)}{\partial\theta_n})^T$$

Policy Optimization using Derivative-free Methods

- Sometimes we cannot compute the derivative, i.e., $\nabla_\theta J(\theta)$
- Derivative free methods:
  - Cross Entropy Method (CEM)
  - Finite Difference

Cross-Entropy Method

- $\theta^* = \arg\max J(\theta)$
- Treat $J(\theta)$ as a black box score function (not differentiable)

Algorithm 1 CEM for black-box function optimization

for iter $i = 1$ to N **do**
  $\mathcal{C} = \{\}$
  for parameter set $e = 1$ to N **do**
    sample $\theta^{(e)} \sim P_{\mu^{(i)}}(\theta)$
    execute roll-outs under $\theta^{(e)}$ to evaluate $J(\theta^{(e)})$
    store $(\theta^e, J(\theta^{(e)}))$ in $\mathcal{C}$
  end for
  $\mu^{(i+1)} = \arg\max_\mu \sum_{k \in \hat{\mathcal{C}}} \log P_\mu(\theta^{(k)})$
  where $\hat{\mathcal{C}}$ are the top 10% of $\mathcal{C}$ ranked by $J(\theta^{(e)})$
end for

Approximate Gradients by Finite Difference

- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension $k \in [1, n]$
  - estimate kth partial derivative of objective function by perturbing θ by a 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 else where
- uses n evaluations to compute policy gradient in total n dimensions
- though noisy and inefficient, but works for arbitrary policies, even if policy is not differentiable.

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Computing the Policy Gradient Analytically

- Assume policy π_θ is differentiable whenever it is no-zero
- and we can compute the gradient $\nabla_\theta \pi_\theta(s, a)$
- Likelihood ratios exploit the following tricks

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$

Policy Example: Softmax Policy

- Simple policy model: weight actions using linear combination of features $\phi(s,a)^T\theta$
- Probability of action is proportional to the exponentiated weight

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

① $\pi_\theta(s,a) = \frac{e^{\phi(s,a)^T\theta}}{\sum_{a'} e^{\phi(s,a')^T\theta}}$

- The score function is

② $\log \pi_\theta(s,a) = \phi(s,a)^T\theta - \log\left(\sum_{a'} e^{\phi(s,a')^T\theta}\right)$

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

③ $\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \frac{\sum_{a'} \phi(s,a') \cdot e^{\phi(s,a')^T\theta}}{\sum_{a'} e^{\phi(s,a')^T\theta}}$

$= \phi(s,a) - \sum_{a'} \phi(s,a')\,\pi_\theta(s,a')$

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

Policy Example: Gaussian Policy

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

$$\pi_\theta(s,a) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}[a-\mu(s)]^2}$$

$$\log \pi_\theta(s,a) = \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}[a-\mu(s)]^2$$

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

$$\nabla_\theta \log \pi_\theta(s,a) = \frac{1}{\sigma^2}(a-\mu(s))\cdot\phi(s)$$

Policy Gradient for 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

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) r \end{aligned}$$

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

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

- The gradient is as

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) r \\ &= \mathbb{E}_{\pi_\theta}[r \nabla_\theta \log \pi_\theta(s, a)] \end{aligned}$$

Policy Gradient for Multi-step MDPs

- Denote a state-action trajectory from one episode as $\tau = (s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T, s_T) \sim \pi_\theta$
- Use $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t)$ to be the sum of rewards for a trajectory τ
- The policy objective is

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t)\right] = \sum_{\tau} P(\tau; \theta) R(\tau)$$

where $P(\tau; \theta)$ denotes the probability over trajectories when executing the policy π_θ

- Then our goal is to find the policy parameter θ

$$\theta^* = \arg\max_{\theta} J(\theta) = \arg\max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

Policy Gradient for Multi-step MDPs

- Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Take the gradient with respect to θ:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_\tau P(\tau;\theta)R(\tau) \\
&= \sum_\tau \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\
&= \sum_\tau P(\tau;\theta)R(\tau)\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \\
&= \sum_\tau P(\tau;\theta)R(\tau)\nabla_\theta \log P(\tau;\theta)
\end{aligned}$$

Policy Gradient for Multi-step MDPs

- Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta J(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta) R(\tau)$$

- Take the gradient with respect to θ:

$$\nabla_\theta J(\theta) = \sum_\tau P(\tau;\theta) R(\tau) \nabla_\theta \log P(\tau;\theta)$$

- Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i;\theta)$$

Decomposing the Trajectories into States and Actions

- Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

- Decompose $\nabla_\theta \log P(\tau; \theta)$

Chain rule + Markov.

$$\nabla_\theta \log P(\tau; \theta) = \nabla_\theta \log \left[\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right]$$

$$= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T-1} (\log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)) \right]$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

model information, transition prob.

Likelihood Ratio Policy Gradient

- Our goal is to find the policy parameter θ

$$\theta^* = \arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau;\theta)R(\tau)$$

- Approximate with empirical estimate for m sample paths under policy π_θ:

$$\nabla_\theta V(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i)\nabla_\theta \log P(\tau_i;\theta)$$

- And we have $\nabla_\theta \log P(\tau_i;\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

$$\nabla_\theta J(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

  - Do not need to know the dynamics model!

Comparison to Maximum Likelihood

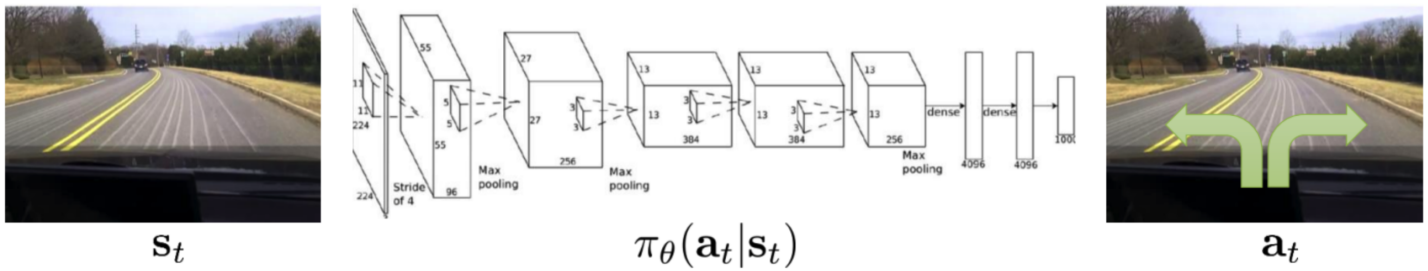
- Policy gradient: $\nabla_\theta J(\theta) \approx$

$$\frac{1}{M}\sum_{m=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{t,m}|s_{t,m})\right)\left(\sum_{t=1}^{T} r(s_{t,m}, a_{t,m})\right)$$

- Maximum likelihood:

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{M}\sum_{m=1}^{M}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{t,m}|s_{t,m})\right)$$

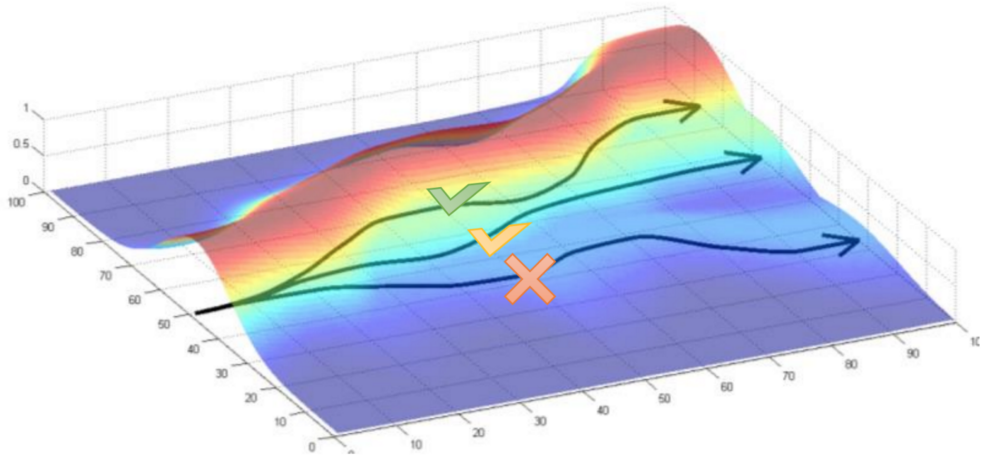
- Interpretation: good action is made more likely, bad action is made less likely



Comparison to Maximum Likelihood

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \nabla_\theta \log P(\tau_i; \theta)$$

- If going up the hill leads to higher reward, change the policy parameters to increase the likelihood of trajectories that move higher



Large Variance of Policy Gradient

- We have the following approximate update

$$\nabla_\theta J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i)$$

- Unbiased but very noisy
- Two fixes:
  - Use temporal causality
  - Include a baseline

Reduce Variance of Policy Gradient using Causality

- Previously $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau\left[\left(\sum_{t=0}^{T-1} r_t\right)\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\right]$
- We can derive the gradient estimator for a single reward term $r_{t'}$ as

$$\nabla_\theta \mathbb{E}_\tau[r_{t'}] = \mathbb{E}_\tau\left[r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- Summing this formula over t, we obtain

$$\begin{aligned}
\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] &= \mathbb{E}_\tau\left[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&= \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} r_{t'}\right] \\
&= \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]
\end{aligned}$$

Reduce Variance of Policy Gradient using Causality

- Therefore we have

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory at step t
- Causality: policy at time t' cannot affect reward at time t when $t < t'$
- Then we can have the following estimated update

$$\nabla_\theta \mathbb{E}[R] \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)$$

REINFORCE: A Monte-Carlo policy gradient algorithm

- The algorithm simply samples multiple trajectories following the policy π_θ while updating θ using the estimated gradient

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t, \boldsymbol{\theta})$ → Eligibility Vector.

- Classic paper: Williams (1992). REINFORCE is an acronym for "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility"

Reducing Variance Using a Baseline

Gradient Bandit

- The original update

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

  - $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a trajectory which might have high variance
- We subtract a baseline $b(s)$ from the policy gradient to reduce variance

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1} (G_t - b(s_t)) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- A good baseline is the expected return

$$b(s_t) = \mathbb{E}[r_t + r_{t+1} + \ldots + r_{T-1}]$$

Reducing Variance Using a Baseline

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}[R] = \mathbb{E}_\tau\left[\sum_{t=0}^{T-1}(G_t - b(s_t))\cdot\nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- Interpretation: increase the logprob of action a_t proportional to how much returns G_t are better than the expected return
- Baseline $b(s)$ can reduce variance, without changing the expectation
- We can prove that

try it!

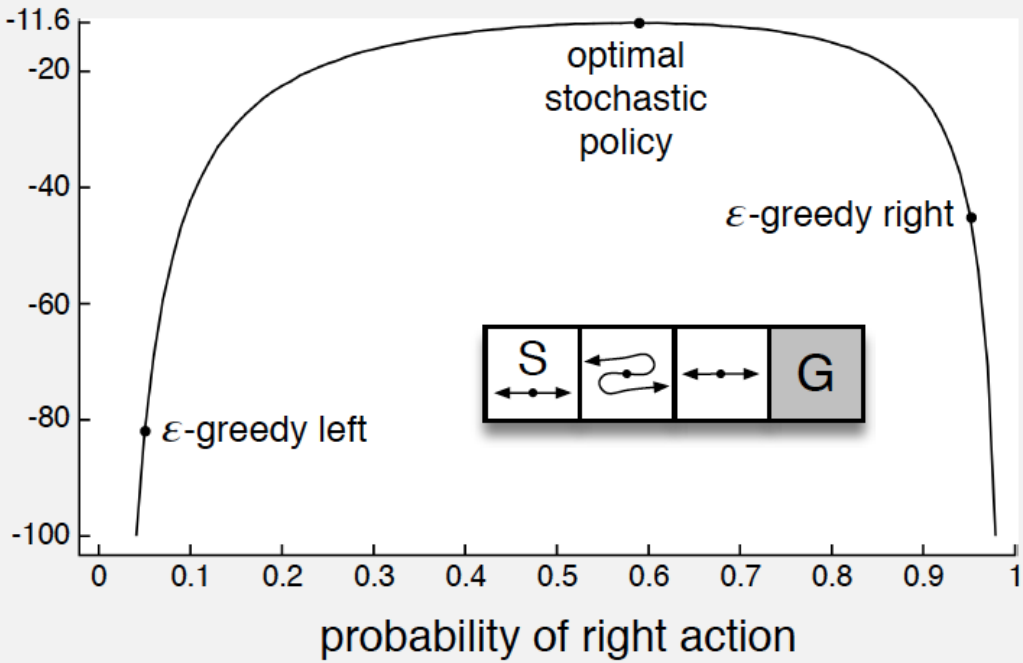
$$\mathbb{E}_\tau\left[\nabla_\theta \log \pi_\theta(a_t|s_t)b(s_t)\right] = 0$$

- Subtracting a baseline is unbiased in expectation

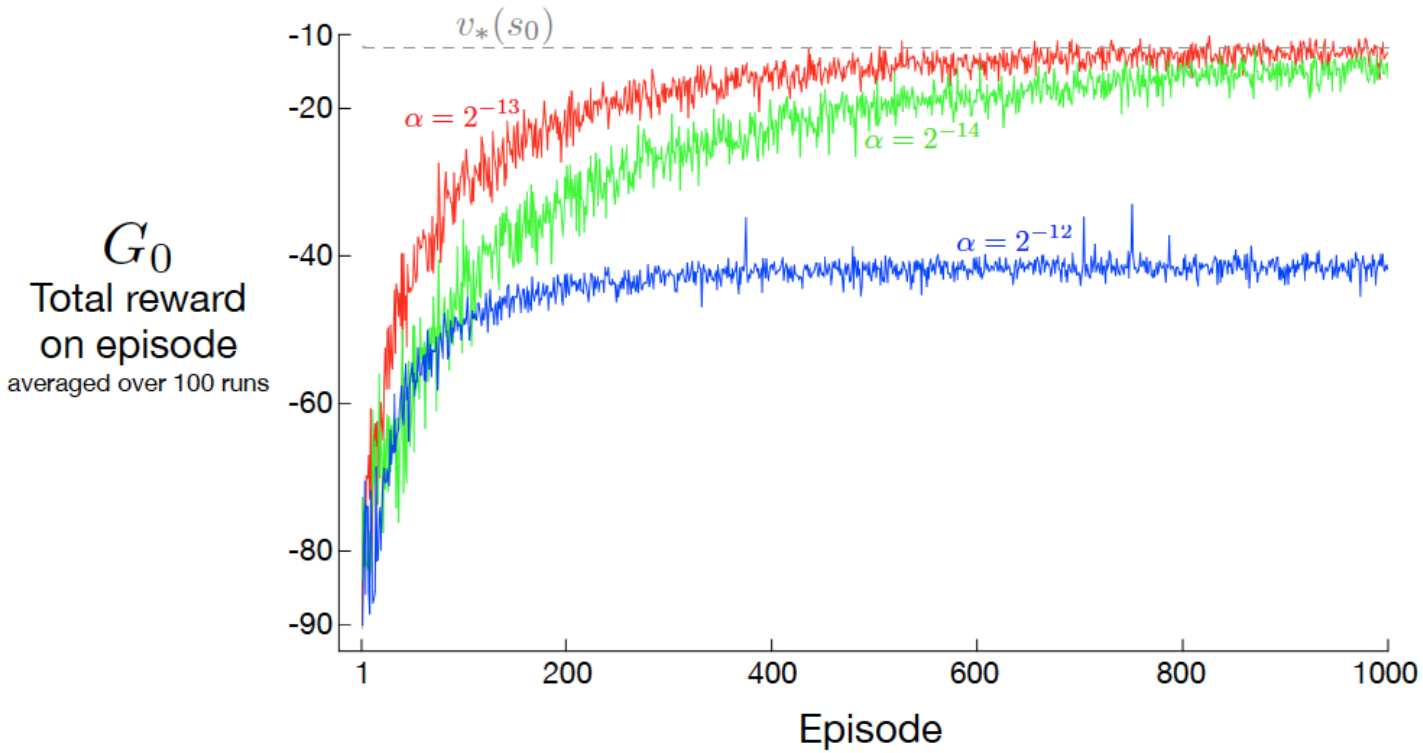
Example: Short Corridor with Switched Actions

do it!

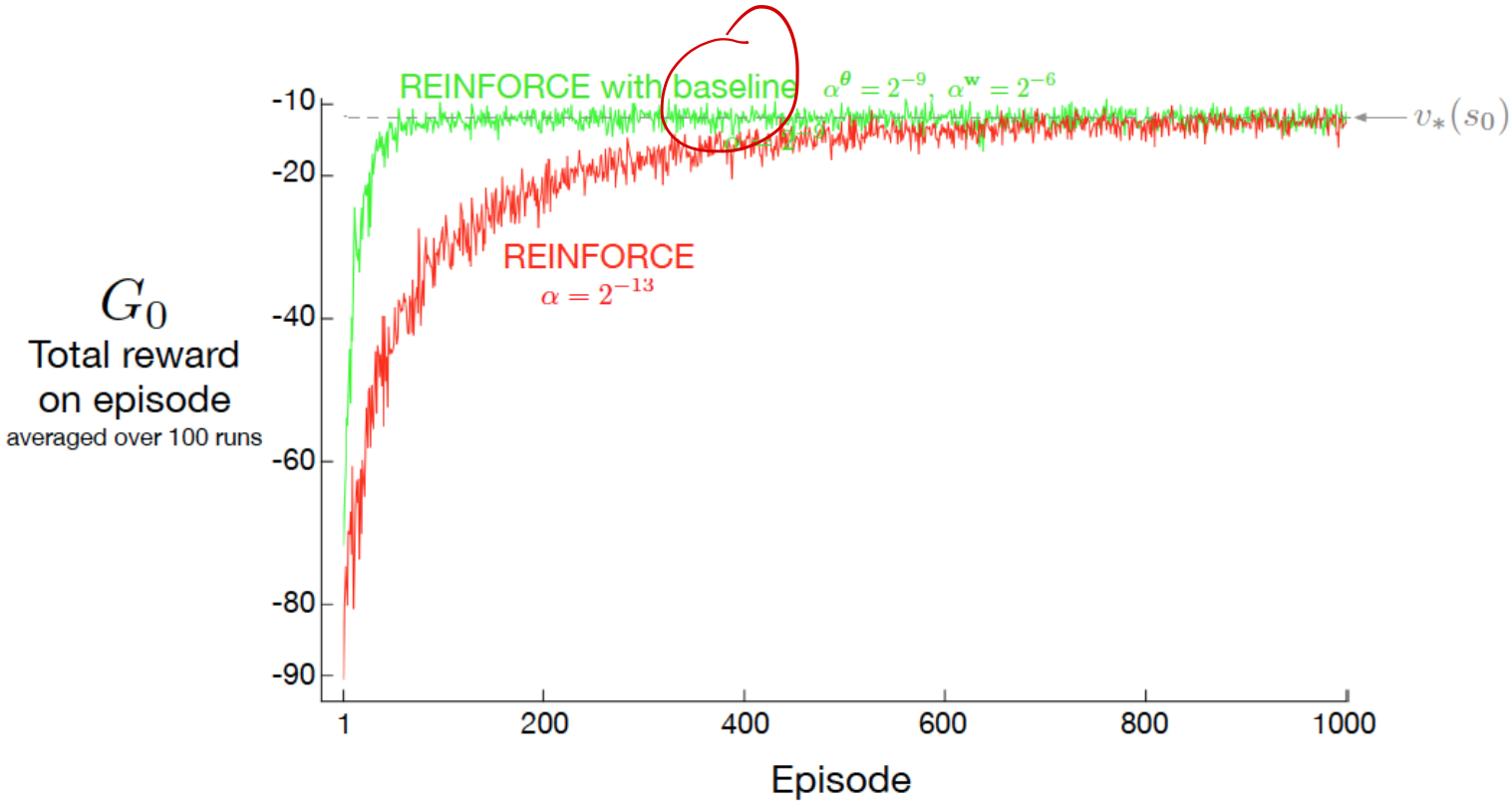
$$J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(\mathrm{S})$$



REINFORCE for Short Corridor with Switched Actions



REINFORCE with Baseline for Short Corridor with Switched Actions



Vanilla Policy Gradient Algorithm with Baseline

procedure POLICY GRADIENT(α)

Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0

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

Collect a set of m trajectories by executing the current policy π_θ

for each time step t of each trajectory $\tau^{(i)}$ **do**

Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$

Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$

Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

$$\sum_{i=1}^{m} \sum_{t=0}^{T-1} \| b(s_t) - G_t^{(i)} \|^2$$

Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m} \sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} | s_t^{(i)})$$

with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam

return θ and baseline values $b(s)$

Policy Gradient is On-Policy RL

- $\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$
- In REINFORCE algorithm: there is the on-policy sampling (sample $\{\tau\}$ from π_θ)
- On-policy learning can be extremely inefficient (Sample)

Off-Policy Learning using Importance Sampling

- $\theta^* = \arg\max_\theta J(\theta)$ where $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau)]$
- Let's say we have a behavior policy $\hat{\pi}$ where we can generate samples, then

$$J(\theta) = \mathbb{E}_{\tau \sim \hat{\pi}}\left[\frac{\pi_\theta(\tau)}{\hat{\pi}(\tau)} r(\tau)\right]$$

- Then we can derive the off-policy policy gradient
  - Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient. ICML

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Reducing Variance Using a Critic

- The update is

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- In practice, G_t is a sample from Monte Carlo policy gradient, which is the unbiased but noisy estimate of $Q^{\pi_\theta}(s_t, a_t)$
- Instead we can use a critic to estimate the action-value function,

$$Q_{\mathbf{w}}(s, a) \approx Q^{\pi_\theta}(s, a)$$

- Then the update becomes

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Reducing Variance Using a Critic

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} Q_{\mathbf{w}}(s_t, a_t) \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

- It becomes Actor-Critic Policy Gradient
  - Actor: the policy function used to generate the action
  - Critic: the value function used to evaluate the reward of the actions
- Actor-critic algorithms maintain two sets of parameters
  - Actor: Updates policy parameters θ, in direction suggested by critic
  - Critic: Updates action-value function parameters $\mathbf{w}$

Estimating the Action-Value Function

- The critic is solving a familiar problem: policy evaluation
  - How good is policy π_θ for current parameter θ
- Policy evaluation was explored in previous lectures, e.g.
  - Monte-Carlo policy evaluation
  - Temporal-Difference learning
  - Least-squares policy evaluation

Action-Value Actor-Critic Algorithm

- Using a linear value function approximation: $Q_{\mathbf{w}}(s, a) = \psi(s, a)^T \mathbf{w}$
  - Critic: update $\mathbf{w}$ by a linear TD(0)
  - Actor: update θ by policy gradient

Algorithm 1 Simple QAC

1: **for** each step **do**
2: generate sample s, a, r, s', a' following π_θ
3: $\delta = r + \gamma Q_{\mathbf{w}}(s', a') - Q_{\mathbf{w}}(s, a)$ #TD error
4: $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \psi(s, a)$
5: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_{\mathbf{w}}(s, a)$
6: **end for**

Bias in Actor-Critic Algorithms

- Approximating the policy gradient introduces bias
- A biased policy gradient may not find the right solution
  - e.g. if $Q_w(s, a)$ uses aliased features, can we solve gridworld example?
- Luckily, if we choose value function approximation carefully
- Then we can avoid introducing any bias
- i.e. We can still follow the *exact* policy gradient

Compatible Function Approximation

Theorem (Compatible Function Approximation Theorem)

If the following two conditions are satisfied:

- *Value function approximator is compatible to the policy*

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

- *Value function parameters w minimize the mean-squared error*

$$\epsilon = \mathbb{E}_{\pi_\theta}\left[(Q^{\pi_\theta}(s,a) - Q_w(s,a))^2\right]$$

Then the policy gradient is exact,

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

Proof of Compatible Function Approximation Theorem

If w is chosen to minimize mean-squared error, gradient of ϵ w.r.t. w must be zero,

$$\nabla_w \epsilon = 0$$

$$\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_w Q_w(s,a)\right] = 0$$

$$\mathbb{E}_{\pi_\theta}\left[(Q^\theta(s,a) - Q_w(s,a))\nabla_\theta \log \pi_\theta(s,a)\right] = 0$$

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

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

So $Q_w(s,a)$ can be substituted directly into the policy gradient,

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

Reducing the Variance of Actor-Critic by a Baseline

- Recall Q-function / state-action-value function:

$$Q^{\pi,\gamma}(s,a) = \mathbb{E}_{\pi}[r_1 + \gamma r_2 + \ldots | s_1 = s, a_1 = a]$$

- State value function can serve as a great baseline

$$\begin{aligned} V^{\pi,\gamma}(s) &= \mathbb{E}_{\pi}[r_1 + \gamma r_2 + \ldots | s_1 = s] \\ &= \mathbb{E}_{a\sim\pi}[Q^{\pi,\gamma}(s,a)] \end{aligned}$$

- Advantage function: combing Q with baseline V

$$A^{\pi,\gamma}(s,a) = Q^{\pi,\gamma}(s,a) - V^{\pi,\gamma}(s)$$

- Then the policy gradient becomes:

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

N-step estimators

- We used the Monte-Carlo estimates of the reward
- We can also us TD methods for the policy gradient update, or any intermediate blend between TD and MC methods:
- Consider the following n-step returns for $n = 1, 2, \infty$

$$
\begin{aligned}
n = 1 (TD) \quad & G_t^{(1)} = r_{t+1} + \gamma v(s_{t+1}) \\
n = 2 \quad & G_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) \\
n = \infty (MC) \quad & G_t^{(\infty)} = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T
\end{aligned}
$$

- Then the advantage estimators become

$$
\begin{aligned}
\hat{A}_t^{(1)} &= r_{t+1} + \gamma v(s_{t+1}) - v(s_t) \\
\hat{A}_t^{(2)} &= r_{t+1} + \gamma r_{t+2} + \gamma^2 v(s_{t+2}) - v(s_t) \\
\hat{A}_t^{(\infty)} &= r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T - v(s_t)
\end{aligned}
$$

$\hat{A}^{(1)}$ has low variance and high bias. $\hat{A}^{(\infty)}$ has high variance but low bias

Estimating the Advantage Function (1)

try it!

- 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

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

- 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 $r + \gamma V(s')$

$$\Delta\theta = \alpha(r + \gamma V(s') - V_\theta(s))\phi(s)$$

  - 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

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$
$$e_t = \gamma\lambda e_{t-1} + \phi(s_t)$$
$$\Delta\theta = \alpha\delta_t e_t$$

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

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

Policy Gradient with Eligibility Traces

- Just like forward-view TD(λ), we can mix over time-scales

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

- where $v_t^\lambda - V_v(s_t)$ is a biased estimate of advantage fn
- Like backward-view TD(λ), we can also use eligibility traces
  - By equivalence with TD(λ), substituting $\phi(s) = \nabla_\theta \log \pi_\theta(s, a)$

$$\begin{aligned}
\delta &= r_{t+1} + \gamma V_v(s_{t+1}) - V_v(s_t) \\
e_{t+1} &= \lambda e_t + \nabla_\theta \log \pi_\theta(s, a) \\
\Delta\theta &= \alpha\delta e_t
\end{aligned}$$

- This update can be applied online, to incomplete sequences

Natural Policy Gradient

- The vanilla policy gradient is parameter sensitive
- A small change in the policy parameter can cause a big change in the conditional probability density $\pi_\theta(a|s)$.
- The vanilla gradient can be defined as the steepest ascent direction under the Euclidean metric
- Nature gradients: gradients that treat all dimensions equally in the space of probability densities $\pi_\theta(a|s)$
- The natural policy gradient is parameter independent
- The nature gradient can be defined as the steepest ascent direction under the Riemannian metric

Natural Policy Gradient

- It finds ascent direction that is closest to vanilla gradient, when changing policy by a small, fixed amount

$$\nabla_\theta^{nat} \pi_\theta(s,a) = G_\theta^{-1} \nabla_\theta \pi_\theta(s,a)$$

- where G_θ is the Fisher information matrix

$$G_\theta = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^\top\right]$$

- G_θ is the Riemannian metric induced by the KL divergence

Natural Actor-Critic

- Using compatible function approximation,

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

- So the natural policy gradient simplifies,

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a)^\top w\right] \\ &= G_\theta w\end{aligned}$$

$$\nabla_\theta^{nat} J(\theta) = w$$

- i.e. update actor parameters in direction of critic parameters

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) \delta e\right] && \text{TD}(\lambda)\text{ Actor-Critic} \\
G_\theta^{-1} \nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$ or $V^\pi(s)$

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Challenges with Policy Gradient

- Poor sample efficiency as PG is on-policy learning

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

- Large policy update or improper step size destroy the training
  - This is different from supervised learning where the learning and data are independent
  - In RL, step too far $\rightarrow$ bad policy $\rightarrow$ bad data collection
  - May not be able to recover from a bad policy, collapse in performance

Gradient Ascend in Policy Gradient

Gradient ascend approaches update the weights a small step in direction of gradient on the loss function

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta)$$

- SGD: First order / linear approximation of the value functions depends on the policy parameterization
- The standard gradient ascend $\nabla_\theta J(\theta)$ is not necessarily the steepest ascend

Extending Policy Gradient with Importance Sampling

- We can modify PG into off-policy learning using importance sampling
- **Importance sampling**: IS calculates the expected value of $f(x)$ where x has a data distribution p
  - we can sample data from another distribution q and use the probability ratio between p and q to re-calibrate the result

$$\mathbb{E}_{x\sim p}[f(x)] = \mathbb{E}_{x\sim q}\left[\frac{p(x)}{q(x)}f(x)\right]$$

- Using important sampling in policy objective

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau)] = \mathbb{E}_{\tau\sim\hat{\pi}}\left[\frac{\pi_\theta(\tau)}{\hat{\pi}(\tau)}r(\tau)\right]$$

Increasing the Robustness with Trust Regions

- The behavior policy could be the old policy directly, thus we can have a surrogate objective function

$$\theta = \arg\max_{\theta} J_{\theta_{\text{old}}}(\theta) = \arg\max_{\theta} \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t \right]$$

- The estimate might be excessively large when $\pi_\theta / \pi_{\theta_{\text{old}}}$ is too large
- Solution: to limit the difference between subsequent policies
- For instance, use the Kullbeck-Leibler (KL) divergence to measure the distance between two policies

$$KL(\pi_{\theta_{\text{old}}} \| \pi_\theta) = \sum_a \pi_{\theta_{\text{old}}}(a|s) \log \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

Increasing the Robustness with Trust Regions

- Thus our objective with trust region becomes

$$\begin{aligned} &\text{maximize}_\theta \quad \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A_t\right] \\ &\text{subject to} \quad KL(\pi_{\theta_{\text{old}}}(\cdot|s_t)\|\pi_\theta(\cdot|s_t)) < \delta \end{aligned}$$

- In the trust region, we limit our parameter search within a region controlled by δ

Gradient ascend

Trust region

Trust Region Optimization

- Following Taylors series expansion on both terms above up to the second-order
- After some derivations we can have

$$L_{\theta_t}(\theta) \approx g^T(\theta - \theta_t)$$

$$KL(\theta \| \theta_t) \approx \frac{1}{2}(\theta - \theta_t)^T H(\theta - \theta_t)$$

where $g = \nabla_\theta L_{\theta_t}(\theta)$ and $H = \nabla^2_\theta KL(\theta \| \theta_t)$ and θ_t is the old policy parameter

- Then the objective turns to:

$$\theta_{t+1} = \arg\max_\theta g^T(\theta - \theta_t) \text{ s.t. } \frac{1}{2}(\theta - \theta_t)^T H(\theta - \theta_t) \leq \delta$$

- This is a quadratic equation and can be solved analytically:

$$\theta_{t+1} = \theta_t + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

Natural Policy Gradient

- Natural gradient is the steepest ascent direction with respect to the Fisher information

$$\theta_{t+1} = \theta_t + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

- H is the Fisher Information Matrix (FIM) which can be computed explicitly as

$$H = \nabla_\theta^2 KL(\pi_{\theta_t} \| \pi_\theta) = E_{x \sim \pi_{\theta_t}} \left[(\nabla_\theta \log \pi_\theta(x))^T (\nabla_\theta \log \pi_\theta(x)) \right]$$

- Learning rate (δ) can be thought of as choosing a step size that is normalized with respect to the change in the policy
- This is beneficial because it means that we don't do any parameter updates that will drastically change the output of the policy network.

Natural Policy Gradient

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **do**
Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian / Fisher Information Matrix $\hat{H}_k$

Compute Natural Policy Gradient update:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$$

end for

- Sham Kakade. A Natural Policy Gradient. NIPS 2001
- A nice read on natural gradient: `https://wiseodd.github.io/techblog/2018/03/14/natural-gradient/`

Trust Region Policy Optimization (TRPO)

- FIM and its inverse are very expensive to compute
- TRPO estimates the term $x = H^{-1}g$ by solving the following linear equation $Hx = g$
- Consider the optimization for a quadratic equation

$$\text{Solving } Ax = b \text{ is equivalent to}$$

$$x = \arg\max_x f(x) = \frac{1}{2}x^T Ax - b^T x$$

$$\text{since } f'(x) = Ax - b = 0$$

- Thus we can optimize the quadratic equation as

$$\min_x \frac{1}{2}x^T Hx - g^T x$$

- Use conjugate gradient method to solve it. It is very similar to the gradient ascent but can be done in fewer iterations

Trust Region Policy Optimization (TRPO)

- Resulting algorithm is a refined version of natural policy gradient

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **do**
Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$
Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$
Perform backtracking line search with exponential decay to obtain final update

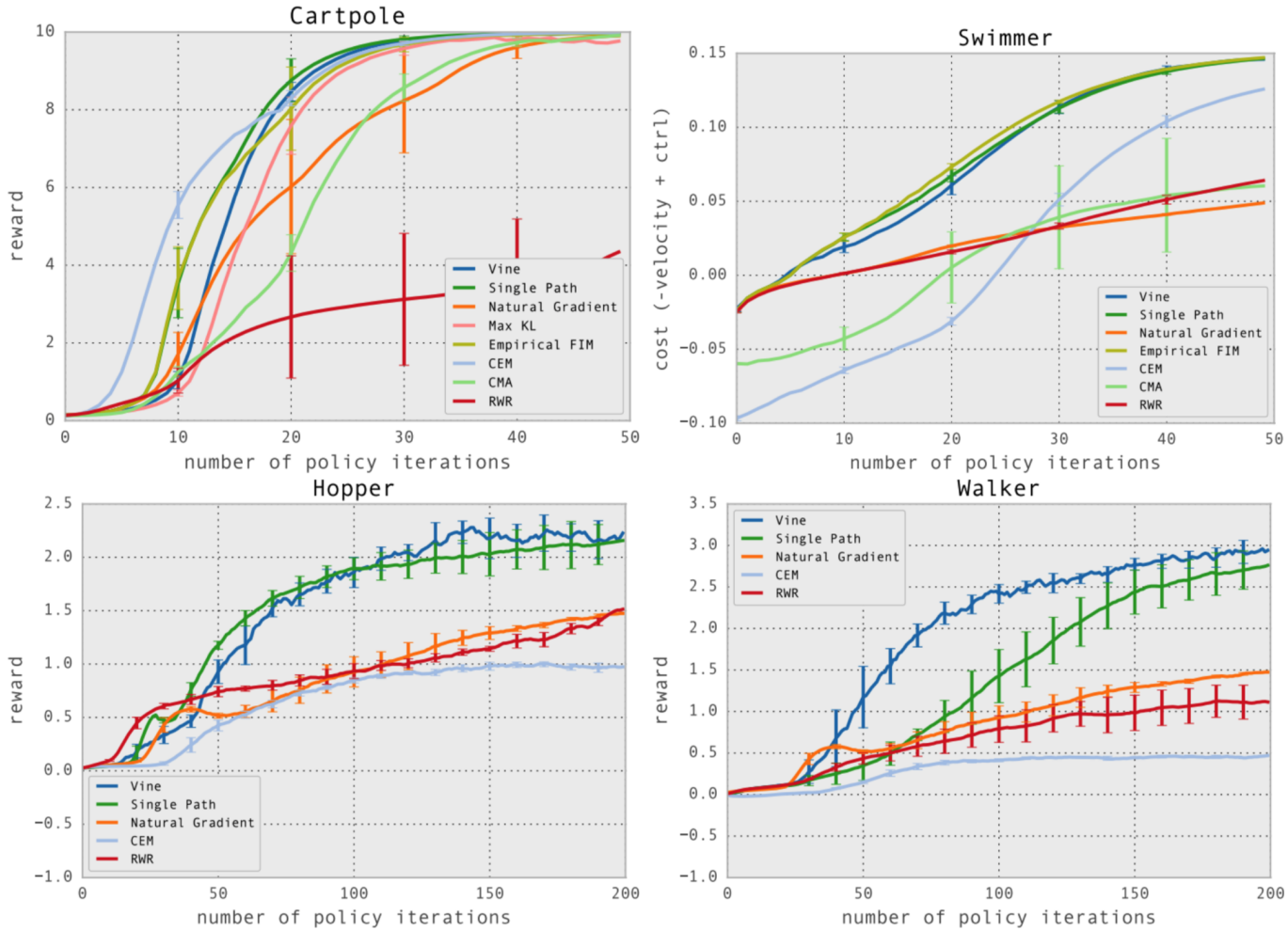
$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for

Trust Region Policy Optimization (TRPO)

- Schulman, et al. ICML 2015: a lot of proofs
- The appendix A of the TRPO paper provides a 2-page proof that establishes the guaranteed monotonic improvement that **the policy update in each TRPO iteration creates a better policy**

Result and Demo of TRPO



- Demo video is at
  `https://www.youtube.com/watch?v=KJ15iGGJFvQ`

Limitations of TRPO

- Scalability issue for TRPO
  - Computing H every time for the current policy model is expensive
  - It requires a large batch of rollouts to approximate H

$$H = E_{x\sim\pi_{\theta_t}}\left[(\nabla_\theta \log \pi_\theta(x))^T(\nabla_\theta \log \pi_\theta(x))\right]$$

  - Conjugate Gradient(CG) makes implementation more complicated
- TRPO does not work well for deep networks

	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

ACKTR: Calculating Natural Gradient with KFAC

- Y. Wu, et al. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. NIPS 2017.
- ACKTR speeds up the optimization by reducing the complexity of calculating the inverse of the H(FIM) using the Kronecker-factored approximation curvature (K-FAC).

$$F = E_{x\sim\pi_{\theta_t}}\left[(\nabla_\theta \log \pi_\theta(x))^T(\nabla_\theta \log \pi_\theta(x))\right]$$

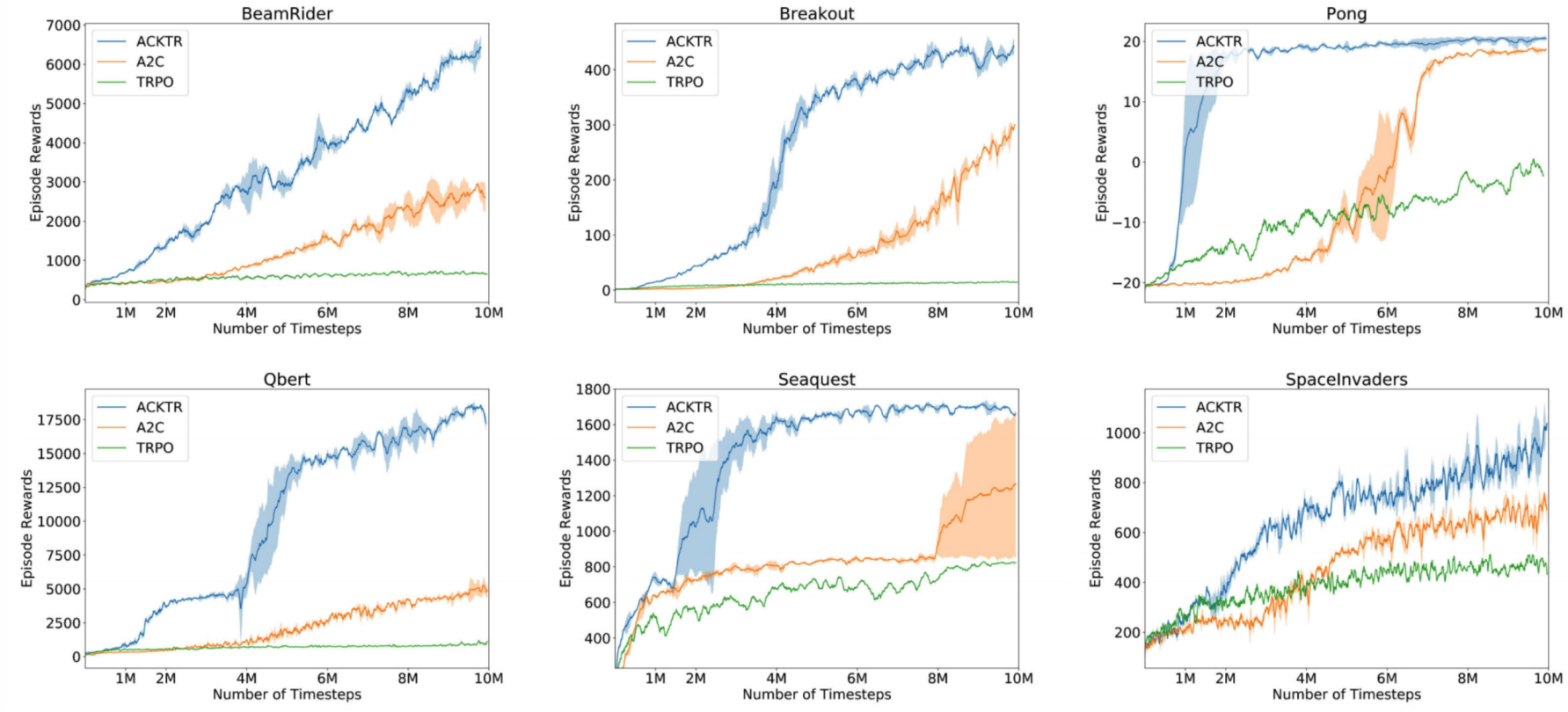
- It is replaced as layer-wise calculation

$$\begin{aligned} F_\iota =& \mathbb{E}[\text{vec}\{\nabla_W L\}\text{vec}\{\nabla_W L\}^T] = \mathbb{E}[aa^T \otimes \nabla_s L(\nabla_s L)^T] \\ \approx& \mathbb{E}[aa^T] \otimes \mathbb{E}[\nabla_s L(\nabla_s L)^T] := A \otimes S := \hat{F}_\iota, \end{aligned}$$

where A denotes $\mathbb{E}[aa^T]$ and S denotes $\mathbb{E}[\nabla_s L(\nabla_s L)^T]$.

Performance of ACKTR

- Performance of ACKTR



- Introductory link:
  `https://blog.openai.com/baselines-acktr-a2c/`

Proximal Policy Optimization (PPO)

- The loss function in the Natural Gradient and TRPO

$$\begin{aligned} &\text{maximize}_\theta \quad \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A_t\right] \\ &\text{subject to} \quad \mathbb{E}_t[KL[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta \end{aligned}$$

- It also can be written into an unconstrained form,

$$\text{maximize}_\theta \ \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}A_t\right] - \beta\mathbb{E}_t[KL[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]]$$

- PPO: include the adaptive KL Penalty, so the optimization will have better insurance that we are optimizing within a trust region

Proximal Policy Optimization (PPO)

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

- Same performance as TRPO, but solved much faster by first-order optimization (SGD)

PPO with clipping

- Let $r_t(\theta)$ denote the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ so different surrogate objectives:
  - PG without trust region: $L_t(\theta) = r_t(\theta)\hat{A}_t$
  - KL constraint: $L_t(\theta) = r_t(\theta)\hat{A}_t$ s.t. $KL[\pi_{\theta_{old}}, \pi_\theta] \leq \delta$
  - KL penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta KL[\pi_{\theta_{old}}, \pi_\theta]$
- A new objective function to clip the estimated advantage function if the new policy is far away from the old policy (r_t is too large)

$$L_t(\theta) = \min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$$

  - If the probability ratio between the new policy and the old policy falls outside the range $(1-\epsilon)$ and $(1+\epsilon)$, the advantage function will be clipped.
  - ϵ is set to 0.2 for the experiments in the PPO paper.

PPO with clipping

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

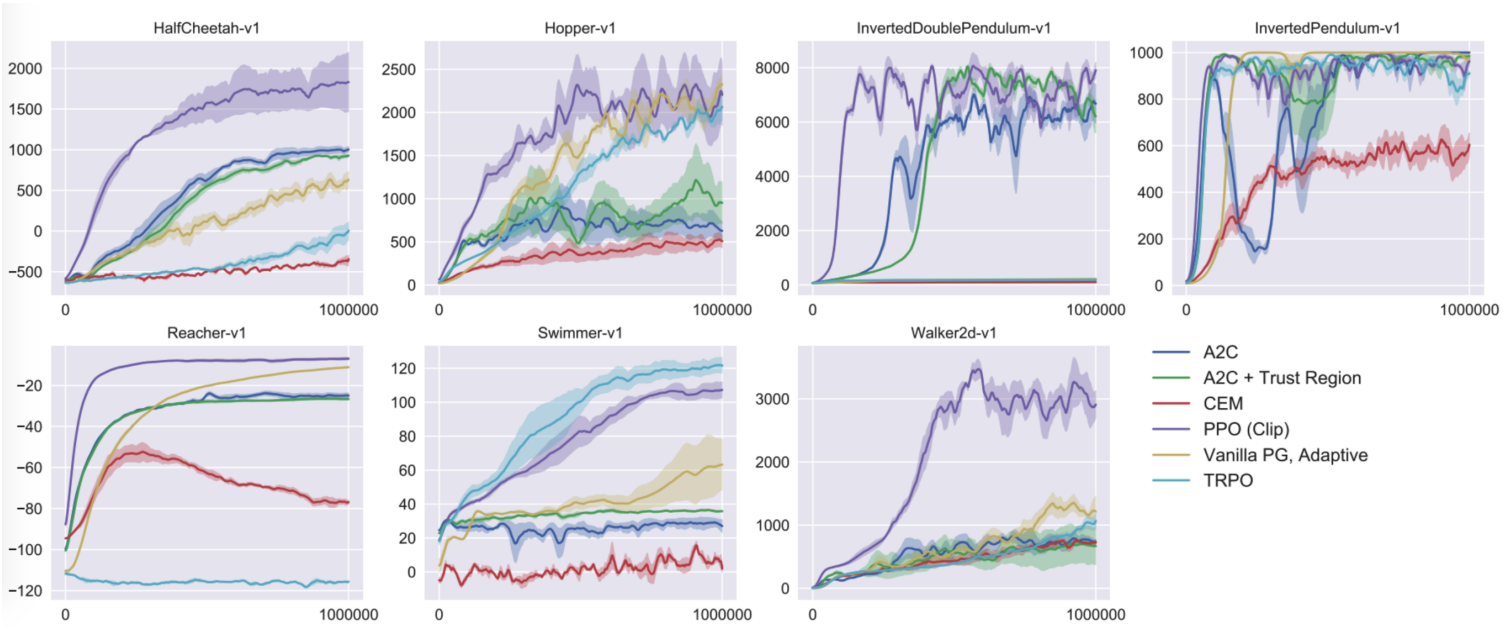
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

- PPOs have the stability and reliability of trust-region methods but are much simpler to implement, requiring only few lines of code change to a vanilla policy gradient implementation

Result of PPO

- Result on the continuous control tasks (MuJuCo)
  `https://gym.openai.com/envs/mujoco`



- Demo of PPO at
  `https://blog.openai.com/openai-baselines-ppo/`
- Emergence of Locomotion Behaviors in Rich Environments by DeepMind (Distributed PPO)
  - `https://www.youtube.com/watch?v=hx_bgoTF7bs`

State of the Art on Policy Gradient

State-of-the-art RL methods are almost all policy-based

- **TRPO**: Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization
  - comment: Solid math proofs and guarantee
- **ACKTR**: Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.
  - comment: numeric optimization-based improvement, scalable to real-problems
- **PPO**: Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithms
  - comment: Easy to read, elegant design of loss function, easy to implement, widely used
- What's next?

OpenAI

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Classical Control Problems

- Commonly used as benchmarks for tabular RL and RL algorithms using linear function approximators
  - Cartpole: balancing a pole on a cart
  - Mountain Car: trying to get a car up a mountain using momentum
  - Acrobot: swinging a pole up using momentum and subsequently balancing it
- Still sometimes used to benchmark deep RL algorithms

Games

- Board games including mastering Go & Poker
- Video Games are also popular benchmarks because
  - many of these games have large observation space and/or large action space
  - they are often non-Markovian, which require specific care
  - they also usually require very long planning horizons (e.g., due to sparse rewards)
- Adopted for several research topics including
  - multi-task learning & transfer learning
  - lifelong-learning & curriculum learning
  - predictive planning & hierarchical planning
  - meta-RL & multi-agent RL
- Limitation: majority investigate discrete action decisions

Popular Video Game Platforms

- Arcade Learning Environment (ALE): a suite of iconic Atari games, including Pong, Asteroids, Montezuma's Revenge, Space Invaders, Seaquest and Breakout
- General Video Game AI (GVGAI): competition framework
- VizDoom: implements the Doom video game as a simulated environment for RL
- Minecraft
- Quake video game
- StarCraft

Continuous Control Systems & Robotics

- MuJoCo simulation framework: provide several locomotion benchmark tasks
- Roboschool: provides the same locomotion tasks along with more complex tasks involving humanoid robot simulations

Easy-to-use Wrappers

- OpenAI Gym: provides ready access to environments such as
  - Atari, board games, Box2d games
  - classical control problems
  - MuJoCo robotics simulations
- Gym Retro
- μniverse
- SerpentAI

Frameworks of Deep RL

Framework	Deep RL	Python interface	Automatic GPU support
DeeR	yes	yes	yes
Dopamine	yes	yes	yes
ELF	yes	no	yes
OpenAI baselines	yes	yes	yes
PyBrain	yes	yes	no
RL-Glue	no	yes	no
RLPy	no	yes	no
rllab	yes	yes	yes
TensorForce	yes	yes	yes

Outline

1. Introduction
2. Policy Optimization
3. Monte-Carlo Policy Gradient
4. Actor-Critic Policy Gradient
5. *State-of-the-Art Algorithm
6. Benchmarking Deep Reinforcement Learning
7. References

Main References

- Reinforcement Learning: An Introduction (second edition), R. Sutton & A. Barto, 2018.
- RL course slides from Richard Sutton, University of Alberta.
- RL course slides from David Silver, University College London.