

EECS 498/598: Deep Learning

Lecture 11. Reinforcement Learning 2

Honglak Lee

03/29/2019

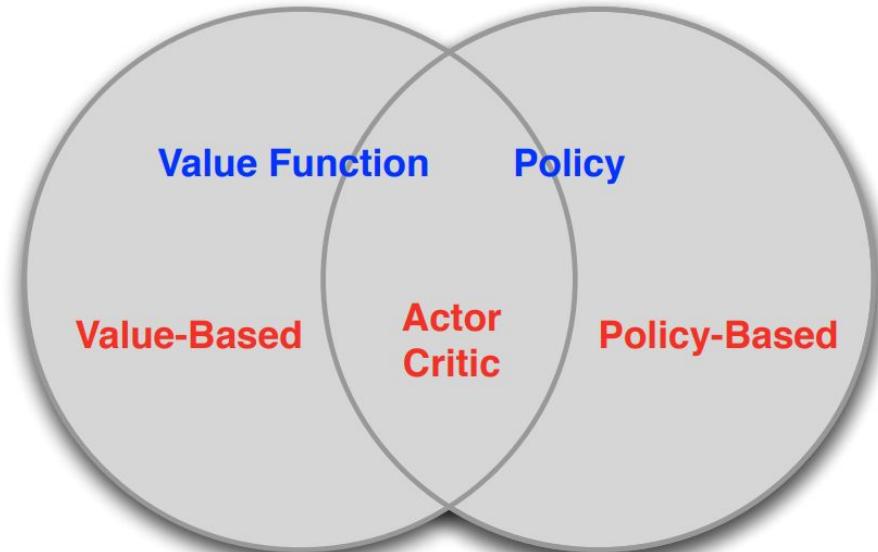


Outline: Policy-based RL Methods

- Introduction
- REINFORCE
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)
- Advanced policy gradient methods
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)

Overview

- Value Based
 - Learnt Value Function
 - Implicit policy
 - Eg: Q-learning, DQN
- Policy Based
 - No Value Function
 - Learnt Policy
 - E.g., REINFORCE, TRPO, PPO
- Actor-Critic
 - Learnt Value Function
 - Learnt Policy
 - E.g., A2C, A3C



Policy-Based RL

Advantages

- Better convergence properties
- 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

Policy Gradients

What is a problem with Q-learning?

The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

Policy Gradients

What is a problem with Q-learning?

The Q-function can be very complicated!

Example: a robot grasping an object has a very high-dimensional state => hard to learn exact value of every (state, action) pair

But the policy can be much simpler: just close your hand

Can we learn a policy directly, e.g. finding the best policy from a collection of policies?

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

We want to find the optimal policy $\theta^* = \arg \max_{\theta} J(\theta)$

How can we do this?

Policy Gradients

Formally, let's define a class of parametrized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$

For each policy, define its value:

$$J(\theta) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

We want to find the optimal policy $\theta^* = \arg \max_{\theta} J(\theta)$

How can we do this?

Gradient ascent on policy parameters!

Aside: Score Function Gradient Estimator

Consider an expectation $E_{x \sim p(x | \theta)}[f(x)]$. Want to compute gradient wrt θ

$$\begin{aligned}\nabla_\theta E_x[f(x)] &= \nabla_\theta \int dx \, p(x | \theta) f(x) \\ &= \int dx \, \nabla_\theta p(x | \theta) f(x) \\ &= \int dx \, p(x | \theta) \frac{\nabla_\theta p(x | \theta)}{p(x | \theta)} f(x) \\ &= \int dx \, p(x | \theta) \nabla_\theta \log p(x | \theta) f(x) \\ &= E_x[f(x) \nabla_\theta \log p(x | \theta)].\end{aligned}$$

$$\boxed{\nabla_\theta E_x[f(x)] = E_x[f(x) \nabla_\theta \log p(x | \theta)]}$$

REINFORCE algorithm

Mathematically, we can write:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau)p(\tau; \theta)d\tau \end{aligned}$$

Where $r(\tau)$ is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \dots)$

REINFORCE algorithm

Expected reward:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau)p(\tau; \theta)d\tau \end{aligned}$$

REINFORCE algorithm

Expected reward:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau)p(\tau; \theta)d\tau \end{aligned}$$

Now let's differentiate this:

$$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau)\nabla_{\theta} p(\tau; \theta)d\tau$$

REINFORCE algorithm

Expected reward:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau)p(\tau; \theta)d\tau \end{aligned}$$

Now let's differentiate this:

$$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau)\nabla_{\theta} p(\tau; \theta)d\tau$$

Intractable! Gradient of an expectation is problematic when p depends on θ

REINFORCE algorithm

Expected reward:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau)p(\tau; \theta)d\tau \end{aligned}$$

Now let's differentiate this:

$$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau)\nabla_{\theta}p(\tau; \theta)d\tau$$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick:

$$\nabla_{\theta}p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_{\theta}p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta)\nabla_{\theta} \log p(\tau; \theta)$$

REINFORCE algorithm

Expected reward:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)] \\ &= \int_{\tau} r(\tau)p(\tau; \theta)d\tau \end{aligned}$$

Now let's differentiate this:

$$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau)\nabla_{\theta}p(\tau; \theta)d\tau$$

Intractable! Gradient of an expectation is problematic when p depends on θ

However, we can use a nice trick: $\nabla_{\theta}p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_{\theta}p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta)\nabla_{\theta} \log p(\tau; \theta)$
If we inject this back:

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \int_{\tau} (r(\tau)\nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta)d\tau \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau)\nabla_{\theta} \log p(\tau; \theta)] \end{aligned}$$

Can estimate with
Monte Carlo sampling

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t)$

Thus: $\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_\theta(a_t | s_t)$

REINFORCE algorithm

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$

Thus: $\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)$

And when differentiating: $\nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Doesn't depend on
transition probabilities!

REINFORCE algorithm

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau \\ &= \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_{\theta} \log p(\tau; \theta)]\end{aligned}$$

Can we compute those quantities without knowing the transition probabilities?

We have: $p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_{\theta}(a_t | s_t)$

Thus: $\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_{\theta}(a_t | s_t)$

And when differentiating: $\nabla_{\theta} \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Doesn't depend on
transition probabilities!

Therefore when sampling a trajectory τ , we can estimate $J(\theta)$ with

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Intuition

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Interpretation:

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

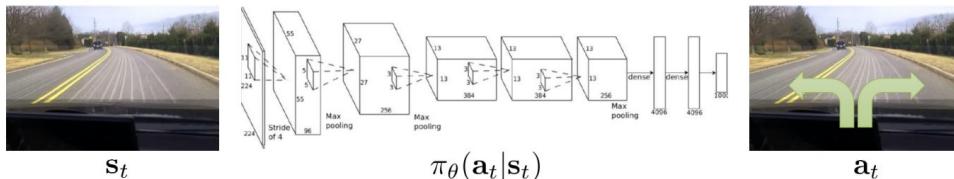
Intuition

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

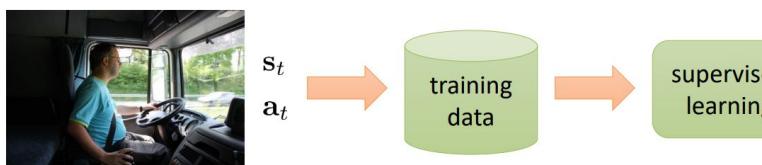
Interpretation:

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

- Quite similar to supervised learning



$$\text{maximum likelihood: } \nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i, s_t^i) \right)$$



$$\text{policy gradient: } \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i, s_t^i) \right) r(\tau^i)$$

Slide credit: Fei fei Li, Justin Johnson, Serena Yeung, Sergey Levine

Intuition

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Interpretation:

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

Intuition

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Interpretation:

- If $r(\tau)$ is high, push up the probabilities of the actions seen
- If $r(\tau)$ is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. **But in expectation, it averages out!**

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Variance reduction

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

Variance reduction

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Variance reduction

Gradient estimator: $\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \boxed{\gamma^{t'-t}} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Variance reduction: Baseline

Problem: The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - \boxed{b(s_t)} \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Variance reduction: Baseline

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

- But.. are we allowed to do that ?
- Subtracting a baseline is unbiased in expectation
 - In expectation, the gradient estimator remains unchanged

How to choose the baseline?

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

How to choose the baseline?

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”

REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

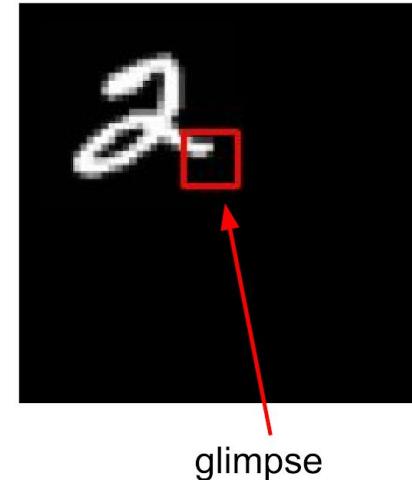
Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

State: Glimpses seen so far

Action: (x,y) coordinates (center of glimpse) of where to look next in image

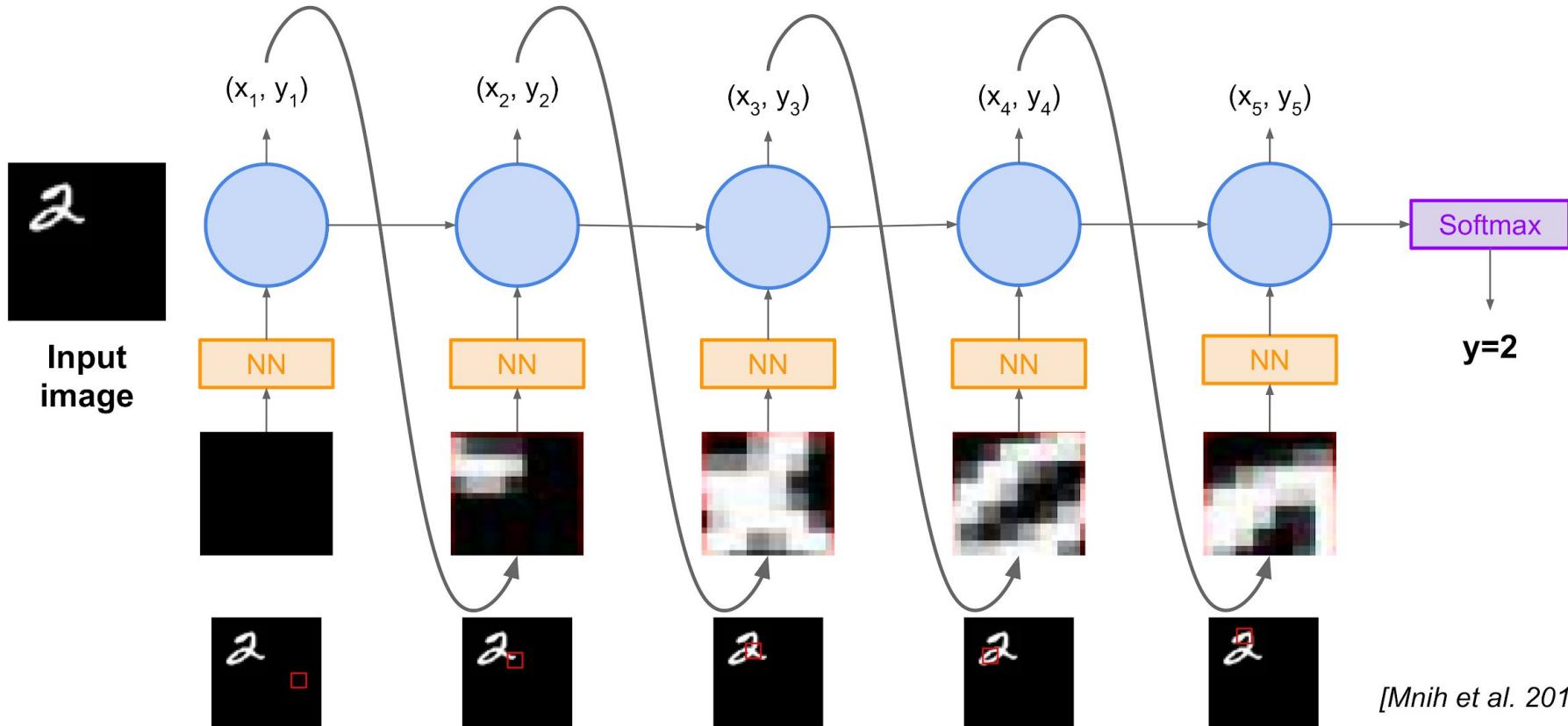
Reward: 1 at the final timestep if image correctly classified, 0 otherwise



Glimpsing is a non-differentiable operation => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

[Mnih et al. 2014]

REINFORCE in action: Recurrent Attention Model (RAM)



[Mnih et al. 2014]

Outline: Policy-based RL Methods

- Introduction
- REINFORCE
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)
- Advanced policy gradient methods
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

How to choose the baseline?

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Option 1: Instead of moving average of cumulative rewards, use a value function estimate $V_{\phi}(s_t) \approx V^{\pi_{\theta}}(s_t)$

$$\nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - V_{\phi}(s_t) \right) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

where

$$V_{\phi}(s_t) \approx V^{\pi_{\theta}}(s_t) = \mathbb{E}_{a_t \sim \pi_{\theta}(a|s_t)} \left[\sum_{t' \geq t} \gamma^{t'-t} r_{t'} | s_t \right]$$

← minimize L2 error
(with SGD)

Actor-Critic Algorithm

Initialize policy parameters θ , critic parameters ϕ

For iteration = 1, 2 ... **do**

 Sample m trajectories under the current policy

$$\Delta\theta \leftarrow 0$$

For i = 1, ..., m **do**

For t = 1, ..., T **do**

$$A_t^i = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t^i \Delta\theta \log \pi_\theta(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \theta + \alpha \Delta\theta$$

$$\phi \leftarrow \phi - \beta \Delta\phi$$

Outline: Policy-based RL Methods

- Introduction
- REINFORCE
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)
- Advanced policy gradient methods
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function! (Option 2: Use both!)

Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function! (Option 2: Use both!)

Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Advantage Actor-Critic algorithm (A2C)

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\pi_\theta} \left[\sum_{t' \geq t} \gamma^{t'-t} r(s_{t'}, a_{t'}) | s_t, a_t \right] \quad \text{Total reward from taking } a_t \text{ in } s_t$$

$$V^\pi(s_t) = E_{a_t \sim \pi_\theta(a_t | s_t)} [Q^\pi(s_t, a_t)] \quad \text{Total reward from } s_t$$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \quad \text{How much better } a_t \text{ is}$$

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

For further simplification, we use the following properties:

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(s_{t+1} | s_t, a_t)} [V^\pi(s_{t+1})] \approx r(s_t, a_t) + \gamma V^\pi(s_{t+1})$$

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

We can learn $V^\pi(s)$ by minimizing $\mathcal{L}(\phi) = ||V_\phi(s_t) - y||^2$ where $y = r(s_t, a_t) + \gamma V_\phi(s_{t+1})$

Advantage Actor-Critic algorithm (A2C)

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

- Advantage estimate $A^{\pi}(s_t, a_t) \approx r(s_t, a_t) + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$
- It is also common to take n-step rewards

$$A^{\pi}(s_t, a_t) \approx \sum_{t'=t}^{t+n-1} \gamma^{t'-t} r(s_{t'}, a_{t'}) + \gamma^n V^{\pi}(s_{t+n}) - V^{\pi}(s_t)$$

- Variance of the estimator increases with n
- Can be tuned as a hyperparameter

Asynchronous Advantage Actor-Critic algorithm (A3C)

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

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$

// Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

 Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

 Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

$t_{start} = t$

 Get state s_t

repeat

 Perform a_t according to policy $\pi(a_t|s_t; \theta')$

 Receive reward r_t and new state s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

until terminal s_t **or** $t - t_{start} == t_{max}$

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

for $i \in \{t - 1, \dots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$

 Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$

 Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

end for

 Perform **asynchronous update** of θ using $d\theta$ and of θ_v using $d\theta_v$.

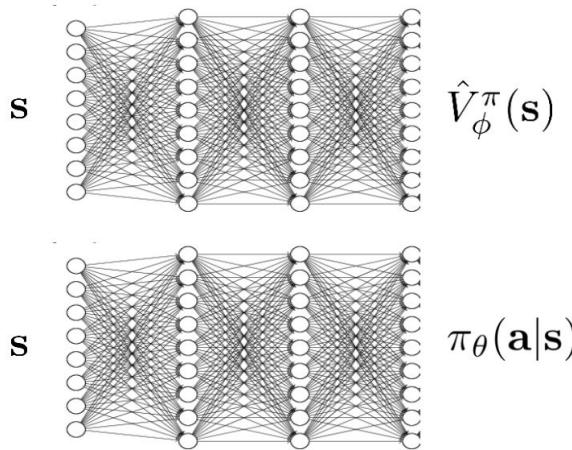
until $T > T_{max}$

Network Design

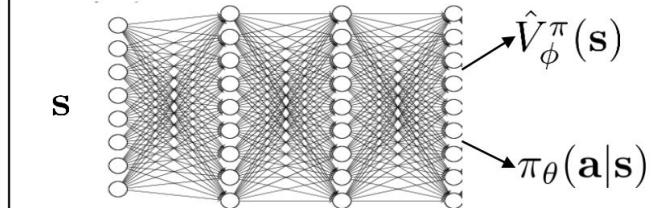
+ simple & stable

- no shared features between actor & critic

two network design



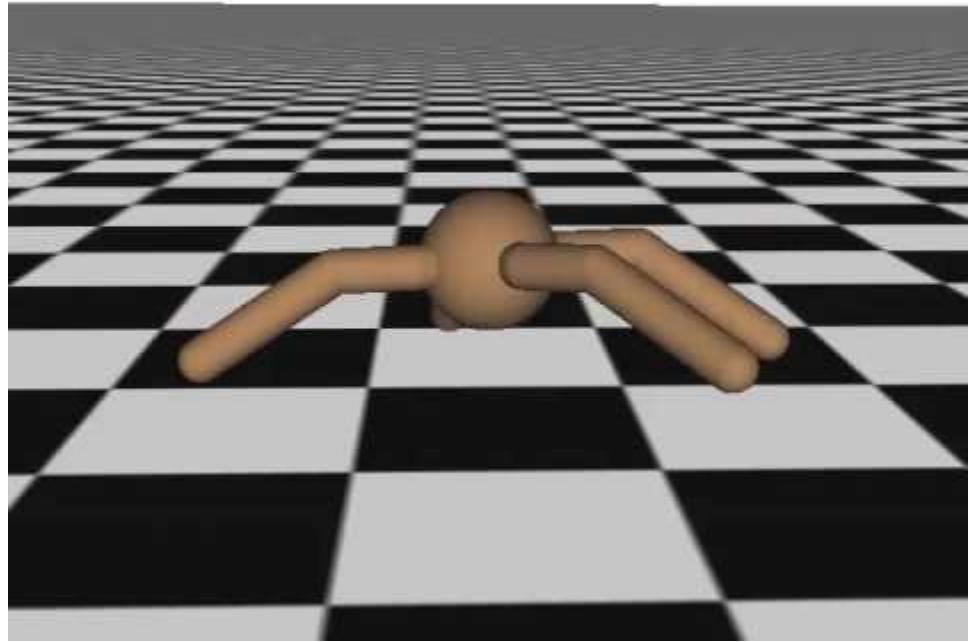
shared network design



Actor-critic examples

- Batch-mode actor-critic
- Blends Monte Carlo and function approximator estimators (GAE)

Iteration 80



<https://www.youtube.com/watch?v=SHLuf2ZBQSw>

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Slide credit: Sergey Levine

Actor-critic examples

- Online actor-critic, parallelized batch
- N-step returns with $N = 4$
- Single network for actor and critic



<https://youtu.be/nMR5mjCFZCw>

Quiz (True/False)

1. Policy gradient methods generally converges more slowly than value-based methods.
2. Adding a baseline term in the REINFORCE algorithm can reduce the bias of the gradient estimator.
3. Adding a baseline term in the REINFORCE algorithm can reduce the variance of the gradient estimator.
4. A2C has lower variance compared to REINFORCE.

Submit your answer here: <https://bit.ly/2HVAXff>

Outline: Policy-based RL Methods

- Introduction
- REINFORCE
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)
- Advanced policy gradient methods
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)

Relative performance of two policies

In a policy optimization algorithm, we want an update step that

- Uses rollouts collected from the most recent policy as efficiently as possible.
- Takes steps that respect **distance in policy space** as opposed to distance in parameter space.

To figure out the right update rule, we need to exploit relationships between the performance of two policies.

Relative policy performance identity: for any policies π, π'

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right]$$

Relative performance of two policies

Proof:

$$\begin{aligned} J(\pi') - J(\pi) &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right] \\ &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)) \right] \\ &= J(\pi') + \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^{t+1} V^\pi(s_{t+1}) - \sum_{t=0}^{\infty} \gamma^t V^\pi(s_t) \right] \\ &= J(\pi') + \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=1}^{\infty} \gamma^t V^\pi(s_{\textcolor{red}{t}}) - \sum_{t=0}^{\infty} \gamma^t V^\pi(s_t) \right] \\ &= J(\pi') - \mathbb{E}_{\tau \sim \pi'} [V^\pi(s_0)] \\ &= J(\pi') - J(\pi) \end{aligned}$$

Relative performance of two policies

Can we use this for policy improvement, where π' represents the new policy and π represents the old one?

$$\begin{aligned}\max_{\pi'} J(\pi') &= \max_{\pi'} J(\pi') - J(\pi) \\ &= \max_{\pi'} \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right]\end{aligned}$$

This is suggestive but not useful yet.

Nice feature of this optimization problem: defines the performance of π' in terms of the advantages of π !

But problematic feature: still requires trajectories sampled from π'

Relative performance of two policies

In terms of the **discounted future state distribution** d^π , defined by

$$d^\pi(s) = (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \pi),$$

we can rewrite the relative policy performance identity:

$$\begin{aligned} J(\pi') - J(\pi) &= \mathbb{E}_{\tau \sim \pi'} \left[\sum_{t=0}^{\infty} \gamma^t A^\pi(s_t, a_t) \right] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{\substack{s \sim d^{\pi'} \\ a \sim \pi'}} [A^\pi(s, a)] \\ &= \frac{1}{1 - \gamma} \mathbb{E}_{\substack{s \sim d^{\pi'} \\ a \sim \pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \end{aligned}$$

... almost there! The only problem is $s \sim d^{\pi'}$

Relative performance of two policies

what if we just said $d^{\pi'} \approx d^\pi$ and did not worry about it?

$$\begin{aligned} J(\pi') - J(\pi) &\approx \frac{1}{1 - \gamma} \mathbb{E}_{\substack{s \sim d^\pi \\ a \sim \pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \\ &\doteq \mathcal{L}_\pi(\pi') \end{aligned}$$

Turns out: this approximation is pretty good when π' and π are close! But why, and how close do they have to be?

Relative performance bounds:

$$|J(\pi') - (J(\pi) + \mathcal{L}_\pi(\pi'))| \leq C \sqrt{\mathbb{E}_{s \sim d^\pi} [D_{KL}(\pi' || \pi)[s]]}$$

If policies are close in KL-divergence — the approximation is good!

Relative performance of two policies

What did we learn from making that approximation?

$$\begin{aligned} J(\pi') - J(\pi) \approx \mathcal{L}_\pi(\pi') &= \frac{1}{1-\gamma} \mathbb{E}_{\substack{s \sim d^\pi \\ a \sim \pi}} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^\pi(s, a) \right] \\ &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)} A^\pi(s_t, a_t) \right] \end{aligned}$$

- This is something that we can optimize using trajectories sampled from the old policy π !
- Similar to using importance sampling, but because weights only depend on current timestep (and not preceding history), they don't vanish or explode.

Something else cool—the approximation matches $J(\pi_\theta) - J(\pi_{\theta_k})$ to first order in policy parameters! That is, $\nabla_\theta \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$ is equal to policy gradient:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{\theta_k}(\theta)|_{\theta_k} &= \mathbb{E}_{\tau \sim \pi_{\theta_k}} \left[\sum_{t=0}^{\infty} \gamma^t \frac{\nabla_\theta \pi_\theta(a_t|s_t)|_{\theta_k}}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta_k}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} A^{\pi_{\theta_k}}(s_t, a_t) \right] \end{aligned}$$

Natural policy gradients

So we have this nice optimization problem:

$$\begin{aligned}\pi_{k+1} &= \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \\ \text{s.t. } \bar{D}_{KL}(\pi' || \pi_k) &\leq \delta\end{aligned}$$

but how do we solve it? Solution: approximately!

$$\begin{aligned}\mathcal{L}_{\theta_k}(\theta) &\approx \mathcal{L}_{\theta_k}(\theta_k) + g^T (\theta - \theta_k) & g &\doteq \nabla_{\theta} \mathcal{L}_{\theta_k}(\theta) |_{\theta_k} \\ \bar{D}_{KL}(\theta || \theta_k) &\approx \frac{1}{2} (\theta - \theta_k)^T H (\theta - \theta_k) & H &\doteq \nabla_{\theta}^2 \bar{D}_{KL}(\theta || \theta_k) |_{\theta_k}\end{aligned}$$

Note: zeroth and first-order terms for \bar{D}_{KL} are zero at θ_k .

$$\begin{aligned}\theta_{k+1} &= \arg \max_{\theta} g^T (\theta - \theta_k) \\ \text{s.t. } \frac{1}{2} (\theta - \theta_k)^T H (\theta - \theta_k) &\leq \delta\end{aligned}$$

Solution to approximate problem:

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

Natural policy gradients

- Recall that $\nabla_{\theta} \mathcal{L}_{\theta_k}(\theta)|_{\theta_k}$ is equal to the policy gradient—so this update gives a policy gradient algorithm where we pre-multiply by H^{-1} .
- The KL-divergence Hessian H is equal to a special matrix called the **Fisher information matrix**, which comes up in a few other places:

$$H = \underset{s, a \sim \theta^k}{\mathbb{E}} \left[\nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k} \nabla_{\theta} \log \pi_{\theta}(a|s)|_{\theta_k}^T \right]$$

- The NPG direction $H^{-1}g$ is **covariant**—that is, **it points in the same direction regardless of the parametrization used to compute it**.

Natural policy gradients

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **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

Outline: Policy-based RL Methods

- Introduction
- REINFORCE
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)
- Advanced policy gradient methods
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)

Trust Region Policy Optimization (TRPO)

Small problems with NPG update:

- Might not be robust to trust region size δ ; at some iterations δ may be too large and performance can degrade.
- Because of quadratic approximation, KL-divergence constraint may be violated.

Solution:

- Require improvement in surrogate (make sure that $\mathcal{L}_{\theta_k}(\theta_{k+1}) \geq 0$).
- Enforce KL-constraint.

How? Backtracking line search with exponential decay (decay coeff $\alpha \in (0, 1)$, budget L)

Trust Region Policy Optimization (TRPO)

TRPO: Backtracking line search based on the Natural Policy Gradient:

Recall: Natural Policy
Gradient update

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

Algorithm 2 Line Search for TRPO

Compute proposed policy step $\Delta_k = \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$

for $j = 0, 1, 2, \dots, L$ **do**

 Compute proposed update $\theta = \theta_k + \alpha^j \Delta_k$

if $\mathcal{L}_{\theta_k}(\theta) \geq 0$ and $\bar{D}_{KL}(\theta || \theta_k) \leq \delta$ **then**

 accept the update and set $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$

 break

end if

end for

Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization: NPG + line search + **conjugate gradient (CG)**
(CG can compute $H^{-1}g$ without inverting H).

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0

for $k = 0, 1, 2, \dots$ **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

Outline: Policy-based RL Methods

- Introduction
- REINFORCE
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)
- Advanced policy gradient methods
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a family of methods that approximately enforce KL constraint **without computing natural gradients**. Two variants:

- Adaptive KL penalty:
 - Policy update solves the unconstrained optimization problem:
$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta || \theta_k)$$
 - Penalty coefficient β_k changes between iterations to approximately enforce KL-divergence constraint.
- Clipped objective: let $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$. Then
 - New objective function:
$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathbb{E}_{\tau \sim \pi_k} \left[\sum_{t=0}^T \left[\min(r_t(\theta) \hat{A}_t^{\pi_k}, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t^{\pi_k}) \right] \right]$$
where ϵ is a hyperparameter
 - Policy update is $\theta_{k+1} = \arg \max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$

Proximal Policy Optimization with adaptive KL Penalty

Algorithm 4 PPO with Adaptive KL Penalty

Input: initial policy parameters θ_0 , initial KL penalty β_0 , target KL-divergence δ

for $k = 0, 1, 2, \dots$ **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

- Initial KL penalty not that important—it adapts quickly
- Some iterations may violate KL constraint, but most do not

Proximal Policy Optimization with clipped Objective

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0 , clipping threshold ϵ

for $k = 0, 1, 2, \dots$ **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

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

end for

- Clipping prevents policy from having incentive to go far away from θ_{k+1}
- **Clipping seems to work at least as well as PPO with KL penalty, but is simpler to implement**

Proximal Policy Optimization with clipped Objective

But how does clipping keep policy close? By **making objective as pessimistic as possible about performance far away from θ_k** :

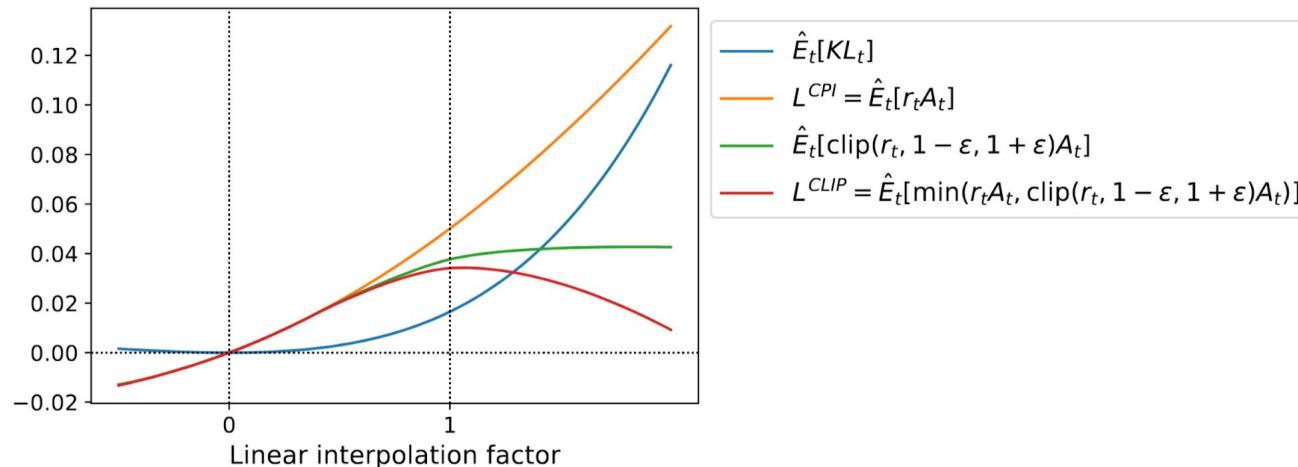


Figure: Various objectives as a function of interpolation factor α between θ_{k+1} and θ_k after one update of PPO-Clip⁹

Summary: NPG → TRPO → PPO

- Natural policy gradient (NPG)
 - Policy gradient + natural gradient

$$\begin{aligned}\pi_{k+1} = \arg \max_{\pi'} \mathcal{L}_{\pi_k}(\pi') \\ \text{s.t. } \bar{D}_{KL}(\pi' || \pi_k) \leq \delta\end{aligned}\quad \longrightarrow \quad \theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$$

- Trust region policy optimization (TRPO)
 - NPG + conjugate gradient + line search
- Proximal policy optimization (PPO)
 - Penalty version (instead of constrained optimization)
 - Clipping without explicit KL calculation

Empirical performance of PPO

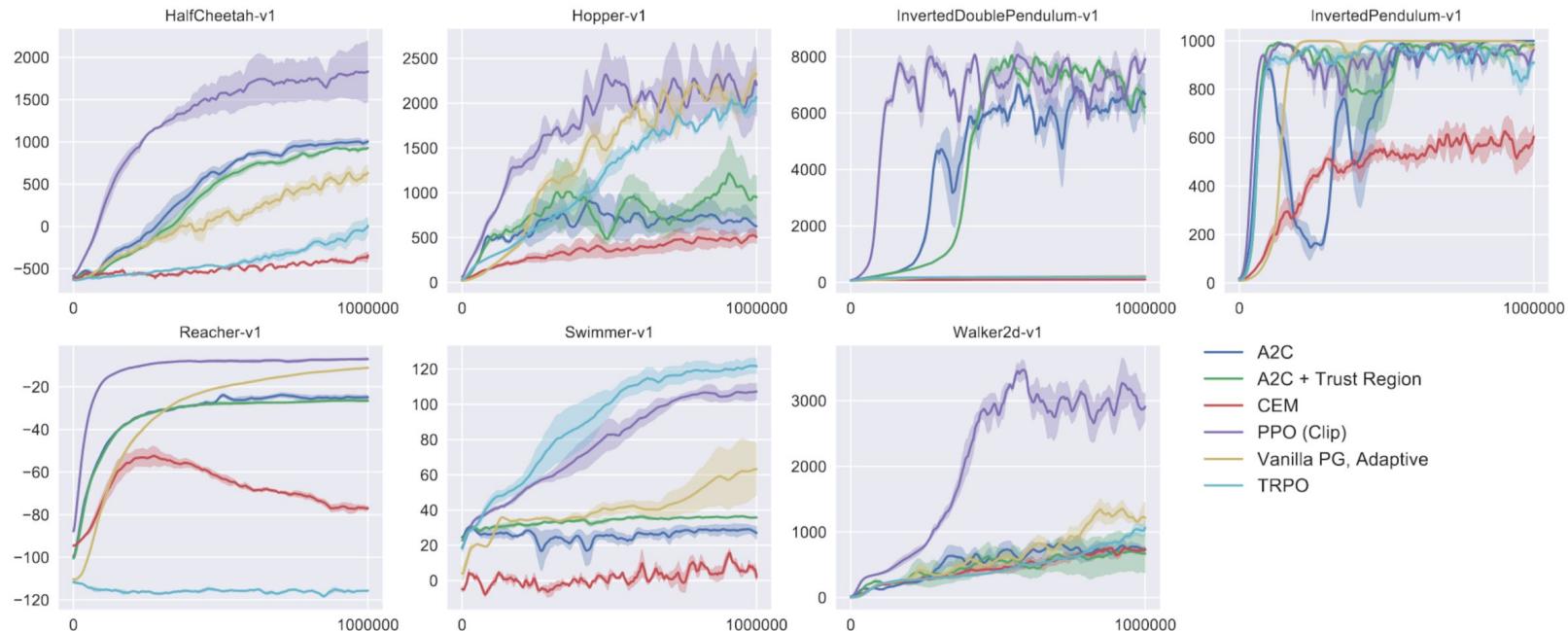


Figure: Performance comparison between PPO with clipped objective and various other deep RL methods on a slate of MuJoCo tasks.¹⁰

Learning continuous control with PPO

Episode 40,000



<https://youtu.be/Tg0Dyu3iQek>

Episode 8,000



<https://youtu.be/irkXnpZP89s>

SOTA application of PPO: Montezuma's revenge

PPO + contingency-aware exploration



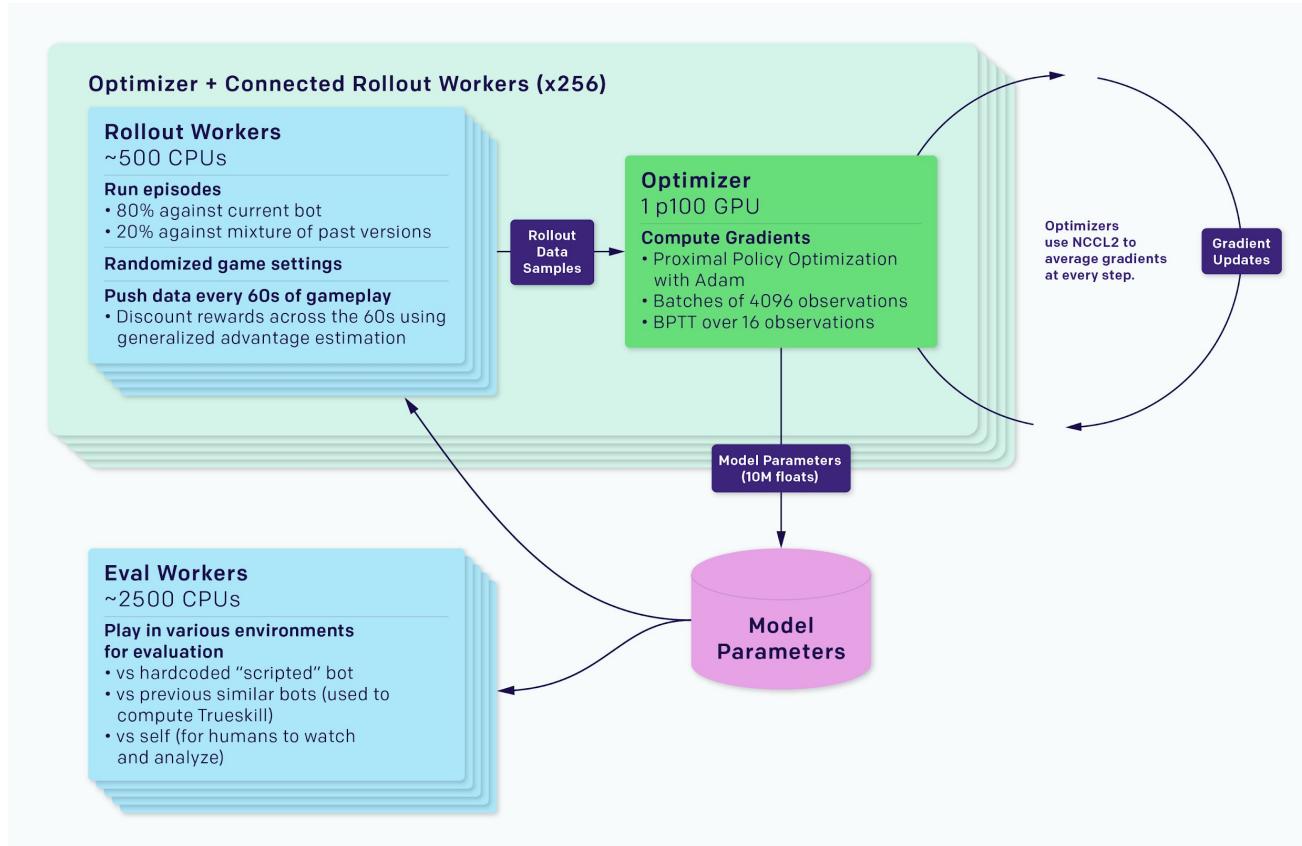
SOTA application of PPO: OpenAI Five

- **Self-play:**
 - Plays 180 years worth of games against itself every day.
- **Scaling up:**
 - Trains using a scaled-up version of [PPO](#) running on 256 GPUs and 128,000 CPU cores.
- **LSTM policy for partial observability and long-term dependencies:**
 - Using a separate [LSTM](#) for each hero and no human data, it learns recognizable strategies.
 - Dota games run at 30 frames per second for an average of 45 minutes, resulting in 80,000 ticks per game. OpenAI Five observes every fourth frame, yielding 20,000 moves.



<https://openai.com/blog/openai-five/>

SOTA application of PPO: OpenAI Five



Quiz (True/False)

1. DQN is an on-policy method that's sample efficient.
2. TRPO is a variant of natural policy gradient methods.
3. PPO is an actor-critic method.
4. Training PPO requires a large replay buffer.

Submit your answer here: <https://bit.ly/2FKgubO>

Summary: Policy-based RL Methods

- Introduction
- Policy gradient methods:
 - REINFORCE
 - Natural policy gradient (NPG)
 - Trust region policy optimization (TRPO)
 - Proximal policy optimization (PPO)
- Actor-Critic methods
 - Advantage Actor Critic (A2C & A3C)

