Policy Based Reinforcement Learning: A Tutorial

Peng Sun pythonsun@tencent.com

Vision Group, Tencent AI Lab

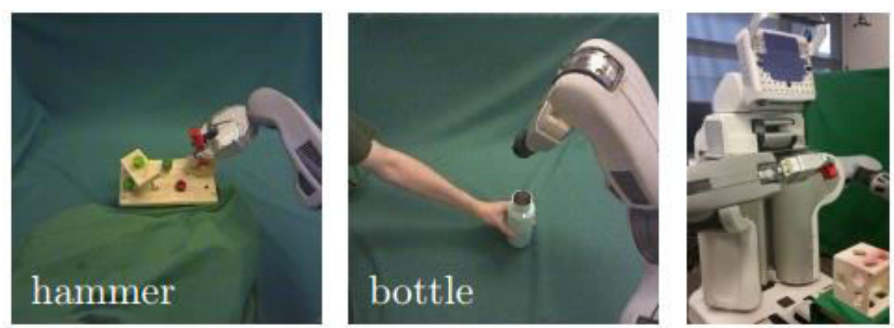
(Deep) Reinforcement Learning

Viewing (raw) states, making serial decisions

Playing Video Game

In-door Robot Navigation

Robot Arm Control



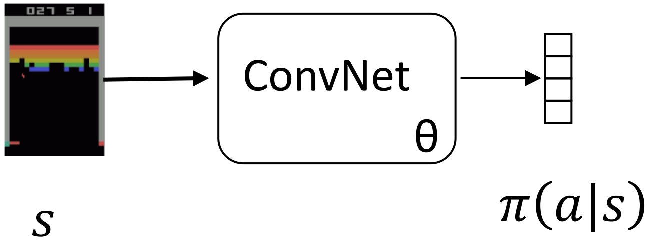
Policy Based (Deep) Reinforcement Learning

Learning the policy directly

A function approximator $\pi(a|s;\theta)$, mapping raw states s to actions a

E.g., the Atari game "breakout"

- Raw states s is 3x84x84 RGB image
- Actions {left, right, fire, no-op}



How It Works: a Quick Overview

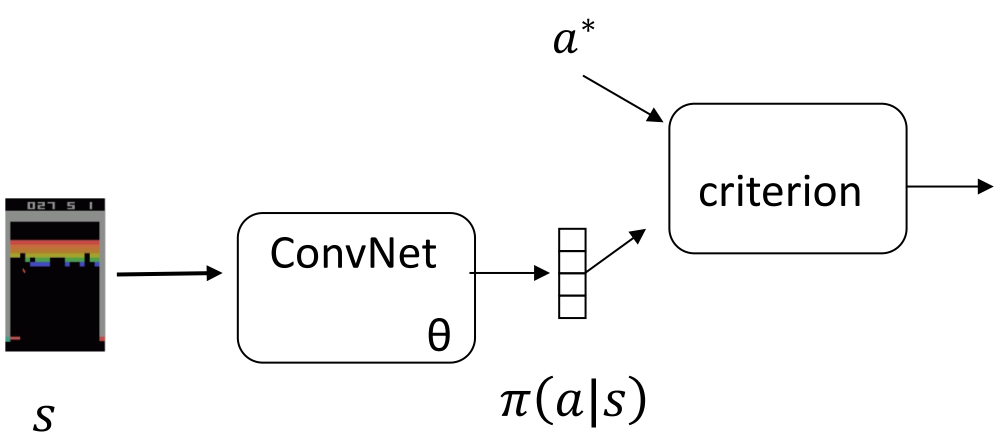
Inputs: image s

Outputs: policy probability $\pi(a|s;\theta)$

Ground Truth: action a^*

Criterion: class log likelihood $\log p(a^*|s;\theta)$

Gradients: $\nabla_\theta \log p(a^*|s;\theta)$



Supervised learning

How It Works: a Quick Overview (Cont.)

Inputs: image s

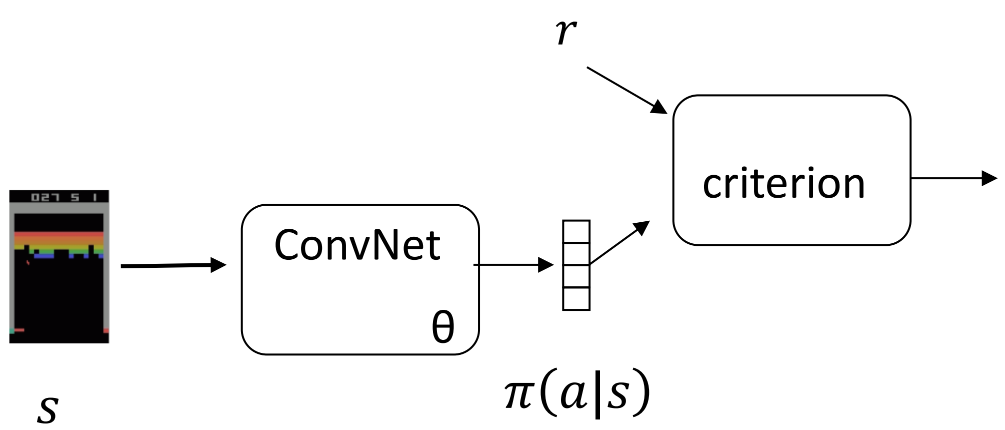
Outputs: policy probability $\pi(a|s;\theta)$

Ground Truth: reward r

Criterion: sample $a \sim \pi(a|s;\theta)$, then

weighted log likelihood $r \log p(a|s;\theta)$

Gradients: $\nabla_\theta\, r \log p(a|s;\theta)$



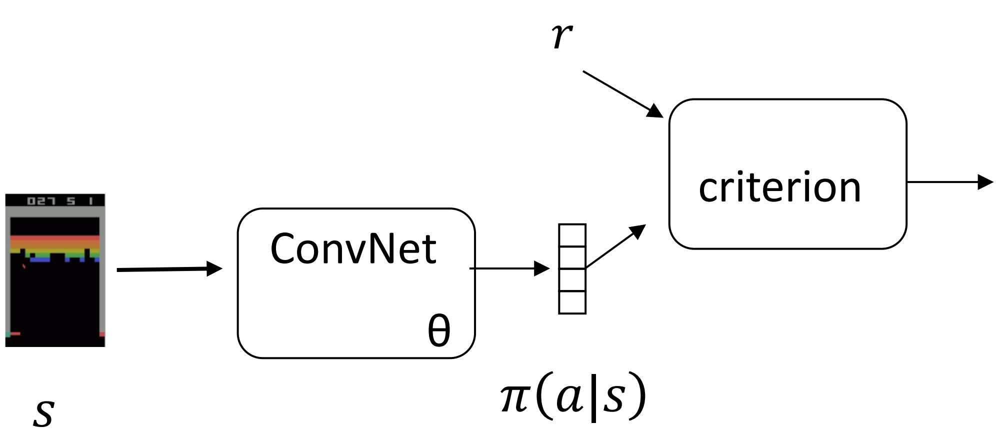
Reinforcement learning

How It Works: a Quick Overview (Cont.)

Gradients: $\nabla_\theta \, r \log p(a|s;\theta)$

- Estimation quality (bias, variance)?
- Delayed reward/credit assignment?

Need formalization

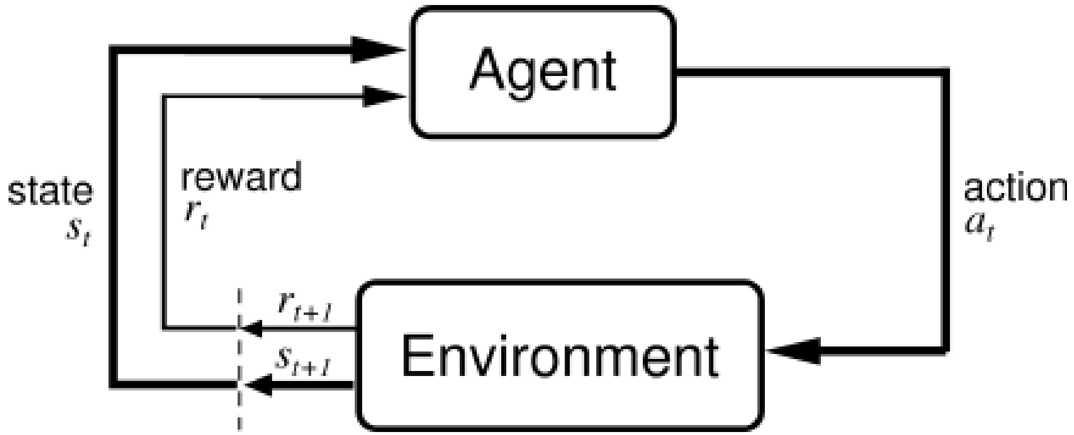


Reinforcement learning

Outline

- Log-Likelihood Objective Optimization
- “Vanilla” Policy Gradient
- Asynchronous Advantage Actor-Critic (A3C)
- Trust Region Policy Optimization (TRPO)
- Guided Policy Search

Agent Environment Interaction



$$\begin{array}{ccccc} & s_{t-1} & s_t & \\ \cdots & a_{t-1} & a_t & \cdots \\ & r_{t-1} & r_t & \end{array}$$

Markovian Process

Assumptions:

current action over current state: $a_t \sim \pi(a_t | s_t)$

current state over previous state, action: $s_t \sim p(a_t | s_{t-1}, a_{t-1})$

current reward over current state, action: $r_t = r(s_t, a_t)$

The trajectory

$$
\begin{tikzcd}
 & s_{t-1} \arrow[r] \arrow[d] \arrow[dd, bend right] & s_t \arrow[d] \arrow[dd, bend right] & \\
\cdots & a_{t-1} \arrow[d] \arrow[ur] & a_t \arrow[d] & \cdots \\
 & r_{t-1} & r_t &
\end{tikzcd}
$$

Probabilistic Graphic Model

Arrows describe the dependency

Easy to write down how to decompose the joint distribution

Random variables: $x = (x_1, x_2, x_3, x_4)$

The joint distribution:

$$p(x_1, x_2, x_3, x_4) = p(x_1)\, p(x_2|x_1) p(x_3|x_2) p(x_4|x_2, x_3)$$

$$\begin{tikzcd}
x_1 \arrow[r] & x_2 \arrow[r] \arrow[d] & x_4 \\
 & x_3 \arrow[ur] &
\end{tikzcd}$$

Learning

What to approximate?

- Function approximator of the policy $\pi(a|s;\theta)$

What to learn?

- Parameters θ
- Altering $\pi(a|s;\theta)$ changes the whole trajectory…

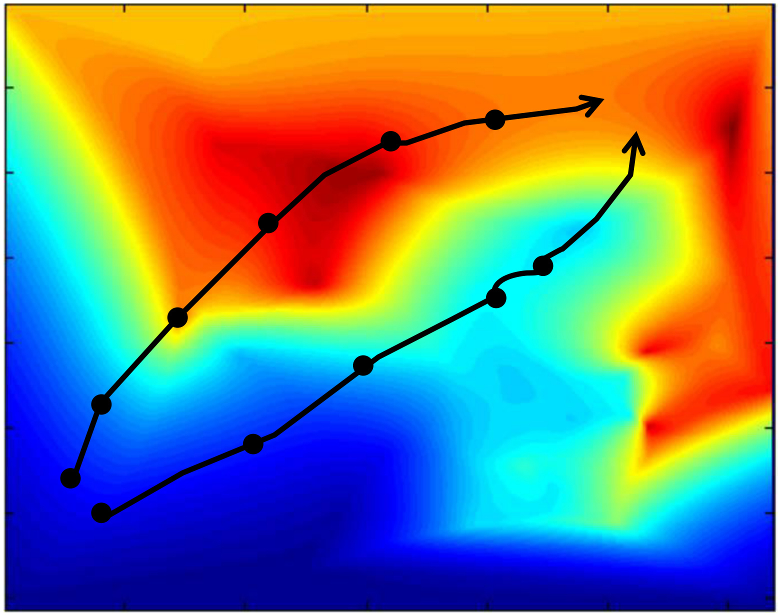
$$
\begin{tikzcd}
 & s_{t-1} \arrow[r] \arrow[d] \arrow[dd, bend right] & s_t \arrow[d] \arrow[dd, bend right] & \\
\cdots & a_{t-1} \arrow[d] \arrow[ur] & a_t \arrow[d] & \cdots \\
 & r_{t-1} & r_t &
\end{tikzcd}
$$

$$s_t \sim p(a_t|s_{t-1}, a_{t-1})$$

$$\boxed{a_t \sim \pi(a_t|s_t)}$$

$$r_t = r(s_t, a_t)$$

Learning Objective



$x = (s, a)$ plane

Define the objective

$$U(\theta) = \mathbb{E}_\tau[R(\tau)]$$

$\tau = (s_1, a_1, \cdots)$ the trajectory

$R(\tau)$ the rewards over τ e.g., the cumulated rewards

The expectation: sums over all possible $s_1, a_1, \cdots$

Derive the gradient

Objective

$$U(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau)$$

Gradient

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

Population mean

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

Sample mean

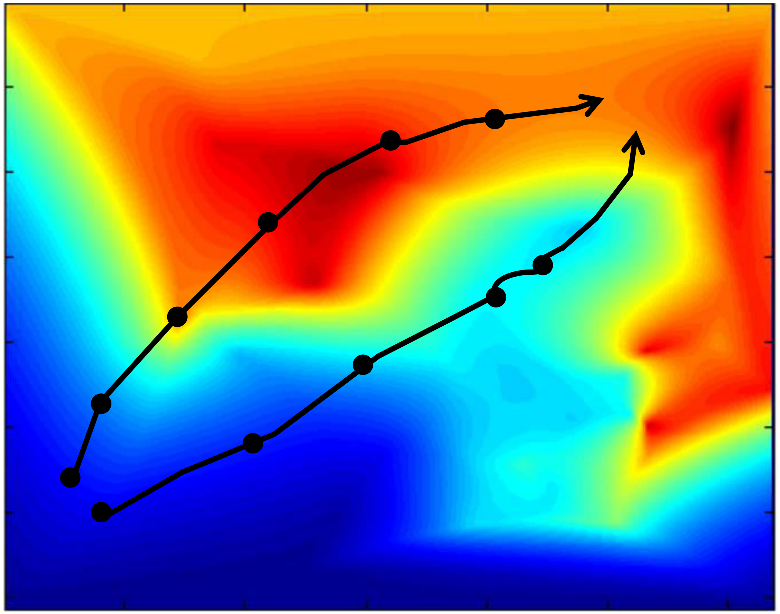
$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)R(\tau^{(i)})$$

Decompose the gradient

$$
\begin{aligned}
\nabla_\theta \log P(\tau^{(i)};\theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}
\end{aligned}
$$

$$
\begin{tikzcd}
 & s_{t-1} \arrow[r] \arrow[d] & s_t \arrow[d] & \\
\cdots & a_{t-1} \arrow[ur] & a_t & \cdots
\end{tikzcd}
$$

Remarks/Propositions



$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)R(\tau^{(i)})$$

- make high reward trajectory more possible; low reward trajectory less possible
- $R(\tau)$ can be non-decomposable, or even non-continuous!!

Application: Image Captioning

Image captioning as a sequential decision making task

- non-differentiable metric CIDER

The trajectory

- State is the hidden cells of LSTM
- Action is what word to choose
- A trajectory corresponds to a sentence

$$\cdots \quad \begin{matrix} s_{t-1} & s_t \\ a_{t-1} & a_t \end{matrix} \quad \cdots$$

Rennie, Steven J., et al. "Self-critical Sequence Training for Image Captioning." (2016).

Outline

- Log-Likelihood Objective Optimization
- “Vanilla” Policy Gradient
- Asynchronous Advantage Actor-Critic (A3C)
- Trust Region Policy Optimization (TRPO)
- Guided Policy Search

Bias and Variance

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)R(\tau^{(i)})$$

- The gradient estimate is unbiased.
- But it’s noisy, i.e., high variance
- Develop techniques to lower variance and make it practical in real word application
  - Baseline
  - Temporal structure

Introduce baseline

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

- If R happens to be non-negative, all trajectories would be always pushed.
- How to push only the “good enough” trajectories?

$$\nabla U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b)$$

- Push good enough trajectory (bigger than b), avoid poor trajectory (less than b)
- Much like why we need a bias term for binary linear classifier
- Reasonable $b = \mathbb{E}\left[R(\tau)\right] \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)})$

William. “REINFORCE." (1992).

Consider temporal structure

Current policy be responsible for all rewards

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b)$$

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=0}^{H-1} R(s_t^{(i)}, u_t^{(i)}) - b\right)$$

Intuitive: current policy be only responsible for future rewards, not the past awards

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) - b(s_k^{(i)})\right)$$

- R term over the future
- Baseline term the expectation over the future

$$b(s_t) = \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \ldots + r_{H-1}\right]$$

Peters & Schaal, IROS 2006 (A survey)

Pseudo code for "Vanilla" Policy Gradient

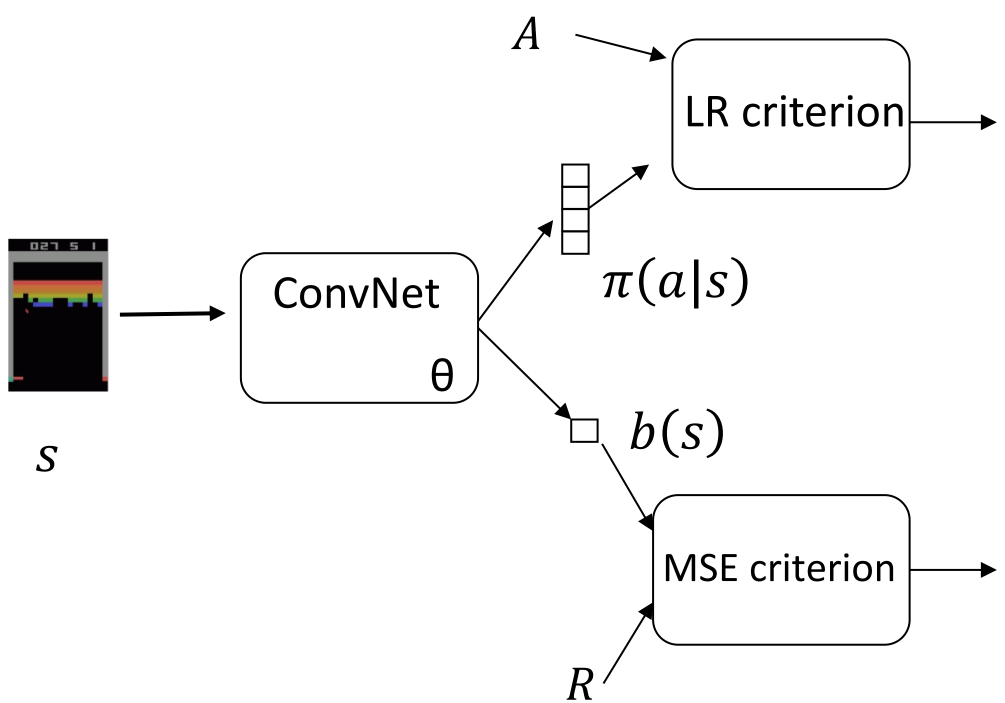
Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration$=1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

Remarks: actually works very well… will see this next…

Neural Network Implementation

The "pseudo ground truth" A and R must be generated on-the-fly.



Outline

- Log-Likelihood Objective Optimization
- “Vanilla” Policy Gradient
- Asynchronous Advantage Actor-Critic (A3C)
- Trust Region Policy Optimization (TRPO)
- Guided Policy Search

Advantage Actor Critic

Almost the same with “vanilla” policy gradient, but different terminology.

- Actor: the policy $\pi(a|s;\theta)$
- Critic: the baseline $V(s)$, criticize how the state is
- Advantage: the minus term

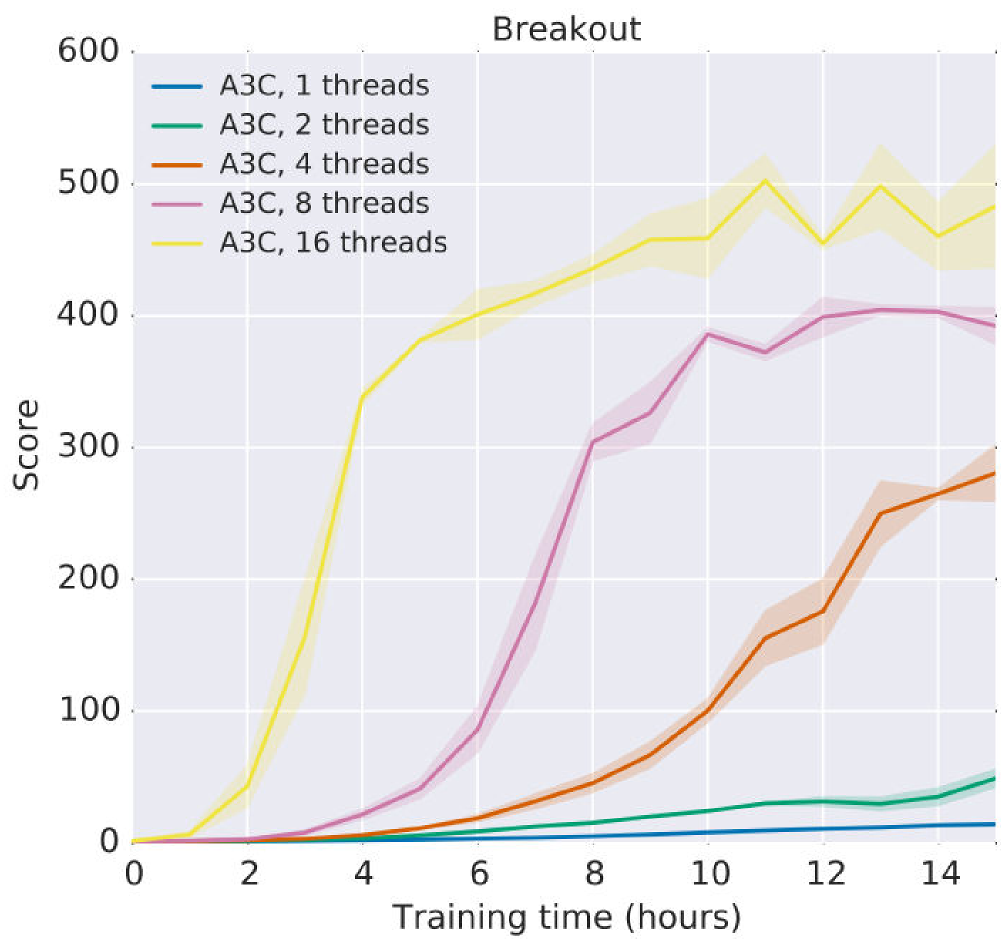
Asynchronous Advantage Actor Critic (A3C)

- Start many agent-environment interaction threads
- Only one nn, whose parameters are shared across threads
- Parameters are updated asynchronously during training
  - easier implementation, as it is lock-free
  - unreasonable in regards of numeric accuracy, but work well in practice.

V. Mnih, A. P. Badia, M. Mirza, et al. “Asynchronous Methods for Deep Reinforcement Learning”. ICML (2016)

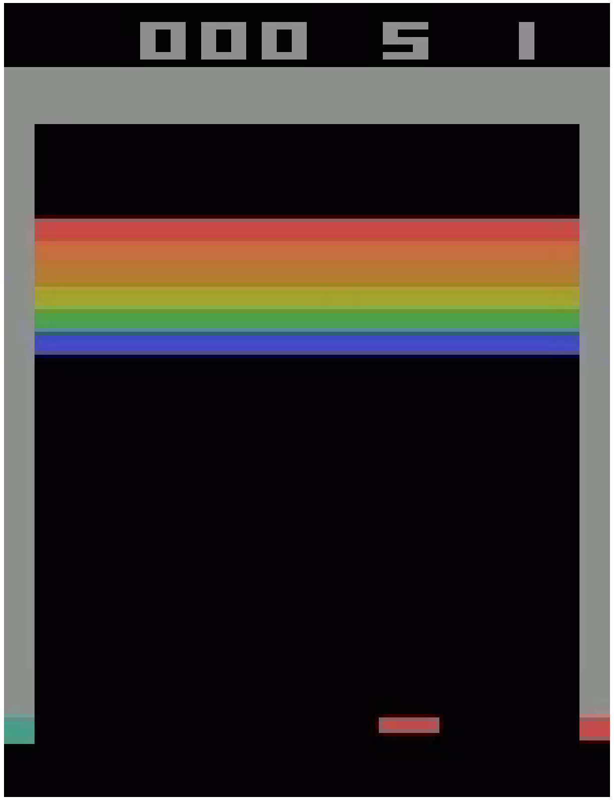
Asynchronous Advantage Actor Critic (A3C): performance

Significant speed-up (figure)



Demo: Atari Breakout

Show movie here



Demo: 3D car racing

https://youtu.be/0xo1Ldx3L5Q

Remark: the same nn and RL algo with the 2D Atari Game!

Demo: ViZDoom

Remarks

Single-thread also works well, although mush slower

- This is not true for other RL method (e.g., Q-learning), which requires high-quality random sampling. Multi threading ensures this.

Strong empirical results

- A3C is uniformly better than other Q-learning based methods
- Could be due to the introduction of V(s) term.
- Didn’t compare to asynchronous dueling-network using similar idea

Anyway, A3C should be an off-the-shelf RL method for problem on hand! Why it was “overlooked” before, though? Hmm…

Outline

- Log-Likelihood Objective Optimization
- "Vanilla" Policy Gradient
- Asynchronous Advantage Actor-Critic (A3C)
- Trust Region Policy Optimization (TRPO)
- Guided Policy Search

Step Size Matters

Too small step size: very slow

Too big step size:

- Supervised Learning: fixed at next step
- Reinforcement Learning: lead to poor state regions; sample poor trajectories; cannot fix…

Trust Region Policy Optimization

During each iteration

- Get trajectories $\{s_n, a_n, r_n\}$ using $\pi(\cdot \mid \cdot\, ; \theta_{old})$
- Maximize

$$L(\theta),$$
$$\text{s.t. } D(\theta_{old}, \theta) < \delta$$

- Update $\theta_{old} \leftarrow \theta$

The sample version, surrogate "local" rewards

$$L(\theta) = \sum_{n=1}^{N} \frac{\pi(a_n|s_n;\theta\ )}{\pi(a_n|s_n;\theta_{old})} \widehat{A_n}$$

The sample version, average constraint

$$D(\theta_{old}, \theta) = \sum_{n=1}^{N} KL\big(\pi(\cdot\,|s_n;\theta_{old}), \pi(\cdot\,|s_n;\theta)\big)$$

The numeric solver

Objective $L(\theta)$ expanded to first order, get its gradient g

Constraint $D(\theta_{old}, \theta)$

- KL: zero-order and first-order expansion are
- Second order expansion, get Hessian matrix F

Find the updating direction $s = F^{-1}g$

Do line search along s by $\theta \leftarrow \theta + \eta s$, step size η ensuring the constraint on $D(\theta_{old}, \theta)$ and improvement over $L(\theta)$

- Similar to Natural Gradient (NG): updating by $\theta \leftarrow \theta + \lambda F^{-1}g$
- TRPO turns out to outperform NG (although the subtle difference!).
- Careful step size is critical!

S. Kakade and J. Langford. "Approximately optimal approximate reinforcement learning". ICML 2012

Demo

URL here

Outline

- Log-Likelihood Objective Optimization
- “Vanilla” Policy Gradient
- Asynchronous Advantage Actor-Critic (A3C)
- Trust Region Policy Optimization (TRPO)
- Guided Policy Search

Background

Robot control

- States: positions, velocity, joint angles…
- Actions: torques
- Rewards: negative distance to expected positions, angles (by design)

RL training on real environment, instead of simulator. Low “put-through” due to physics, even though the policy gradient algorithm itself can work very fast.

Require data-efficient training!

Data Efficiency v.s. Domain Knowledge

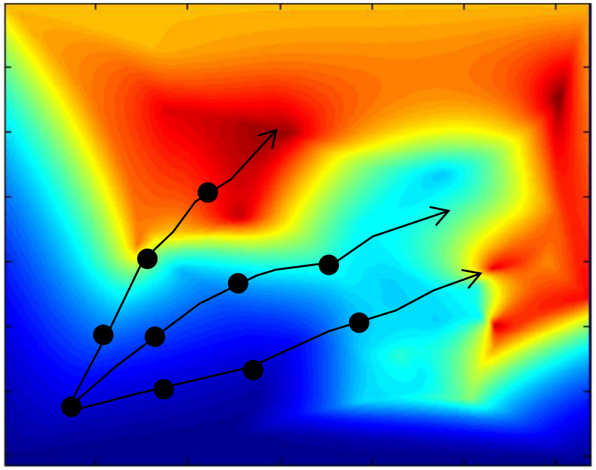
Why PG consumes so many training data?

- Does not model the environment dynamics (model-free).

Explicitly model the environment dynamics

- Can hopefully learn with fewer data
- E.g., parabolic curve fitting

Optimal Control/Trajectory Optimization



Given an initial state, find a good trajectory along which high rewards are collected.

- Simultaneous rollout and local policy $q_i(a_t|s_t)$ learning
- Analytic form! E.g., Dynamics is Gaussian, reward is quadratic, then $q_i(a_t|s_t)$ is time-varying linear Gaussian
- $q_i(a_t|s_t)$ good for current trajectory, but generalizes poorly to other states or trajectories

General Pipeline

Given

- Known environment dynamics $p(a_t|s_{t-1}, a_{t-1})$ or fitted from data
- Known reward function $r_t = r(s_t, a_t)$ by design

When not convergent, iterate over

1. Sample on initial state and do trajectory optimization to get trajectories and $q_1(a|s)$, $q_2(a|s)$, …
2. Supervised Learning: Fit $\pi(a|s;\theta)$ to $q_1(a|s)$, $q_2(a|s)$, … on the points on the trajectories

The local policy is called guided policy, hence the name

Sergey Levine, Pieter Abbeel. Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics. NIPS 2014.

Demo

https://youtu.be/mSzEyKaJTSU

Visual Motor Control

Control robot by viewing image sequence

Methodology: Guided Policy Search with asymmetrical states in the two phases

- Guiding policy $q_i(a|s)$ over "intrinsic" states (position, velocity…)
- Learn policy $\pi(a|s;\theta)$ over "visual" states (pixels from camera)

Sergey Levine*, Chelsea Finn*, Trevor Darrell, Pieter Abbeel. End-to-End Training of Deep Visual motor Policies. JMLR 17, 2016.

Remarks

- A promising framework to exploit environment dynamics
- How to generalize to other domains than robotics? (E.g., playing 3D video game)