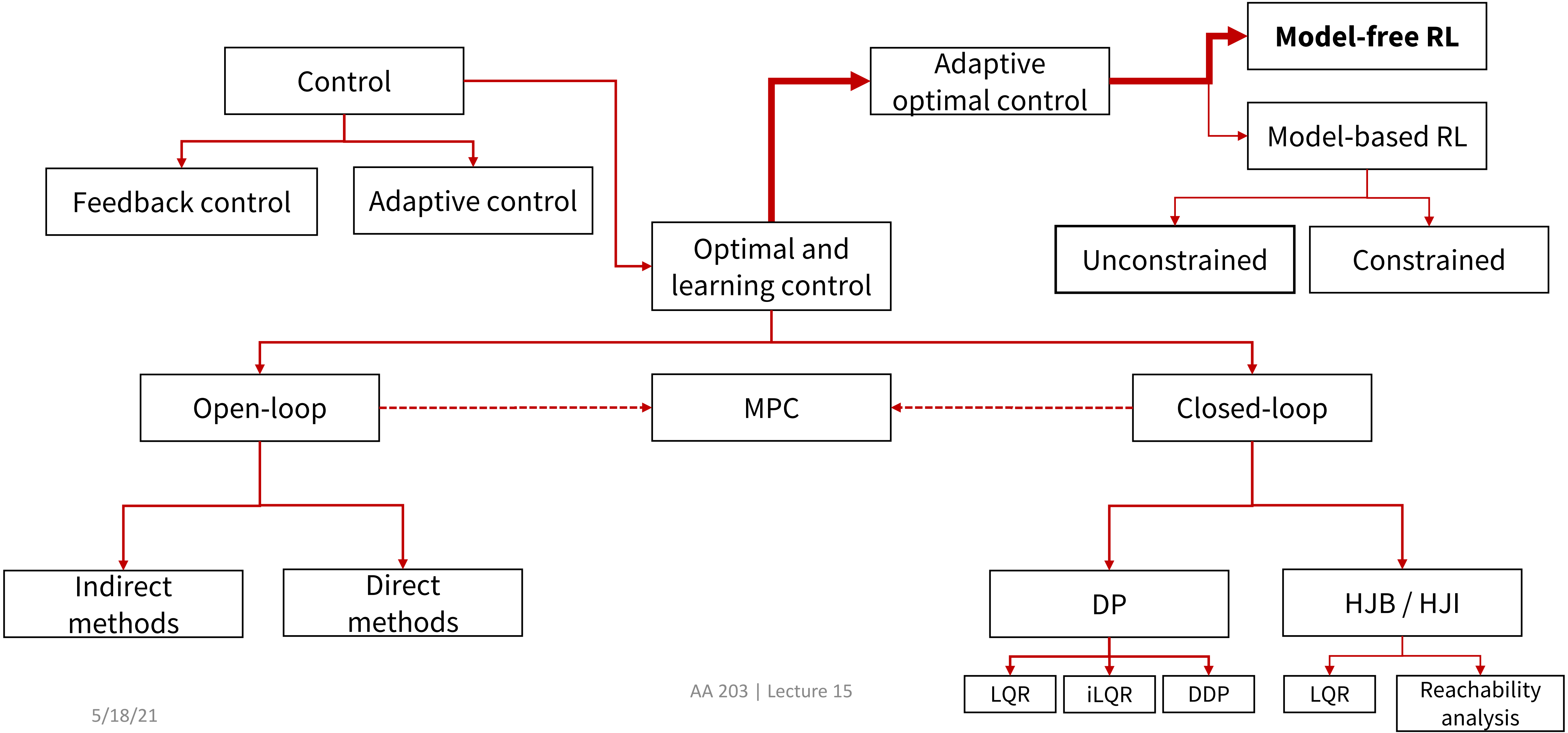
AA203
Optimal and Learning-based Control

Policy gradient and actor-critic

Stanford University

Autonomous Systems Lab

Roadmap


Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Unconstrained
Constrained
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI
LQR
iLQR
DDP
LQR
Reachability analysis

Model-free RL: deep RL and policy gradient

- Review Q-learning
- Policy gradient
- Introduce variance reduction methods for policy gradient estimation
- Brief survey of the modern model-free RL landscape

- Readings:
  - R. Sutton and A. Barto. *Reinforcement Learning: An Introduction,* 2018.

Review: Q-Learning

Policy evaluation for Q^* via

$$\min_{\theta}\left(r_t + \gamma \max_{u} Q_{\theta'}(x_{t+1}, u) - Q_{\theta}(x_t, u_t)\right)^2$$

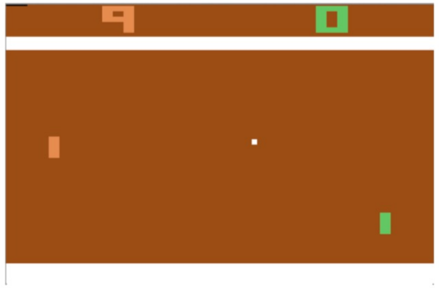
with a greedy policy improvement step, $\pi(x) = \max_{u} Q_{\theta}(x, u)$.

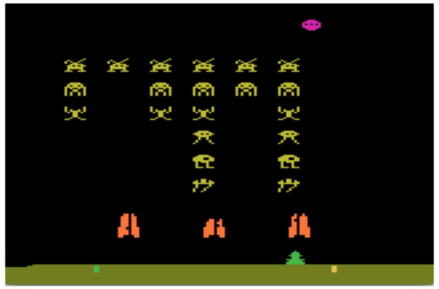
Deep Q-Learning

- Many possible function approximators for Q
  - Linear, nearest neighbors, aggregation
- Recent success: neural networks with loss function

$$\left(r_t + \gamma \max_u Q_{\theta'}(x_{t+1}, u) - Q_\theta(x_t, u_t)\right)^2$$

- Deep Q Network (DQN; Mnih et al. 2013)
  - Experience replay






Model-free, policy based: Policy Gradient

Instead of learning the Q function, learn the policy directly!

Define a class of policies π_θ where θ are the parameters of the policy.

Can we learn the optimal θ from interaction?

Goal: use trajectories to estimate a gradient of policy performance w.r.t parameters θ

Policy Gradient

A particular value of θ induces a distribution of possible trajectories.

Objective function:

$$J(\theta) = E_{\tau \sim p(\tau;\theta)}[r(\tau)]$$

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

where $r(\tau)$ is the total discounted cumulative reward of a trajectory.

Policy Gradient

Gradient of objective w.r.t. parameters:

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

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

$$\nabla_\theta J(\theta) = \int_\tau (r(\tau) \nabla_\theta \log p(\tau;\theta)) p(\tau;\theta)\, d\tau$$

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

Policy Gradient

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

$$\log p(\tau;\theta) = \log\left(\prod_{t\geq 0} T(x_{t+1}|x_t,u_t)\pi_\theta(u_t|x_t)\right)$$

$$= \sum_{t\geq 0} \log T(x_{t+1}|x_t,u_t) + \log \pi_\theta(u_t|x_t)$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(u_t|x_t)$$

We don't need to know the transition model to compute this gradient!

Policy Gradient

If we use π_θ to sample a trajectory, we can approximate the gradient via N Monte Carlo samples:

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

$$\approx \frac{1}{N}\sum_{i=1}^{N}\left(r\left(\tau^{(i)}\right)\sum_{t\geq 0}\nabla_\theta \log \pi_\theta(u_t^{(i)}|x_t^{(i)})\right)$$

Intuition: adjust theta to:

- Boost probability of actions taken if reward is high
- Lower probability of actions taken if reward is low

Learning by trial and error

Policy Gradient Recap

Pros:

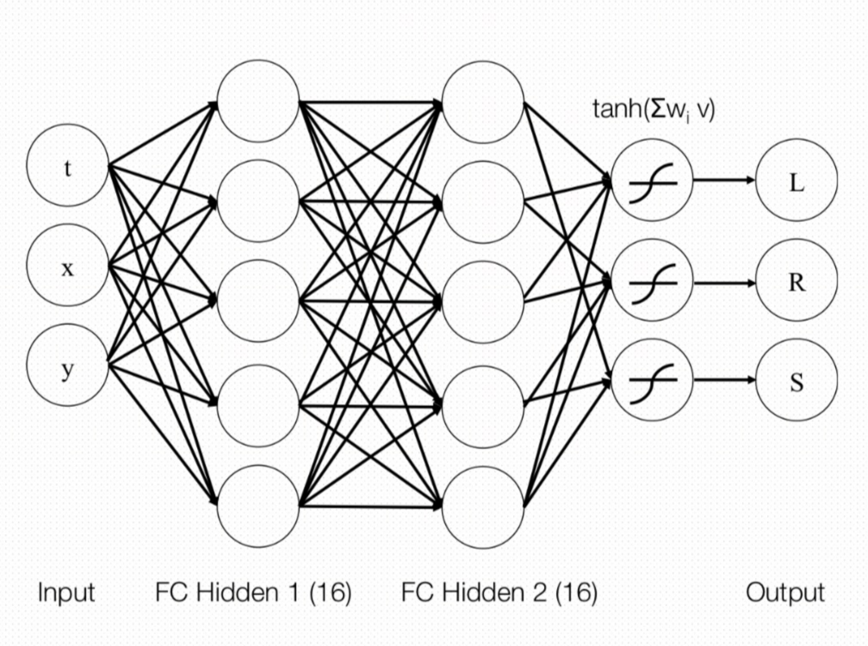
- Learns policy directly – often more stable
- Works for continuous action spaces
- Converges to local maximum of $J(\theta)$

Cons:

- Needs data from current policy to compute gradient – data inefficient
- Gradient estimates can be very noisy

Deep policy gradient

- Parametrize policy as deep neural network
- In practice, very unstable
  - Need to reduce variance of gradient estimator: baselines and actor-critic



Time dependency of policy gradient theorem

- Previous estimator for policy gradient was

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(r\left(\tau^{(i)}\right)\sum_{t\geq 0}\nabla_\theta \log \pi_\theta(u_t^{(i)}|x_t^{(i)})\right)$$

Action u_t, can not change reward r_t for $t' > t$:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t\geq 0}\nabla_\theta \log \pi_\theta(u_t^{(i)}|x_t^{(i)})\sum_{\tau\geq t} r(x_\tau^{(i)}, u_\tau^{(i)})\right)$$

REINFORCE

Loop forever:

Generate episode $x_0, u_0, r_0, x_1, u_1, r_1$ … with π_θ

Loop for all $t = 0, \ldots, N-1$:

$G \leftarrow \sum_{k=t}^{N} r_k$

$\theta \leftarrow \theta + \alpha\, G\, \nabla_\theta \log \pi_\theta(u_t | x_t)$

Adding baselines to policy evaluation

- Monte Carlo policy gradient estimator has **extremely high variance.**
- We want to search for gradient estimators that have lower variance
- Add in **baseline**

$$\tilde{G}_t = G_t - b(x_t)$$
$$J(\theta) = \mathrm{E}_{x_t, u_t, \dots}[\tilde{G}_t]$$

Policy gradient theorem yields

$$\nabla_\theta J(\theta) = \mathrm{E}_{x_0, u_0, \dots}[\sum_{t \geq 0} \tilde{G}_t \nabla_\theta \log \pi(u_t | x_t, \theta)]$$

A closer look at the baseline

Claim: adding baseline does not change the value of the expected gradient

$$\nabla_\theta J(\theta) = \mathrm{E}[\sum_{t\geq 0}(G_t \ - b(x_t))\nabla_\theta \log \pi(u_t|x_t,\theta)]$$

$$= \mathrm{E}[\sum_{t\geq 0} G_t \nabla_\theta \log \pi(u_t|x_t,\theta)] \ - \mathrm{E}[\sum_{t\geq 0} b(x_t)\nabla_\theta \log \pi(u_t|x_t,\theta)]$$

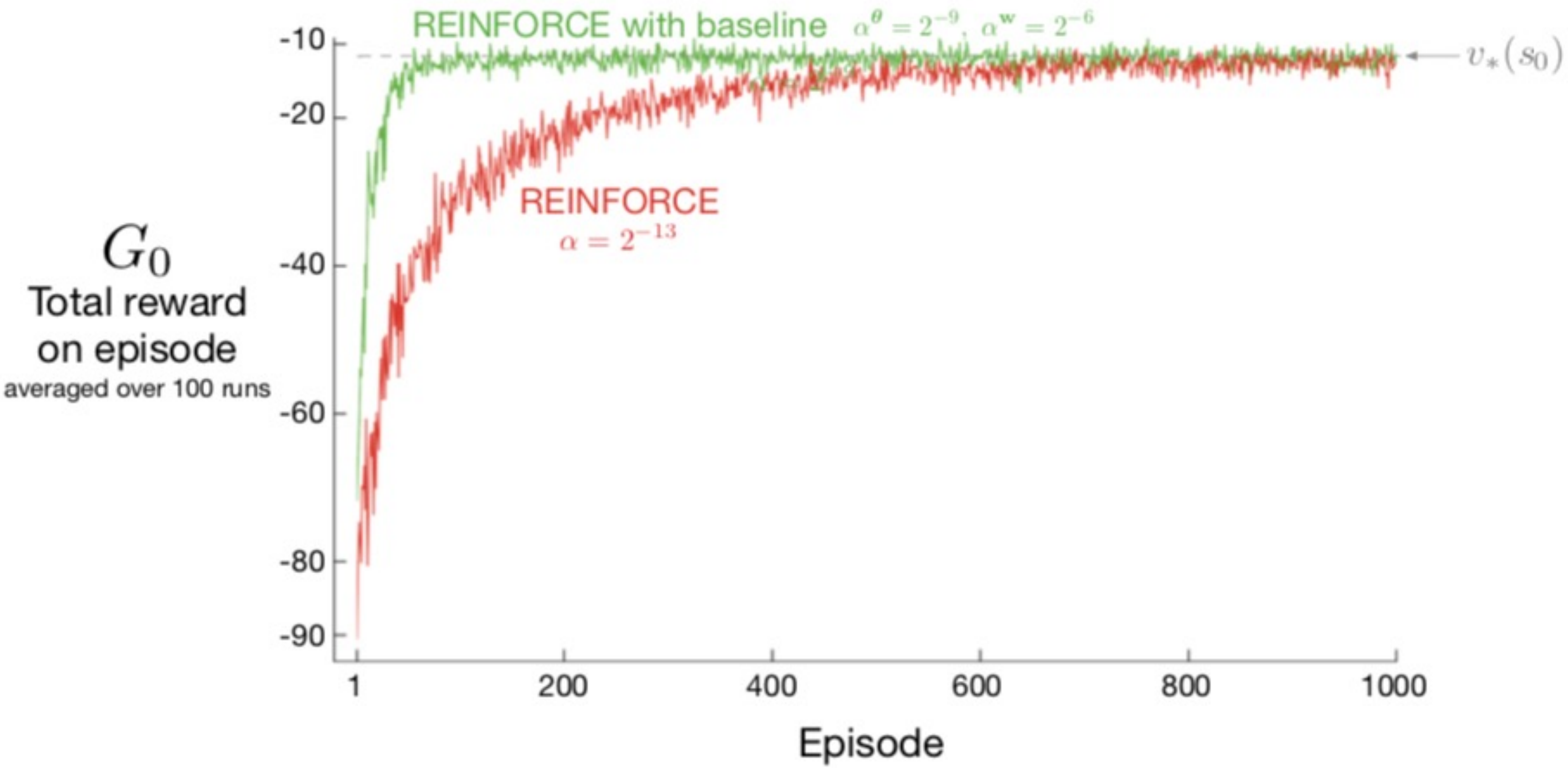
$$\mathrm{E}[b(x_t)\nabla_\theta \log \pi(u_t|x_t,\theta)] = \mathrm{E}_{x_t}[b(x_t)\mathrm{E}_{u_t}[\nabla_\theta \log \pi(u_t|x_t,\theta)]]$$

$$\mathrm{E}_{u_t}[\nabla_\theta \log \pi(u_t|x_t,\theta)] = \nabla_\theta \ \mathrm{E}_{u_t}[1] = 0$$

Any state-dependent function, indep. of action, works.

Example

Performance improvement on gridworld



Actor-critic

Particularly good baseline choice: value function

Actor-critic: use both **actor** (policy) and **critic** (value function).

Loop forever:

Generate episode $x_0, u_0, r_0, x_1, u_1, r_1$ … with π_θ

Loop for all $t = 0, \dots, N-1$:

$G \leftarrow \sum_{k=t+1}^{N} r_k$

$\delta_w \leftarrow G - V_w(x_t)$

$w \leftarrow w + \alpha_w \delta_w \nabla_w V_w(x_t)$

$\theta \leftarrow \theta + \alpha_\theta \delta_w \nabla_\theta \log \pi_\theta(u_t | x_t)$

Policy gradient theorem with Q function

- Previously, have used $J(\theta) = \mathrm{E}_{x_0,u_0,\dots}[\sum_{t\geq 0} r(x_t, u_t)]$
- Note that

$$J(\theta) = \mathrm{E}_{u_t \sim \pi(\cdot|x_t)}[Q^\pi(x_t, u_t)]$$

Yields policy gradient

$$\nabla_\theta J(\theta) = \mathrm{E}_{u_t \sim \pi(\cdot|x_t)}[Q^\pi(x_t, u_t) \nabla_\theta \log \pi(u_t|x_t)]$$

Note that $Q^\pi(x_t, u_t) = \mathrm{E}_{u_t \sim \pi(\cdot|x_t), x_{t+1}}[r(x_t, u_t) + V^\pi(x_{t+1})]$

Advantage policy gradient

- Combining the Q function policy gradient and the value baselines, we have

$$\nabla_\theta J(\theta) = \mathrm{E}[\, \delta^\pi \nabla_\theta \log \pi(u_t|x_t)]$$

For $\delta^\pi = \left(r_t + V^\pi(x_{t+1}) - V^\pi(x_t)\right)$. This is the TD error for policy evaluation!

- Note that $\mathrm{E}_\pi[\delta^\pi|x,u] = Q^\pi(x,u) - V^\pi(x) = A^\pi(x,u)$.
  - This is called the **advantage**.

Advantage actor-critic

Loop forever:

Generate episode $x_0, u_0, r_0, x_1, u_1, r_1 \ldots$ with π_θ

Loop for all $t = 0, \ldots, N-1$:

$$\delta_w \leftarrow r_t + V_w(x_{t+1}) \; - V_w(x_t)$$

$$w \leftarrow w + \alpha_w \delta_w \nabla_w V_w(x_t)$$

$$\theta \leftarrow \theta + \alpha_\theta \delta_w \nabla_\theta \log \pi_\theta(u_t | x_t)$$

Alternative estimators

- Many possible estimators for the advantage

- Multistep TD error:

$$\delta \leftarrow r_t + r_{t+1} + \ \ldots + r_{t+\tau} + V_w(x_{t+\tau+1}) \ - V_w(x_t)$$

As τ gets larger, this gets closer to Monte Carlo with value baseline.

Trust region policy optimization (TRPO) [Schulman et al., ICML 2015]

- Main idea : instead of choosing step size, use trust region

$$\max \mathrm{E}[\frac{\pi_\theta(u_t|x_t)}{\pi_{\theta_{old}}(u_t|x_t)}\hat{A}_t]$$

$$s.t. \mathrm{E}_{x\sim\rho_{old}}\left[D_{KL}\left(\pi_{\theta_{old}}(\cdot\,|x)||\pi_\theta(\cdot\,|x)\right)\right] \leq \delta$$

- Can show that this leads to monotonic improvement in the ideal case.
- Simpler, more popular version: proximal policy optimization (PPO).
  - Replaces TRPO CG solve with simple adaptive KL penalty.

Deterministic policy gradient (DPG) [Silver et al., ICML 2014]

- Instead of using stochastic policy with value estimation baseline:
  - Maintain estimate of Q function via minimizing TD error
  - Optimize deterministic policy via

$$\max_{\theta} \mathrm{E}_x[Q(x, \pi_\theta(x))]$$

- Policy simply *amortizes* optimization of the Q function.
- Can be used off policy, relatively unstable in practice.

Maximization bias

- Even though state-action value estimates are unbiased, may still have biased value estimates
- Example:

Double Q-learning

- Several possible solutions; in general, want to avoid using *max of estimates* as *estimate of max*.
- Double Q-learning [van Hasselt, NeurIPS 2010]: use two independent estimates Q_1, Q_2
  - $u^* = \mathrm{argmax}_u Q_1(x, u)$
  - Use value estimate $Q_2(x, u^*)$
- Alternative approach: maintain two independent critics, always use min [Fujimoto et al, ICML 2018]

Criticism of model-free methods

- Despite recent progress (including much not discussed here), questions about whether model-free methods are doing more than random search in parameter space.

Why did TD-Gammon Work?

Jordan B. Pollack & Alan D. Blair
Computer Science Department
Brandeis University
Waltham, MA 02254
{pollack,blair}@cs.brandeis.edu

Abstract

Although TD-Gammon is one of the major successes in machine learning, it has not led to similar impressive breakthroughs in temporal difference learning for other applications or even other games. We were able to replicate some of the success of TD-Gammon, developing a competitive evaluation function on a 4000 parameter feed-forward neural network, without using back-propagation, reinforcement or temporal difference learning methods. Instead we apply simple hill-climbing in a relative fitness environment. These results and further analysis suggest that the surprising success of Tesauro's program had more to do with the co-evolutionary structure of the learning task and the dynamics of the backgammon game itself.

Simple random search of static linear policies is competitive for reinforcement learning

Horia Mania hmania@berkeley.edu
Aurelia Guy lia@berkeley.edu
Benjamin Recht brecht@berkeley.edu

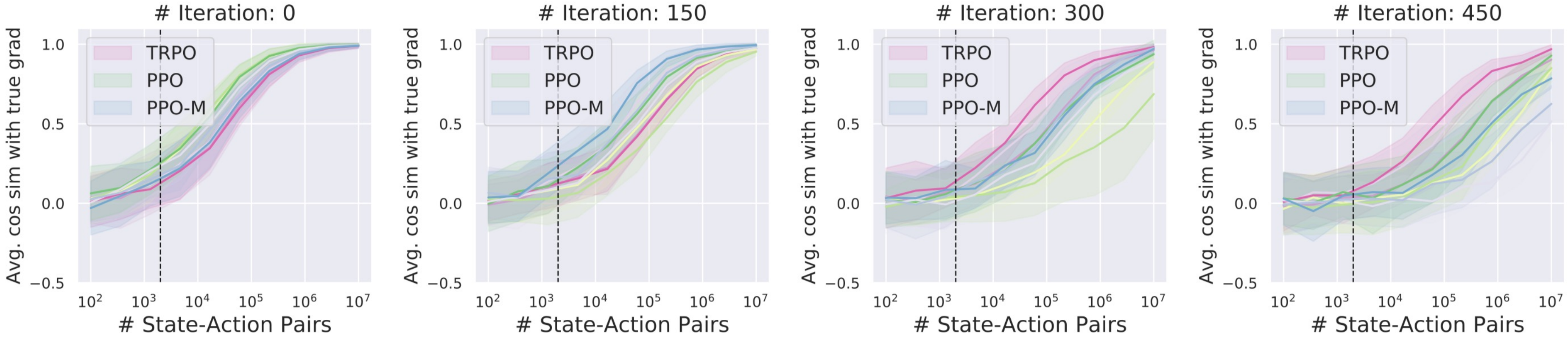
Department of Electrical Engineering and Computer Science
University of California, Berkeley

Abstract

Model-free reinforcement learning aims to offer off-the-shelf solutions for controlling dynamical systems without requiring models of the system dynamics. We introduce a model-free random search algorithm for training static, linear policies for continuous control problems. Common evaluation methodology shows that our method matches state-of-the-art sample efficiency on the benchmark MuJoCo locomotion tasks. Nonetheless, more rigorous evaluation reveals that the assessment of performance on these benchmarks is optimistic. We evaluate the performance of our method over hundreds of random seeds and many different hyperparameter configurations for each benchmark task. This extensive evaluation is possible because of the small computational footprint of our method. Our simulations highlight a high variability in performance in these benchmark tasks, indicating that commonly used estimations of sample efficiency do not adequately evaluate the performance of RL algorithms. Our results stress the need for new baselines, benchmarks and evaluation methodology for RL algorithms.

Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?

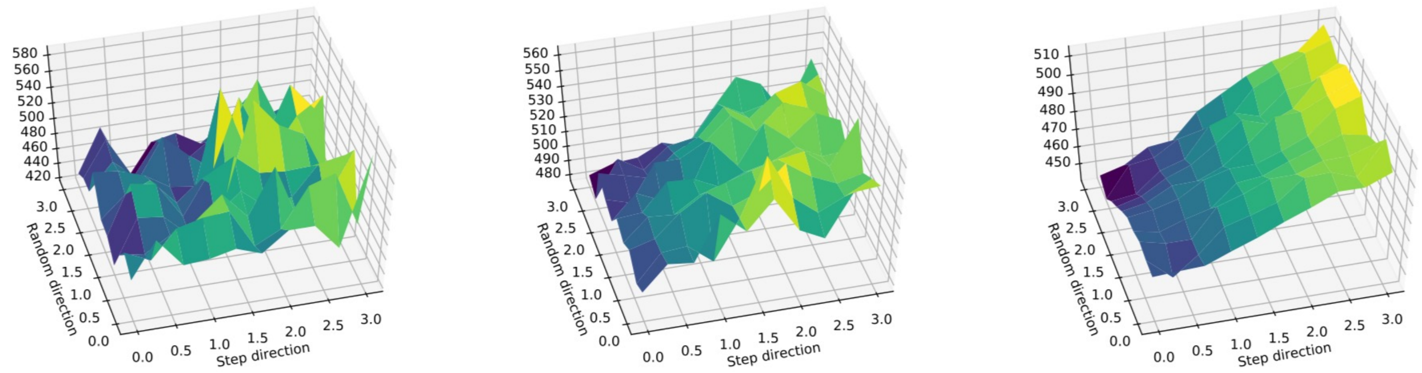
Andrew Ilyas*[1], Logan Engstrom*[1], Shibani Santurkar[1], Dimitris Tsipras[1], Firdaus Janoos[2], Larry Rudolph[1,2], and Aleksander Mądry[1]



2,000 state-action pairs (19 trajectories)

20,000 state-action pairs (198 trajectories)

100,000 state-action pairs (1068 trajectories)



Why model-free?

- Advantages
  - Very few assumptions
  - Many state of the art methods reach better performance than model-based methods

- Weaknesses
  - Extremely high sample complexity

Next time

- Combining policy optimization with model learning