The Mathematical Foundations of Policy Gradient Methods

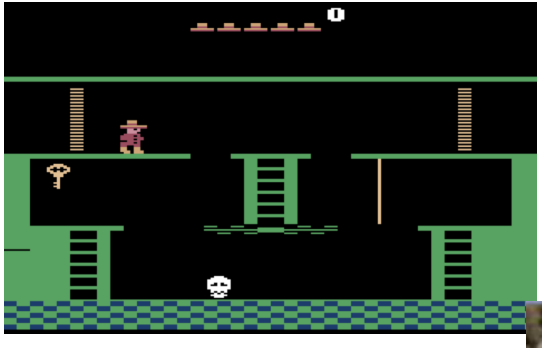
Sham M. Kakade

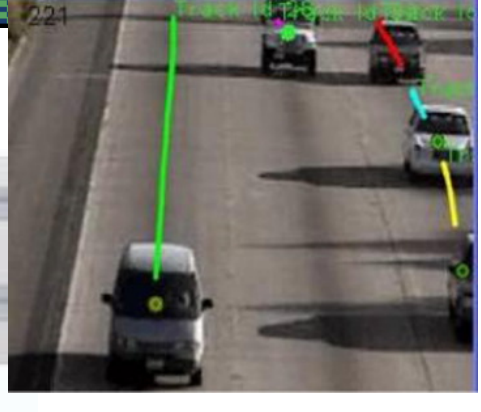
University of Washington

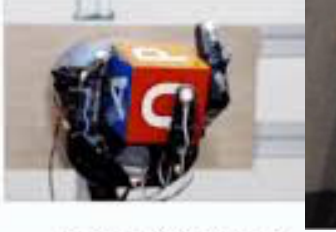
&

Microsoft Research

Reinforcement (interactive) learning (RL):

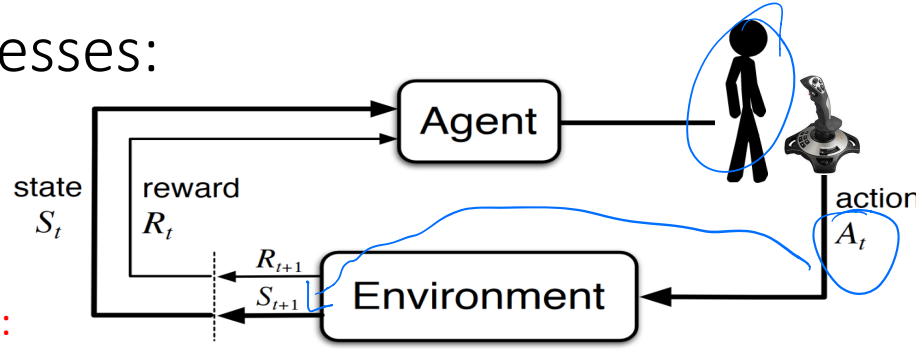






FINGER PIVOTING SLIDING FINGER GAITING

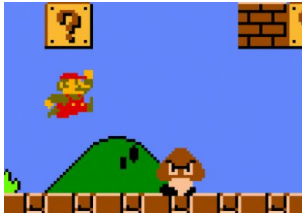
Markov Decision Processes:
a framework for RL



- A policy:
  π: States → Actions
- We execute π to obtain a trajectory:
  $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$
- Total γ-discounted reward:

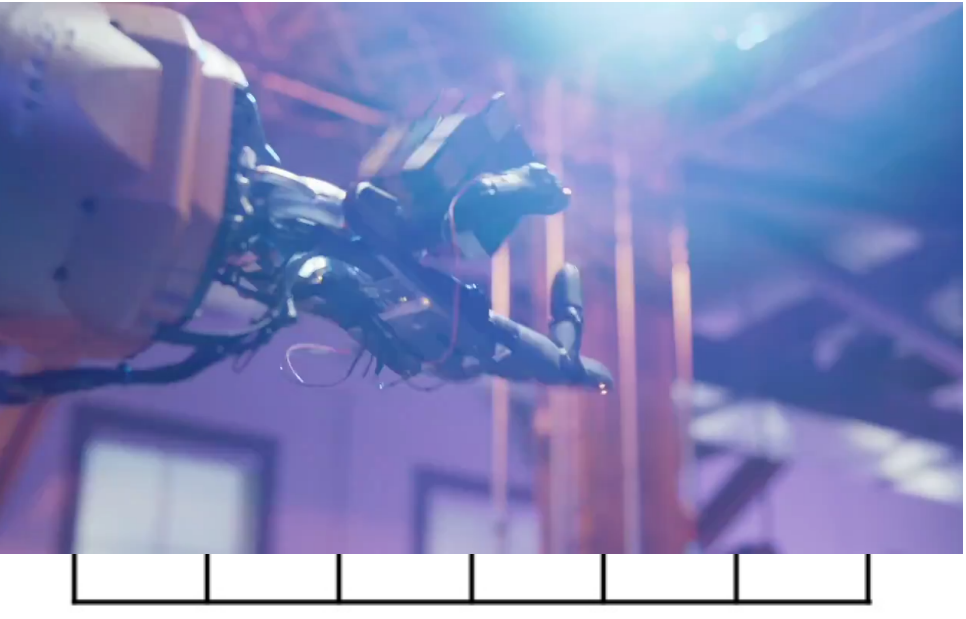
$\gamma \leq 1$

$$V^{\pi}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0, \pi\right]$$



Goal: Find a policy that maximizes our value, $V^{\pi}(s_0)$.

Dexterous Robotic Hand Manipulation
OpenAI, Oct 15, 2019



Challenges in RL

1. Exploration
   (the environment may be unknown)
2. Credit assignment problem
   (due to delayed rewards)
3. Large state/action spaces:
   hand state: joint angles/velocities
   cube state: configuration
   actions: forces applied to actuators

Values, State-Action Values, and Advantages

$$V^{\pi}(s_0) = E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0, \pi\right]$$

$$Q^{\pi}(s_0, a_0) = E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \,\middle|\, s_0, a_0, \pi\right]$$

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

- Expectation with respect to sampled trajectories under π
- Have S states and A actions.
- Effective "horizon" is $1/(1-\gamma)$ time steps.

The "Tabular" Dynamic Programming approach

State s: (joint angles, … cube config,…)	**Action a:** (forces at joints)	**$Q^\pi(s,a)$: state-action value** "one step look-ahead value" using π
$(31^\circ, 12^\circ, \dots, 8134, \dots)$	$(1.2 \text{ Newton}, 0.1 \text{ Newton}, \dots)$	8 units of reward
⋮	⋮	⋮

- Table: 'bookkeeping' for dynamic programming (with known rewards/dynamics)
  1. Estimate the state-action value $Q^\pi(s,a)$ for *every entry* in the table.
  2. Update the policy π & goto step 1

start with π_0

$\pi(s) \in \arg\max_a Q^\pi(s,a)$

- Generalization: how can we deal with this infinite table? using sampling/supervised learning?

This Tutorial:

Mathematical Foundations of Policy Gradient Methods

generalization

- Part – I: Basics
  A. Derivation and Estimation
  B. Preconditioning and the Natural Policy Gradient
- Part – II: Convergence and Approximation
  A. Convergence: This is a non-convex problems!
  B. Approximation: How to the think about the role of deep learning?

Part-1: Basics

State-Action Visitation Measures!

- This helps to clean up notation!
- "Occupancy frequency" of being in state s and action a, after following π starting in s_0

$$d^{\pi}_{s_0}(s) = (1-\gamma)E\left[\sum_{t=0}^{\infty}\gamma^t\, I(s_t = s)\,\middle|\, s_0, \pi\right]$$

- $d^{\pi}_{s_0}$ is a probability distribution
- With this notation:

$$V^{\pi}(s_0) = \frac{1}{1-\gamma}E_{s\sim d^{\pi}_{s_0}, a\sim\pi}\left[r(s,a)\right]$$

$d^{\pi}_{s_0}(s) \simeq$ chance of visiting s starting at s_0 following π.

Direct Policy Optimization over Stochastic Policies

- $\pi_\theta(a|s)$ is the probability of action a given s, parameterized by

$$\pi_\theta(a|s) \propto \exp(f_\theta(s,a))$$

$$\pi(a|s) = e^{\theta_{sa}} / Z$$

- Softmax policy class: $f_\theta(s,a) = \theta_{s,a}$
- Linear policy class: $f_\theta(s,a) = \vec{\theta} \cdot \vec{\phi}(s,a)$ where $\vec{\phi}(s,a) \in R^d$
- Neural policy class: $f_\theta(s,a)$ is a neural network

In practice, policy gradient methods rule…

They are the most effective method for obtaining state of the art.

$$\theta \leftarrow \theta + \eta \nabla V^{\pi_\theta}(s_0)$$

- Why do we like them?
  - They easily deal with large state/action spaces (through the neural net parameterization)
  - We can estimate the gradient using only simulation of our current policy π_θ (the expectation is under the state actions visited under π_θ)
  - They directly optimize the cost function of interest!

Two (equal) expressions for the policy gradient!

over trajectories

$$\nabla V^{\theta}(s_0) = \frac{1}{1-\gamma} E_{s\sim d^{\pi}, a\sim\pi}\left[Q^{\theta}(s,a)\nabla \log \pi_{\theta}(a|s)\right]$$

$$\nabla V^{\theta}(s_0) = \frac{1}{1-\gamma} E_{s\sim d^{\pi}, a\sim\pi}\left[A^{\theta}(s,a)\nabla \log \pi_{\theta}(a|s)\right]$$

(some shorthand notation above)

- Where do these expression come from?
- How do we compute this?

exercise

hint:

$$\sum_x p(x) \nabla_\theta \log p_\theta(x) = 0$$

Example: an important special case!

- Remember the softmax policy class (a "tabular" parameterization)

$$\pi_\theta(a|s) \propto \exp(\theta_{s,a})$$

- Complete class with SA params:
  one parameter per state action, so it contains the optimal policy
- Expression for softmax class:

$$\frac{\partial V^\theta(s_0)}{\partial \theta_{s,a}} = d^{\pi_\theta}(s)\pi_\theta(a|s)A^\theta(s,a)$$

- Intuition: increase $\theta_{s,a}$ if the 'weighted' advantage is large.

Part-1A: Derivations and Estimation

General Derivation

$$
\begin{aligned}
& \nabla V^{\pi_\theta}(s_0) \\
= \; & \nabla \sum_{a_0} \pi_\theta(a_0|s_0) Q^{\pi_\theta}(s_0, a_0) \\
= \; & \sum_{a_0} \Big(\nabla \pi_\theta(a_0|s_0) \Big) Q^{\pi_\theta}(s_0, a_0) + \sum_{a_0} \pi_\theta(a_0|s_0) \nabla Q^{\pi_\theta}(s_0, a_0) \\
= \; & \sum_{a_0} \pi_\theta(a_0|s_0) \Big(\nabla \log \pi_\theta(a_0|s_0) \Big) Q^{\pi_\theta}(s_0, a_0) \\
& + \sum_{a_0} \pi_\theta(a_0|s_0) \nabla \Big(r(s_0, a_0) + \gamma \sum_{s_1} P(s_1|s_0, a_0) V^{\pi_\theta}(s_1) \Big) \\
= \; & \sum_{a_0} \pi_\theta(a_0|s_0) \Big(\nabla \log \pi_\theta(a_0|s_0) \Big) Q^{\pi_\theta}(s_0, a_0) + \gamma \sum_{a_0, s_1} \pi_\theta(a_0|s_0) P(s_1|s_0, a_0) \nabla V^{\pi_\theta}(s_1) \\
= \; & \mathbb{E}\left[Q^{\pi_\theta}(s_0, a_0) \nabla \log \pi_\theta(a_0|s_0) \right] + \gamma \mathbb{E}\left[\nabla V^{\pi_\theta}(s_1) \right].
\end{aligned}
$$

SL vs RL: How do we obtain gradients?

- In supervised learning, how do we compute the gradient of our loss $\nabla L(\theta)$?

$$\theta \leftarrow \theta + \eta \nabla L(\theta)$$

  - Hint: can we compute our loss?

- In reinforcement learning, how do we compute the policy gradient $\nabla V^{\theta}(s_0)$?

$$\theta \leftarrow \theta + \eta \nabla V^{\theta}(s_0)$$

even computing $V^{\pi}(s_0)$ is tricky.

$$\nabla V^{\theta}(s_0) = \frac{1}{1-\gamma} E_{s,a}\left[Q^{\theta}(s,a) \nabla \log \pi_{\theta}(a|s)\right]$$

Monte Carlo Estimation

- Sample a trajectory: execute π_θ and $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

$$\widehat{Q}(s_t, a_t) = \sum_{t'=0}^{\infty} \gamma^{t'} r(s_{t'+t}, a_{t'+t})$$

$$\widehat{\nabla V^\theta} = \sum_{t=0}^{\infty} \gamma^t \widehat{Q}(s_t, a_t) \nabla \log \pi_\theta(a_t|s_t)$$

- Lemma: [Glynn '90, Williams '92]] This gives an unbiased estimate of the gradient:

$$\mathrm{E}[\widehat{\nabla V^\theta}] = \nabla V^\theta(s_0)$$

This is the "likelihood ratio" method.

Back to the softmax policy class…

$$\pi_\theta(a|s) \propto \exp(\theta_{s,a})$$

$$= \frac{e^{\theta_{sa}}}{\sum_{a'} e^{\theta_{sa'}}}$$

- Expression for softmax class:

$$\frac{\partial V^\theta(s_0)}{\partial \theta_{s,a}} = d^{\pi_\theta}(s)\pi_\theta(a|s)A^\theta(s,a)$$

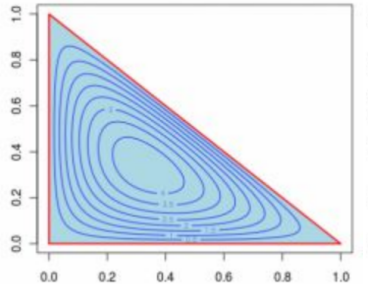
- What might be making gradient ~~estimation~~ method difficult here?
  (hint: when does gradient descent "effective" stop?)

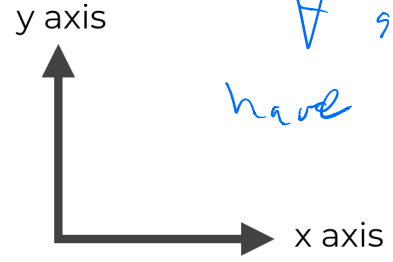
GD stops when grad is "small"

Suppose A_{sa} is >>0 but if $\pi(a|s)$ is small $\pi(a|s)$ is small. X

Part-1B: Preconditioning and the Natural Policy Gradient

A closer look at Natural Policy Gradient (NPG)





∀ state have a dist $\pi(\cdot|s)$

- Practice: (almost) all methods are gradient based, usually variants of: Natural Policy Gradient [K. '01]; TRPO [Schulman '15]; PPO [Schulman '17]
- NPG warps the distance metric to stretch the corners out (using the Fisher information metric) move 'more' near the boundaries. The update is:

$$F(\theta) = E_{s\sim d^{\pi}, a\sim\pi}[\nabla \log \pi_\theta(a|s) \nabla \log \pi_\theta(a|s)^T]$$

$$\theta \leftarrow \theta + \eta F(\theta)^{-1} \nabla V^{\theta}(s_0)$$

- Sampling estimation
- state space size. ↔ # params.

TRPO (Trust Region Policy Optimization)

- TRPO [Schulman '15] (related: PPO [Schulman '17]):
  move staying "close" in KL to previous policy:

$$\theta_{t+1} = \text{argmin}_\theta \; V^\theta(s_0)$$

$$\text{s. t. } E_{s\sim d^{\pi_t}}\left[KL\left(\pi^\theta(\cdot\,|s)\,\|\,\pi^{\theta_t}(\cdot\,|s)\right)\right] \leq \delta$$

- NPG=TRPO: they are first order equivalent (and have same practical behavior)

NPG intuition. But first….

- NPG as preconditioning:

$$\theta \leftarrow \theta + \eta\, F(\theta)^{-1} \nabla V^{\theta}(s_0)$$

OR

$$\theta \leftarrow \theta + \frac{\eta}{1-\gamma} (E[\nabla \log \pi_\theta(a|s) \nabla \log \pi_\theta(a|s)^T])^{-1} E\left[\nabla \log \pi_\theta(a|s) A^{\theta}(s,a)\right]$$

- What does the following problem remind you of?

$$E[XX^{\top}]^{-1} E[XY]$$

- What is NPG is trying to approximate?

Equivalent Update Rule (for the softmax)

- Take the best linear fit of Q^θ in "policy space"-features": this gives

$$W^*_{s,a} = A^\theta(s,a)$$

exercise

- Using the NPG update rule :

$$\theta_{s,a} \leftarrow \theta_{s,a} + \frac{\eta}{1-\gamma} A^\theta(s,a)$$

- And so an equivalent update rule to NPG is:

$$\pi_\theta(a|s) \leftarrow \pi_\theta(a|s) \exp\left(\frac{\eta}{1-\gamma} A^\theta(s,a)\right)/Z$$

"Soft Policy Iteration"

$$\pi(a|s) \cdot e^{\frac{\eta}{1-\gamma} Q(s,a)} \Big/ Z$$

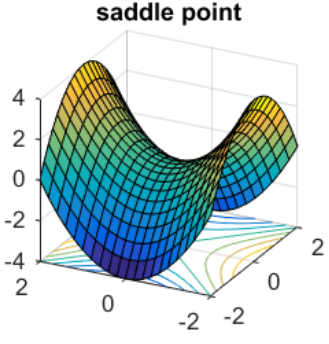
- What algorithm does this remind you of?

as $\eta \to \infty$, next policy π is policy if update.

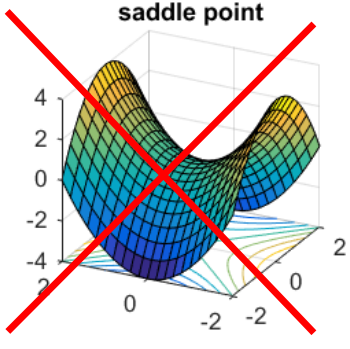
Questions: convergence? General case/approximation?

But does gradient descent even work in RL??

Supervised Learning



Reinforcement Learning

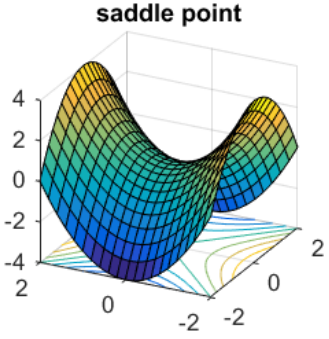


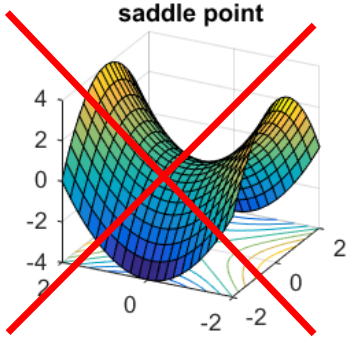
What about approximation?

Stay tuned!!

Part-2: Convergence and Approximation

The Optimization Landscape





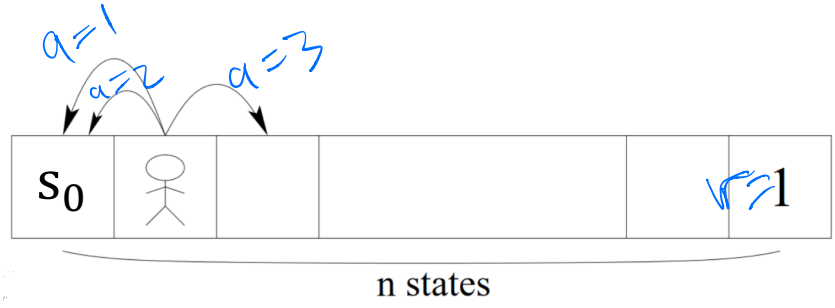
Supervised Learning:

- Gradient descent tends to 'just work' in practice and is not sensitive to initialization
- Saddle points not a problem…

Reinforcement Learning:

- Local search depends on initialization in many real problems, due to "very" flat regions.
- Gradients can be exponentially small in the "horizon"

RL and the vanishing gradient problem



Thrun '92

Reinforcement Learning:

- The random init. has "very" flat regions in real problems (lack of 'exploration')
- Lemma: [Agarwal, Lee, K., Mahajan 2019]
  With random init, all k-th higher-order gradients are $2^{-H/2}$ in magnitude for up to $\mathrm{k} < \mathrm{H}/\ln(H)$ orders, $H = 1/(1-\gamma)$.
- This is a landscape/optimization issues.
  (also a statistical issue if we used random init).

Part 2:

Understanding the convergence properties of the (NPG) policy gradient methods!

- A: Convergence
  - Let's look at the tabular/"softmax" case
- B: Approximation
  - Approximation: "linear" policies and neural nets

NPG: back to the "soft" policy iteration interpretation

- Remember the softmax policy class

$$\pi_\theta(a|s) \propto \exp(\theta_{s,a})$$

has $S \cdot A$ params

- At iteration t, the NPG update rule:

$$\theta^{t+1} \leftarrow \theta^t + \eta\, F(\theta^t)^{-1} \nabla V^t(s_0)$$

is equivalent to a "soft" (exact) policy iteration update rule:

$$\pi^{t+1}(a|s) \leftarrow \pi^t(a|s) \exp\left(\frac{\eta}{1-\gamma} A^t(s,a)\right) / Z_s$$

- What happens for this non-convex update rule?

"partition function"

$$Z_s = \sum_a \pi(a|s) e^{\frac{\eta}{1-\gamma} A^t(s,a)}$$

A^{π_θ} A^{θ} $A^t \rightsquigarrow \pi_{\theta_t}$