

# Lecture 3: Deep Reinforcement Learning

## Policy Gradients

Chunlin Chen & Zhi Wang

Department of Control and Systems Engineering  
Nanjing University

Nov. 19th, 2019

# Table of Contents

- 1 From Tabular Algorithms to Deep RL
- 2 Policy Gradients
- 3 Advanced Policy Gradient Methods

## For large/continuous state/action spaces

- **Curse of dimensionality:** Computational requirements grow exponentially with the number of state variables
  - Theoretically, all state-action pairs need to be visited infinite times to guarantee an optimal policy
  - In many practical tasks, almost every state encountered will never have been seen before
- **Generalization:** How can experience with a limited subset of the state space be usefully generalized to produce a good approximation over a much larger subset?

# Function approximation

- It takes examples from a desired function (e.g., a value function) and attempts to generalize from them to construct an approximation to the entire function
  - Linear function approximation:  $Q(s, a) = \sum_i \phi_i(s, a)w_i$
  - Neural network approximation:  $Q(s, a) = Q(s, a; \theta)$
- Function approximation is an instance of supervised learning, the primary topic studied in machine learning, artificial neural networks, pattern recognition, and statistical curve fitting
  - In theory, any of the methods studied in these fields can be used in the role of function approximator within RL algorithms
  - RL with function approximation involves a number of **new issues** that do not normally arise in conventional supervised learning, e.g., non-stationarity, bootstrapping, and delayed targets

# Deep reinforcement learning (DRL)

- The revolution of Deep Learning
  - Turing award 2018 to deep learning godfathers
- DRL = theories of RL + the help of deep function approximators



# Deep reinforcement learning (DRL)



# What do you think machine learning is?

## Machine Learning



what society thinks I  
do



what my friends think  
I do



what my parents think  
I do

$$L_2 = \|\mathbf{w}\|^2 + \sum_i \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_i \sigma_i$$

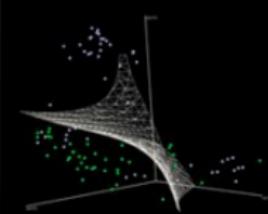
$$\alpha_i \geq 0, \forall i$$

$$\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i, \sum_i \alpha_i y_i = 0$$

$$\nabla \hat{y}(\theta_t) = \frac{1}{n} \sum_{i=1}^n \nabla \ell(x_i, y_i; \theta_t) + \nabla r(\theta_t),$$

$$\theta_{t+1} = \theta_t - \eta_t \nabla \ell(x_{i(t)}, y_{i(t)}; \theta_t) - \eta_t \cdot \nabla r(\theta_t)$$

$$\mathbb{E}_{i(t)} [\ell(x_{i(t)}, y_{i(t)}; \theta_t)] = \frac{1}{n} \sum_{i=1}^n \ell(x_i, y_i; \theta_t),$$



```
>>> from sklearn import svm
```

what other programmers  
think I do

what I think I do

what I really do

# What do you think deep learning is?

## Deep Learning



What society thinks I do



What my friends think I do



What other computer scientists think I do



What mathematicians think I do

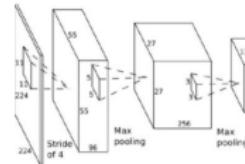


What I think I do

```
from theano import *
```

What I actually do

# Imitation learning - Imitating expert demonstrations



$$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$$



$$\mathbf{o}_t$$



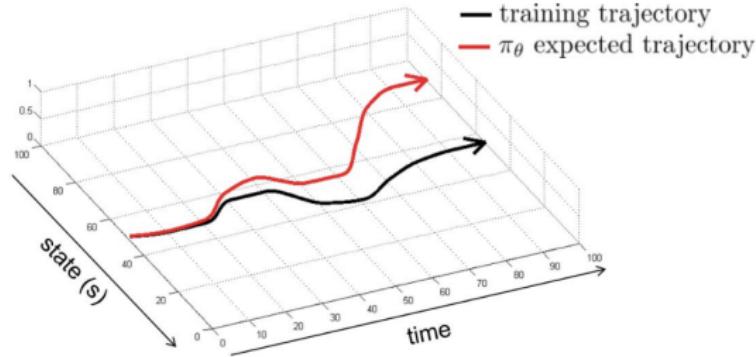
supervised  
learning

$$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$$

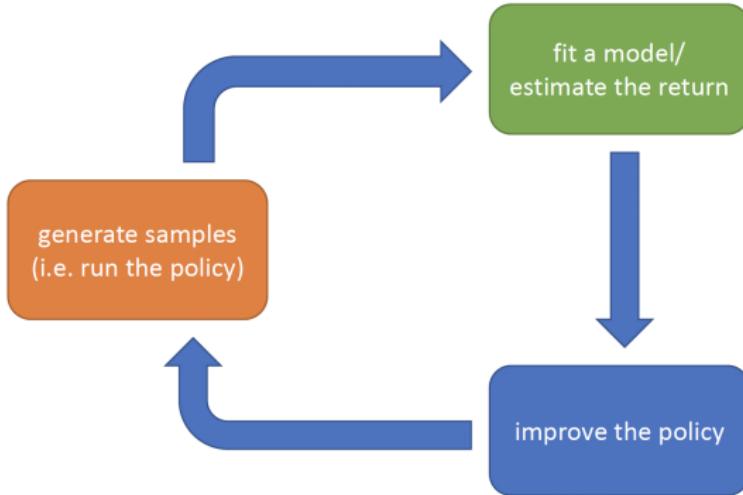
- Supervised training has better stability than many RL methods

# Imitation learning - Imitating expert demonstrations

- One of the biggest challenge is collecting expert demonstrations
  - Unless it has a huge business potential, the attached cost can be prohibitive
- The trained policy is only as good as the demonstrations
  - Initialize a policy from expert demonstrations, finetune it using RL
- Error accumulates fast in a trajectory and put us into situations that we never deal with before
  - Analogous to the key challenge in mode-based RL



# The anatomy of a DRL algorithm



- Following the **Generalized Policy Iteration** framework
  - ... Generate samples => Policy evaluation => Policy improvement ...

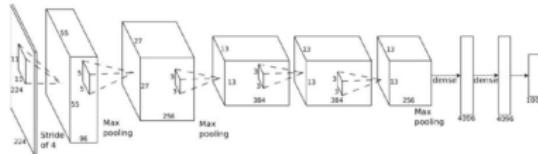
$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

# Types DRL algorithms

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

- **Policy gradients:** Directly approximate the policy and differentiate the above objective
- **Value function-based:** Estimate state- or action-value function or of the optimal policy (no explicit policy)
- **Actor-critic:** Estimate state- or action-value function or of the current policy, use it to improve policy
- **Model-based RL:** Estimate the transition model,  $p(s', r|s, a)$ 
  - Use it for planning (no explicit policy), use it to improve a policy, etc

# DRL - Directly approximate policies



$\mathbf{o}_t$

$$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$$

$\mathbf{a}_t$

$\mathbf{s}_t$  – state

$\mathbf{o}_t$  – observation

$\mathbf{a}_t$  – action

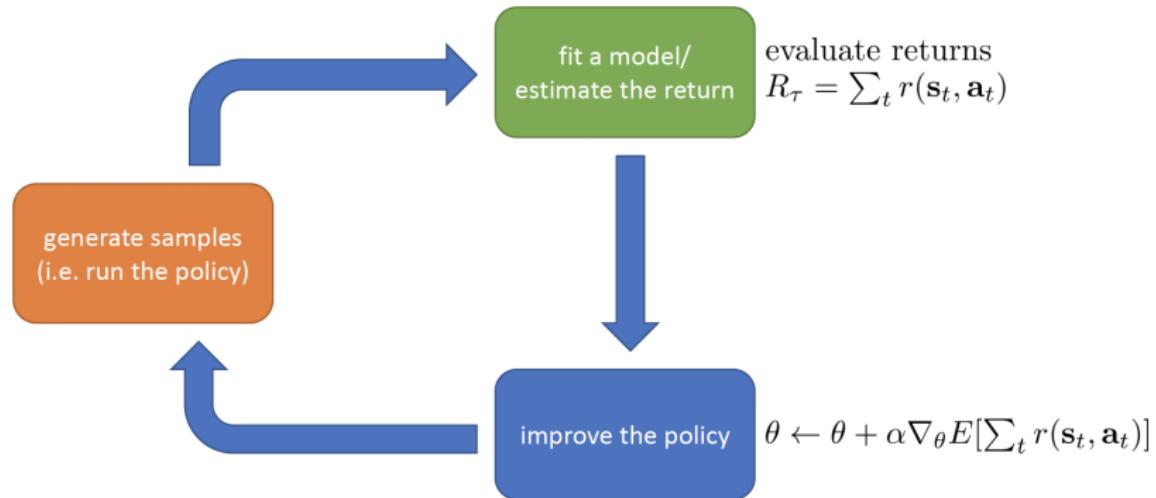
$\pi_\theta(\mathbf{a}_t | \mathbf{o}_t)$  – policy

$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$  – policy (fully observed)

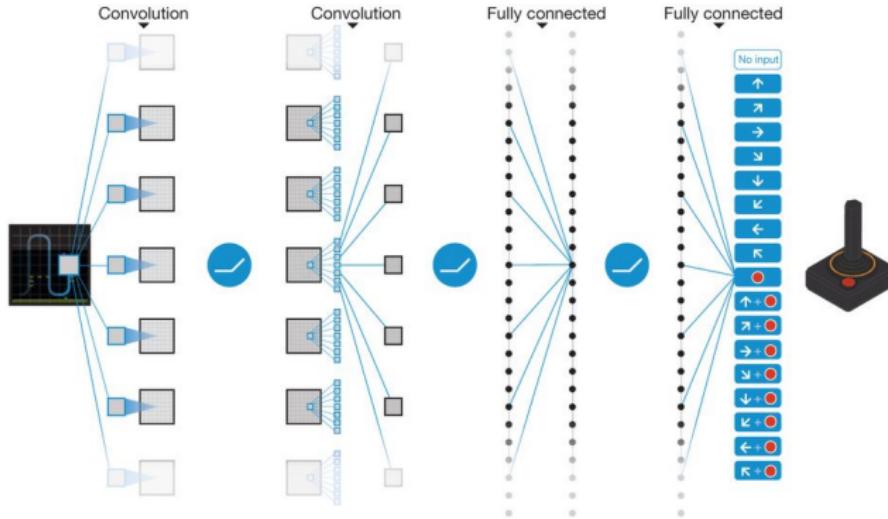


- Usually using deep neural networks
- Optimize the policy network by backpropagation

# DRL - Directly approximate policies

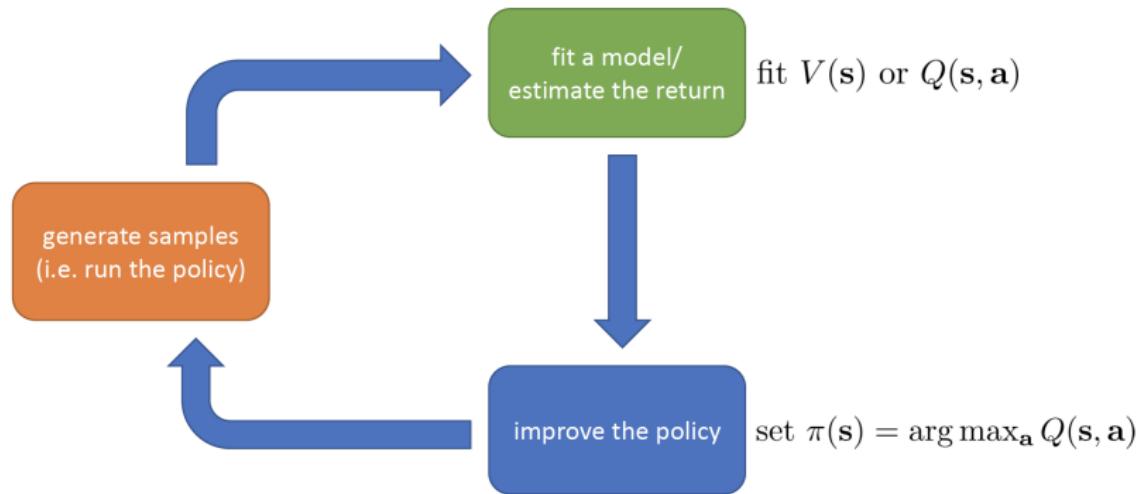


# DRL - Approximate value functions



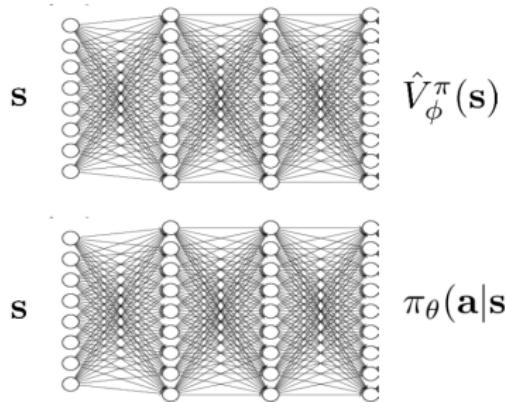
- Usually using deep neural networks
- Optimize the deep Q-network by minimizing the **Bellman residual**

# DRL - Approximate value functions

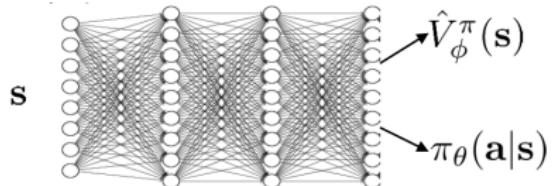


# DRL - Approximate both

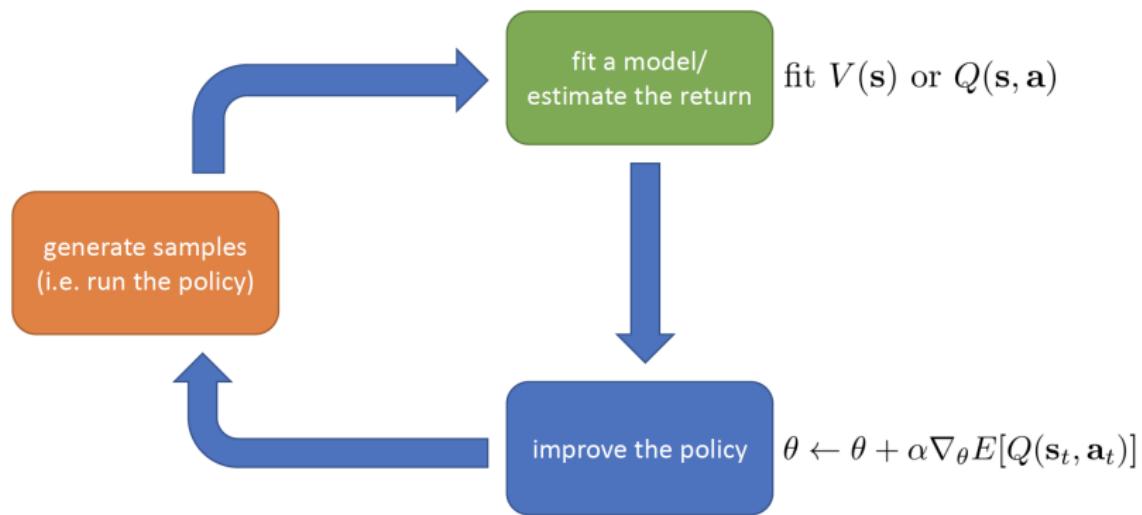
two network design



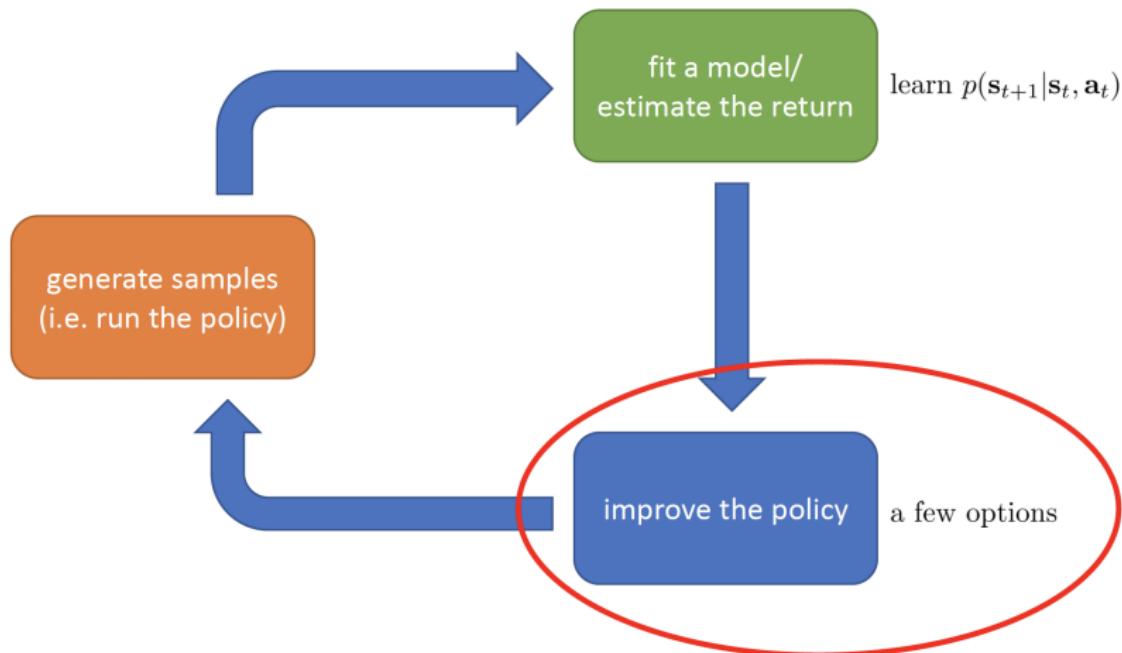
shared network design



# DRL - Approximate both



# DRL - Model-based



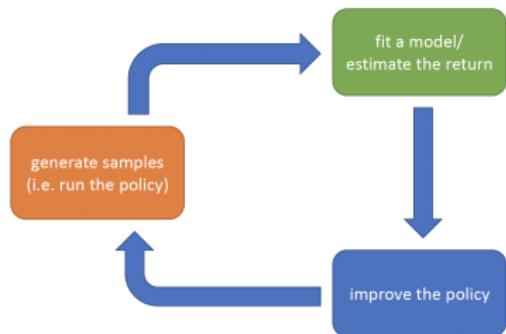
improve the policy

a few options

- Just use the model to plan (no explicit policy)
  - Trajectory optimization / optimal control
  - Discrete planning in discrete action spaces, e.g., Monte Carlo tree search
- Backpropagate gradients into the policy
- Use the model to learn a value function
  - Dynamic programming
  - Generate simulated experience for model free learner

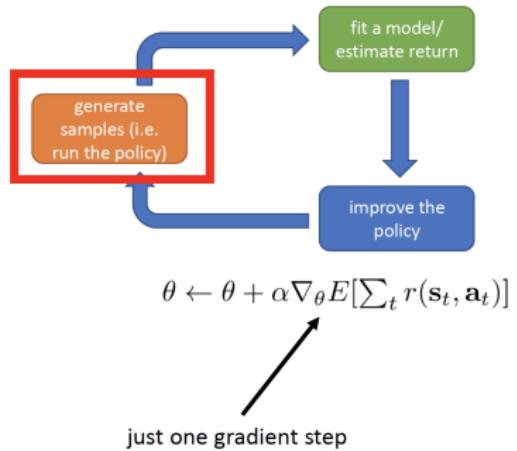
# Why so many DRL algorithms?

- Different tradeoffs
  - Sample efficiency – model-based
  - Stability & easy to use – model-free
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?

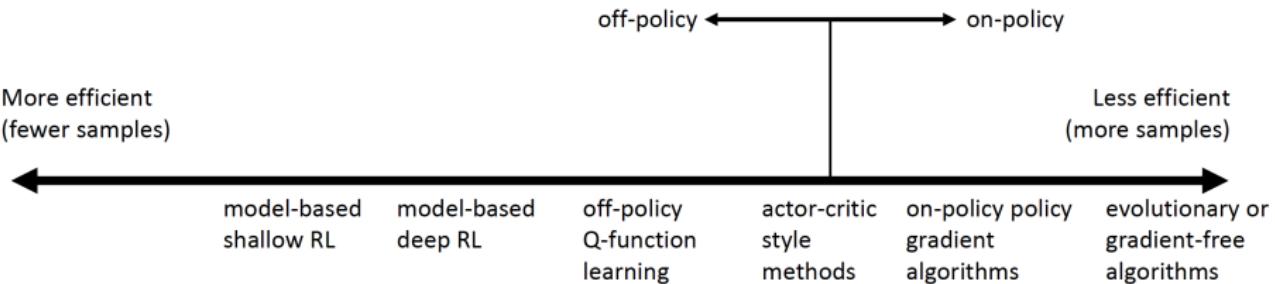


# Comparison: Sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?
- Most important question: is the algorithm off-policy?
  - Off-policy: able to improve the policy without generating new samples from that policy
  - On-policy: each time the policy is changed, even a little bit, we need to generate new samples



# Comparison: Sample efficiency



# Comparison: Stability and ease of use

- Does it converge?
- And if it converges, to what?
- And does it converge every time?

Why is any of this even a question???

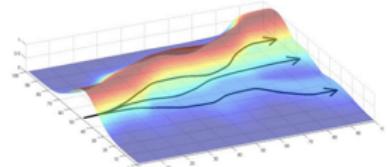
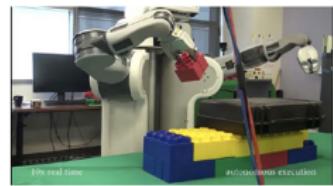
- Supervised learning: almost always gradient descent
- Reinforcement learning: often not gradient descent
  - Q-learning: Bellman fixed point iteration
  - Model-based RL: model is not optimized for expected reward
  - Policy gradient: is gradient descent, but also often the least efficient!

# Comparison: Stability and ease of use

- Value function fitting
  - At best, minimizes error of fit (“Bellman error”), not the same as expected reward
  - At worst, doesn’t optimize anything. Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case
- Model-based RL
  - Model minimizes error of fit. This will converge.
  - No guarantee that better model = better policy
- Policy gradient
  - The only one that actually performs gradient descent (ascent) on the true objective

# Comparison: Assumptions

- Common assumption 1: full observability
  - Generally assumed by value function fitting methods
  - Can be mitigated by adding recurrence
- Common assumption 2: episodic learning
  - Often assumed by pure policy gradient methods
  - Assumed by some model-based RL methods
- Common assumption 3: continuity or smoothness
  - Assumed by some continuous value function learning methods
  - Often assumed by some model-based RL methods



# Examples of specific algorithms

- Policy gradient methods
  - REINFORCE
  - Natural policy gradient
  - Trust region policy optimization
- Value function fitting methods
  - Fitted value iteration, fitted Q-iteration
  - Deep Q-network, deep Q-learning
- Actor critic algorithms
  - Deep deterministic policy gradient (DDPG)
  - Asynchronous advantage actor critic (A3C)
  - Soft actor critic (SAC)
- Model based RL algorithms
  - Dyna
  - Guided policy search

We'll learn about most of these in the next few weeks!

# Example 1: Atari games with deep Q-networks

- Playing Atari with deep reinforcement learning
- Q-learning with convolutional neural networks



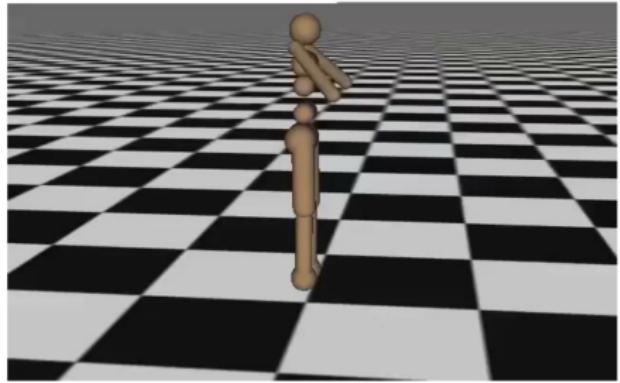
## Example 2: robots and model-based RL

- End-to-end training of deep visuomotor policies
- Guided policy search (model-based RL) for image based robotic manipulation

**Various Experiments**  
Including the policy input

## Example 3: walking with policy gradients

- High-dimensional continuous control with generalized advantage estimation
- Trust region policy optimization with value function approximation



## Example 4: robotic grasping with Q-functions

- Q learning from images for real world robotic grasping



# Table of Contents

1 From Tabular Algorithms to Deep RL

2 Policy Gradients

3 Advanced Policy Gradient Methods

# What we'll cover

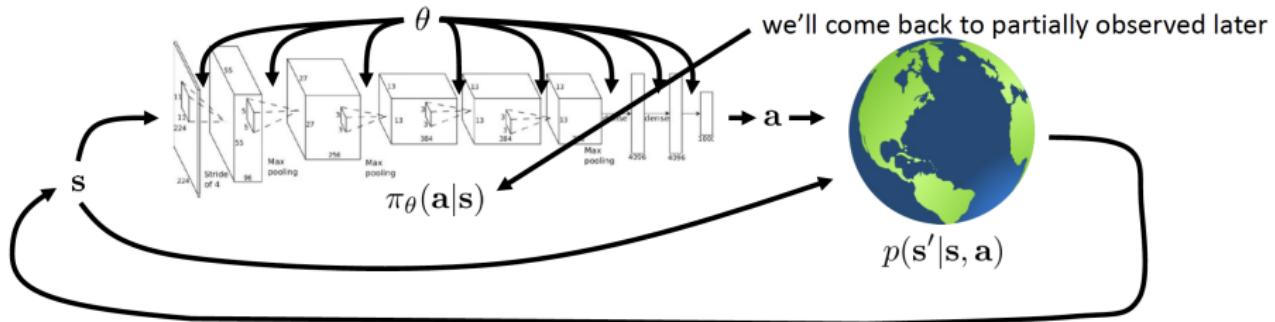
## — Contents

- The policy gradient algorithm
- What does the policy gradient do?
- Basic variance reduction: causality
- Basic variance reduction: baselines
- Policy gradient examples

## — Goals

- Understand policy gradient RL
- Understand practical considerations for policy gradients

# Recall the goal of RL

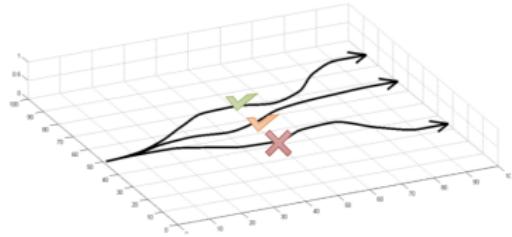


$$p_{\theta}(\tau) = p_{\theta}(s_0, a_0, \dots, s_T, a_T) = p(s_0) \prod_{t=0}^T \pi_{\theta}(a_t | s_t) p(s_{t+1} | s_t, a_t)$$

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

# Evaluating the objective

$$\theta^* = \arg \max_{\theta \in \mathbb{R}^d} \underbrace{\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right]}_{J(\theta)}$$



$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \underbrace{\frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t})}_{\text{sum over samples from } \pi_\theta}$$

# Direct policy differentiation

- Objective function / cost function

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \underbrace{r(\tau)}_{\sum_{t=0}^T r(s_t, a_t)} \right] = \int \pi_\theta(\tau) r(\tau) d\tau$$

- The gradient – differentiate the objective function

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

- A convenient identity

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

# Direct policy differentiation

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

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

$$\nabla_{\theta} \left[ \cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)} \right]$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

log of both sides

$$\begin{aligned} \pi_{\theta}(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) &= p(\mathbf{s}_1) \prod_{t=1}^T \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \\ \log \pi_{\theta}(\tau) &= \log p(\mathbf{s}_1) + \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) + \log p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \end{aligned}$$

# Evaluating the policy gradient

$$\text{recall: } J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

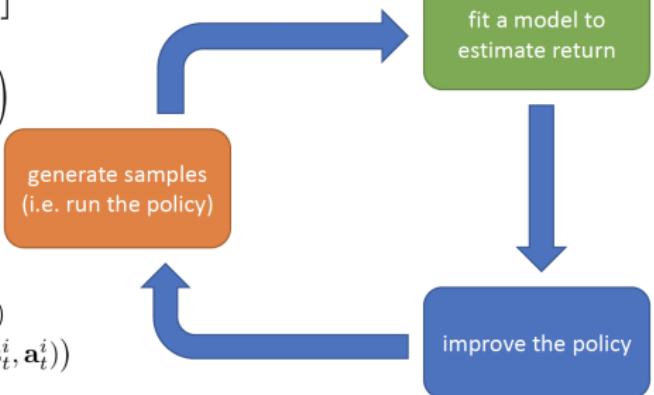
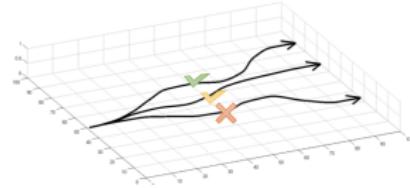
$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

REINFORCE algorithm:

- 1. sample  $\{\tau^i\}$  from  $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$  (run the policy)
- 2.  $\nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i | \mathbf{s}_t^i)) (\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))$
- 3.  $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



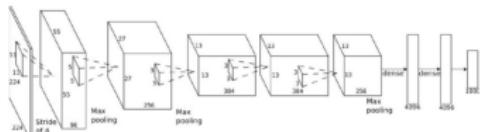
# Evaluating the policy gradient

$$\text{recall: } J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

what is this?

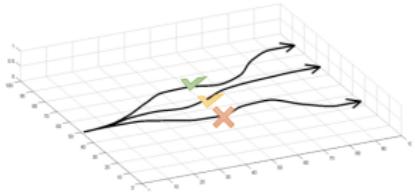


$\mathbf{s}_t$

$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$



$\mathbf{a}_t$



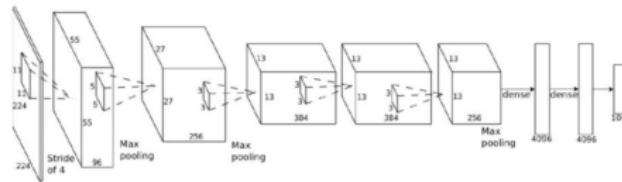
# Comparison to maximum likelihood

$$\text{policy gradient: } \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

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



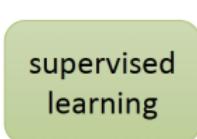
$\mathbf{s}_t$



$\mathbf{a}_t$



$\mathbf{s}_t$   
 $\mathbf{a}_t$



$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$

# Example: Gaussian policies

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

example:  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

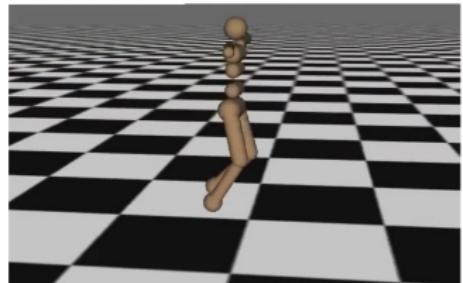
$$\log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \|f(\mathbf{s}_t) - \mathbf{a}_t\|_{\Sigma}^2 + \text{const}$$

$$\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) = -\frac{1}{2} \Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t) \frac{df}{d\theta}$$

REINFORCE algorithm:

- 1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$  (run it on the robot)
- 2.  $\nabla_{\theta} J(\theta) \approx \sum_i (\sum_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t^i | \mathbf{s}_t^i)) (\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))$
- 3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

Iteration 2000



# What did we just do?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \right) \left( \sum_{t=1}^T r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \underbrace{\sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\tau_i)}_{\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} r(\tau_i)$$

maximum likelihood:  $\nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau_i)$

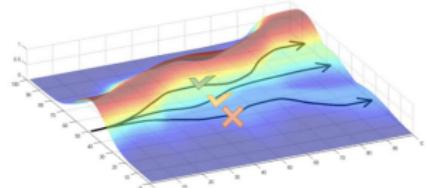
good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of “trial and error”!

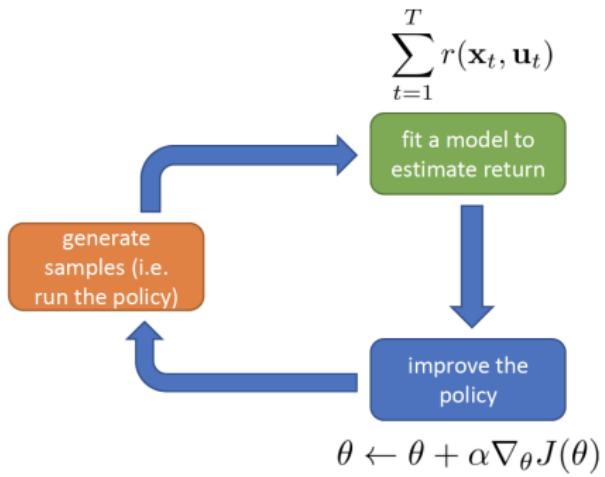
REINFORCE algorithm:

- 1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$  (run it on the robot)
- 2.  $\nabla_{\theta} J(\theta) \approx \sum_i (\sum_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t^i | \mathbf{s}_t^i)) (\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))$
- 3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$



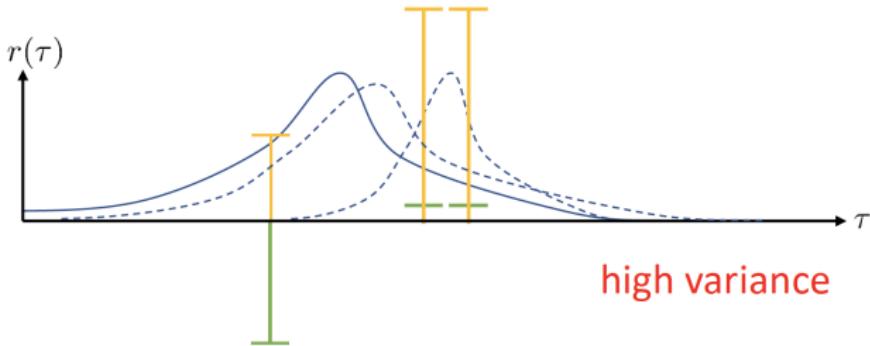
# Review of the policy gradient

- Evaluating the RL objective
  - Generate samples
- Evaluating the policy gradient
  - Log gradient trick
  - Generate samples
- Understand policy gradient
  - Formalization of trial-and-error



# What is wrong with the policy gradient?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)$$



- Even worse: what if the two “good” samples have  $r(\tau) = 0$ ?

# Reducing variance - Causality

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

$\Downarrow$

- **Causality:** policy at time  $t'$  cannot affect reward at time  $t$  when  $t < t'$



$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t} | s_{i,t}) \underbrace{\left( \sum_{t'=t}^{\textcolor{red}{T}} r(s_{i,t'}, a_{i,t'}) \right)}_{Q(s_{i,t}, a_{i,t})}$$

“reward to go”

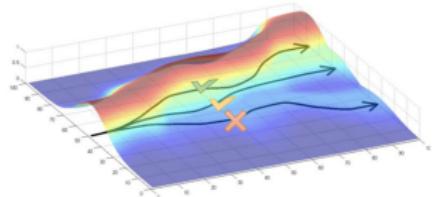
# Reducing variance - Baselines

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log \pi_{\theta}(\tau) [r(\tau) - b]$$

a convenient identity

$$\pi_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau) = \nabla_{\theta} \pi_{\theta}(\tau)$$

$$b = \frac{1}{N} \sum_{i=1}^N r(\tau)$$



- But... are we allowed to do that?

$$\begin{aligned}\mathbb{E}[\nabla_{\theta} \log \pi_{\theta}(\tau) b] &= \int \pi_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau) b d\tau = \int \nabla_{\theta} \pi_{\theta}(\tau) b d\tau \\ &= b \nabla_{\theta} \int \pi_{\theta}(\tau) d\tau = b \nabla_{\theta} 1 = 0\end{aligned}$$

- Subtracting a baseline is unbiased in expectation!
- Average reward is not the best baseline, but it's pretty good!

# Analyzing the variance

can we write down the variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)} [\nabla_{\theta} \log \pi_{\theta}(\tau) (r(\tau) - b)]$$

$$\text{Var} = E_{\tau \sim \pi_{\theta}(\tau)} [(\nabla_{\theta} \log \pi_{\theta}(\tau) (r(\tau) - b))^2] - E_{\tau \sim \pi_{\theta}(\tau)} [\nabla_{\theta} \log \pi_{\theta}(\tau) (r(\tau) - b)]^2$$

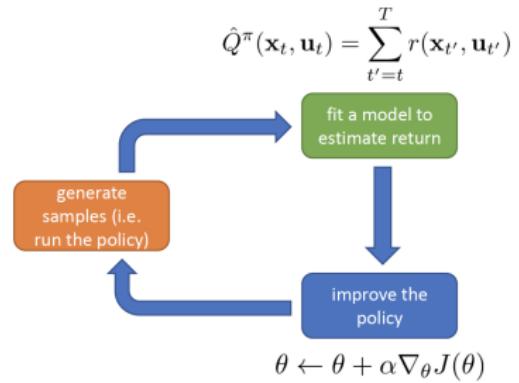
this bit is just  $E_{\tau \sim \pi_{\theta}(\tau)} [\nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)]$   
(baselines are unbiased in expectation)

$$\begin{aligned} \frac{d\text{Var}}{db} &= \frac{d}{db} E[g(\tau)^2 (r(\tau) - b)^2] = \frac{d}{db} (E[g(\tau)^2 r(\tau)^2] - 2E[g(\tau)^2 r(\tau)b] + b^2 E[g(\tau)^2]) \\ &= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0 \end{aligned}$$

$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]} \quad \leftarrow \quad \text{This is just expected reward, but weighted by gradient magnitudes!}$$

# Review

- The high variance of policy gradient
- Exploiting causality
  - Future doesn't affect the past
- Baselines
  - Unbiased!
- Analyzing variance
  - Can derive optimal baselines



# Policy gradient is on-policy

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

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



this is trouble...

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

can't just skip this!

REINFORCE algorithm:

- 
1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$  (run it on the robot)
  2.  $\nabla_{\theta} J(\theta) \approx \sum_i (\sum_t \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t^i | \mathbf{s}_t^i)) (\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))$
  3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

# Off-policy learning & importance sampling

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

what if we don't have samples from  $\pi_\theta(\tau)$ ?

(we have samples from some  $\bar{\pi}(\tau)$  instead)

$$J(\theta) = E_{\tau \sim \bar{\pi}(\tau)} \left[ \frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} r(\tau) \right]$$

$$\pi_\theta(\tau) = p(\mathbf{s}_1) \prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)} \prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t) \cancel{p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)} \prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t) \cancel{p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)}} = \frac{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \bar{\pi}(\mathbf{a}_t | \mathbf{s}_t)}$$

importance sampling

$$\begin{aligned} E_{x \sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x \sim q(x)} \left[ \frac{p(x)}{q(x)}f(x) \right] \end{aligned}$$

# Policy gradient with importance sampling

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

a convenient identity

$$\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

can we estimate the value of some *new* parameters  $\theta'$ ?

$$J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[ \frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} r(\tau) \right]$$

the only bit that depends on  $\theta'$

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[ \frac{\nabla_{\theta'} \pi_{\theta'}(\tau)}{\pi_\theta(\tau)} r(\tau) \right] = E_{\tau \sim \pi_\theta(\tau)} \left[ \frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau) \right]$$

now estimate locally, at  $\theta = \theta'$ :  $\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$

# The off-policy policy gradient

$$\theta^* = \arg \max_{\theta} J(\theta)$$

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

$$\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}$$

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[ \frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau) \right] \quad \text{when } \theta \neq \theta'$$

$$= E_{\tau \sim \pi_\theta(\tau)} \left[ \left( \prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)} \right) \left( \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \right) \left( \sum_{t=1}^T r(\mathbf{s}_t, \mathbf{a}_t) \right) \right] \text{ what about causality?}$$

$$= E_{\tau \sim \pi_\theta(\tau)} \left[ \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \underbrace{\left( \prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right)}_{\text{future actions don't affect current weight}} \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

# A first-order approximation for importance sampling

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[ \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \underbrace{\left( \prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right)}_{\text{exponential in } T...} \right]$$

let's write the objective a bit differently...

on-policy policy gradient:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

off-policy policy gradient:  $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

$= \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \cancel{\frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{s}_{i,t})}} \frac{\pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$

ignore this part

# Research outputs using off-policy policy gradient

## Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting

Zhi Wang<sup>✉</sup>, Student Member, IEEE, Han-Xiong Li<sup>✉</sup>, Fellow, IEEE, and Chunlin Chen<sup>✉</sup>, Member, IEEE

---

### Algorithm 1 Policy Relaxation With Importance Sampling

---

**Input:** Number of burn-in episodes  $k$ ;  
learning rate  $\alpha$ ; batch size  $m$   
**Output:** Optimal policy parameters  $\theta^*$

1 Initialize the number of learning episodes:  $\eta \leftarrow 0$   
2 **while** not converged **do**  
3   **if**  $\eta \leq k$  **then**  
4      $\pi_r(a|s) = \text{Uniform}(A(s)), \forall s$   
5     Sample  $m$  episodes from  $\pi_r$ :  $\tau^i \sim \pi_r$   
6      $\nabla_{\theta} J(\theta) = \sum_{i=1}^m \frac{\pi_{\theta}(\tau^i)}{\pi_r(\tau^i)} \nabla_{\theta} \log \pi_{\theta}(\tau^i) r(\tau^i)$   
7   **else**  
8     Sample  $m$  episodes from  $\pi_{\theta}$ :  $\tau^i \sim \pi_{\theta}$   
9      $\nabla_{\theta} J(\theta) = \sum_{i=1}^m \nabla_{\theta} \log \pi_{\theta}(\tau^i) r(\tau^i)$   
10   **end**  
11    $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$   
12    $\eta \leftarrow \eta + m$   
13 **end**

---

### B. Policy Relaxation

In the new environment  $M_t$ , the agent tends to visit a small part of the whole state-action space when executing the previously learned policy, thus probably leading to a local optimum due to insufficient exploration. Hence, we propose a policy relaxation mechanism to encourage a proper exploration. Specifically, in the  $k$  burn-in learning episodes, the agent is forced to execute a relaxed policy where actions are randomly selected from the available set. For better readability, let  $\theta$  denote the current parameters in  $M_t$ , and  $\pi_{\theta}$  be the policy derived from  $\theta$ . Regarding the number of learning episodes  $\eta$ , the agent's behavior policy  $\pi_r$  is relaxed as

$$\pi_r(a|s) = \begin{cases} \text{Uniform}(A(s)), & \eta \leq k \\ \pi_{\theta}(a|s), & \eta > k \end{cases} \quad (9)$$

# Policy gradient with automatic differentiation

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \underbrace{\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}}_{\text{pretty inefficient to compute these explicitly!}}$$

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

maximum likelihood:  $\nabla_{\theta} J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})$        $J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t})$

Just implement “pseudo-loss” as a weighted maximum likelihood:

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

 cross entropy (discrete) or squared error (Gaussian)

# Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Maximum likelihood:

```
# Given:  
# actions - (N*T) x Da tensor of actions  
# states - (N*T) x Ds tensor of states  
# Build the graph:  
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits  
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)  
loss = tf.reduce_mean(negative_likelihoods)  
gradients = loss.gradients(loss, variables)
```

# Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:  
# actions - (N*T) x Da tensor of actions  
# states - (N*T) x Ds tensor of states  
# q_values - (N*T) x 1 tensor of estimated state-action values  
# Build the graph:  
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits  
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)  
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)  
loss = tf.reduce_mean(weighted_negative_likelihoods)  
gradients = loss.gradients(loss, variables)
```

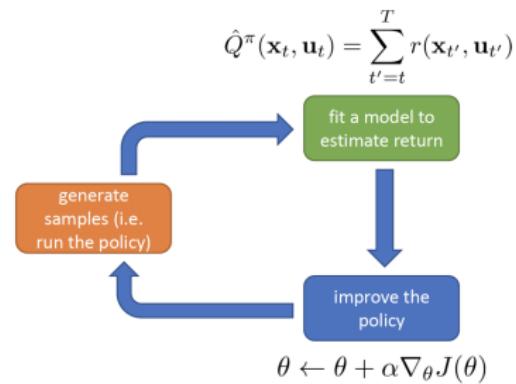
$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \circ \hat{Q}_{i,t}$$

# Policy gradient in practice

- Remember that the gradient has high variance
  - This isn't the same as supervised learning!
  - Gradients will be really noisy!
- Consider using much larger batches
- Tweaking learning rates is very hard
  - Using adaptive step size rules like ADAM
  - More policy gradient specific learning rate adjustment methods...

# Review

- Policy gradient is on policy
- Can derive off policy variant
  - Use importance sampling
  - Exponential scaling in  $T$
  - Can ignore state portion (first-order approximation)
- Can implement with automatic differentiation – need to know what to backpropagate
- Practical considerations: batch size, learning rates, optimizers



# Table of Contents

- 1 From Tabular Algorithms to Deep RL
- 2 Policy Gradients
- 3 Advanced Policy Gradient Methods

# Problems of vanilla policy gradient (REINFORCE)

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) Q(s_{i,t}, a_{i,t})$$
$$\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$$

- Hard to select the step size  $\alpha$ 
  - Too big step: Bad policy  $\rightarrow$  data collected under bad policy  $\rightarrow$  we cannot recover (in Supervised Learning, data does not depend on neural network weights)
  - Too small step: Not efficient use of experience (in Supervised Learning, data can be trivially re-used)

# Problems of vanilla policy gradient (REINFORCE)



- Small changes in the policy parameters can unexpectedly lead to big **changes** in the policy

# Gradient descent in parameter space

- The step size in gradient descent results from solving the following optimization problem, e.g., using line search

$$d^* = \arg \max_{\|d\| \leq \epsilon} J(\theta + d)$$

- Euclidean distance in parameter space
- Stochastic gradient descent (SGD)

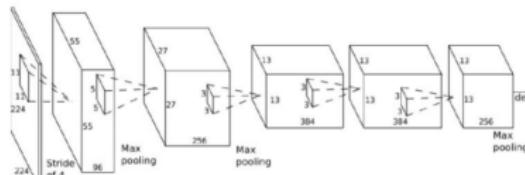
$$\theta \leftarrow \theta + d^*$$

# Hard to pick the threshold $\epsilon$

- It is hard to predict the result on the parameterized distribution
  - Especially for nonlinear function approximators, e.g., neural networks



$s_t$



$$\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t)$$



$\mathbf{a}_t$



$$\mu_{\theta}(s)$$
$$\sigma_{\theta}(s)$$

# Gradient descent in distribution space

- Gradient descent in parameter space

$$d^* = \arg \max_{\|d\| \leq \epsilon} J(\theta + d)$$

- **Natural gradient descent**: the step size in parameter space is determined by considering the KL divergence in the distributions before and after the update

$$d^* = \arg \max_d J(\theta + d), \quad s.t. D_{KL}(\pi_\theta || \pi_{\theta+d}) \leq \epsilon$$

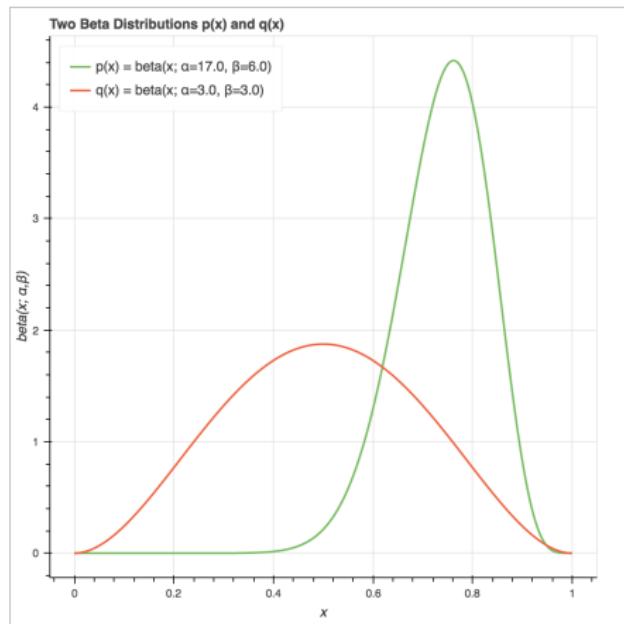
- **KL divergence** in distribution space
- Easier to pick the distance threshold!!!

# Distance for probability distributions

- How to calculate the distance between two points in a 2D coordinate?

$$\text{distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Euclidean distance



- How to calculate the distance between two **probability distributions**,  $p(x)$  and  $q(x)$ ?

# Kullback-Leibler (KL) divergence

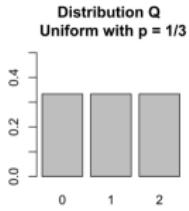
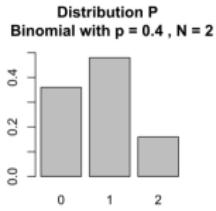
- A measure of how one probability distribution,  $p(x)$ , is different from a second, reference probability distribution,  $q(x)$

$$D_{\text{KL}}(p(x)||q(x)) = \sum_i p(x_i) \log \frac{p(x_i)}{q(x_i)}$$

$$D_{\text{KL}}(p(x)||q(x)) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

- A KL divergence of 0 indicates that the two distributions are identical

# Kullback-Leibler (KL) divergence: An example



$$\begin{aligned} D_{\text{KL}}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \ln \left( \frac{P(x)}{Q(x)} \right) \\ &= 0.36 \ln \left( \frac{0.36}{0.333} \right) + 0.48 \ln \left( \frac{0.48}{0.333} \right) + 0.16 \ln \left( \frac{0.16}{0.333} \right) \\ &= 0.0852996 \\ D_{\text{KL}}(Q \parallel P) &= \sum_{x \in \mathcal{X}} Q(x) \ln \left( \frac{Q(x)}{P(x)} \right) \\ &= 0.333 \ln \left( \frac{0.333}{0.36} \right) + 0.333 \ln \left( \frac{0.333}{0.48} \right) + 0.333 \ln \left( \frac{0.333}{0.16} \right) \\ &= 0.097455 \end{aligned}$$

x	0	1	2
Distribution P(x)	0.36	0.48	0.16
Distribution Q(x)	0.333	0.333	0.333

# Back to natural gradient descent

- How to solve this constrained optimization problem?

$$d^* = \arg \max_d J(\theta + d), \quad s.t. D_{\text{KL}}(\pi_\theta || \pi_{\theta+d}) \leq \epsilon$$

- Use the **Lagrangian multiplier**  $\lambda$ , turn to the **unconstrained penalized objective**

$$d^* = \arg \max_d J(\theta + d) - \lambda(D_{\text{KL}}(\pi_\theta || \pi_{\theta+d}) - \epsilon)$$

# Taylor expansion for the unconstrained penalized objective

$$d^* = \arg \max_d J(\theta + d) - \lambda(D_{\text{KL}}(\pi_\theta || \pi_{\theta+d}) - \epsilon)$$

- First-order Taylor expansion for the loss

$$J(\theta + d) \approx J(\theta) + \nabla_{\theta'} J(\theta')|_{\theta'=\theta} \cdot d$$

- Second-order Taylor expansion for the KL

$$D_{\text{KL}}(\pi_\theta || \pi_{\theta+d}) \approx \frac{1}{2} d^T \cdot \nabla_{\theta'}^2 D_{\text{KL}}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} \cdot d$$

# Taylor series/expansion

- A representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point

$$\begin{aligned}f(x) &= \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n \\&= f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2 + \dots\end{aligned}$$

- Examples

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \dots$$

# Taylor expansion of KL

$$D_{KL}(\pi_\theta || \pi_{\theta'}) \approx D_{KL}(\pi_\theta || \pi_\theta) + d^T \nabla_{\theta'} D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} + \frac{1}{2} d^T \nabla_{\theta'}^2 D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} d$$

$$D_{KL}(\pi_\theta || \pi_{\theta'}) = \int \pi_\theta(x) \log \frac{\pi_\theta(x)}{\pi_{\theta'}(x)} dx = \underbrace{\int \pi_\theta(x) \log \pi_\theta(x) dx}_{\text{independent of } \theta'} - \int \pi_\theta(x) \log \pi_{\theta'}(x) dx$$

$$\begin{aligned}\nabla_{\theta'} D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} &= -\nabla_{\theta'} \int \pi_\theta(x) \log \pi_{\theta'}(x) dx|_{\theta'=\theta} \\ &= -\int \pi_\theta(x) \nabla_{\theta'} \log \pi_{\theta'}(x) dx|_{\theta'=\theta} \\ &= -\int \frac{\pi_\theta(x)}{\pi_{\theta'}(x)} \nabla_{\theta'} \pi_{\theta'}(x) dx|_{\theta'=\theta} \\ &= -\nabla_{\theta'} \int \pi_{\theta'}(x) dx|_{\theta'=\theta} \\ &= 0\end{aligned}$$

# Taylor expansion of KL

$$D_{KL}(\pi_\theta || \pi_{\theta'}) \approx D_{KL}(\pi_\theta || \pi_\theta) + d^T \nabla_{\theta'} D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} + \frac{1}{2} d^T \nabla_{\theta'}^2 D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} d$$

$$\begin{aligned}\nabla_{\theta'}^2 D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} &= - \int \pi_\theta(x) \nabla_{\theta'}^2 \log \pi_{\theta'}(x) dx|_{\theta'=\theta} \\ &= - \int \pi_\theta(x) \frac{\pi_{\theta'}(x) \nabla_{\theta'}^2 \pi_{\theta'}(x) - \nabla_{\theta'} \pi_{\theta'}(x) \nabla_{\theta'} \pi_{\theta'}(x)^T}{\pi_{\theta'}(x)^2} dx|_{\theta'=\theta} \\ &= - \underbrace{\nabla_{\theta'}^2 \int \pi_{\theta'}(x) dx|_{\theta'=\theta}}_0 + \int \pi_\theta(x) \nabla_{\theta'} \log \pi_{\theta'}(x) \nabla_{\theta'} \log \pi_{\theta'}(x)^T dx|_{\theta'=\theta} \\ &= \mathbb{E}_{x \sim \pi_\theta} [\nabla_{\theta'} \log \pi_{\theta'}(x) \nabla_{\theta'} \log \pi_{\theta'}(x)^T|_{\theta'=\theta}]\end{aligned}$$

# Hessian of KL = Fisher information matrix (FIM)

- **Hessian:** A square matrix of second-order partial derivatives of a scalar-valued function, which describes the local curvature of a function of many variables

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- **Fisher information:** a way of measuring the amount of information that an observable random variable  $X$  carries about an unknown parameter  $\theta$  upon which the probability of  $X$  depends

$$\mathbf{F}(\theta) = \mathbb{E}_{x \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(x) \nabla_\theta \log \pi_\theta(x)^T]$$

# Hessian of KL = Fisher information matrix (FIM)

- The FIM is exactly the **Hessian matrix** of KL divergence

$$\underbrace{\nabla_{\theta'}^2 D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta}}_{\text{Hessian of KL}} = \underbrace{\mathbb{E}_{x \sim \pi_\theta} [\nabla_{\theta'} \log \pi_{\theta'}(x) \nabla_{\theta'} \log \pi_{\theta'}(x)^T |_{\theta'=\theta}]}_{\text{FIM}}$$

$$\begin{aligned} D_{KL}(\pi_\theta || \pi_{\theta'}) &\approx \underbrace{D_{KL}(\pi_\theta || \pi_\theta)}_0 + d^T \underbrace{\nabla_{\theta'} D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta}}_0 + \frac{1}{2} d^T \underbrace{\nabla_{\theta'}^2 D_{KL}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta}}_{\mathbf{F}(\theta)} d \\ &= \frac{1}{2} d^T \mathbf{F}(\theta) d \\ &= \frac{1}{2} (\theta' - \theta)^T \mathbf{F}(\theta) (\theta' - \theta) \end{aligned}$$

## Back to Taylor expansion of KL

$$D_{KL}(\pi_\theta || \pi_\theta + d) \approx \frac{1}{2} d^T \mathbf{F}(\theta) d$$

- KL divergence is roughly analogous to a distance measure between distributions
- Fisher information serves as a local distance metric between distributions: how much you change the distribution if you move the parameters a little bit in a given direction

## Back to solving the KL constrained problem

$$\begin{aligned} d^* &= \arg \max_d J(\theta + d) - \lambda (\text{D}_{\text{KL}}(\pi_\theta || \pi_{\theta+d}) - \epsilon) \\ &\approx \arg \max_d J(\theta) + \nabla_{\theta'} J(\theta')|_{\theta'=\theta} \cdot d - \lambda \left( \frac{1}{2} d^T \nabla_{\theta'}^2 \text{D}_{\text{KL}}(\pi_\theta || \pi_{\theta'})|_{\theta'=\theta} d - \epsilon \right) \\ &= \arg \max_d \nabla_{\theta'} J(\theta')|_{\theta'=\theta} \cdot d - \frac{1}{2} \lambda d^T \mathbf{F}(\theta) d \end{aligned}$$

- Set the gradient to 0:

$$\begin{aligned} 0 &= \frac{\partial}{\partial d} \left( \nabla_{\theta'} J(\theta')|_{\theta'=\theta} \cdot d - \frac{1}{2} \lambda d^T \mathbf{F}(\theta) d \right) \\ &= \nabla_{\theta'} J(\theta')|_{\theta'=\theta} - \lambda \mathbf{F}(\theta) d \end{aligned}$$

$$d^* = \frac{1}{\lambda} \mathbf{F}^{-1}(\theta) \nabla_{\theta'} J(\theta')|_{\theta'=\theta} = \frac{1}{\lambda} \mathbf{F}^{-1}(\theta) \nabla_{\theta} J(\theta)$$

# Natural gradient descent

- The natural gradient:

$$\tilde{\nabla}_{\theta} J(\theta) = \mathbf{F}^{-1}(\theta) \nabla_{\theta} J(\theta)$$

- Natural gradient descent:

$$\theta' = \theta + \alpha \cdot \mathbf{F}^{-1}(\theta) \hat{g}$$

- How to determine the learning rate  $\alpha$ :

$$D_{KL}(\pi_{\theta} || \pi_{\theta} + d) \approx \frac{1}{2} (\theta' - \theta)^T \mathbf{F}(\theta) (\theta' - \theta) \leq \epsilon$$

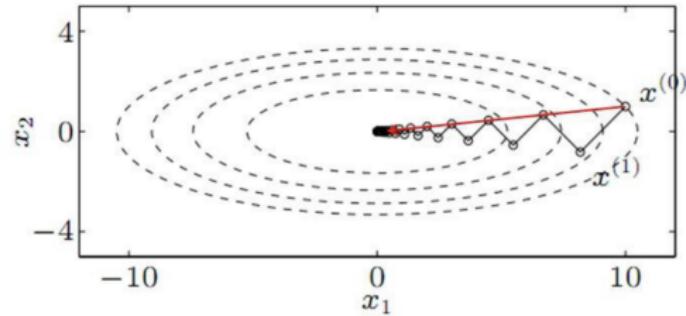
$$\frac{1}{2} (\alpha \hat{g})^T \mathbf{F}(\alpha \hat{g}) = \epsilon$$

$$\boxed{\alpha = \sqrt{\frac{2\epsilon}{\hat{g}^T \mathbf{F} \hat{g}}}}$$

# Geometric interpretation of natural policy gradient

- Find **the steepest direction** for parameter updating

Essentially the same problem as this:



# Natural gradient descent → Natural policy gradient (NPG)

---

**Algorithm 1** Natural Policy Gradient

---

Input: initial policy parameters  $\theta_0$

**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:

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

**end for**

---

- Originated from **natural gradient descent** in supervised learning
- Very **expensive** to compute the **inverse of Hessian matrix** for a large number of parameters

# Review of natural policy gradient

- The gradient
  - Constrain parameter update in parameter space (using Euclidean distance)
- The natural gradient
  - Constrain parameter update in distribution space (using KL divergence)
  - The meaning of “natural”: the distance metric is **invariant** to function parameterization
- Fisher information matrix (FIM)
  - Second-order information: a local distance metric between distributions
  - The FIM is exactly the Hessian matrix of KL divergence
  - Expensive to compute for a large number of parameters

# Trust region policy optimization (TRPO)

- John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel, **Trust Region Policy Optimization**, ICML, 2015.
- The family of **statistical learning**
  - John Schulman → Pieter Abbeel → Andrew Ng → Michael Jordan

## John Schulman's Homepage

I'm a research scientist at [OpenAI](#). I co-lead the reinforcement learning (RL) team, where we work on (1) designing better RL algorithms that enable agents to learn much faster in novel situations; (2) designing better training environments that teach agents transferrable skills. We mostly use [games](#) as a [testbed](#).



Previously, I received my [PhD](#) in Computer Science from UC Berkeley, where I had the good fortune of being advised by [Pieter Abbeel](#). Prior to my recent work in RL, I spent some time working on robotics, enabling robots to [tie knots](#) and [stitches](#) and plan movement using [trajectory optimization](#).

- [Publications](#)
- [Presentations](#)
- [Code](#)
- [Awards](#)

Email: [joschu@openai.com](mailto:joschu@openai.com).

# Trust region policy optimization (TRPO)



Michael I. Jordan

[FOLLOW](#)

Professor of EECS and Professor of Statistics, [University of California, Berkeley](#).

Verified email at cs.berkeley.edu - [Homepage](#)

machine learning statistics computational biology artificial intelligence optimization

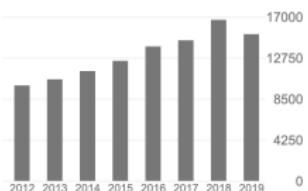
TITLE	CITED BY	YEAR
<a href="#">Latent dirichlet allocation</a> DM Blei, AY Ng, MI Jordan <i>Journal of machine Learning research</i> 3 (Jan), 993-1022	29247	2003
<a href="#">On spectral clustering: Analysis and an algorithm</a> AY Ng, MI Jordan, Y Weiss <i>Advances in neural information processing systems</i> , 849-856	7927	2002
<a href="#">Adaptive mixtures of local experts.</a> RA Jacobs, MI Jordan, SJ Nowlan, GE Hinton <i>Neural computation</i> 3 (1), 79-87	4089	1991

Cited by

[VIEW ALL](#)

All Since 2014

	All	Since 2014
Citations	165762	84682
h-index	160	114
i10-index	540	425



# TRPO - The KL constrained problem

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

- ▶ Also worth considering using a penalty instead of a constraint

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

Again the KL penalized problem!

---

**Algorithm 3** Trust Region Policy Optimization

---

Input: initial policy parameters  $\theta_0$

**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**

---

# Line search with monotonic policy improvement

---

**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**

---

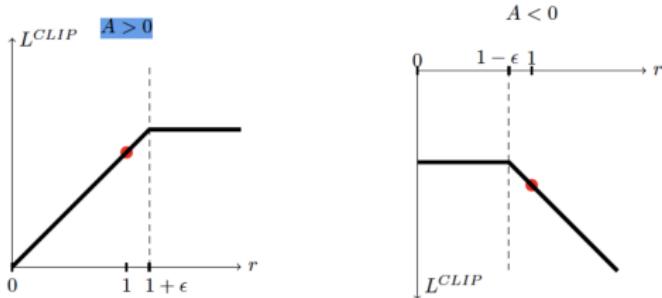
# Proximal policy optimization (PPO): Clipped objective

- Recall the surrogate objective

$$L^{IS}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t [r_t(\theta) \hat{A}_t].$$

- Form a lower bound via clipped importance ratios

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right]$$



- Prevent large changes of policies, constrain the policy update
- Achieve similar performance to TRPO without second-order information (no Fisher matrix!)

# Proximal policy optimization (PPO): Adaptive KL penalty

Input: initial policy parameters  $\theta_0$ , initial KL penalty  $\beta_0$ , target KL-divergence  $\delta$

**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**

Don't use second order approximation for KL which is expensive, use standard gradient descent

- Penalty coefficient  $\beta$  changes between iterations to approximately enforce KL-divergence constraint
- Achieve similar performance to TRPO without second-order information (no Fisher matrix!)

# Review

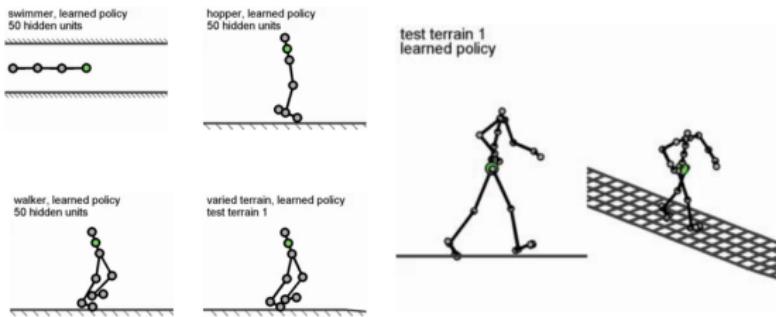
- TRPO: again the KL penalty problem
  - Natural policy gradient + Monotonic policy improvement + Line search
  - Still need to compute the natural gradient with Hessian matrix
- PPO
  - Achieve TRPO-like performance without second-order computation
  - Clipped objective, adaptive KL penalty

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

# Example: policy gradient with importance sampling

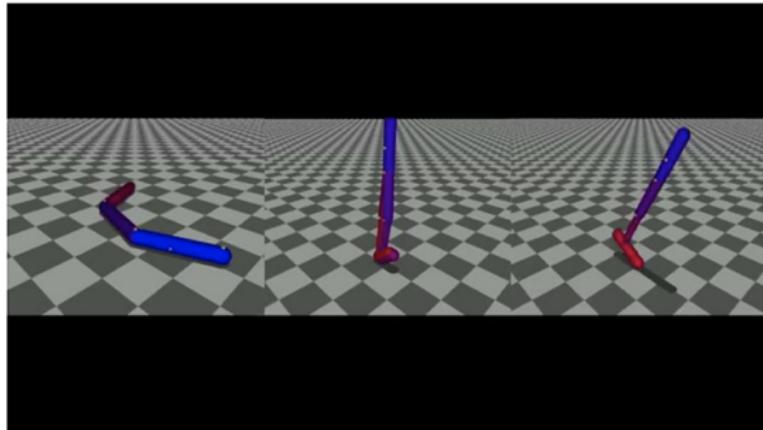
$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_{\theta}(\tau)} \left[ \sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t | \mathbf{s}_t) \left( \prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'} | \mathbf{s}_{t'})}{\pi_{\theta}(\mathbf{a}_{t'} | \mathbf{s}_{t'})} \right) \left( \sum_{t'=t}^T r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

- Incorporate example demonstrations using importance sampling
- Neural network policies



## Example: trust region policy optimization

- Natural gradient with automatic step adjustment
- Discrete and continuous actions
- Code available (see Duan et al. '16)



# Learning objectives of this lecture

- You should be able to...
  - Understand the function approximation mechanism for RL problems with large or continuous state-action spaces
  - Understand and be able to use the vanilla policy gradient method
  - Be able to use the baseline to reduce the variance of policy gradient
  - Know the importance sampling technique for off-policy policy gradient
  - Know the implementation tricks in practice
  - Be aware of several advanced algorithms, natural policy gradient, TRPO, PPO
  - Enhance your mathematical skills

# References

- Lecture 4 & 5 & 9 of CS285 at UC Berkeley, *Deep Reinforcement Learning, Decision Making, and Control*
  - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-4.pdf>
  - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf>
  - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-9.pdf>
- Classic papers
  - Williams (1992). **Simple statistical gradient following algorithms for connectionist reinforcement learning**: introduces REINFORCE algorithm.
  - Peters & Schaal (2008). **Reinforcement learning of motor skills with policy gradients**: very accessible overview of optimal baselines and natural gradient.
- DRL policy gradient papers
  - Schulman, L., Moritz, Jordan, Abbeel (2015). **Trust region policy optimization**: deep RL with natural policy gradient and adaptive step size.
  - Schulman, Wolski , Dhariwal , Radford, Klimov (2017). **Proximal policy optimization algorithms**: deep RL with importance sampled policy gradient.
  - Y. Duan, et al., **Benchmarking Deep Reinforcement Learning for Continuous Control**, ICML, 2016.
  - Z. Wang, et al., **Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting**, TNNLS, 2019.

# THE END