Policy Gradient Descent

Lucas Janson

CS/Stat 184(0): Introduction to Reinforcement Learning

Fall 2024

Today

- Feedback from last lecture
- Recap+
- Gradient Descent (ok this is also sort of recap)
- Policy Gradient
- Likelihood ratio method

Feedback from feedback forms

1. Thank you to everyone who filled out the forms!
2.

Today

- ✓ Feedback from last lecture
- Recap+
- Gradient Descent (ok this is also sort of recap)
- Policy Gradient
- Likelihood ratio method

Q-Value Dynamic Programming Algorithm:

Recall from HW1 the Bellman equations for $Q^\star$:

$$Q_h^\star(s,a) = r(s,a) + \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[\max_{a'} Q_{h+1}^\star(s',a')\right]$$

Analogous Q-value DP, with same notational change as last lecture: h as argument

1. Initialization: $Q(s,a,H) = 0 \quad \forall s,a$
2. Solve (via dynamic programming):

$$Q(s,a,h) = r(s,a) + \mathbb{E}_{s' \sim P(s,a)}\left[\max_{a' \in A} Q(s',a',h+1)\right] \quad \forall s,a,h$$

3. Return:

$$\pi_h(s) = \arg\max_a \left\{ Q(s,a,h) \right\}$$

What if we can't just evaluate the expectations?

If S and/or A are very large, computing expectations could be very expensive

We may not have a way to directly compute those expectations, but instead only have access to a simulator (or the real world), where we can collect data

Suppose:

This is now full RL!!

- We have N trajectories $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

  Each trajectory is of the form $\tau_i = \{s_0^i, a_0^i, \ldots s_{H-1}^i, a_{H-1}^i, s_H^i\}$

- π_{data} is often referred to as our data collection policy.

Want: $Q(s, a, h) \approx r(s, a) + \mathbb{E}_{s' \sim P(s,a)} \left[\max_{a' \in A} Q(s', a', h+1) \right] \quad \forall s, a, h$

Since we're trying to approximate conditional expectations, seems like it kind of fits into supervised learning—can we use an approach like that? Yes!

Connection to Supervised Learning

$$Q(s,a,h) \approx r(s,a) + \mathbb{E}_{s' \sim P(s,a)} \left[\max_{a' \in A} Q(s', a', h+1) \right] \quad \forall s, a, h$$

What are the y and x?

Note that the RHS can also be written as

$$\mathbb{E}\left[r(s_h, a_h) + \max_{a'} Q(s_{h+1}, a', h+1) \middle| s_h, a_h, h \right]$$

This suggests that $y = r(s_h, a_h) + \max_{a'} Q(s_{h+1}, a', h+1)$ and $x = (s_h, a_h, h)$

Then we'd be happy if we found a

$$Q(s_h, a_h, h) = f(x) = \mathbb{E}[y \,|\, x] = \mathbb{E}\left[r(s_h, a_h) + \max_{a'} Q(s_{h+1}, a', h+1) \middle| s_h, a_h, h \right]$$

Connection to Supervised Learning (cont'd)

We can convert our data $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$, into (y, x) pairs; how many? NH

BUT, to compute each y, we need to already know Q!

Setting that aside for the moment, to fit supervised learning, we'd minimize a least-squares objective function: $\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{NH} (y_i - f(x_i))^2$

Then if we have enough data, choose a good $\mathcal{F}$, and optimize well,

$$Q(s_h, a_h, h) := \hat{f}(x) \approx \mathbb{E}[y \,|\, x] = \mathbb{E}\left[r(s_h, a_h) + \max_{a'} Q(s_{h+1}, a', h+1) \,\middle|\, s_h, a_h, h \right]$$

Fitted (Q-)Value Iteration

To address the circularity problem of not knowing Q for computing the y, we have an algorithmic tool… what is it?
Hint: we used it for another VI algorithm before…

Fixed point iteration! Initialize, then at each step, pretend Q is known by plugging in the previous time step's Q to compute the y's, and then use that to get next Q

Input: **offline dataset** $\tau_1, \dots \tau_N \sim \rho_{\pi_{data}}$

1. Initialize fitted Q function at f_0
2. For $k = 1, \dots, K$:

$$f_k = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \sum_{h=1}^{H-1} \left(f(s_h^i, a_h^i, h) - \left(r(s_h^i, a_h^i) + \max_a f_{k-1}(s_{h+1}^i, a, h+1) \right) \right)^2$$

3. With f_K as an estimate of $Q^\star$, return $\pi_h(s) = \arg\max_a \left\{ f^K(s, a, h) \right\}$

Q-Learning is an online version, i.e., draw new trajectories at each k based on f_k as Q-function

Bonus: Q-learning

(Tabular) Q-Learning

- Initialize: $Q(s, a, h)$
- For $k = 1,2,\ldots K$ episodes
  - Within each episode, for $h = 0,1,\ldots H-1$
    - Act: choose actions however you like! (but try to maintain exploration)
    - Update:

$$Q(s_h, a_h, h) \leftarrow Q(s_h, a_h, h) - \eta \left(Q(s_h, a_h, h) - r(s_h, a_h) - \max_a Q(s_{h+1}, a, h+1) \right)$$

- Return $Q(s, a, h)$

- Update step is a single stochastic gradient step of size η
- Q-learning is online: actions are taken within the algorithm
- Q-learning is an “off-policy” algorithm.
- Guarantee: Assuming states, actions visited infinitely often (which can be guaranteed with the action policy), $Q \to Q^\star$.

Q-Learning with Function Approximation (extra material: read later if interested)

- Init: $Q(s, a, h)$
- For $k = 1,2,\ldots K$ episodes
  - Within each episode, for $h = 0,1,\ldots H-1$
    - Act: choose actions however you like!
      (but try to maintain exploration)
    - Update:
      $$\theta \leftarrow \theta - \eta\Big(f_\theta(s_h, a_h, h) - r(s_h, a_h) - \max_a f_\theta(s_{h+1}, a, h+1)\Big)\nabla f_\theta(s_h, a_h, h)$$
- Return $Q(s, a, h)$

- How to understand this expression?
  Consider doing a small step of SGD on the fitted-Q objective function.

Recall: Policy Iteration (PI)

- Initialization: choose a policy $\pi^0 : S \mapsto A$
- For $k = 0,1,\ldots$
  1. **Policy Evaluation**: Solve (via dynamic programming):
  $$Q^{\pi^k}(s,a,h) = r(s,a) + \mathbb{E}_{s' \sim P(\cdot|s,a)}\left[Q^{\pi^k}(s', \pi^k(s), h+1)\right] \quad \forall s,a,h$$
  2. **Policy Improvement**: set $\pi_h^{k+1}(s) := \arg\max_a Q^{\pi^k}(s,a,h)$

Again: what if we're in full RL setting where we can't just evaluate expectations?

This breaks the Policy Evaluation step, so can we do a fitted version?

Yes! RHS can be written as $\mathbb{E}\left[r(s_h,a_h) + Q^{\pi^k}(s_{h+1}, \pi^k(s_h), h+1) \middle| s_h, a_h, h\right]$

Spot the difference!

Fitted Policy Evaluation

Use exact same strategy as before: **fixed point iteration**

Input: policy π, **dataset** $\tau_1, \ldots \tau_N \sim \rho_\pi$

1. Initialize fitted Q^π function at f_0
2. For $k = 1, \ldots, K$:

$$f_k = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \sum_{h=1}^{H-1} \Big(f(s_h^i, a_h^i, h) - \big(r(s_h^i, a_h^i) + f_{k-1}(s_{h+1}^i, \pi(s_h^i), h+1) \big) \Big)^2$$

3. Return the function f_K as an estimate of Q^π

Fitted Policy Iteration:

- Initialization: choose a policy $\pi^0 : S \mapsto A$ and a sample size N
- For $k = 0,1,\ldots$
    1. **Fitted Policy Evaluation**: Using N sampled trajectories $\tau_1, \ldots \tau_N \sim \rho_{\pi^k}$, obtain approximation $\hat{Q}^{\pi^k} \approx Q^{\pi^k}$
    2. **Policy Improvement**: set $\pi_h^{k+1}(s) := \arg\max_a \hat{Q}^{\pi^k}(s, a, h)$

(Another) Fitted Policy Evaluation option

Using the definition of the Q function, can do a non-iterative fitted policy evaluation

$$Q^\pi(s,a,h) = \mathbb{E}\left[\sum_{t=h}^{H-1} r(s_t,a_t) \middle| s_h, a_h, h\right]$$

Input: policy π, **dataset** $\tau_1, \ldots \tau_N \sim \rho_\pi$

Return:

$$\hat{Q}^\pi = \arg\min_{f\in\mathcal{F}} \sum_{i=1}^{N}\sum_{h=1}^{H-1}\left(f(s_h^i, a_h^i, h) - \sum_{t=h}^{H-1} r(s_t^i, a_t^i)\right)^2$$

Bonus: TD(0)

(see posted slides)

The “tabular” TD(0) Algorithm of Q^π

- Init: $Q^\pi(s, a, h)$
- For $k = 1,2,\ldots K$ episodes
  - Within each episode, for $h = 0,1,\ldots H-1$
    - Act according to π
    - update:
      $Q^\pi(s_h, a_h, h) \leftarrow Q^\pi(s_h, a_h, h) - \eta \left(Q^\pi(s_h, a_h, h) - r(s_h, a_h) - Q^\pi(s_{h+1}, a_{h+1}, h+1)\right)$
- Return $Q^\pi(s, a, h)$

- Just like Q-learning, TD(0) is an “online” approach for policy evaluation.
  - It can be helpful for variance reduction.
- Recall Bellman consistency conditions for Q^π:
  $Q^\pi(s, a, h) = r(s, a) + \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[Q^\pi(s', \pi_{h+1}(s'), h+1)\right]$
- “TD” stands for “Temporal Difference”

TD(0) Algorithm for Q^π, with function approximation

- Init: $Q(s, a, h)$
- For $k = 1,2,\ldots K$ episodes
  - Within each episode, for $h = 0,1,\ldots H-1$
    - Act according to π
    - update:

$$\theta \leftarrow \theta - \eta\Big(f_\theta(s_h, a_h, h) - r(s_h, a_h) - f_\theta(s_{h+1}, a_{h+1}, h+1)\Big)\nabla f_\theta(s_h, a_h, h)$$

- Return $Q(s, a, h)$

- Again, this is an "online" approach for policy evaluation, but with function approximation.

Today

- ✓ Feedback from last lecture
- ✓ Recap+
- Gradient Descent (ok this is also sort of recap)
- Policy Gradient
- Likelihood ratio method

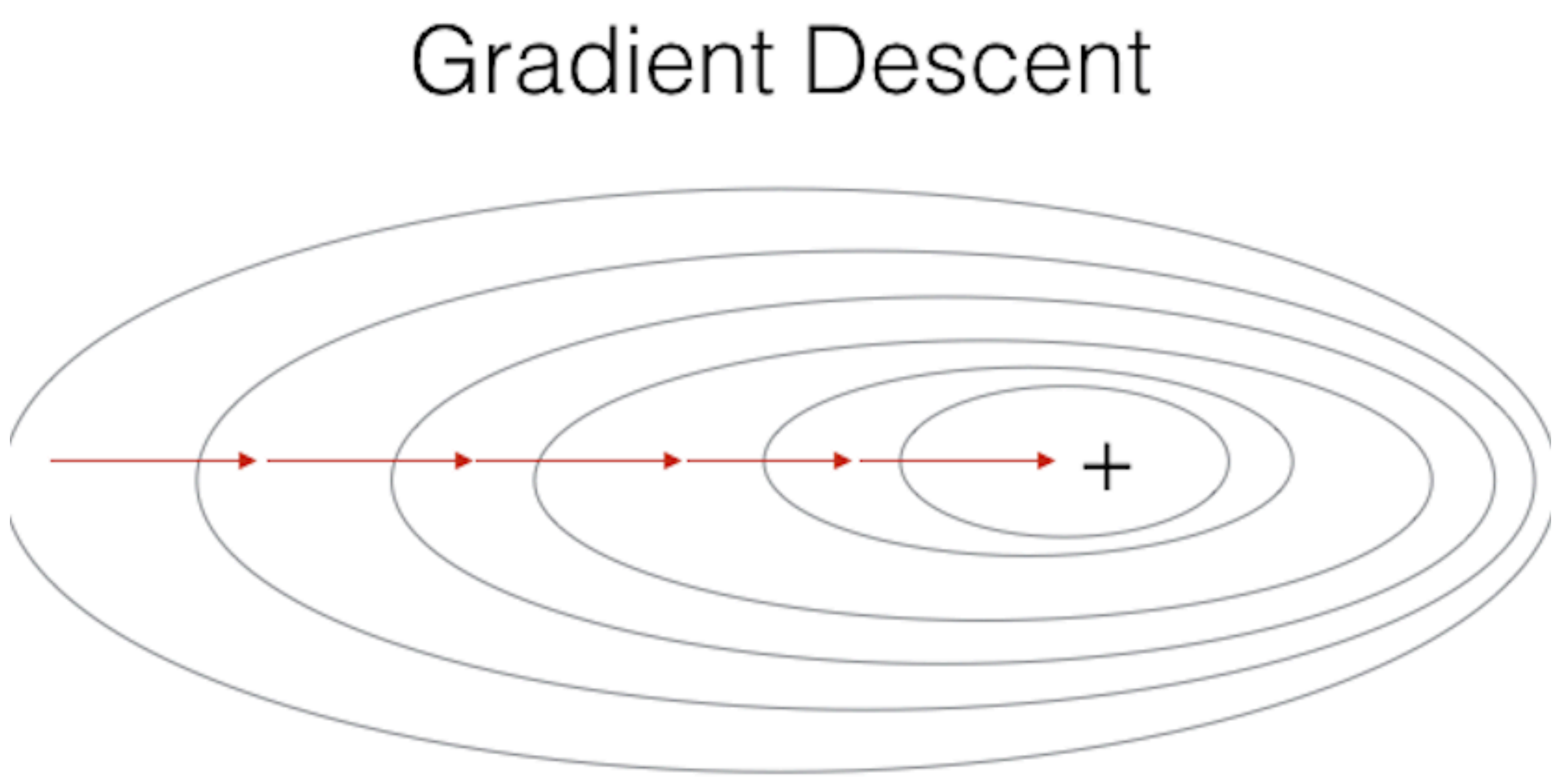
Gradient Descent (GD) and Stochastic Gradient Descent (SGD)

(we really do *ascent* in RL, so we should say GA and SGA)

- Given an objective function

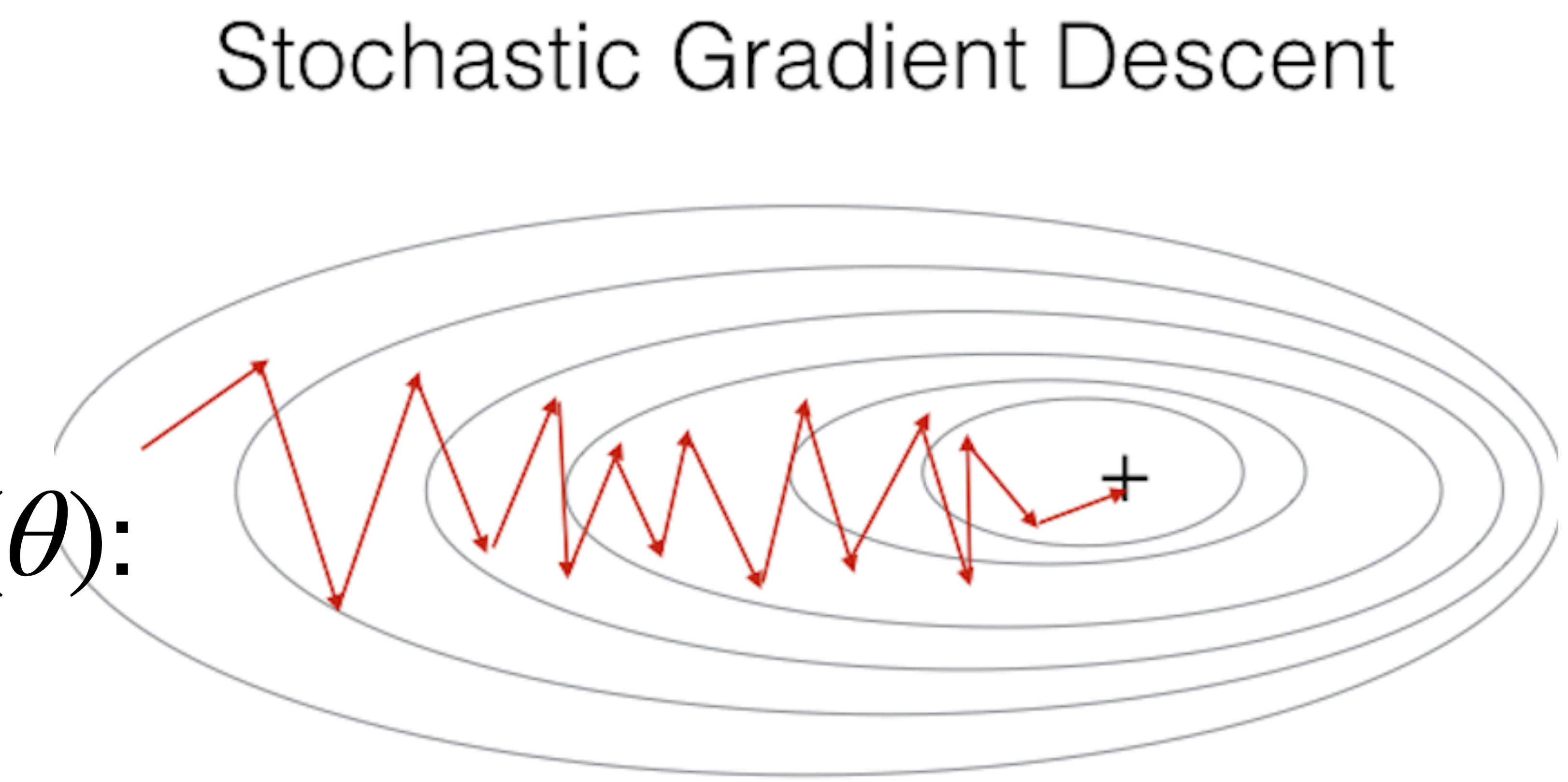
  $$J(\theta) : \mathbb{R}^d \mapsto \mathbb{R}, \quad \text{(e.g., } J(\theta) = \mathbb{E}_{x,y}(f_\theta(x) - y)^2\text{)},$$

  our objective is: $\min_\theta J(\theta)$



- Gradient Descent is an iterative approach, to decrease the objective function as follows:
  - Initialize θ^0, for $k = 0, \ldots$:

    $$\theta^{k+1} = \theta^k - \eta \nabla J(\theta^k)$$



- Stochastic Gradient Descent uses (unbiased) estimates of $\nabla J(\theta)$:
  - Initialize θ^0, for $k = 0, \ldots$:

    $$\theta^{k+1} = \theta^k - \eta^k g^k, \quad \text{where } \mathbb{E}[g^k] = \nabla_\theta J(\theta^k)$$

Example of GD

- Given an objective function

$$J(\theta) : \mathbb{R} \mapsto \mathbb{R}, \quad J(\theta) = \frac{1}{2}(\theta - c)^2,$$

  our objective is: $\min_{\theta} J(\theta)$

- We have $\nabla J(\theta) = \theta - c$, so GD is:
  - Initialize $\theta^0 = 0$,
  - for $k = 0, \ldots$:

$$\theta^{k+1} = \theta^k - \eta(\theta^k - c)$$

- Note with $\eta = 1$, we find the optima, $\theta^\star = c$, in one step.

Brief overview of GD/SGD:

- Different types of “stationary points” (e.g. points with 0 gradients): global optima, local optima, and saddle points (by picture)

- For convex functions (with certain regularity conditions, such as “smoothness”),
  - GD (with an appropriate constant learning rate) converges to the global optima.
  - SGD (with an appropriately decaying learning rate) converges to the global optima. (lower variance is better for SGD)

- For non-convex functions, we could hope to find a local minima.

- What we can prove (under some regularity conditions) is a little weaker: Both GD (with some constant learning rate) and SGD (with some decaying learning rate) converge to a stationary point, i.e.

$$\text{As } k \to \infty,\ \nabla J(\theta^k) \to 0$$

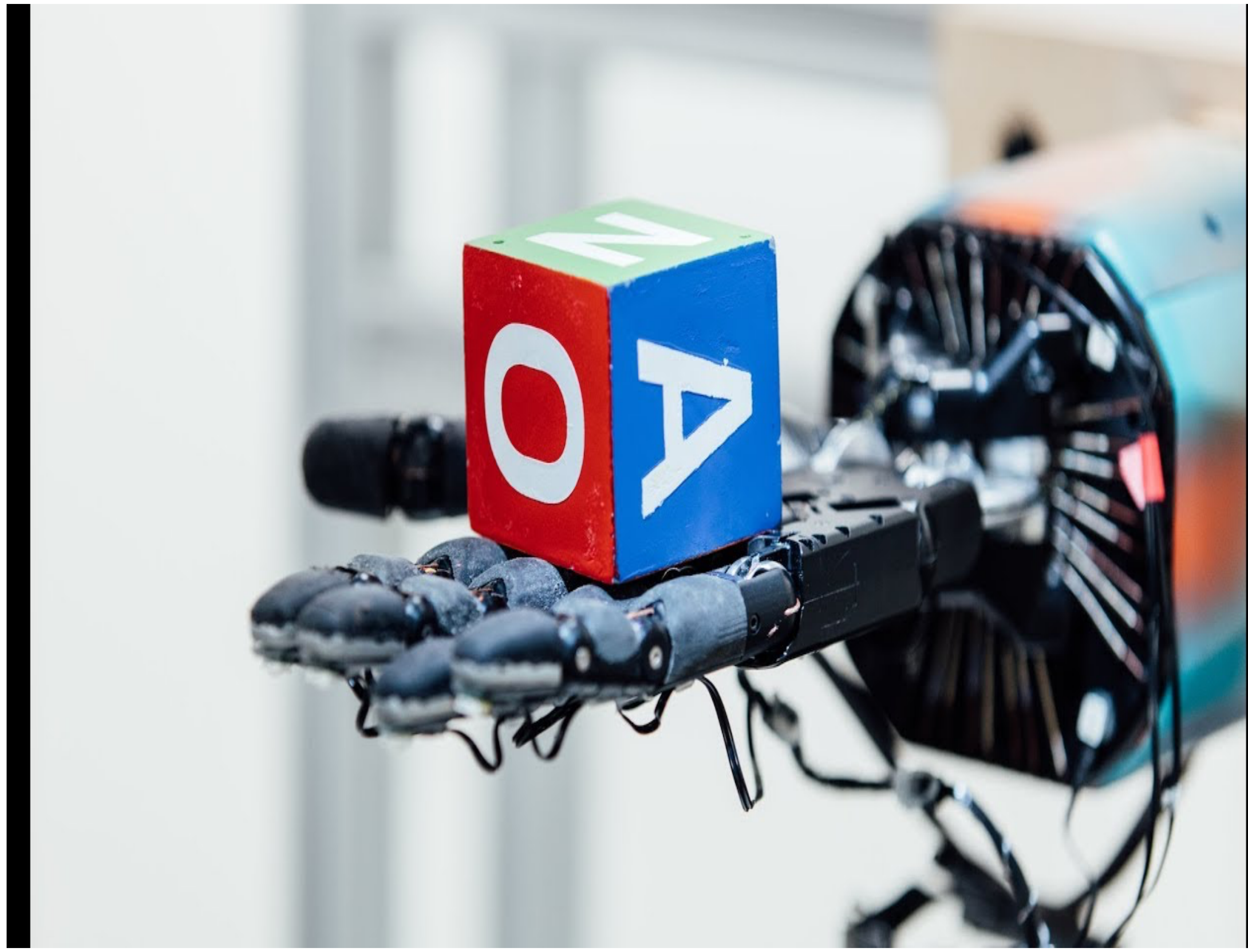
Today

- ✓ Feedback from last lecture
- ✓ Recap+
- ✓ Gradient Descent (ok this is also sort of recap)
- Policy Gradient
- Likelihood ratio method

Policy Optimization

[AlphaZero, Silver et.al, 17]

[OpenAI Five, 18]



[OpenAI,19]

The Learning Setting:

We don't know the MDP, but we can obtain trajectories.

The Finite Horizon, Learning Setting. We can obtain trajectories as follows:

- We start at $s_0 \sim \mu$.
- We can act for H steps and observe the trajectory $\tau = \{s_0, a_0, s_1, a_1, \ldots, s_{H-1}, a_{H-1}\}$

Note that with a simulator, we can sample trajectories as specified in the above.

Optimization Objective

- Consider a parameterized class of policies:

$$\{\pi_\theta(a \mid s) \mid \theta \in \mathbb{R}^d\}$$

(why do we make it stochastic?)

- Objective $\max_\theta J(\theta)$, where

$$J(\theta) := \mathbb{E}_{s_0 \sim \mu}\left[V^{\pi_\theta}(s_0)\right] = \mathbb{E}_{\tau \sim \rho_{\pi_\theta}}\left[\sum_{h=0}^{H-1} r(s_h, a_h)\right]$$

- Policy Gradient Descent:

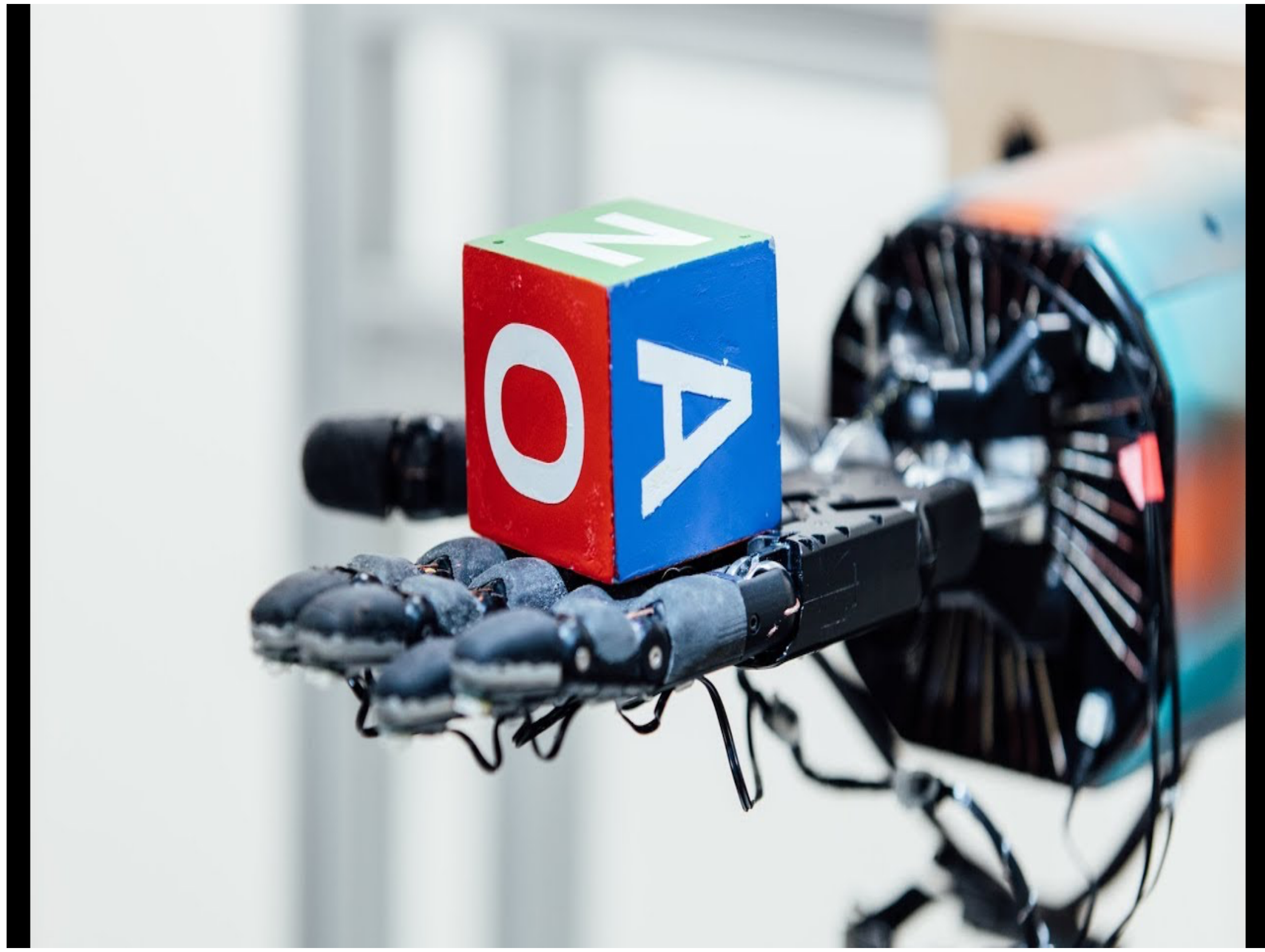
$$\theta^{k+1} = \theta^k + \eta \nabla J(\theta^k)$$

Main question for today's lecture:
how to compute the gradient?

What are parameterized policies?

[AlphaZero, Silver et.al, 17]

[OpenAI Five, 18]



[OpenAI,19]

A state:

- Tabular case: an index in $[|S|] = \{1, \ldots |S|\}$
- Real world: a list/array of the relevant info about the world that makes the process Markovian.
  - e.g. sometimes make a feature vector $\phi(s, a, h) \in \mathbb{R}^d$ which we believe is a "good representation" of the world
  - we sometimes append history info into the current state

Example Policy Parameterizations

Recall that we consider parameterized policy $\pi_\theta(\cdot \mid s) \in \Delta(A), \forall s$

1. Softmax linear Policy

Feature vector $\phi(s, a, h) \in \mathbb{R}^d$, and parameter $\theta \in \mathbb{R}^d$

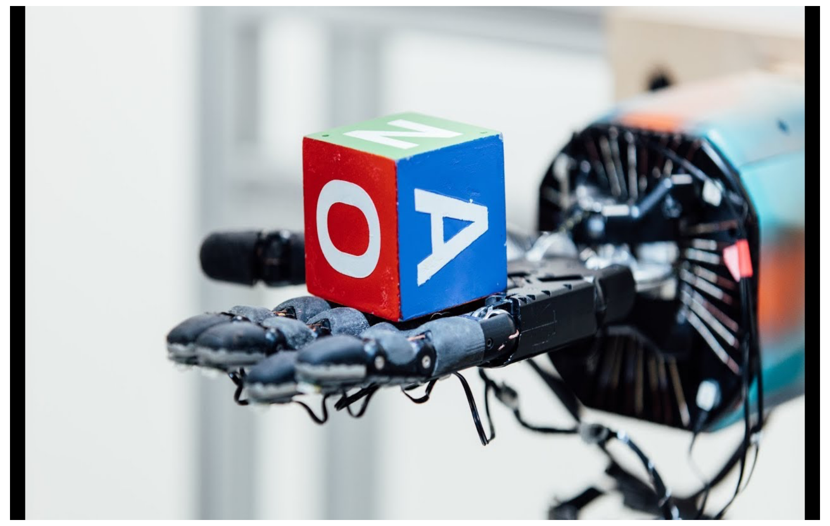
$$\pi_\theta(a \mid s, h) = \frac{\exp(\theta^\top \phi(s, a, h))}{\sum_{a'} \exp(\theta^\top \phi(s, a', h))}$$

2. Neural Policy:

Neural network $f_\theta : S \times A \times [H] \mapsto \mathbb{R}$

$$\pi_\theta(a \mid s, h) = \frac{\exp(f_\theta(s, a, h))}{\sum_{a'} \exp(f_\theta(s, a', h))}$$

Example Policy Parameterization for "Controls"



Suppose $a \in R^k$, as it might be for a control problem.

3. Gaussian + Linear Model

- Feature vector: $\phi(s, h) \in \mathbb{R}^d$,
- Parameters: $\theta \in \mathbb{R}^{k \times d}$,
  (and maybe $\sigma \in R^+$)

- Policy: sample action from a (multivariate) Normal with mean $\theta \cdot \phi(s, h)$ and variance $\sigma^2 I$, i.e.

$$\pi_{\theta,\sigma}(\cdot \mid s, h) = \mathcal{N}\left(\theta \cdot \phi(s, h), \sigma^2 I\right)$$

- Sampling:

$$a = \theta \cdot \phi(s, h) + \eta, \text{ where } \eta \sim \mathcal{N}(0, \sigma^2 I)$$

4. Gaussian + Neural Model

- Neural network $g_\theta : S \times [H] \mapsto \mathbb{R}^k$
- Parameters: $\theta \in R^d$,
  (and maybe $\sigma \in R^+$)

- Policy: a (multivariate) Normal with mean $g_\theta(s)$ and variance $\sigma^2 I$, i.e.

$$\pi_{\theta,\sigma}(\cdot \mid s, h) = \mathcal{N}(g_\theta(s, h), \sigma^2 I)$$

- Sampling:

$$a = g_\theta(s, h) + \eta, \text{ where } \eta \sim \mathcal{N}(0, \sigma^2 I)$$

Today

- ✓ Feedback from last lecture
- ✓ Recap+
- ✓ Gradient Descent (ok this is also sort of recap)
- ✓ Policy Gradient
- Likelihood ratio method

The Likelihood Ratio Method

- Suppose $J(\theta) = \mathbb{E}_{x \sim P_\theta}\left[f(x)\right] = \sum_x P_\theta(x) f(x)$, and our objective is $\max_\theta J(\theta)$.
- Computing $\nabla_\theta J(\theta)$ exactly may be difficult (due to the sum over x=trajectories)
  - So GD not an option—what about SGD?
  - In supervised learning, stochastic gradient was just gradient on one sample—will that work here?
  - Won't work: θ-dependence is inside the distribution, not inside the expectation
  - So how can we unbiasedly estimate $\nabla_\theta J(\theta)$?
- Suppose we can compute $f(x)$, $P_\theta(x)$, and $\nabla P_\theta(x)$, and we can sample $x \sim P_\theta$
- We have that:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x \sim P_\theta(x)}\left[\nabla_\theta \log P_\theta(x)\, f(x)\right]$$

Proof:

$$\nabla_\theta J(\theta) = \sum_x \nabla_\theta P_\theta(x) f(x)$$

$$= \sum_x P_\theta(x) \frac{\nabla_\theta P_\theta(x)}{P_\theta(x)} f(x)$$

$$= \sum_x P_\theta(x)\, \nabla_\theta \log P_\theta(x) f(x)$$

The Likelihood Ratio Method, continued

- We have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x \sim P_\theta(x)} \left[\nabla_\theta \log P_\theta(x) \, f(x) \right]$$

- An unbiased estimate is given by:

$\widehat{\nabla}_\theta J(\theta) = \nabla_\theta \log P_\theta(x) \cdot f(x)$, where $x \sim P_\theta$

- We can lower variance by drawing N i.i.d. samples from P_θ and averaging:

$$\widehat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log P_\theta(x_i) \, f(x_i)$$

Today

- Feedback from last lecture
- Recap+
- Gradient Descent (ok this is also sort of recap)
- Policy Gradient
- Likelihood ratio method

Summary:

- Q-learning and TD(0) are online variants of fitted DP that use SGD
- PG approach: let's directly try to optimize the objective function of interest!

Attendance:
bit.ly/3RcTC9T

Feedback:
bit.ly/3RHtlxy