CS 4/5789 9 March 2022 Prof Sarah Dean

Lecture 13: Optimization & Gradient Descent

1) Function approximation

Bellman-based supervision (like rollout based) gives us labels that we can use to train models: $\{(s_i, a_i, y_i)\}_{i=1}^N$

ERM: $$\min_{Q \in \mathcal{Q}} \sum_{i=1}^N (Q(s_i,a_i) - y_i)^2$$

suppose parametrized model class
$$\mathcal{Q} = \{ Q_\theta \mid \theta \in \mathbb{R}^d \}$$

Bellman-based supervision is online & incremental. So rather than full ERM minimization, it is common to do gradient descent updates to θ using incoming data.

$$\nabla_\theta (Q_\theta(s_i,a_i) - y_i)^2 = 2(Q_\theta(s_i,a_i) - y_i) \nabla_\theta Q_\theta(s_i,a_i)$$

update looks like
$$\theta \leftarrow \theta + \alpha (Q_\theta(s_i,a_i) - y_i) \nabla Q_\theta(s_i,a_i))$$

could be Bellman-exp (SARSA) or Bellman-opt (Q-learning)

Reminiscent of SGD for ERM in supervised learning. But in SL, datapoints (x_i, y_i) are usually sampled randomly from a static dataset.

In the context of Approx. Dynamic Programming:

1) online GD: gradient update on incoming data (s_t, a_t, y_t)

2) "experience replay": Store incoming data, then Sample minibatches $\{(s_i, a_i, y_i)\}$ at random & perform SGD updates. Especially common in Deep Q-learning.

2) Optimization & Gradient Descent

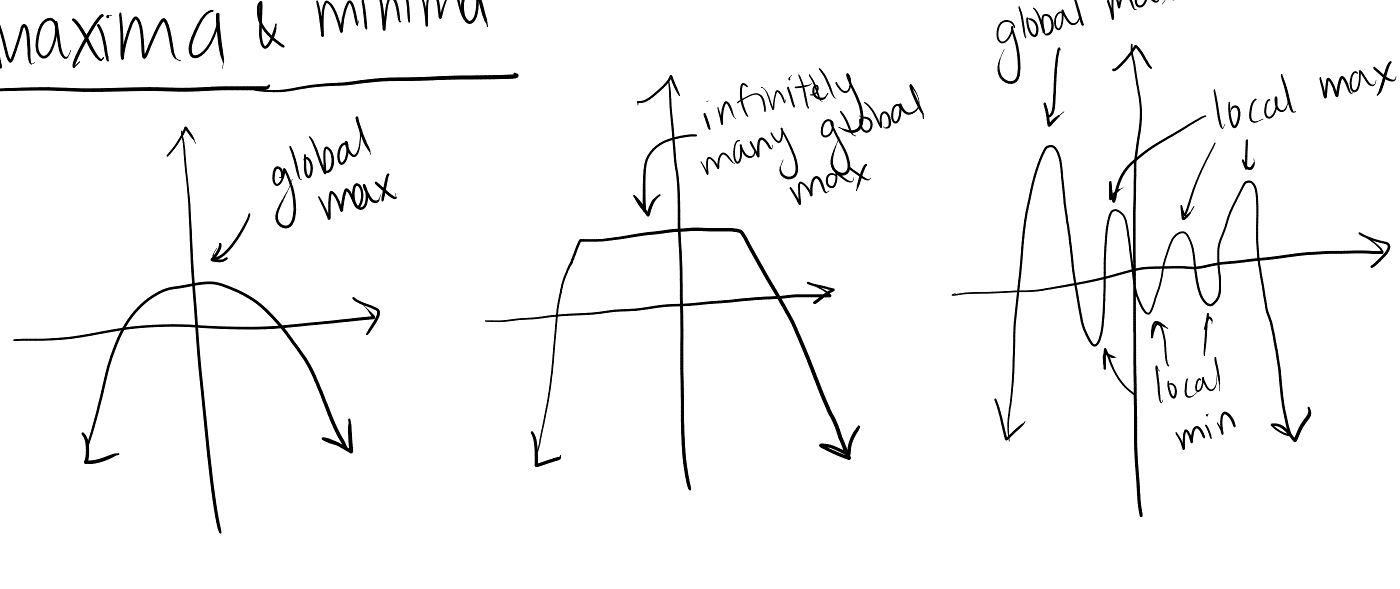
Motivation: our ultimate goal is to find a (near) optimal policy. So maybe we should optimize the policy directly?

Parametrized Policy: π_θ

Objective function: $J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid P, \pi_\theta, \mu_0\right]$

We will come back to this, but for now let's review optimization concepts.

Maxima & minima

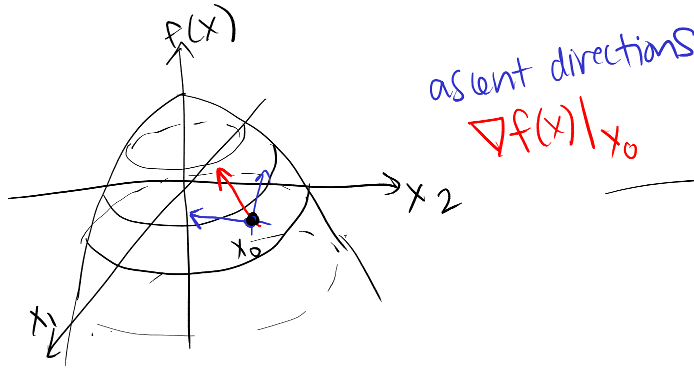


Consider a general function $f(x): \mathbb{R}^d \to \mathbb{R}$

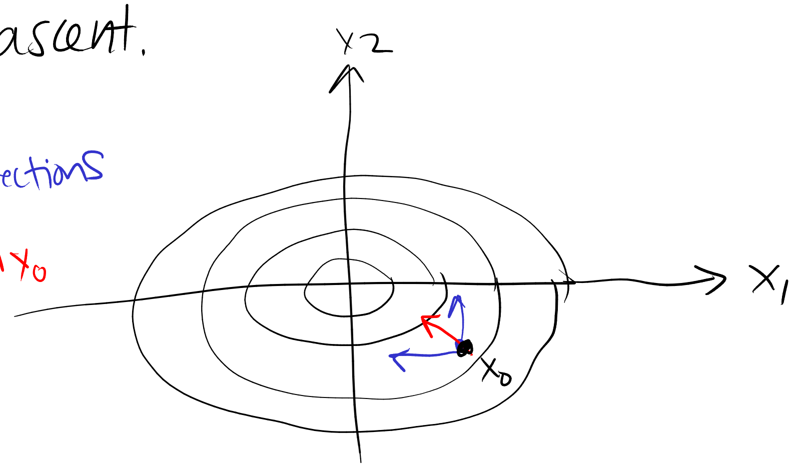
A global maximum is a point x_0 such that $f(x) \geq f(x_0)$ $\forall x \in \mathbb{R}^d$. A local max is when the inequality holds for all $\|x - x_0\| \leq \varepsilon$ for some $\varepsilon > 0$.

An ascent direction at point x_0 is any v such that $f(x_0 + \alpha v) > f(x_0)$ for some $\alpha > 0$.
Ascent directions can help us search for maxima.

The gradient of a differentiable function is the direction of steepest ascent.



2D quadratic fn.



level sets of $f(x)$

Gradient Ascent

initalize x_0
for $t = 0, 1, \ldots$

$$x_{t+1} = x_t + \underset{\uparrow \text{ step size}}{\alpha} \nabla f(x_t)$$

First order method:
We are minimizing a first order approximation.

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t)$$

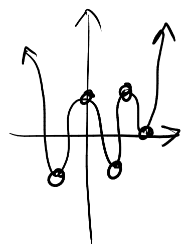
maximized when $x - x_t$ is parallel to $\nabla f(x_t)$

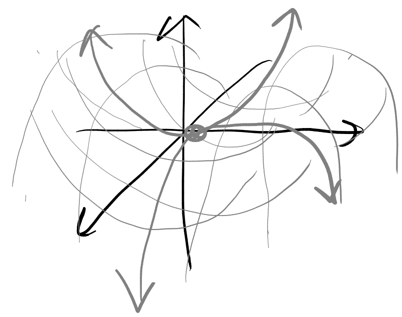
step size α prevents us from moving too far (where the approx. becomes invalid)

The gradient is equal to zero at a local max. Why? Because by definition there must not be any ascent direction.

<u>Critical point</u> is a point x_0 where $\nabla f(x)|_{x=x_0} = 0$.

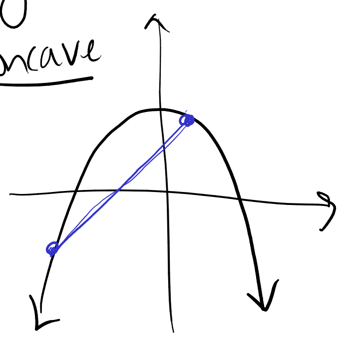
Not only local max! Also local min, saddle points.



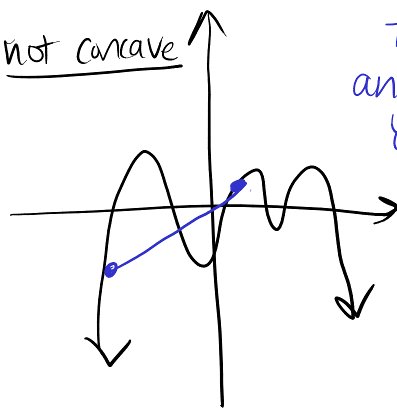


If f is <u>concave</u> then $\nabla f(x)|_{x=x_0} = 0 \Rightarrow x_0$ is a global maximum.

<u>concave</u>



<u>not concave</u>



the line connecting any two points on a concave function lies entirely below the function

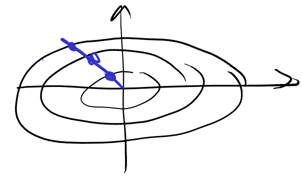
Even when a function is not concave, we can still guarantee that gradient ascent converges towards a critical point.

3) Stochastic Gradient Ascent

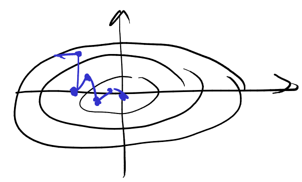
Instead of exact gradient evaluations, SGA uses estimates g_t such that $\mathbb{E}[g_t] = \nabla f(x_t)$.

Alg: SGA
init x_0
for $t = 0, 1, \ldots$
$\quad x_{t+1} = x_t + \alpha g_t$

GA:



SGA:



Example: ERM via SGD.

$$f(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

$$\nabla f(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f_\theta(x_i), y_i)$$

select x_i, y_i uniformly at random

$$g = \nabla_\theta \ell(f_\theta(x_i), y_i)$$

$$\mathbb{E}[g] = \mathbb{E}[\nabla_\theta \ell(f_\theta(x_i), y_i)] = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f_\theta(x_i), y_i)$$

Theorem: Suppose that $f(x)$ is β-smooth i.e. $\|\nabla f(x) - \nabla f(x')\|_2 \leq \beta \|x - x'\|_2$ and $\sup_x |f(x)| \leq M$. Then SGA with gradient estimates $g(x)$ satisfying

1) $\mathbb{E}[g(x)] = \nabla f(x)$ 2) $\mathbb{E}[\|g(x)\|_2^2] \leq \sigma^2$

Satisfies:

$$\mathbb{E}\left[\frac{1}{T} \sum_{t=1}^{T} \|\nabla f(x_t)\|_2\right] \lesssim \sqrt{\frac{\beta \sigma^2 M}{T}} \quad \text{for} \quad \alpha = \sqrt{\frac{M}{\beta \sigma^2 T}}$$

Depends on <u>variance</u> of gradient estimates.

Example: minibatching with SGD for ERM

suppose $i_0, \ldots, i_M$ chosen uniformly at random.

$$g_M = \frac{1}{M} \sum_{j=0}^{M} \nabla_\theta \ell(f_\theta(x_{i_j}), y_{i_j})$$

Still an unbiased estimate of the gradient.

$$\mathbb{E}\|g_M - \nabla R(\theta)\|_2^2 = \frac{1}{M^2} \sum_{j=0}^{M} \mathbb{E}\|\nabla_\theta \ell(f_\theta(x_{i_j}), y_{i_j}) - \nabla R(\theta)\|_2^2$$

$$= \frac{\sigma^2}{M}$$

$\sigma^2 \leftarrow$ variance of loss w/ single datapoint

reduction by minibatching $\rightarrow M$

Question: in RL can we use sampled trajectories to do SGA similar to how ERM uses single datapoints for SGD?

Simple example

$$J(\theta) = \mathbb{E}_{w}\left[s_1^2 \,\middle|\, s_1 = f(s_0, a, w),\ a = \pi_\theta(s_0) \right]$$

$$\nabla_\theta J = \nabla_\theta \mathbb{E}_{w} (f(s_0, \pi_\theta(s_0), w))^2$$

$$= \mathbb{E}_{w}\left[\nabla_\theta f(s_0, \pi_\theta(s_0), w)^2 \right] \neq \mathbb{E}_{w}[s_1^2]$$

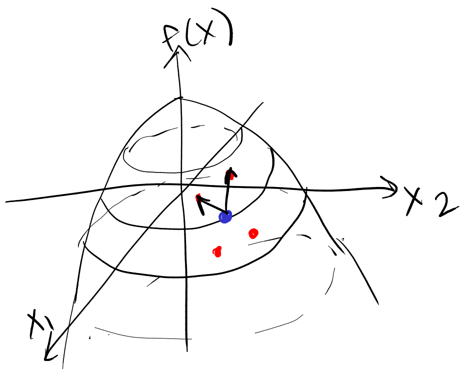
to sample from this expected value we can look at sampled trajectories. But for the trajectory we still somehow need to differentiate through f. We don't know f and therefore don't have access to its gradients!

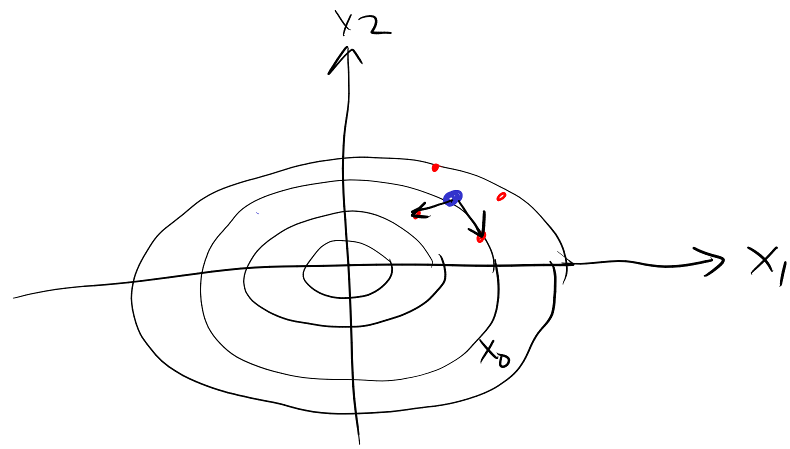
4) Derivative-Free Optimization

How can we find maxima only using function evaluation?
i.e. we can query $f(x)$ but not $\nabla f(x)$.

Goal: find a descent direction

Simple idea: randomly test a few directions & see which lead to increase.





There are many variations of this simple idea: simulated annealing, cross-entropy method, genetic algorithms, evolutionary strategies. They differ in how random samples are aggregated into update step.

1) Random Search

Recall when we discussed iLQR the finite difference approximation:

$$f'(x) \approx \frac{f(x+\delta)-f(x-\delta)}{2\delta}$$

This idea can help us build an approximation of the gradient based only on function evaluation.
direction of steepest ascent

For vector functions:

$$\langle \nabla f(x), v \rangle \approx \frac{f(x+\delta v)-f(x-\delta v)}{2\delta}$$

Alg: Random Search
initialize x_0
for $t=0,1,\dots$
    sample $v_1,\dots v_N \sim N(0, I)$
    update $x_{t+1} = x_t + \frac{\alpha}{N} \sum_{k=1}^{N} (f(x+\delta v_k) - f(x-\delta v_k)) v_k$

we can understand this as stochastic gradient descent:

$$\mathbb{E}\left(\left(f(x+\delta v_k) - f(x-\delta v_k)\right)v_k\right) \approx \mathbb{E}\left(2\delta \nabla f(x)^T v_k \cdot v_k\right)$$
$$= 2\delta\, \mathbb{E}\left[v_k v_k^T\right] \nabla f(x)$$
$$= 2\delta \nabla f(x)$$

This method samples/searches in parameter space.

2) Importance Weighting

Distribution trick: in general, we can write:

$$f(x) = \mathbb{E}_{y \sim P_x}[h(y)]$$

for some class of distributions P_x.
(In RL setting, P_θ could represent the distribution over trajectories induced by π_θ.)
Now suppose a sampling distribution ρ where $\frac{P_x(y)}{\rho(y)} < \infty$.

$$\mathbb{E}_{y \sim P_x}[h(y)] = \sum_y h(y) P_x(y) \cdot \frac{\rho(y)}{\rho(y)} = \mathbb{E}_{y \sim \rho}\left[\frac{P_x(y)}{\rho(y)} h(y)\right].$$

This allows us to write the gradient:

$$\nabla_x f(x) = \mathop{\mathbb{E}}_{y\sim p}\left[\frac{\nabla_x P_x(y)}{p(x)} h(y)\right]$$

If $p(x) = P_x(y)$ then

$$\nabla_x f(x) = \mathop{\mathbb{E}}_{y\sim P_x(y)}\left[\frac{\nabla_x P_x(y)}{P_x(y)} h(y)\right] = \mathop{\mathbb{E}}_{y\sim P_x(y)}\left[\nabla_x \log(P_x(y))\, h(y)\right]$$

Now if $P_x(y)$ factors, $\log(P_x(y))$ will be sum of factors, and the gradient will depend only on factors which depend on optimization variable x. (This is very useful for policy optimization – next lecture)

Therefore, our stochastic maximization algorithm:

Alg: sampling-DFO

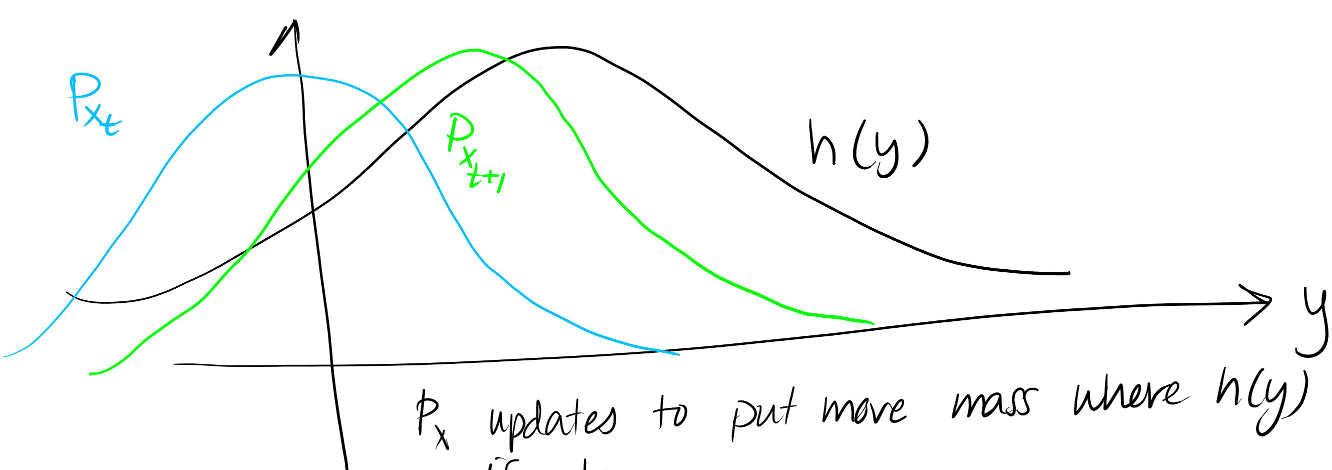
initialize x_0

for $t = 0, 1, \ldots$

  sample $y \sim P_{x_t}$ and observe $h(y)$

  $x_{t+1} = x_t + \alpha \nabla_{x_t} \log(P_{x_t}(y))\, h(y)$

This method samples in y-space rather than parameter space.



P_x updates to put more mass where $h(y)$ is large