CS-GY 6763: Lecture 7
Second Order Conditions, Online and Stochastic Gradient Decent

NYU Tandon School of Engineering, Prof. Christopher Musco

ADMINISTRATIVE

Midterm in class next Wednesday.

- List of topics covered and practice problems will be posted on the course webpage.
- You are allowed a double sided sheet of paper.
- If you are taking it at the Moses Center, please send me an email just to make sure I don't forget.
- I will go over Problem Set 2 in office hours on Monday and record it. Recording of Majid going over Problem Set 1 already posted.

First Order Optimization: Given a function f and a constraint set $\mathcal{S}$, assume we have:

- **Function oracle**: Evaluate $f(\mathbf{x})$ for any $\mathbf{x}$.
- **Gradient oracle**: Evaluate $\nabla f(\mathbf{x})$ for any $\mathbf{x}$.
- **Projection oracle**: Evaluate $P_{\mathcal{S}}(\mathbf{x})$ for any $\mathbf{x}$.

Goal: Find $\hat{\mathbf{x}} \in \mathcal{S}$ such that $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x}) + \epsilon$.

Projected gradient descent:

- Select starting point $\mathbf{x}^{(0)}$, learning rate η.
- For $i = 0, \ldots, T$:
    - $\mathbf{z} = \mathbf{x}^{(i)} - \eta \nabla f(\mathbf{x}^{(i)})$
    - $\mathbf{x}^{(i+1)} = P_{\mathcal{S}}(\mathbf{z})$
- Return $\hat{\mathbf{x}} = \arg\min_i f(\mathbf{x}^{(i)})$.

Conditions for convergence:

- **Convexity:** f is a convex function, $\mathcal{S}$ is a convex set.
- **Bounded initial distance:**

$$\|\mathbf{x}^{(0)} - \mathbf{x}^*\|_2 \leq R$$

- **Bounded gradients (Lipschitz function):**

$$\|\nabla f(\mathbf{x})\|_2 \leq G \text{ for all } \mathbf{x} \in \mathcal{S}.$$

Theorem: Projected Gradient Descent returns $\hat{\mathbf{x}}$ with $f(\hat{\mathbf{x}}) \leq \min_{\mathbf{x}\in\mathcal{S}} f(\mathbf{x}) + \epsilon$ after

$$T = \frac{R^2 G^2}{\epsilon^2}$$

iterations.

OTHER CONVERGENCE GUARANTEES

Convexity:

$$0 \le f''(x) \le \beta$$

$$\overset{\alpha}{\cancel{0}} \le [f(\mathbf{y}) - f(\mathbf{x})] - \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) \le \frac{\beta}{2}\|x-y\|_2^2$$

β-smoothness:

$$[f(\mathbf{y}) - f(\mathbf{x})] - \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) \le \frac{\beta}{2}\|\mathbf{x} - \mathbf{y}\|_2^2.$$

Number of iterations for ϵ error:

	G-Lipschitz	β-smooth
R bounded start	$O\left(\frac{G^2R^2}{\epsilon^2}\right)$	$O\left(\frac{\beta R^2}{\epsilon}\right)$
α-strong convex	$O\left(\frac{G^2}{\alpha\epsilon}\right)$	$O\left(\frac{\beta}{\alpha}\log(R/\epsilon)\right)$

Definition (α-strongly convex)

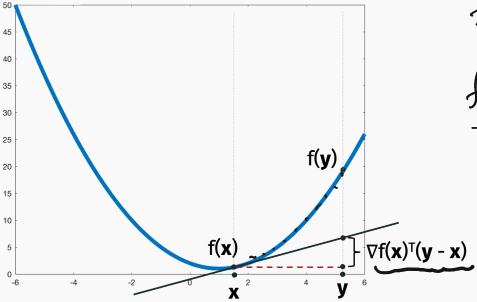
A convex function f is α-strongly convex if, for all $\mathbf{x}$, $\mathbf{y}$

$$\frac{\alpha}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \leq [f(\mathbf{y})-f(\mathbf{x})] - \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x})$$

For a twice-differentiable scalar function f, equivalent to $f''(x) \geq \alpha$.

$f''(x) \geq 0$

$f''(x) \geq \alpha$



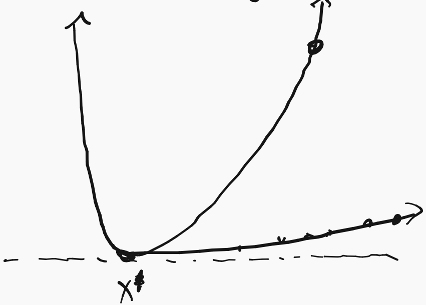
Gradient descent for strongly convex functions:

- Choose number of steps T.
- For $i = 1, \dots, T$:
  - $\eta = \frac{2}{\alpha \cdot (i+1)}$
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \arg\min_{\mathbf{x}^{(i)}} f(\mathbf{x}^{(i)})$.

$f'(x^*) = 0$

$f'(x) - \underbrace{\cancel{f'(x^*)}}_{0} \approx f''(y)(x - x^*)$

$y \in [x^*, x]$



Theorem (GD convergence for α-strongly convex functions.)

Let f be an α-strongly convex function and assume we have that, for all $\mathbf{x}$, $\|\nabla f(\mathbf{x})\|_2 \leq G$. If we run GD for T steps (with adaptive step sizes) we have:

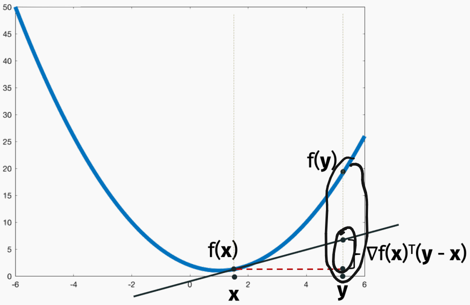
$T = O\left(\frac{G^2}{\alpha \varepsilon}\right)$

$$f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \frac{2G^2}{\alpha(T-1)}$$

Corollary: If $T = O\left(\frac{G^2}{\alpha\epsilon}\right)$ we have $f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \epsilon$

We could also have that f is both β-smooth and α-strongly convex.

$$\frac{\alpha}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \leq [f(\mathbf{y})-f(\mathbf{x})] - \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) \leq \frac{\beta}{2}\|\mathbf{x}-\mathbf{y}\|_2^2.$$



$\kappa \in [1, \infty)$

$$\frac{\alpha}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \leq [f(\mathbf{y})-f(\mathbf{x})] - \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) \leq \frac{\beta}{2}\|\mathbf{x}-\mathbf{y}\|_2^2.$$

Theorem (GD for β-smooth, α-strongly convex.)

Let f be a β-smooth and α-strongly convex function. If we run GD for T steps (with step size $\eta = \frac{1}{\beta}$) we have:

$T = \log(1/\epsilon) \cdot \frac{\beta}{\alpha}$

$$\frac{2}{\beta}\left(f(x^T) - f(x^*)\right) \leq \|\mathbf{x}^{(T)} - \mathbf{x}^*\|_2^2 \leq e^{-T\frac{\alpha}{\beta}}\|\mathbf{x}^{(0)} - \mathbf{x}^*\|_2^2 \leq e^{-T\alpha/\beta} R^2$$

$$f(x^T) - f(x^*) \leq \frac{R^2 \beta}{2} \cdot e^{-T\alpha/\beta}$$

$\kappa = \frac{\beta}{\alpha}$ is called the "condition number" of f.

~~Is it better if κ is large or small?~~

Converting to more familiar form: Using that fact the $\nabla f(\mathbf{x}^*) = \mathbf{0}$ along with $x = x^*$ $y = x^{(T)}$

$$\frac{\alpha}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 \leq [f(\mathbf{y}) - f(\mathbf{x})] - \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) \leq \frac{\beta}{2}\|\mathbf{x}-\mathbf{y}\|_2^2,$$

we have: $f(x^T) - f(x^*) - \cancel{\nabla f(x^*)(x^T - x^*)} \leq \frac{\beta}{2}\|x^* - x^T\|_2^2$

$$\|\mathbf{x}^{(T)} - \mathbf{x}^*\|_2^2 \geq \frac{2}{\beta}\left[f(\mathbf{x}^{(T)}) - f(\mathbf{x}^*)\right].$$

We also assume

$$\|\mathbf{x}^{(0)} - \mathbf{x}^*\|_2^2 \leq R^2.$$

Theorem (GD for β-smooth, α-strongly convex.)

Let f be a β-smooth and α-strongly convex function. If we run GD for T steps (with step size $\eta = \frac{1}{\beta}$) we have:

$$f(\mathbf{x}^{(T)}) - f(\mathbf{x}^*) \leq \frac{\beta}{\alpha} e^{-T\frac{\alpha}{\beta}} \cdot \left[f(\mathbf{x}^{(0)}) - f(\mathbf{x}^*)\right]$$

$\log(\beta^2) = O(\log \beta)$

Corollary: If $T = O\left(\frac{\beta}{\alpha}\log(R\beta/\epsilon)\right)$ we have:

$$f(\mathbf{x}^{(T)}) - f(\mathbf{x}^*) \leq \epsilon$$

Only depend on $\log(1/\epsilon)$ instead of on $1/\epsilon$ or $1/\epsilon^2$!

We're going to prove this theorem for the special case of a quadratic function:

$$\min_{x} \; \|Ax - b\|_2^2.$$

Underwhelming, yes, but the analysis is really helpful pedagogically! Also if there is one class of algorithms that use more of the worlds computing power than training neural networks, it's GD like iterative methods for solving linear systems.

$j,k \in 1,\dots,d$

Let f be a twice differentiable function from $\mathbb{R}^d \to \mathbb{R}$. Let the **Hessian** $\mathbf{H} = \nabla^2 f(\mathbf{x})$ contain all of its second derivatives at a point $\mathbf{x}$. So $\mathbf{H} \in \mathbb{R}^{d \times d}$. We have:

$$\mathbf{H}_{j,k} = \left[\nabla^2 f(\mathbf{x})\right]_{j,k} = \frac{\partial^2 f}{\partial x_j x_k}.$$

For vector $\mathbf{x}, \mathbf{v}$:

$$\nabla f(\mathbf{x} + t\mathbf{v}) \approx \nabla f(\mathbf{x}) + t \underbrace{\left[\nabla^2 f(\mathbf{x})\right]}_{H} \mathbf{v}.$$

$x+tv$

$+ t\, Hv$

$\nabla f(x) \quad \nabla f(x+tv)$

Let f be a twice differentiable function from $\mathbb{R}^d \to \mathbb{R}$. Let the **Hessian** $\mathbf{H} = \nabla^2 f(\mathbf{x})$ contain all of its second derivatives at a point $\mathbf{x}$. So $\mathbf{H} \in \mathbb{R}^{d\times d}$. We have:

$$\mathbf{H}_{j,k} = \left[\nabla^2 f(\mathbf{x})\right]_{j,k} = \frac{\partial^2 f}{\partial x_j x_k}.$$

$a^{(i)}$ = ith row in A

$y^{(i)}$ = ith entry in A

Example: $f(\mathbf{x}) = \sum_{i=1}^n \left(\mathbf{x}^T\mathbf{a}^{(i)} - y^{(i)}\right)^2 = \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2$

$$\frac{\partial f}{\partial x_j} = \sum_{i=1}^n 2\left(\mathbf{x}^T\mathbf{a}^{(i)} - y^{(i)}\right)\cdot a_j^{(i)}$$

$2x^T a^{(i)} \cdot a_j^{(i)}$

$$\frac{\partial^2 f}{\partial x_k \partial x_j} = \boxed{\sum_{i=1}^n 2a_k^{(i)} a_j^{(i)}} = 2\langle a_k, a_j\rangle$$

$\to$ jth column

kth column of a

$$\mathbf{H} = 2A^TA$$

$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$. Recall that $\nabla f(\mathbf{x}) = 2\mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b})$.

$$\nabla f(x+tv) = 2A^T(A(x+tv)-b)$$

$$= 2A^T(Ax-b) + \boxed{2A^TA}\,tv$$

A twice-differentiable function $f : \mathbb{R} \rightarrow \mathbb{R}$ is :

- convex if and only if $f''(x) \geq 0$ for all x.
- β-smooth if $f''(x) \leq \beta$.
- α-strongly convex if $f''(x) \geq \alpha$.

$H \succeq 0$

How do these statements generalize to the case when f has a vector in put, so the second derivative is a matrix $\mathbf{H}$?

Claim: If f is twice differentiable, then it is convex if and only if the matrix $\mathbf{H} = \nabla^2 f(\mathbf{x})$ is positive semidefinite for all $\mathbf{x}$.

Definition (Positive Semidefinite (PSD))

A square, symmetric matrix $\mathbf{H} \in \mathbb{R}^{d \times d}$ is positive semidefinite (PSD) for any vector $\mathbf{y} \in \mathbb{R}^d$, $\mathbf{y}^T\mathbf{H}\mathbf{y} \geq 0$.

This is a natural notion of "positivity" for symmetric matrices. To denote that $\mathbf{H}$ is PSD we will typically use "Loewner order" notation (`\succeq` in LaTex):

$$\mathbf{H} \succeq 0.$$

$B - A \succeq 0$

We write $\mathbf{B} \succeq \mathbf{A}$ or equivalently $\mathbf{A} \preceq \mathbf{B}$ to denote that $(\mathbf{B} - \mathbf{A})$ is positive semidefinite. This gives a partial ordering on matrices.

Claim: If f is twice differentiable, then it is convex if and only if the matrix $\mathbf{H} = \nabla^2 f(\mathbf{x})$ is positive semidefinite for all $\mathbf{x}$.

Definition (Positive Semidefinite (PSD))

A square, symmetric matrix $\mathbf{H} \in \mathbb{R}^{d \times d}$ is positive semidefinite (PSD) for any vector $\mathbf{y} \in \mathbb{R}^d$, $\mathbf{y}^T \mathbf{H} \mathbf{y} \geq 0$.

For the least squares regression loss function: $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, $\mathbf{H} = \nabla^2 f(\mathbf{x}) = 2\mathbf{A}^T\mathbf{A}$ for all $\mathbf{x}$. Is $\mathbf{H}$ PSD?

want: $y^T(2A^TA)y \geq 0$

$y^T A^T A y \geq 0$

$= (Ay)^T Ay = \|Ay\|_2^2 \geq 0$

If f is β-smooth and α-strongly convex then at any point $\mathbf{x}$, $\mathbf{H} = \nabla^2 f(\mathbf{x})$ satisfies:

Identity

$$\alpha\mathbf{I} \preceq \mathbf{H} \preceq \beta\mathbf{I},$$

$$y^T H y \leq \beta y^T I y$$

where $\mathbf{I}$ is a $d \times d$ identity matrix.

This is the natural matrix generalization of the statement for scalar valued functions:

$$\alpha \leq f''(x) \leq \beta.$$

$$\alpha \mathbf{I}_{d\times d} \preceq \mathbf{H} \preceq \beta \mathbf{I}_{d\times d}.$$

Equivalently for any $\mathbf{z}$,

$$\alpha\|\mathbf{z}\|_2^2 \leq \mathbf{z}^T\mathbf{H}\mathbf{z} \leq \beta\|\mathbf{z}\|_2^2.$$

$$z^T H z \leq z^T(\beta I)z$$
$$= \beta z^T I z$$
$$= \beta z^T z = \beta\|z\|_2^2$$

Let $f(\mathbf{x}) = \|\mathbf{D}\mathbf{x} - \mathbf{b}\|_2^2$ where $\mathbf{D}$ is a diagaonl matrix. For now imagine we're in two dimensions: $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, $\mathbf{D} = \begin{bmatrix} d_1 & 0 \\ 0 & d_2 \end{bmatrix}$.

$H = 2D^TD$
$= 2D^2$

What are α, β for this problem?

$\beta =$ largest z^THz for $\|z\|_2^2 = 1$

$2\max(d_1^2, d_2^2) = \beta$

$$\alpha\|\mathbf{z}\|_2^2 \leq \mathbf{z}^T\mathbf{H}\mathbf{z} \leq \beta\|\mathbf{z}\|_2^2$$

$2\min(d_1^2, d_2^2) = \alpha$

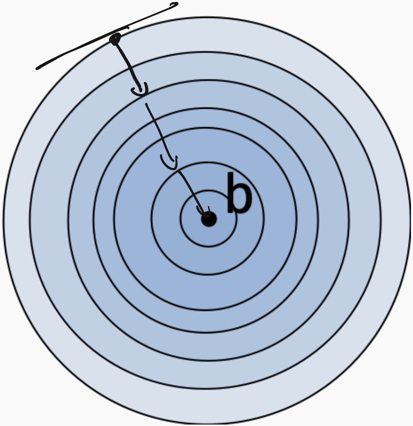
$\alpha =$ smallest z^THz for $\|z\|_2^2 = 1$

$d_1^2 \geq d^2$

$z^THz = 2d_1^2 z_1^2 + 2d_2^2 z_2^2$

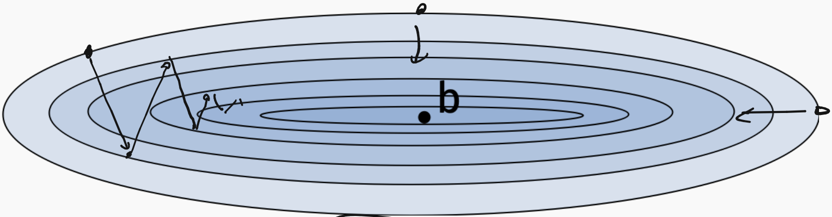
$H = 2\begin{bmatrix} d_1^2 & 0 \\ 0 & d_2^2 \end{bmatrix}$

$k = 1$



Level sets of $\|\mathbf{Dx} - \mathbf{b}\|_2^2$ when $d_1^2 = 1, d_2^2 = 1$.

$\|x - b\|_2^2$



Level sets of $\|\mathbf{D}\mathbf{x} - \mathbf{b}\|_2^2$ when $d_1^2 = \frac{1}{3}, d_2^2 = 2.$

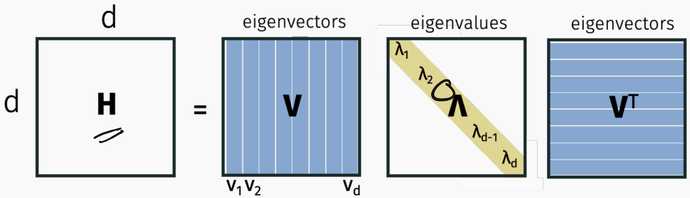
$\kappa = 2/(1/3) = 6$

What about non-diagonal $\mathbf{D}$?

EIGENDECOMPOSITION VIEW

$A^T A$

Any symmetric matrix $\mathbf{H}$ has an orthogonal, real valued eigendecomposition.



Here $\mathbf{V}$ is square and orthogonal, so $\mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}$. And for each $\mathbf{v}_i$, we have:

scalar

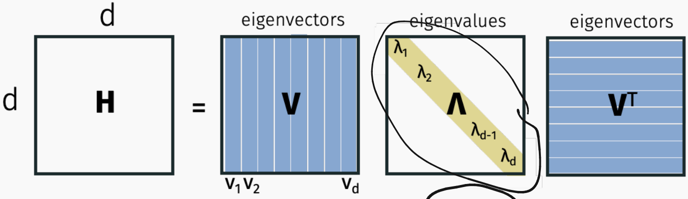
$$\mathbf{H}\mathbf{v}_i = \lambda_i \mathbf{v}_i.$$

By definition, that's what makes $\mathbf{v}_1, \ldots, \mathbf{v}_d$ eigenvectors.

Recall $\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$.

$$y^T H y \geq 0$$

$$v_i^T H v_i = v_i^T \lambda v_i = \lambda_i \|v_i\|_2^2 = \lambda_i$$



$$V^T H V = \Lambda$$

$\Rightarrow$

$\Leftarrow$

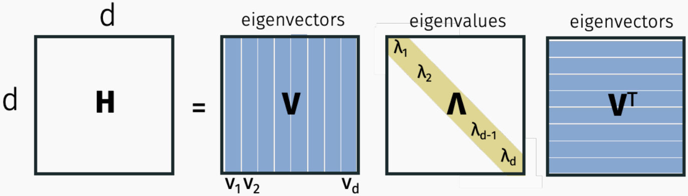
Claim: H is PSD $\Leftrightarrow \lambda_1, \ldots, \lambda_d \geq 0$.

$$H = V\sqrt{\Lambda}\sqrt{\Lambda}^T V^T \qquad H = (V\sqrt{\Lambda})(V\sqrt{\Lambda})^T$$

$$y^T H y = \|(V\sqrt{\Lambda})^T y\|_2^2 \geq 0$$

EIGENDECOMPOSITION VIEW

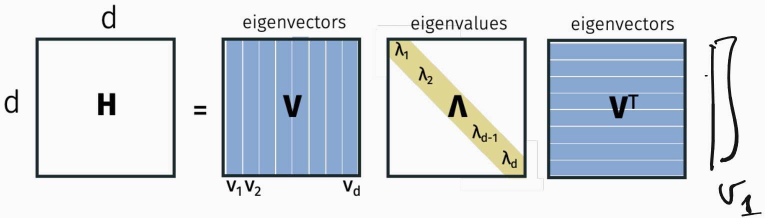
Recall $\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$.



Claim: $\alpha\mathbf{I} \preceq \mathbf{H} \preceq \beta\mathbf{I} \Leftrightarrow \alpha \leq \lambda_d \leq \ldots \leq \lambda_1 \leq \beta.$

EIGENDECOMPOSITION VIEW

Recall $\mathbf{V}\mathbf{V}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$. $z = c_1 v_1 + c_2 v_2 + \dots c_d v_d$



In other words, if we let $\lambda_{\max}(\mathbf{H})$ and $\lambda_{\min}(\mathbf{H})$ be the smallest and largest eigenvalues of $\mathbf{H}$, then for all $\mathbf{z}$ we have:

$$\mathbf{z}^T\mathbf{H}\mathbf{z} \leq \lambda_{\max}(\mathbf{H}) \cdot \|\mathbf{z}\|^2$$
$$\mathbf{z}^T\mathbf{H}\mathbf{z} \geq \lambda_{\min}(\mathbf{H}) \cdot \|\mathbf{z}\|^2$$

If the maximum eigenvalue of $\mathbf{H} = \nabla^2 f(\mathbf{x}) = \beta$ and the minimum eigenvalue of $\mathbf{H} = \nabla^2 f(\mathbf{x}) = \alpha$ then $f(\mathbf{x})$ is β-smooth and α-strongly convex.

$$\lambda_{\text{max}}(\mathbf{H}) = \beta$$
$$\lambda_{\text{min}}(\mathbf{H}) = \alpha$$

Theorem (GD for β-smooth, α-strongly convex.)

Let f be a β-smooth and α-strongly ~~convex function~~. If we run GD for S steps (with step size $\eta = \frac{1}{\beta}$) we have:

the quadratic least squares function

$$\|\mathbf{x}^{(S)} - \mathbf{x}^*\|_2 \leq e^{-S/\kappa}\|\mathbf{x}^{(0)} - \mathbf{x}^*\|_2$$

$\lambda_{max}(H) = 2\lambda_{max}(A^TA)$

Goal: Prove for $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$.

$\lambda_{max} = \lambda_{max}(A^TA)$

Let $\lambda_{\max} = \lambda_{\max}(\mathbf{A}^T\mathbf{A})$ and set step size $\eta = \frac{1}{2\lambda_{max}}$. Gradient descent update is:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{2\,\lambda_{\max}} 2\mathbf{A}^T(\mathbf{A}\mathbf{x}^{(t)} - \mathbf{b})$$

Richardson Iteration view:

$A^T(Ax^* - b) = 0$

$A^T A x^* - A^T b = 0$

$$(x^{(t+1)} - x^*) = \left(I - \frac{1}{\lambda_{max}} A^T A\right)(x^{(t)} - x^*)$$

$$x^{(t+1)} = x^{(t)} - \frac{1}{\cancel{2}\lambda_{max}} \cancel{2} A^T(Ax^{(t)} - b)$$

$$x^{(t+1)} - x^* = x^{(t)} - \frac{1}{\lambda_{max}} A^T A x^{(t)} + \frac{1}{\lambda_{max}} A^T b - x^*$$

$$x^{(t+1)} - x^* = \left(x^{(t)} - \frac{1}{\lambda_{max}} A^T A x^{(t)} + \frac{1}{\lambda_{max}} A^T A x^* - x^*\right)$$

UNROLLED GRADIENT DESCENT

$$x^{(1)} - x^* = \left(I - \frac{1}{\lambda_{max}} A^T A\right)(x^0 - x^*)$$

$$x^{(2)} - x^* = \;\ldots\; (x^{(1)} - x^*)$$

$$(\mathbf{x}^{(S)} - \mathbf{x}^*) = \left(\mathbf{I} - \frac{1}{\lambda_{\max}}\mathbf{A}^T\mathbf{A}\right)^S (\mathbf{x}^{(0)} - \mathbf{x}^*)$$

$\|Cy\|_2^2 = y^T C^T C y$

$$(\mathrm{x}^{(S)} - \mathrm{x}^*) = \Big(\underbrace{\mathrm{I} - \frac{1}{\lambda_{\max}}\mathrm{A}^T\mathrm{A}}_{\to B}\Big)^S (\mathrm{x}^{(0)} - \mathrm{x}^*)$$

Approach: Show that the maximum eigenvalue of $\left(\mathrm{I} - \frac{1}{\lambda_{\max}}\mathrm{A}^T\mathrm{A}\right)^{2S}$ is small – i.e., bounded by $e^{-S/\kappa} = \epsilon$.

Conclusion: $\|\mathrm{x}^{(S)} - \mathrm{x}^*\|_2^2 \leq$ $\|B^S(x^{(0)} - x^*)\|_2^2 = (x^0 - x^*)^T B^S B^S (x^0 - x^*)$

$= (x^0 - x^*)^T B^{2S} (x^0 - x^*)$

So we have $\|\mathrm{x}^{(S)} - \mathrm{x}^*\|_2 \leq$ $\lambda_{\max}(B^{2S}) \cdot \|x^0 - x^*\|_2^2$

B^{2S} $\qquad$ B eigs: $\lambda_1, \dots \lambda_d$

B^{2S} eigs: $\lambda_1^{2S} \dots \lambda_d^{2S}$

$$(\mathbf{x}^{(S)} - \mathbf{x}^*) = \left(\mathbf{I} - \frac{1}{\lambda_{\max}}\mathbf{A}^T\mathbf{A}\right)^S (\mathbf{x}^{(0)} - \mathbf{x}^*)$$

What is the maximum eigenvalue of the symmetric matrix $\left(\mathbf{I} - \frac{1}{\lambda_{\max}}\mathbf{A}^T\mathbf{A}\right)$ in terms of the eigenvalues of $\mathbf{A}^T\mathbf{A}$?

$I - \frac{1}{\lambda_{max}} V \Lambda V^T$ $\quad = VV^T - \frac{1}{\lambda_{max}} V \Lambda V^T$

$= V\left(I - \frac{1}{\lambda_{max}} \Lambda\right) V^T$

$I - \frac{1}{\lambda_{max}} \Lambda$

$1 - \lambda_1 / \lambda_{max} = 0$

$1 - \lambda_2 / \lambda_{max}$

$\vdots$

$1 - \lambda_d / \lambda_{max} = 1 - \lambda_{min} / \lambda_{max} = 1 - 1/\kappa$

$$(\mathbf{x}^{(S)} - \mathbf{x}^*) = \left(\mathbf{I} - \frac{1}{\lambda_{\max}}\mathbf{A}^T\mathbf{A}\right)^S (\mathbf{x}^{(0)} - \mathbf{x}^*)$$

What is the maximum eigenvalue of $\underline{\left(\mathbf{I} - \frac{1}{\lambda_{\max}}\mathbf{A}^T\mathbf{A}\right)^{2S}}$?

$$\left(1 - \frac{1}{\kappa}\right)^{2S} = \Big(\underbrace{\left(1 - \frac{1}{\kappa}\right)^{\kappa}}_{\leq \frac{1}{e}}\Big)^{2S/\kappa}$$

$$\leq \frac{1}{e}^{2S/\kappa} = e^{-2S/\kappa}$$

ACCELERATION

Nesterov's accelerated gradient descent:

$\sqrt{\kappa} \log(1/\varepsilon)$

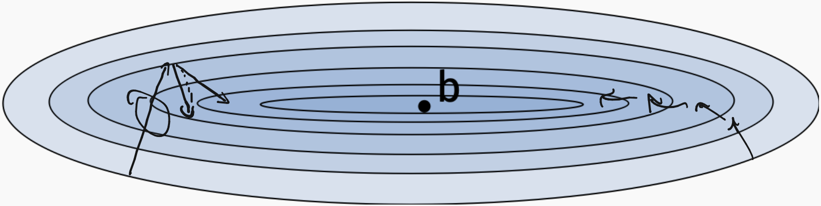
- $\mathbf{x}^{(0)} = \mathbf{y}^{(1)} = \mathbf{z}^{(1)}$
- For $t = 1, \ldots, T$
  - $\mathbf{y}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{\beta}\nabla f(\mathbf{x}^{(t)})$
  - $\mathbf{x}^{(t+1)} = \left(1 + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)\mathbf{y}^{(t+1)} + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\left(\mathbf{y}^{(t+1)} - \mathbf{y}^{(t)}\right)$

Theorem (AGD for β-smooth, α-strongly convex.)

Let f be a β-smooth and α-strongly convex function. If we run AGD for S steps we have:

$$f(\mathbf{x}^{(S)}) - f(\mathbf{x}^*) \leq \kappa e^{-S/\sqrt{\kappa}}\left[f(\mathbf{x}^{(0)}) - f(\mathbf{x}^*)\right]$$

Corollary: If $T = O\left(\sqrt{\kappa}\log(\kappa/\epsilon)\right)$ achieve error ϵ.



Level sets of $\|\mathbf{A}\mathbf{x}-\mathbf{b}\|_2^2$.

Other terms for similar ideas:

- Momentum
- Heavy-ball methods

What if we look back beyond <u>two iterates</u>?

BREAK

Second part of class:

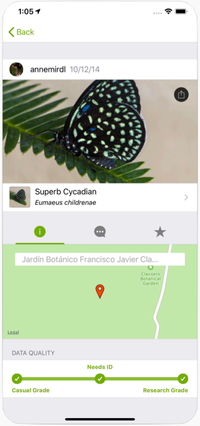
- Basics of Online Learning + Optimization.
- Introduction to Regret Analysis.
- Application to analyzing Stochastic Gradient Descent.

Many machine learning problems are solved in an online setting with constantly changing data.

- Spam filters are incrementally updated and adapt as they see more examples of spam over time.
- Image classification systems learn from mistakes over time (often based on user feedback).
- Content recommendation systems adapt to user behavior and clicks (which may not be a good thing...)

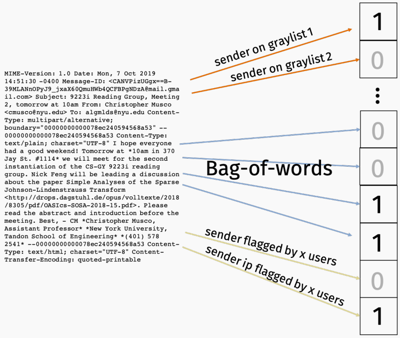
Plant identification via iNaturalist app.

(California Academy of Science + National Geographic)



- When the app fails, image is classified via crowdsourcing (backed by huge network of amateurs and experts).
- Single model that is updated constantly, not retrained in batches.

ML based email spam/scam filtering.



Markers for spam change overtime, so model might change.

ML based email spam/scam filtering.

Markers for spam change overtime, so model might change.

Choose some model $M_{\mathbf{x}}$ parameterized by parameters $\mathbf{x}$ and some loss function ℓ. At time steps $1, \ldots, T$, receive data vectors $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(T)}$.

$x^{(1)} \ldots x^{(T)}$

- At each time step, we pick ("play") a parameter vector $\mathbf{x}^{(i)}$.
- Make prediction $\tilde{y}^{(i)} = M_{\mathbf{x}^{(i)}}(\mathbf{a}_i)$. (i)
- Then told true value or label $y^{(i)}$.
- Goal is to minimize cumulative loss:

$\sum_{i=1}^{n} \ell(x, a^{(i)}, y^{(i)})$

$$L = \sum_{i=1}^{n} \ell(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)})$$

For example, for a regression problem we might use the ℓ_2 loss:

$$\ell(\mathbf{x}^{(i)}, \mathbf{a}^{(i)}, y^{(i)}) = \left| \langle \mathbf{x}^{(i)}, \mathbf{a}^{(i)} \rangle - y^{(i)} \right|^2 .$$

For classification, we could use logistic/cross-entropy loss.

$\ell(\cdot, a^{(i)}, y^{(i)})$

Abstraction as optimization problem: Instead of a single objective function f, we have a single (initially unknown) function $f_1, \ldots, f_T : \mathbb{R}^d \to \mathbb{R}$ for each time step.

- For time step $i \in 1, \ldots, T$, select vector $\mathbf{x}^{(i)}$.
- Observe f_i and pay cost $f_i(\mathbf{x}^{(i)})$
- Goal is to minimize $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})$.

We make no assumptions that $f_1, \ldots, f_T$ are related to each other at all!

In offline optimization, we wanted to find $\hat{x}$ satisfying $f(\hat{x}) \le \min_x f(x)$. Ask for a similar thing here.

"best solution in hindsight"

Objective: Choose $x^{(0)}, \ldots, x^{(T)}$ so that:

$$\frac{1}{T}\sum_{i=1}^{T} f_i(x^{(i)}) \le \frac{1}{T}\left[\min_{x} \sum_{i=1}^{T} f_i(x)\right] + \frac{\epsilon}{T}.$$

Here ϵ is called the regret of our solution sequence $x^{(0)}, \ldots, x^{(T)}$.

$c\sqrt{T}$

$\log(T)$

T

We typically ϵ to be growing sublinearly in T.

Regret compares to the best fixed solution in hindsight.

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right] + \epsilon.$$

It's very possible that $\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) < \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right]$. Could we hope for something stronger?

Exercise: Argue that the following is impossible to achieve:

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\sum_{i=1}^{T} \min_{\mathbf{x}} f_i(\mathbf{x})\right] + \epsilon.$$

Convex functions:

$$f_1(x) = |x - h_1|$$
$$\vdots$$
$$f_n(x) = |x - h_T|$$

where $h_1, \ldots, h_T$ are i.i.d. uniform $\{0, 1\}$.

$$\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)}) \leq \left[\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})\right] + \epsilon.$$

Beautiful balance:

- Either $f_1, \ldots, f_T$ are similar or changing slowly, so we can learn predict f_i from earlier functions.
- Or $f_1, \ldots, f_T$ are very different, in which case $\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ is large, so regret bound is easy to achieve.
- Or we live somewhere in the middle.

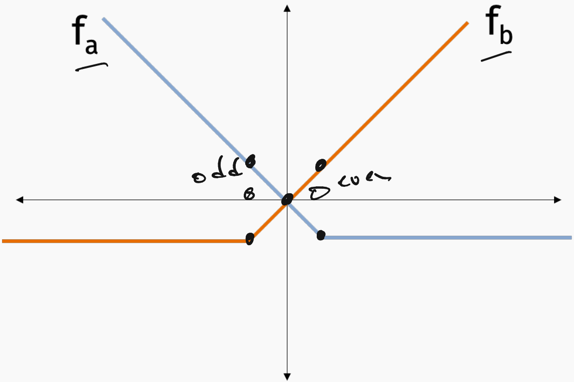
Follow-the-leader algorithm:

- Choose $\mathbf{x}^{(0)}$
- For $i = 1, \ldots, T$:
  - Let $\mathbf{x}^{(i)} = \arg\min_{j=1}^{i-1} f_j(\mathbf{x})$.
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.

$\arg\min \sum_{j=1}^{i-1} f_j(x)$

Simple and intuitive, but there are two issues with this approach. One is computational, one is related to the accuracy.

Hard case: $\underline{\frac{f_b}{2}}$ f_a f_b $\underline{f_a}$ f_b . . . . .



$x^* = \min_{x} \sum_{i=1}^{T} f_i(x)$

$\hookrightarrow$ offline.

Online Gradient descent:

- Choose $\mathbf{x}^{(1)}$ and $\eta = \frac{R}{G\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$

If $f_1, \ldots, f_T = f$ are all the same, this looks a lot like regular gradient descent. We update parameters using the gradient ∇f at each step.

ONLINE GRADIENT DESCENT (OGD)

best solution in hindsight

$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ (the offline optimum)

Assume:

- $f_1, \ldots, f_T$ are all convex.
- Each is G-Lipschitz: for all $\mathbf{x}, i$, $\|\nabla f_i(\mathbf{x})\|_2 \leq G$.
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

Online Gradient descent:

- Choose $\mathbf{x}^{(1)}$ and $\eta = \frac{R}{G\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Play $\mathbf{x}^{(i)}$.
  - Observe f_i and incur cost $f_i(\mathbf{x}^{(i)})$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_i(\mathbf{x}^{(i)})$

Let $\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_i(\mathbf{x})$ (the offline optimum)

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \frac{1}{T}\left[\underline{\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})}\right] - \frac{1}{T}\left[\underline{\sum_{i=1}^{T} f_i(\mathbf{x}^*)}\right] \leq \frac{RG\cancel{\sqrt{T}}}{\sqrt{T}}$

Average regret overtime is bounded by $\frac{\epsilon}{T} \leq \frac{RG}{\sqrt{T}}$.

Goes $\to 0$ as $T \to \infty$.

All this with no assumptions on how $f_1, \ldots, f_T$ relate to each other! They could have even been chosen adversarially – e.g. with f_i depending on our choice of $\mathbf{x}_i$ and all previous choices.

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}.$

Claim 1: For all $i = 1, \ldots, T$,

$$f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(i+1)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

(Same proof as last class. Only uses convexity of f_i.)

$$\|x^{(i+1)} - x^*\|$$

$$= \|x^{(i)} - \eta \nabla f_i(x^{(i)}) - x^*\|$$

Theorem (OGD Regret Bound)

After T steps, $\epsilon = \left[\sum_{i=1}^{T} f_i(\mathbf{x}^{(i)})\right] - \left[\sum_{i=1}^{T} f_i(\mathbf{x}^*)\right] \leq RG\sqrt{T}$.

Claim 1: For all $i = 1, \ldots, T$,

$$f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^*\|_2^2 - \|\mathbf{x}^{(i+1)} - \mathbf{x}^*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$$

Telescoping Sum:

$\leq R^2$ $\quad\quad$ ≥ 0

$$\sum_{i=1}^{T}\left[f_i(\mathbf{x}^{(i)}) - f_i(\mathbf{x}^*)\right] \leq \frac{\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2^2 - \cancel{\|\mathbf{x}^{(T)} - \mathbf{x}^*\|_2^2}}{2\eta} + \frac{T\eta G^2}{2}$$

$$\leq \frac{R^2}{2\eta} + \frac{T\eta G^2}{2}$$

$\frac{B}{G\sqrt{T}}$

$\leq RG\sqrt{T}$

\# of data points

Efficient offline optimization method for functions f with finite sum structure:

total loss

loss for one data point

ours data set

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}).$$

Goal is to find $\hat{\mathbf{x}}$ such that $f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*) + \epsilon$.

- The most widely use optimization algorithm in modern machine learning.
- Easily analyzed as a special case of online gradient descent!

Recall the machine learning setup. In empirical risk minimization, we can typically write:

$O(nd)$

$$f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$$

where f_i is the loss function for a particular data example $(\mathbf{a}^{(i)}, y^{(i)})$.

$f_i(x)$

$a^{(i)} \cdot \underbrace{2(x^T a^{(i)} - y^{(i)})}_{O(d)}$

Example: least squares linear regression.

$$f(\mathbf{x}) = \sum_{i=1}^{n} (\mathbf{x}^T\mathbf{a}^{(i)} - y^{(i)})^2$$

Note that by linearity, $\nabla f(\mathbf{x}) = \sum_{i=1}^{n} \nabla f_i(\mathbf{x})$.

Main idea: Use random approximate gradient in place of actual gradient.

Pick random $j \in 1, \ldots, n$ and update $\mathbf{x}$ using $\nabla f_j(\mathbf{x})$.

$$\mathbb{E}\left[\nabla f_j(\mathbf{x})\right] = \frac{1}{n}\nabla f(\mathbf{x}).$$

$$\mathbb{E}\left[\nabla f_j(x)\right] = \sum_{i=1}^{n} \frac{1}{n} \nabla f_i(x) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x) = \frac{1}{n} \nabla f(x)$$

$n\nabla f_j(\mathbf{x})$ is an unbiased estimate for the true gradient $\nabla f(\mathbf{x})$, but can often be computed in a $1/n$ fraction of the time!

Trade slower convergence for cheaper iterations.

Stochastic first-order oracle for $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$.

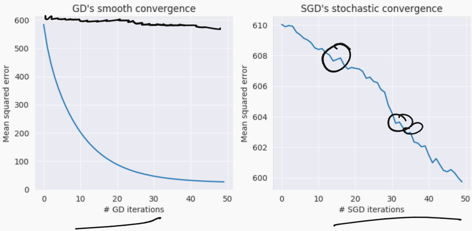
- **Function Query:** For any chosen $j, \mathbf{x}$, return $f_j(\mathbf{x})$
- **Gradient Query:** For any chosen $j, \mathbf{x}$, return $\nabla f_j(\mathbf{x})$

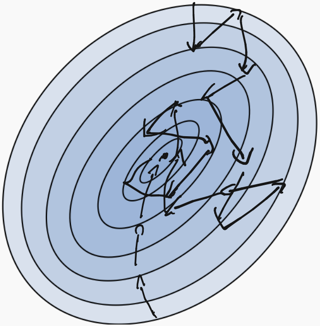
Stochastic Gradient descent:

- Choose starting vector $\underline{\mathbf{x}^{(1)}}$, learning rate $\underline{\eta}$
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, n$.
  - $\underline{\mathbf{x}^{(i+1)}} = \underline{\mathbf{x}^{(i)}} - \eta \underline{\nabla f_{j_i}(\mathbf{x}^{(i)})}$
- Return $\hat{\mathbf{x}} = \underline{\frac{1}{T}\sum_{i=1}^{T}} \mathbf{x}^{(i)}$

$j_1, \ldots j_T \in 1, \ldots, n$

VISUALIZING SGD





Assume:

- Finite sum structure: $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$, with $f_1, \ldots, f_n$ all convex.
- Lipschitz functions: for all $\mathbf{x}$, j, $\|\nabla f_j(\mathbf{x})\|_2 \leq \frac{G'}{n}$.
  - What does this imply about Lipschitz constant of f?
- Starting radius: $\|\mathbf{x}^* - \mathbf{x}^{(1)}\|_2 \leq R$.

$\|\nabla f(x)\|_2 \leq \sum_{j=1}^{n} \|\nabla f_j(x)\|_2$

$\leq n \cdot \frac{G'}{n} = G'$

Stochastic Gradient descent:

- Choose $\mathbf{x}^{(1)}$, steps T, learning rate $\eta = \frac{R}{G'\sqrt{T}}$.
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, n$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla f_{j_i}(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T}\sum_{i=1}^{T} \mathbf{x}^{(i)}$

Approach: View as online gradient descent run on function sequence $f_{j_1}, \ldots, f_{j_T}$.

Only use the fact that step equals gradient in expectation.

For a convex function f and points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}$

$$f\left(\frac{1}{t}\cdot \mathbf{x}^{(1)} + \ldots + \frac{1}{t}\cdot \mathbf{x}^{(t)}\right) \leq \frac{1}{t}\cdot f(\mathbf{x}^{(1)}) + \ldots + \frac{1}{t}\cdot f(\mathbf{x}^{(t)})$$

$$f\left(\frac{1}{2}x + \frac{1}{2}y\right) \leq \frac{1}{2}f(x) + \frac{1}{2}f(y)$$

$$\mathbb{E}[f(x^*)] = \boxed{\sum_{k=1,\dots,n} f_k(x^*)}$$

$$\mathbb{E}[f_{j_i}(x^*)] = \boxed{\sum_{k=1,\dots,n} \frac{1}{n} f_k(x^*)}$$

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$$\hat{x} = \frac{1}{T} \sum_{i=1}^{T} x^{(i)}$$

Claim 1:

$$f(\hat{\mathbf{x}}) - f(\mathbf{x}^*) \leq \frac{1}{T} \sum_{i=1}^{T} \left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*) \right]$$

Prove using Jensen's Inequality:

$$f\left(\frac{1}{T}\sum_{i=1}^{T} x^{(i)}\right) - f(x^*) \leq \left(\frac{1}{T}\sum_{i=1}^{T} f(x^{(i)})\right) - f(x^*)$$

Claim (SGD Convergence)

1. $\mathbb{E} f(x^{(i)}) = n \mathbb{E} f_{j_i}(x^{(i)})$

$f(x) = \sum_{j=1}^{n} f_j(x)$

fixed

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$j \neq j_i \rightarrow$ random

2. $\mathbb{E} f(x^*) = n \mathbb{E} f_{j_i}(x^*)$

$$\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] \leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[\boxed{f(\mathbf{x}^{(i)})} - \boxed{f(\mathbf{x}^*)}\right]$$

prev. slide

$$= \frac{1}{T}\sum_{i=1}^{T} \boxed{n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)})\right.} - \boxed{f_{j_i}(\mathbf{x}^*)}\Big]$$

$\sum_{x^{(i)}} \Pr(x^{(i)}) \boxed{f(x^{(i)})}$

$= \sum_{j \in 1,\dots,n,\, x^{(i)}} \Pr(x^{(i)}) \quad f_j(x^{(i)})$

$= \sum_{x^{(i)}} \Pr(x^{(i)}) \boxed{\sum_{j=1,\dots,n} f_j(x^{(i)})}$

$= \sum_{x^{(i)}} \sum_{j=1,\dots,n} \Pr(x^{(i)})\, f_j(x^{(i)})$

$\sum_{j \in 1,\dots,n,\, x^{(i)}} \Pr(x^{(i)}) \cdot \frac{1}{n} f_j(x^{(i)})$

$j_i \in 1,\dots,n$

STOCHASTIC GRADIENT DESCENT ANALYSIS

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$$\begin{aligned}
\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] &\leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right] \\
&= \frac{1}{T}\sum_{i=1}^{T} n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right] \\
&\leq \frac{n}{T} \cdot \mathbb{E}\left[\sum_{i=1}^{T} f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^{offline})\right],
\end{aligned}$$

$\frac{n}{T}\mathbb{E}\sum_{i=1}^{T}(f_{j_i}(x^{(i)}) - f_{j_i}(x^*))$

where $\mathbf{x}^{offline} = \arg\min_{\mathbf{x}} \sum_{i=1}^{T} f_{j_i}(\mathbf{x})$.

Claim (SGD Convergence)

After $T = \frac{R^2 G'^2}{\epsilon^2}$ *iterations:*

$$\mathbb{E}\left[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)\right] \leq \epsilon.$$

$T = \frac{R^2 G'^2}{\epsilon^2}$

$$\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] \leq \frac{1}{T}\sum_{i=1}^{T} \mathbb{E}\left[f(\mathbf{x}^{(i)}) - f(\mathbf{x}^*)\right]$$

$$= \frac{1}{T}\sum_{i=1}^{T} n\mathbb{E}\left[f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^*)\right]$$

$$\leq= \frac{n}{T}\mathbb{E}\left[\sum_{i=1}^{T} f_{j_i}(\mathbf{x}^{(i)}) - f_{j_i}(\mathbf{x}^{offline})\right] \rightarrow \frac{R\,G'}{n}\sqrt{T}$$

$$\leq \frac{\cancel{n}}{T} \cdot \left(R \cdot \frac{G'}{\cancel{n}} \cdot \sqrt{T}\right) \qquad \text{(by OGD guarantee.)}$$

$$\leq \frac{R G'}{\sqrt{T}}$$

Number of iterations for error ϵ:

- **Gradient Descent**: $T = \frac{R^2G^2}{\epsilon^2}$.
- **Stochastic Gradient Descent**: $T = \frac{R^2G'^2}{\epsilon^2}$.

Always have $G \leq G'$:

$$\begin{aligned}\max_{\mathbf{x}} \|\nabla f(\mathbf{x})\|_2 &\leq \max_{\mathbf{x}} \left(\|\nabla f_1(\mathbf{x})\|_2 + \ldots + \|\nabla f_n(\mathbf{x})\|_2\right) \\ &\leq \max_{\mathbf{x}} \left(\|\nabla f_1(\mathbf{x})\|_2\right) + \ldots + \max_{\mathbf{x}} \left(\|\nabla f_n(\mathbf{x})\|_2\right) \\ &\leq n \cdot \frac{G'}{n} = G'.\end{aligned}$$

$\nabla f(x) = \sum \nabla f_i(x)$

$\leq G'/n$

So GD converges strictly faster than *SGD*.

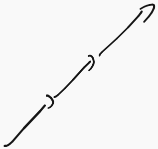
But for a fair comparison:

- SGD cost = (# of iterations) $\cdot\, O(1)$
- GD cost = (# of iterations) $\cdot\, O(n)$

STOCHASTIC VS. FULL BATCH GRADIENT DESCENT

We always have $G \leq G'$. When it is <u>much smaller</u> then GD will perform better. When it is closer to this upper bound, SGD will perform better.

What is an extreme case where $G = G'$?



What if each gradient $\nabla f_i(\mathbf{x})$ looks like random vectors in $\mathbb{R}^d$? E.g. with $\mathcal{N}(0,1)$ entries?

$\frac{G'}{n} = \max \|\nabla f_i(x)\|_2$

$$\mathbb{E}\left[\|\nabla f_i(\mathbf{x})\|_2^2\right] = \mathbb{E}\Big[\sum_{j=1,\ldots,d} n_j^2\Big] = d$$

$n_j \sim \mathcal{N}(0,1)$

$\frac{G'^2}{n^2} \approx d$

$G' \approx \sqrt{d}\, n$

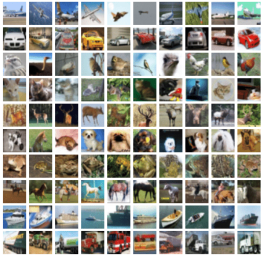
$$\mathbb{E}\left[\|\nabla f(\mathbf{x})\|_2^2\right] = \mathbb{E}\left[\|\sum_{i=1}^{n} \nabla f_i(\mathbf{x})\|_2^2\right] = dn$$

$$\begin{bmatrix} n_1^1 + n_2^1 + \ldots n_n^1 \\ n_1^2 + n_2^2 + \ldots n_n^2 \\ \vdots \end{bmatrix}$$

$\mathcal{N}(0, n)$

$G \approx \sqrt{d}\sqrt{n}$

Takeaway: SGD performs better when there is more structure or repetition in the data set.



PRECONDITIONING

Main idea: Instead of minimizing $f(\mathbf{x})$, find another function $g(\mathbf{x})$ with the same minimum but which is better suited for first order optimization (e.g., has a smaller conditioner number).

Claim: Let $h(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^d$ be an invertible function. Let $g(\mathbf{x}) = f(h(\mathbf{x}))$. Then

$$\min_{\mathbf{x}} f(\mathbf{x}) = \min_{\mathbf{x}} g(\mathbf{x}) \quad \text{and} \quad \arg\min_{\mathbf{x}} f(\mathbf{x}) = h\left(\arg\min_{\mathbf{x}} g(\mathbf{x})\right).$$

First Goal: We need $g(\mathbf{x})$ to still be convex.

Claim: Let $\mathbf{P}$ be an invertible $d \times d$ matrix and let $g(\mathbf{x}) = f(\mathbf{P}\mathbf{x})$.

$g(\mathbf{x})$ is always convex.

Second Goal:

$g(\mathbf{x})$ should have better condition number κ than $f(\mathbf{x})$.

Example:

- $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$. $\kappa_f = \frac{\lambda_1(\mathbf{A}^T\mathbf{A})}{\lambda_d(\mathbf{A}^T\mathbf{A})}$.
- $g(\mathbf{x}) = \|\mathbf{A}\mathbf{P}\mathbf{x} - \mathbf{b}\|_2^2$. $\kappa_g = \frac{\lambda_1(\mathbf{P}^T\mathbf{A}^T\mathbf{A}\mathbf{P})}{\lambda_d(\mathbf{P}^T\mathbf{A}^T\mathbf{A}\mathbf{P})}$.

Third Goal: $\mathbf{P}$ should be easy to compute.

Many, many problem specific preconditioners are used in practice. There design is usually a heuristic process.

Example: Diagonal preconditioner.

- Let $\mathbf{D} = \text{diag}(\mathbf{A}^T\mathbf{A})$
- Intuitively, we roughly have that $\mathbf{D} \approx \mathbf{A}^T\mathbf{A}$.
- Let $\mathbf{P} = \sqrt{\mathbf{D}^{-1}}$

$\mathbf{P}$ is often called a **Jacobi preconditioner**. Often works very well in practice!

DIAGONAL PRECONDITIONER

```
A =

    -734      1      33     9111      0
     -31     -2     108     5946    -19
     232     -1     101     3502     10
     426      0     -65    12503      9
    -373      0      26     9298      0
    -236     -2     -94     2398     -1
    2024      0    -132    -6904    -25
   -2258     -1      92    -6516      6
    2229      0       0    11921    -22
     338      1      -5   -16118    -23
```

```
>> cond(A'*A)

ans =

   8.4145e+07
```

```
>> P = sqrt(inv(diag(diag(A'*A))));
>> cond(P*A'*A*P)

ans =

   10.3878
```

Another view: If $g(\mathbf{x}) = f(\mathbf{P}\mathbf{x})$ then $\nabla g(\mathbf{x}) = \mathbf{P}^T \nabla f(\mathbf{P}\mathbf{x})$.

$\nabla g(\mathbf{x}) = \mathbf{P}\nabla f(\mathbf{P}\mathbf{x})$ when $\mathbf{P}$ is symmetric.

Gradient descent on g:

- For $t = 1, \ldots, T$,
  - $\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \eta \mathbf{P} \left[\nabla f(\mathbf{P}\mathbf{x}^{(t)})\right]$

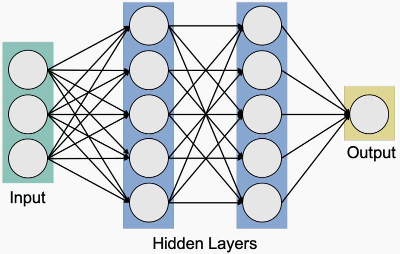
Gradient descent on g:

- For $t = 1, \ldots, T$,
  - $\mathbf{y}^{(t+1)} = \mathbf{y}^{(t)} - \eta \mathbf{P}^2 \left[\nabla f(\mathbf{y}^{(t)})\right]$

When $\mathbf{P}$ is diagonal, this is just gradient descent with a different step size for each parameter!

Algorithms based on this idea:

- AdaGrad
- RMSprop
- Adam optimizer



(Pretty much all of the most widely used optimization methods for training neural networks.)

COORDINATE DESCENT

Main idea: Trade slower convergence (more iterations) for cheaper iterations.

Stochastic Gradient Descent: When $f(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x})$, approximate $\nabla f(\mathbf{x})$ with $\nabla f_i(\mathbf{x})$ for randomly chosen i.

Main idea: Trade slower convergence (more iterations) for cheaper iterations.

Stochastic Coordinate Descent: Only compute a single random entry of $\nabla f(\mathbf{x})$ on each iteration:

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial x_d}(\mathbf{x}) \end{bmatrix} \qquad \nabla_i f(\mathbf{x}) = \begin{bmatrix} 0 \\ \frac{\partial f}{\partial x_i}(\mathbf{x}) \\ \vdots \\ 0 \end{bmatrix}$$

Update: $\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \eta \nabla_i f(\mathbf{x}^{(t)})$.