CS-GY 9223 I: Lecture 7
Preconditioning, acceleration, coordinate decent, etc.

NYU Tandon School of Engineering, Prof. Christopher Musco

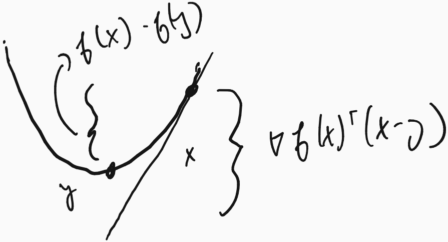
SMOOTH AND STRONGLY CONVEX

Recall from last lecture: a convex function f is β-smooth and α-strongly convex if, <u>for all</u> $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$,

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

$$\nabla f(x)^T (x-y) - [f(x) - f(y)] \geq 0$$

when f is convex.



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:

$$\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_2^2 \leq e^{-(t-1)\frac{\alpha}{\beta}}\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2^2$$

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

$\log 1/\varepsilon^2 = O(\log 1/\varepsilon)$ $\quad e^{-t/\kappa}$

Corollary: If $T = O\left(\kappa \log(1/\epsilon)\right)$ we have:

ε^2

$$\|\mathbf{x}^{(T)} - \mathbf{x}^*\|_2 \leq \epsilon\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2.$$

$\leq \varepsilon$

$e^{-O(\kappa \log(1/\varepsilon))/\kappa} = e^{-O(\log(1/\varepsilon))} = e^{O(\log(\varepsilon))} \leq \varepsilon.$

FROM LAST CLASS

Let $f(\mathbf{x}) = \|\mathbf{D}\mathbf{x} - \mathbf{b}\|_2^2$ where $\mathbf{D}$ is a diagonal matrix.

- $\beta = 2\max(\mathbf{D})^2$
- $\alpha = 2\min(\mathbf{D})^2$

$$\kappa = \frac{\max(D)^2}{\min(D)^2}$$

Gradient descent on f:

- $\mathbf{x}^{(1)} = \mathbf{0}$
- For $t = 1, \ldots, T$
  - $\underline{\mathbf{x}}^{(t+1)} = \underline{\mathbf{x}}^{(t)} - \frac{1}{\beta}\underbrace{(2\mathbf{D}(\mathbf{D}\mathbf{x}^{(t)} - \mathbf{b}))}$

$$f(x) = x^T D^2 x - 2x^T D b + b^T b$$

$$\nabla f(x) = 2D(Dx - b)$$

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:

$$\|\mathbf{x}^{(t)} - \mathbf{x}^*\|_2 \leq e^{-t/\kappa}\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2$$

Prove for $f(\mathbf{x}) = \|\mathbf{D}\mathbf{x} - \mathbf{b}\|_2^2$. You may assume that $\min(\mathbf{D})^2 > 0$.

$$\kappa = \frac{\max(D)^2}{\min(D)^2} \to \infty$$

$$\nabla[\|Dx-b\|_2^2] = 2D(Dx-b)$$

Alternate view:

$$2D(Dx^* - b) = 0$$

$$\boxed{D^2 x^* = Db}$$

$$\underline{(x^{(t+1)} - x^*) = \left(I - \frac{2}{\beta}D^2\right)(x^{(t)} - x^*)}$$

Original gradient step:

$$x^{(t+1)} = x^{(t)} - \frac{1}{\beta} 2D(Dx^{(t)} - b)$$

$$x^{(t+1)} - x^* = x^{(t)} - x^* - \frac{2}{\beta} D^2 x^{(t)} + \frac{2}{\beta} \underbrace{Db}_{D^2 x^*}$$

$$x^{(t+1)} - x^* = I(x^{(t)} - x^*) - \frac{2}{\beta} D^2 (x^{(t)} - x^*)$$

IN-CLASS EXERCISE

$$\underbrace{\left(x^{(t+1)} - x^*\right) = \left(I - \frac{2}{\beta} D^2\right)\left(x^{(t)} - x^*\right)}$$

$\beta = 2\max(D^2)$

$$\underbrace{(\mathbf{x}^{(t+1)} - \mathbf{x}^*)} = \underbrace{\left(\mathbf{I} - \frac{2}{\beta}\mathbf{D}^2\right)^t}_{\text{max entry} \leq 1} \underbrace{(\mathbf{x}^{(1)} - \mathbf{x}^*)}$$

$0 \leq$ entries ≤ 1

$0 \leq$ entries $\leq 1 - \frac{\min(D^2)}{\max(D^2)}$ $\boxed{1 - 1/\kappa}$

$\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_2 \leq$ $= \underbrace{\|(I - \frac{2}{\beta}D^2)^t (x^{(1)} - x^*)\|_2}_{\text{small}}$

$$V = \begin{bmatrix} v_1 & & 0 \\ & \ddots & \\ 0 & & v_d \end{bmatrix} \qquad V^t = \begin{bmatrix} v_1^t & & 0 \\ & \ddots & \\ 0 & & v_d^t \end{bmatrix}$$

$= \|V^t (x^{(1)} - x^*)\|_2$

$\leq e^{-t/\kappa} \|x^{(1)} - x^*\|_2$

$0 \leq$ entries $\leq (1 - 1/\kappa)^t = (1 - 1/\kappa)^{\kappa(t/\kappa)} \leq \boxed{e^{-t/\kappa}}$

$(1 - 1/\kappa)^\kappa \leq e^{-1}$

GENERAL LINEAR REGRESSION

This same analysis holds for any linear system minimized via gradient descent:

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \min_{\mathbf{x}} \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\mathbf{b}$$

$$\boxed{\nabla f(x) = 2A^T(Ax-b)} \qquad \boxed{A^TAx^* = A^Tb}$$

Unrolling gradient decent updates leads to:

$$(\mathbf{x}^{(t+1)} - \mathbf{x}^*) = \underbrace{\left(\mathbf{I} - \eta\mathbf{A}^T\mathbf{A}\right)^t}_{} (\mathbf{x}^{(1)} - \mathbf{x}^*).$$

$\eta = 2/\beta$

$$x^{(t+1)} = x^{(t)} - \eta A^TAx^{(t)} + \underbrace{\eta A^Tb}_{= \eta A^TAx^*}$$

$$x^{(t+1)} - x^* = x^{(t)} - x^* - \eta A^TA(x^{(t)} - x^*)$$

Quick linear algebra review: $A^T A = \begin{bmatrix} & \\ & \end{bmatrix} = d\begin{bmatrix} U \end{bmatrix}\begin{bmatrix} \Lambda \end{bmatrix}\begin{bmatrix} U^T \end{bmatrix}$

- $\mathbf{A}^T\mathbf{A}$ is symmetric so has an orthogonal eigendecomposition: $\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$.
  - $\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}$. U is orthonormal
  - $\boldsymbol{\Lambda}$ is diagonal with entries $\lambda_1 \geq \lambda_2 \geq \ldots, \lambda_d$.

Claim: $\lambda_d \geq 0$ (i.e., $\mathbf{A}^T\mathbf{A}$ is positive semidefinite).

Defn. $A^T A$ is positive semidefinite if for all x, $x^T \underbrace{A^T A x}_{y} \geq 0$.

$y^T y = \|y\|_2^2 \geq 0$

Verify outside of class: $\kappa = \frac{\lambda_1}{\lambda_d} = \frac{\max(\Lambda)}{\min(\Lambda)}$

$f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ is $2\lambda_1$ smooth and $2\lambda_d$ strongly convex. So we have: $\kappa = \frac{\lambda_1}{\lambda_d}$

$I = UU^T \quad I = UIU^T$

$A = U\Lambda U^T$

$\frac{1}{\beta} = \max(\Lambda)$

$$(\mathbf{x}^{(t+1)} - \mathbf{x}^*) = (\mathbf{I} - \eta\mathbf{A}^T\mathbf{A})^t(\mathbf{x}^{(1)} - \mathbf{x}^*).$$

$$(\mathbf{I} - \eta\mathbf{A}^T\mathbf{A})^t = (\mathbf{U}\underbrace{(\mathbf{I} - \eta\mathbf{\Lambda})}_{V}\mathbf{U}^T)^t = \mathbf{U}(\mathbf{I} - \eta\mathbf{\Lambda})^t\mathbf{U}^T$$

$I - \eta A^T A$

has all entries $\in [0, 1 - 1/\kappa]$

$(UVU^T)^t$

$= UV\cancel{U^T U}V\cancel{U^T} \cdots \cancel{U}VU^T$

UV^tU^T

$$\|\mathbf{x}^{(t+1)} - \mathbf{x}^*\|_2 =$$

$= \|U(I - \eta\Lambda)^t U^T(x^{(1)} - x^*)\|_2$

$= \|(I - \eta\Lambda)^t U^T(x^{(1)} - x^*)\|_2 \leq e^{-t/\kappa}\|U^T(x^{(1)} - x^*)\|_2 = e^{-t/\kappa}\|x^{(1)} - x^*\|_2$

We now have a really good understanding of gradient descent.

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	$O\left(\frac{G^2}{\alpha\epsilon}\right)$	$O\left(\frac{\beta}{\alpha}\log(1/\epsilon)\right)$

How do we use this understanding to design faster algorithms?

ACCELERATION

Total runtime for solving linear regression via GD: $d \leq n$

(time per iteration) x (number of iterations)

$\min \|Ax - b\|_2^2$

$2A^T(Ax - b)$

$O(\kappa \log 1/\varepsilon)$

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

$\boxed{O(nd \cdot \kappa \log(1/\epsilon))}$

for $\mathbf{A} \in \mathbb{R}^{n \times d}$, $\mathbf{x} \in \mathbb{R}^d$, $\mathbf{b} \in \mathbb{R}^n$.

$MV(A, v) \to Av$

$x^* = (A^TA)^{-1} A^T b$

$O(d^3)$

Often can be implemented in $\ll O(nd)$

$O(nd)$

$(d \times n)(n \times d)$ $\to O(nd^2)$

$\to O(nd^2)$

ACCELERATION

Theorem (Accelerated Iterative Regression)

Let $\mathbf{x}^* = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$*. There is an algorithm which finds* $\tilde{\mathbf{x}}$ *with* $\|\tilde{\mathbf{x}} - \mathbf{x}^*\|_2 \leq \epsilon \|\mathbf{x}^*\|_2$ *in time:*

$$O(nd \cdot \underbrace{\sqrt{\kappa} \log(1/\epsilon)}_{})$$

$\to$ degree q

Claim: For any η, polynomial $p(z) = c_1 z + c_2 z^2 + \ldots + c_q z^q$ with $p(1) = \sum_{j=1}^{q} c_q = 1$, there is an algorithm running in $\boxed{O(ndq)}$ time which outputs $\tilde{\mathbf{x}}$ satisfying:

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

$x^* - \tilde{x}$

$$\cancel{\mathbf{x} - \mathbf{x}^*} = p(\mathbf{I} - \eta\mathbf{A}^T\mathbf{A})\mathbf{x}^*$$

$$x^* - \tilde{x} = p(I - \eta A^T A)x^*$$

For standard gradient descent, $p(z) = z^q$.

$x^{(1)} = 0$

Gradient descent gives:

z^q

$$x^* - \tilde{x} = (I - \eta A^T A)^q (x^* - \cancel{x^{(1)}})$$

$$x^* - \tilde{x} = (I - \eta A^T A)^q x^*$$

$O(nd\,q)$

Claim: For any η, polynomial $p(z) = c_1 z + c_2 z^2 + \ldots + c_q z^q$ with $p(1) = \sum_{j=1}^{q} c_q = 1$, there is an algorithm running in $O(ndq)$ time which outputs $\tilde{\mathbf{x}}$ satisfying:

$x^* - \tilde{x}$ $\quad p(I - \eta A^T A)x^*$

$$\cancel{\tilde{\mathbf{x}}} - \mathbf{x}^* = \underbrace{c_1 \cdot (\mathbf{I} - \eta \mathbf{A}^T\mathbf{A})\mathbf{x}^* + c_2 \cdot (\mathbf{I} - \eta \mathbf{A}^T\mathbf{A})^2\mathbf{x}^* + \ldots + c_q \cdot (\mathbf{I} - \eta \mathbf{A}^T\mathbf{A})^q\mathbf{x}^*}$$

Claim: $c_j \cdot (\mathbf{I} - \eta \mathbf{A}^T\mathbf{A})^j \mathbf{x}^* = \underline{c_j \cdot \mathbf{x}^*} + \underline{p_j'}(\underline{\mathbf{I} - \eta \mathbf{A}^T\mathbf{A}})\mathbf{A}^T\mathbf{A}\mathbf{x}^*$ where p_j is a polynomial with degree $j - 1$.

$$c_j (I - \eta A^T A)^j x^* = c_j (I - \eta j A^T A + \ldots A^T A^2 + \ldots) x^*$$

$$= c_j x^* - (\qquad\qquad)$$

$$= \underline{c_j x^*} - (\underbrace{\qquad\qquad}) \underline{A^T A x^*}$$

Claim: For any η, polynomial $p(z) = c_1 z + c_2 z^2 + \ldots + c_q z^q$ with $p(1) = \sum_{j=1}^{q} c_q = 1$, there is an algorithm running in $O(ndq)$ time which outputs $\tilde{\mathbf{x}}$ satisfying:

$$\mathbf{x}^* - \tilde{\mathbf{x}} = \underbrace{(c_1 + c_2 + \ldots + c_q)}_{1} \cdot \mathbf{x}^* + p'(\mathbf{I} - \eta \mathbf{A}^T \mathbf{A}) \overbrace{\mathbf{A}^T \mathbf{A} \mathbf{x}^*}^{A^T b}$$

$p' = p_1' + p_2' + \ldots + p_q'$

$\tilde{\mathbf{x}} = p'(\mathbf{I} - \eta \mathbf{A}^T \mathbf{A}) \mathbf{A}^T \mathbf{b}$ where p' is a polynmial with degree $q - 1$.

$(I - \eta A^T A) y$

$y - \eta A^T \underbrace{A y}_{O(nd)}$ $O(nd)$

$\cancel{x^*} - \tilde{x} = \cancel{x^*} + p'(I - \eta A^T A) A^T b$

$\tilde{x} = -p'(I - \eta \underbrace{A^T A}) \underbrace{A^T b}$

depends on p

Can compute $\tilde{x}$ in $O(ndq)$ time.

$O(d)$

p' has degree $q-1$

THE POLYNOMIAL VIEW

$p'(z) = c_1' z + c_2' z^2 + \dots c_q' z^q$

Time to compute

$p'(V) A^T b = c_1' V A^T b + c_2' V^2 A^T b + \dots$

$V^q y$ is $O(ndq)$

$$\tilde{x} - x^* = p(\mathbf{I} - \eta \mathbf{A}^T \mathbf{A}) x^*$$

$$p(\mathbf{I} - \eta \mathbf{A}^T \mathbf{A}) = \mathbf{U} p(\mathbf{I} - \eta \mathbf{\Lambda}) \mathbf{U}^T$$

$V = (I - \eta A^T A)$

eigenvalues

$(I - \eta A^T A)(I - \eta A^T A) \dots (I - \eta A^T A) y$

Compute $\tilde{x}$ in $O(ndq)$ time

$$\|\tilde{x} - x^*\|_2 = \|\mathbf{U} p(\mathbf{I} - \eta \mathbf{\Lambda}) \mathbf{U}^T x^*\|_2$$
$$= \|p(\mathbf{I} - \eta \mathbf{\Lambda}) \mathbf{U}^T x^*\|_2$$

$\leq \epsilon \|U^T x^*\|_2$

$= \epsilon \|x^*\|_2$

$p(1) = 1$

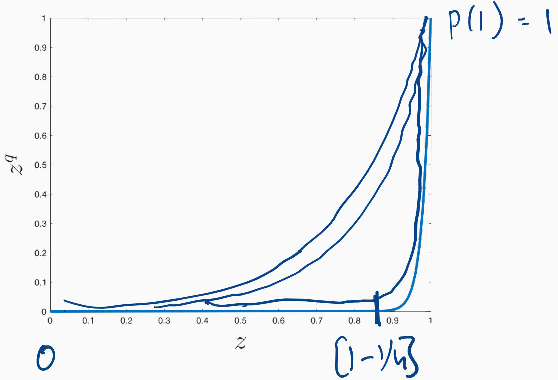
As long as $\max [p(\mathbf{I} - \eta \mathbf{\Lambda})] \leq \epsilon$,

$$\|\tilde{x} - x^*\|_2 \leq \epsilon \|x^*\|_2$$

$\eta = \frac{1}{\max(\Lambda)}$

All entries in $I - \eta\Lambda$ are between $[0, 1 - 1/\kappa]$

Goal: Find polynomial p such that $p(1) = 1$ and $p(z) \leq \epsilon$ for $z \in [0, 1 - \frac{1}{\kappa}]$.



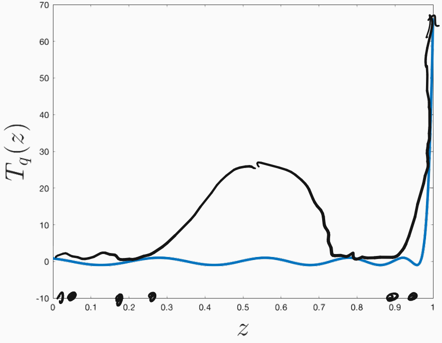
Gradient descent uses $p(z) = z^{O(\kappa \log(1/\epsilon))}$.

$p(z) \leq \varepsilon$ for $z \in [0, 1-1/\kappa]$

A BETTER JUMP POLYNOMIAL

Goal: Find polynomial p such that $p(1) = 1$ and $p(z) \leq \epsilon$ for $z \in [0, 1 - \frac{1}{\kappa}]$.

Gradient Descent for regression : Richardson Iteration



Accelerated:

"Chebyshev Iteration"

"Conjugate Gradient"

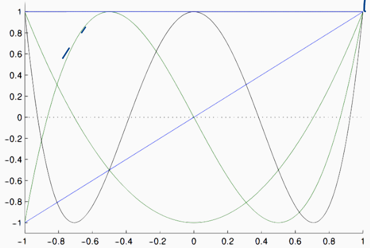
Can be done with degree $O(\sqrt{\kappa}\log(1/\epsilon))$ polynomial instead!

$\kappa \log(1/\epsilon)$

What are these polynomials?

Chebyshev polynomials of the first kind.

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2x^2 - 1$$
$$\vdots$$
$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$$



"There's only one bullet in the gun. It's called the Chebyshev polynomial." – Prof. Rocco Servedio

Nesterov's accelerated gradient descent:

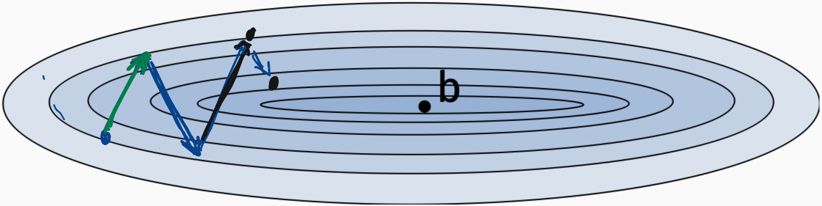
- $\mathbf{x}^{(1)} = \mathbf{y}^{(1)} = \mathbf{z}^{(1)}$
- For $t = 1, \ldots, T$
  - $\mathbf{y}^{(t+1)} = \mathbf{x}^{(t)} - \frac{1}{\beta}\nabla f(\mathbf{x}^{(t)})$
  - $\mathbf{x}^{(t+1)} = \left(1 + \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)\mathbf{y}^{(t+1)} - \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\mathbf{y}^{(t)}$

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

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

$$f(\mathbf{x}^{(t)}) - f(\mathbf{x}^*) \leq \kappa e^{-(t-1)\sqrt{\kappa}}\left[f(\mathbf{x}^{(1)}) - f(\mathbf{x}^*)\right]$$

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:

GD greedy : $O(\kappa)$

- Momentum
- Heavy-ball methods

AGD (two iterates) : $O(\sqrt{\kappa})$

Look at d iterates : $O(\kappa^{1/d})$

What if we look back beyond two iterates?

$O(\sqrt{\kappa})$

PRECONDITIONING

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. $\quad f(h(x)) = g(x)$

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

$h(x) = Px$

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

$$\begin{array}{cc} d & \\ \boxed{P} & d \\ d & \end{array}$$

Need to prove: for all x, y, $\lambda \in [0,1]$

$$\lambda g(x) + (1-\lambda) g(y) \geq g(\lambda x + (1-\lambda) y)$$

$$\lambda f(Px) + (1-\lambda) f(Py) \geq f(\lambda Px + (1-\lambda) Py)$$

$$= f(P(\lambda x + (1-\lambda) y))$$

$$\lambda g(x) + (1-\lambda) g(y) \geq g(\lambda x + (1-\lambda) y)$$

Second Goal: $g(x) = Px$

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

Example:

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

Ideal preconditioner: Choose P so that $\mathbf{P}^T\mathbf{A}^T\mathbf{A}\mathbf{P} = \mathbf{I}$. For example, could set $P = \sqrt{(\mathbf{A}^T\mathbf{A})^{-1}}$. $\rightarrow U\sqrt{\Lambda^{-1}}U^T$

$A^TA = U\Lambda U^T$

$U\sqrt{\Lambda^{-1}}\cancel{U^T U}\Lambda \cancel{U^T U}\sqrt{\Lambda^{-1}}U^T = P^TA^TAP = I$

What's the problem with this choice?

$= \min x^TP^TA^TAPx - x^TP^TA^Tb$

$(x^TIx - x^TP^TA^Tb)$

$\nabla g(x) = 2x - \underline{P^TA^Tb}$

Third Goal: **P** should be easy to compute.

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

$(A^T A)_{ii} = a_i^T a_i$

Example: Diagonal preconditioner.

does not take $O(nd^2)$

takes $O(nd)$

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

$\sqrt{(A^T A)^{-1}}$

P is often called a Jacobi preconditioner. Often works very well in practice!

```
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
```

x $=$ b

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

ans =

   8.4145e+07
```

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

ans =

   10.3878
```

DIAGONAL PRECONDITIONER

Can you think of an example **A** where Jacobi preconditioning doesn't decrease a large κ?

$$\begin{bmatrix} 1 & .99 \\ .99 & 1 \end{bmatrix}$$

Can Jacobi preconditioning <u>increase</u> κ?

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

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

Gradient descent on g: $\nabla g(x^{(t)})$

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

$x^{(t)} \approx x^*_g$

precond **Gradient descent on ~~g~~ f:**

$y^{(t)} = Px^{(t)} \approx x^*_f$

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

ηP_{11}

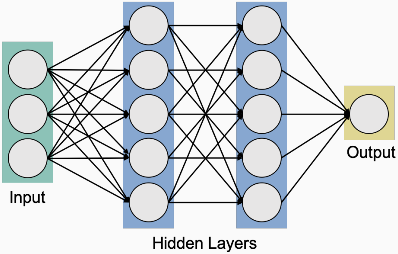
ηP_{22}

ηP_{dd}

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

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 $\underline{f(\mathbf{x})} = \underline{\sum_{i=1}^{n} f_i(\mathbf{x})}$, approximate $\underbrace{\nabla f(\mathbf{x})}$ with $\underbrace{\nabla f_i(\mathbf{x})}$ for randomly chosen i.

takes 1/n of the time to compute as $\nabla f(x)$

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

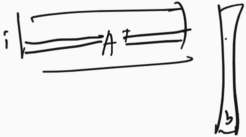
When $\mathbf{x}$ has d parameters, computing $\nabla_i f(\mathbf{x})$ often costs just a 1/d fraction of what it costs to compute $\nabla f(\mathbf{x})$

Example: $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$ for $\mathbf{A} \in \mathbb{R}^{n \times d}, \mathbf{x} \in \mathbb{R}^d, \mathbf{b} \in \mathbb{R}^n$.

- $\nabla f(\mathbf{x}) = \underbrace{2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b}}$.
- $\nabla_i f(\mathbf{x}) = 2\left[\mathbf{A}^T\mathbf{A}\mathbf{x}\right]_i - 2\left[\mathbf{A}^T\mathbf{b}\right]_i$.

$O(nd)$

$O(n)$ time to compute $\nabla_i f(x)$.



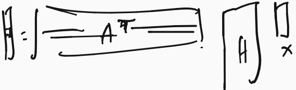
$O(n)$

$x^{(t)} = x^{(t-1)} + v_i$

where v_i is only non-zero in position i.

Have $Ax^{(t-1)} + Av_i$?

can compute $Ax^{(t)}$ in $O(n)$ time



$a_i^T a^{(i)} x \neq a_i^T A x$

i^{th} column i^{th} row of A

STOCHASTIC COORDINATE DESCENT

Stochastic Coordinate Descent:

- Choose number of steps T and step size η.
- For $i = 1, \ldots, T$:
  - Pick random $j_i \in 1, \ldots, d$.
  - $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} - \eta \nabla_{j_i} f(\mathbf{x}^{(i)})$
- Return $\hat{\mathbf{x}} = \frac{1}{T} \sum_{i=1}^{T} \mathbf{x}^{(i)}$.

$$n \, \mathbb{E} \, \nabla_{j_i} f(x) = \nabla f(x)$$

Theorem (Stochastic Coordinate Descent convergence)

Given a G-Lipschitz function f with minimizer $\mathbf{x}^*$ *and initial point* $\mathbf{x}^{(1)}$ *with* $\|\mathbf{x}^{(1)} - \mathbf{x}^*\|_2 \leq R$, *SCD with step size* $\eta = \frac{1}{Rd}$ *satisfies the guarantee:*

Lipschitz constant

$$\mathbb{E}[f(\hat{\mathbf{x}}) - f(\mathbf{x}^*)] \leq \frac{2GR}{\sqrt{T/d}} \rightarrow \text{distance to optimal}$$

γd

$$\frac{2GR}{\sqrt{T}}$$

$$T = O\left(\frac{G^2R^2}{\epsilon^2}\right)$$

Often it doesn't make sense to sample i uniformly at random:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -.5 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 10 \\ 42 \\ -11 \\ -51 \\ 34 \\ -22 \end{bmatrix}$$

Select indices i proportional to $\|\mathbf{a}_i\|_2^2$:

$$\Pr[\text{select index } i \text{ to update}] = \frac{\|\mathbf{a}_i\|_2^2}{\sum_{j=1}^{d} \|\mathbf{a}_j\|_2^2} = \frac{\|\mathbf{a}_i\|_2^2}{\|\mathbf{A}\|_2^2}$$