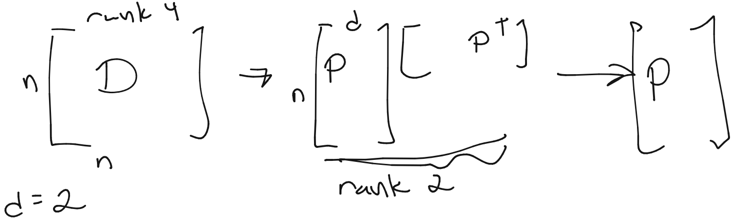


COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Spring 2020.

Lecture 21

$$D = U \Sigma W^T$$
$$D^T D = W \Sigma U^T U \Sigma W^T = W \Sigma^2 W^T$$

Problem Set 3 Solutions Posted.

$\lambda_1 \ldots \lambda_n$ $\sigma_1 \ldots \sigma_n$

eig vs svd

$$D = V \Lambda V^T = V \Lambda S V^T$$
$$D = U \Sigma W^T$$

singular values of D squared
eigenvalues of $D^T D$

Last Class: Fast computation of the SVD/eigendecomposition.

- Power method for computing the top singular vector of a matrix.
- Power method is a simple iterative algorithm for solving the *non-convex* optimization problem:

$$\max_{\vec{v}:\|\vec{v}\|_2^2 \le 1} \vec{v}^T \mathbf{X}^T \mathbf{X} \vec{v}.$$

This Class (and rest of semester):

- More general iterative algorithms for optimization, specifically gradient descent and its variants.
- What are these methods, when are they applied, and how do you analyze their performance?
- Small taste of what you can find in COMPSCI 590OP or 690OP.

$\vec{v}_1$ z^0 $z^{(t)} = X^T X z^{(t-1)}$

Theorem (Basic Power Method Convergence)

Let $\gamma = \frac{\sigma_1 - \sigma_2}{\sigma_1}$ be the relative gap between the first and second largest singular values. If Power Method is initialized with a random Gaussian vector $\vec{z}^{(0)}$ then, with high probability, after $t = O\left(\frac{\log d/\epsilon}{\gamma}\right)$ steps:

$$\|\vec{z}^{(t)} - \vec{v}_1\|_2 \leq \epsilon.$$

$O(nd^2)$

Total runtime: t matrix-vector multiplications with $\mathbf{X}^T\mathbf{X} \rightarrow 2t$ matrix-vector multiplications with $\mathbf{X}$.

$$O\left(\text{nnz}(\mathbf{X}) \cdot \frac{\log(d/\epsilon)}{\gamma} \cdot\right) = O\left(nd \cdot \frac{\log(d/\epsilon)}{\gamma}\right).$$

number of nonzeros

Krylov subspace methods (Lanczos method, Arnoldi method.)

- How `svds`/`eigs` are actually implemented. Only need $t = O\left(\frac{\log d/\epsilon}{\sqrt{\gamma}}\right)$ steps for the same guarantee.

Krylov subspace methods (Lanczos method, Arnoldi method.)

- How `svds`/`eigs` are actually implemented. Only need $t = O\left(\frac{\log d/\epsilon}{\sqrt{\gamma}}\right)$ steps for the same guarantee.

$X = U \Sigma V^T$

Main Idea: Need to separate σ_1 from σ_i for $i \geq 2$.

$V \Sigma^{2t} V^T z^{(0)}$

- Power method: $\vec{z}^{(t)} \propto (\mathbf{X}^T\mathbf{X})^t \cdot \vec{z}^{(0)}$ so component in the direction of v_i goes from $c_i \rightarrow (\sigma_i^2)^t \cdot c_i$.

$\sigma_1^{2t} >> \sigma_i^{2t}$

Krylov subspace methods (Lanczos method, Arnoldi method.)

- How `svds`/`eigs` are actually implemented. Only need $t = O\left(\frac{\log d/\epsilon}{\sqrt{\gamma}}\right)$ steps for the same guarantee.

Main Idea: Need to separate σ_1 from σ_i for $i \geq 2$.

- Power method: $\vec{z}^{(t)} \propto (\mathbf{X}^T\mathbf{X})^t \cdot \vec{z}^{(0)}$ so component in the direction of v_i goes from $c_i \to (\sigma_i^2)^t \cdot c_i$.
- Krylov methods: $\vec{z}^{(t)} \propto p_t(\mathbf{X}^T\mathbf{X}) \cdot \vec{z}^{(0)}$ where p_t is any degree t polynomial. So $c_i \to p_t(\sigma_i^2) \cdot c_i$

$3(X^TX)^t + 4(X^TX)^{t-1} + 5(X^TX)^{t-2} \ldots$

Krylov subspace methods (Lanczos method, Arnoldi method.)

- How `svds`/`eigs` are actually implemented. Only need $t = O\left(\frac{\log d/\epsilon}{\sqrt{\gamma}}\right)$ steps for the same guarantee.

Main Idea: Need to separate σ_1 from σ_i for $i \geq 2$.

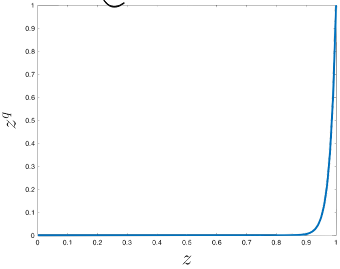
- Power method: $\vec{z}^{(t)} \propto (\mathbf{X}^T\mathbf{X})^t \cdot \vec{z}^{(0)}$ so component in the direction of v_i goes from $c_i \rightarrow (\sigma_i^2)^t \cdot c_i$.
- Krylov methods: $\vec{z}^{(t)} \propto p_t(\mathbf{X}^T\mathbf{X}) \cdot \vec{z}^{(0)}$ where p_t is any degree t polynomial. So $c_i \rightarrow p_t(\sigma_i^2) \cdot c_i$
- Still requires just $2t$ matrix vector multiplies. Why?

$(\sigma_1^2)^t >> (\sigma_i^2)^t$

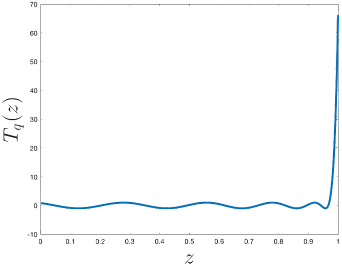
$z^{(t)} = [a_1 X^T X + a_2 (X^T X)^2 + \ldots a_t (X^T X)^t] z^{(0)}$

$X^T X z^{(0)}$ $\quad (X^T X) X^T X z^{(0)}$

$(\sigma_i^2)^t$ $\quad$ $p_t(\sigma_i^2)$



vs.



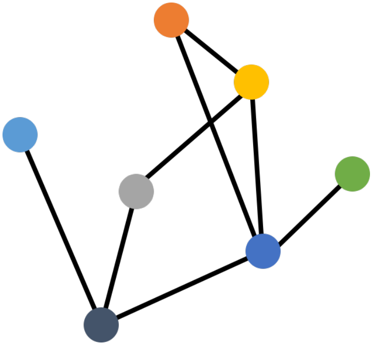
eigs svds.

Optimal 'jump' polynomial in general is given by a degree t Chebyshev polynomial. Krylov methods find a polynomial tuned to the input matrix $\mathbf{X}$ that does at least as well.

CONNECTION TO RANDOM WALKS

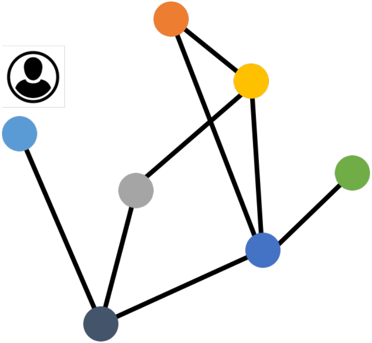
The power method is closely related to Markov chain convergence, random walks on graphs, and the PageRank algorithm.

Consider a random walk on a graph G with adjacency matrix $\mathbf{A}$.



The power method is closely related to Markov chain convergence, random walks on graphs, and the PageRank algorithm.

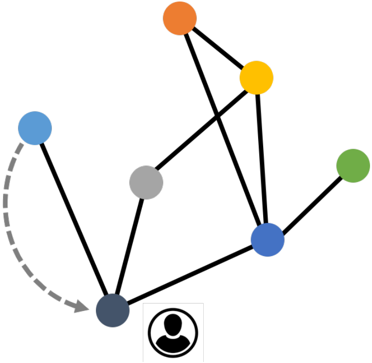
Consider a random walk on a graph G with adjacency matrix $\mathbf{A}$.



At each step, move to a random vertex, chosen uniformly at random from the neighbors of the current vertex.

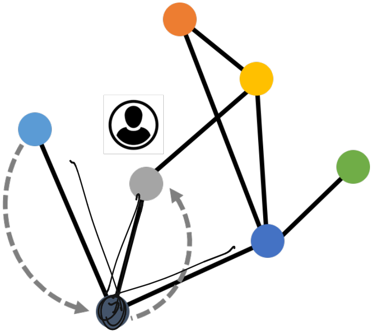
The power method is closely related to Markov chain convergence, random walks on graphs, and the PageRank algorithm.

Consider a random walk on a graph G with adjacency matrix $\mathbf{A}$.



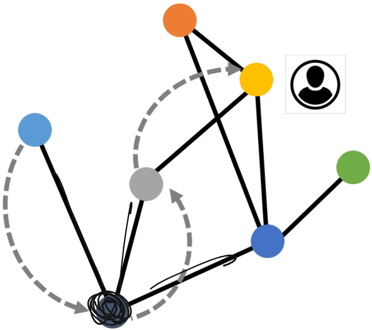
The power method is closely related to Markov chain convergence, random walks on graphs, and the PageRank algorithm.

Consider a random walk on a graph G with adjacency matrix $\mathbf{A}$.



The power method is closely related to Markov chain convergence, random walks on graphs, and the PageRank algorithm.

Consider a random walk on a graph G with adjacency matrix $\mathbf{A}$.



$p^{(0)} = [0\ 1\ 0\ 0\ 0\ 0]$ $p^{(1)} = [0\ 0\ \frac{1}{3}\ \frac{1}{3}\ \frac{1}{3}\ 0\ 0]$

CONNECTION TO RANDOM WALKS

Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \Pr(\text{walk at node i at step t})$.

Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \Pr(\text{walk at node i at step t})$.

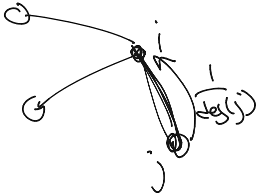
- **Initialize:** $\vec{p}^{(0)} = [1, 0, 0, \ldots, 0]$.

CONNECTION TO RANDOM WALKS

Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \text{Pr(walk at node i at step t)}$.

- **Initialize:** $\vec{p}^{(0)} = [1, 0, 0, \ldots, 0]$.
- **Update:**

$$\text{Pr(walk at i at step t)} = \sum_{j \in neigh(i)} \text{Pr(walk at j at step t-1)} \cdot \frac{1}{degree(j)}$$



Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \Pr(\text{walk at node i at step t})$.

- **Initialize:** $\vec{p}^{(0)} = [1, 0, 0, \ldots, 0]$.
- **Update:**

$$\Pr(\text{walk at i at step t}) = \sum_{j \in neigh(i)} \Pr(\text{walk at j at step t-1}) \cdot \frac{1}{degree(j)}$$

$$p^{(t)}(i) = \vec{z}^T \vec{p}^{(t-1)}$$

where $\vec{z}(j) = \frac{1}{degree(j)}$ for all $j \in neigh(i)$, $\vec{z}(j) = 0$ for all $j \notin neigh(i)$.

$$z^T p^{(t-1)} = \begin{bmatrix} 0 & \frac{1}{deg(j)} & 0 & 0 & \frac{1}{deg(j)} & 0 \end{bmatrix} \begin{bmatrix} \text{Pr at node 1 step t-1} \\ \text{Pr at node 2 step t-1} \\ \vdots \end{bmatrix}$$

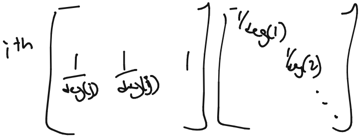
Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \text{Pr(walk at node i at step t)}$.

- **Initialize:** $\vec{p}^{(0)} = [1, 0, 0, \ldots, 0]$.
- **Update:**

$$\text{Pr(walk at i at step t)} = \sum_{j \in neigh(i)} \text{Pr(walk at j at step t-1)} \cdot \frac{1}{degree(j)}$$
$$= \vec{z}^T \vec{p}^{(t-1)}$$

where $\vec{z}(j) = \frac{1}{degree(j)}$ for all $j \in neigh(i)$, $\vec{z}(j) = 0$ for all $j \notin neigh(i)$.

- $\vec{z}$ is the i^{th} row of the right normalized adjacency matrix $\mathbf{AD}^{-1}$.



Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \Pr(\text{walk at node i at step t})$.

- **Initialize:** $\vec{p}^{(0)} = [1, 0, 0, \ldots, 0]$.
- **Update:**

$$\Pr(\text{walk at i at step t}) = \sum_{j \in neigh(i)} \Pr(\text{walk at j at step t-1}) \cdot \frac{1}{degree(j)}$$
$$= \vec{z}^T \vec{p}^{(t-1)}$$

where $\vec{z}(j) = \frac{1}{degree(j)}$ for all $j \in neigh(i)$, $\vec{z}(j) = 0$ for all $j \notin neigh(i)$.

- $\vec{z}$ is the i^{th} row of the right normalized adjacency matrix $\mathbf{AD}^{-1}$.
- $\vec{p}^{(t)} = \mathbf{AD}^{-1}\vec{p}^{(t-1)}$

$p^{(0)}$

$p^{(t)} = (AD^{-1})^t p^{(0)}$

Let $\vec{p}^{(t)} \in \mathbb{R}^n$ have i^{th} entry $\vec{p}^{(t)}(i) = \Pr(\text{walk at node i at step t})$.

- **Initialize:** $\vec{p}^{(0)} = [1, 0, 0, \ldots, 0]$.
- **Update:**

$$\begin{aligned}\Pr(\text{walk at i at step t}) &= \sum_{j \in neigh(i)} \Pr(\text{walk at j at step t-1}) \cdot \frac{1}{degree(j)} \\ &= \vec{z}^T \vec{p}^{(t-1)}\end{aligned}$$

where $\vec{z}(j) = \frac{1}{degree(j)}$ for all $j \in neigh(i)$, $\vec{z}(j) = 0$ for all $j \notin neigh(i)$.

- $\vec{z}$ is the i^{th} row of the right normalized adjacency matrix $\mathbf{AD}^{-1}$.
- $\vec{p}^{(t)} = \mathbf{AD}^{-1}\vec{p}^{(t-1)} = \underbrace{\mathbf{AD}^{-1}\mathbf{AD}^{-1}\ldots\mathbf{AD}^{-1}}_{t \text{ times}}\vec{p}^{(0)}$

Claim: After t steps, the probability that a random walk is at node i is given by the i^{th} entry of

$$\vec{p}^{(t)} = \underbrace{\mathbf{A}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}\ldots\mathbf{A}\mathbf{D}^{-1}}_{t \text{ times}}\vec{p}^{(0)}.$$

Claim: After t steps, the probability that a random walk is at node i is given by the i^{th} entry of

$$\vec{p}^{(t)} = \underbrace{\mathbf{A}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}\ldots\mathbf{A}\mathbf{D}^{-1}}_{t \text{ times}}\vec{p}^{(0)}.$$

$$\mathbf{D}^{-1/2}\vec{p}^{(t)} = \underbrace{(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\ldots(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})}_{t \text{ times}}(\mathbf{D}^{-1/2}\vec{p}^{(0)}).$$

$$D^{-1/2}p^{t} = D^{-1/2}AD^{-1}AD^{-1}\ldots AD^{-1}p^{0}$$

Claim: After t steps, the probability that a random walk is at node i is given by the i^{th} entry of

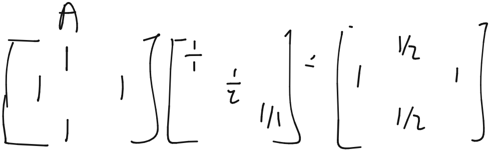
$$\vec{p}^{(t)} = \underbrace{\mathbf{A}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}\ldots\mathbf{A}\mathbf{D}^{-1}}_{t \text{ times}}\vec{p}^{(0)}.$$

$(X^T \quad X) \quad (X^T X) \ldots$

$$\mathbf{D}^{-1/2}\vec{p}^{(t)} = \underbrace{(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\ldots(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})}_{t \text{ times}}(\mathbf{D}^{-1/2}\vec{p}^{(0)}).$$

normalized adjacency matrix

- $\mathbf{D}^{-1/2}\vec{p}^{(t)}$ is exactly what would obtained by applying $t/2$ iterations of power method to $\mathbf{D}^{-1/2}\vec{p}^{(0)}$!



RANDOM WALKING AS POWER METHOD

Claim: After t steps, the probability that a random walk is at node i is given by the i^{th} entry of

$$\vec{p}^{(t)} = \underbrace{\mathbf{A}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}\ldots\mathbf{A}\mathbf{D}^{-1}}_{t \text{ times}}\vec{p}^{(0)}.$$

$$\mathbf{D}^{-1/2}\vec{p}^{(t)} = \underbrace{(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\ldots(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})}_{t \text{ times}}(\mathbf{D}^{-1/2}\vec{p}^{(0)}).$$

- $\mathbf{D}^{-1/2}\vec{p}^{(t)}$ is exactly what would obtained by applying $t/2$ iterations of power method to $\mathbf{D}^{-1/2}\vec{p}^{(0)}$! $D^{-1/2} p^{(t)}$
- Converges to the top eigenvector of the normalized adjacency matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. $\vec{p}^{(t)} \rightarrow$ stationary distribution.

$p^{(t)} = D^{1/2} D^{-1/2} p^{(t)}$

RANDOM WALKING AS POWER METHOD

Claim: After t steps, the probability that a random walk is at node i is given by the i^{th} entry of

$$\vec{p}^{(t)} = \underbrace{\mathbf{A}\mathbf{D}^{-1}\mathbf{A}\mathbf{D}^{-1}\ldots\mathbf{A}\mathbf{D}^{-1}}_{t \text{ times}}\vec{p}^{(0)}.$$

$$\mathbf{D}^{-1/2}\vec{p}^{(t)} = \underbrace{(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\ldots(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})}_{t \text{ times}}(\mathbf{D}^{-1/2}\vec{p}^{(0)}).$$

- $\mathbf{D}^{-1/2}\vec{p}^{(t)}$ is exactly what would obtained by applying $t/2$ iterations of power method to $\mathbf{D}^{-1/2}\vec{p}^{(0)}$!
- Converges to the top eigenvector of the normalized adjacency matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. $\vec{p}^{(t)} \rightarrow$ stationary distribution.
- Like the power method, the time a random walk takes to converge to its stationary distribution (mixing time) is dependent on the gap between the top two ~~eigenvalues~~ singular values of $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The spectral gap.

Questions on Power/Krylov Methods?

Discrete (Combinatorial) Optimization: (traditional CS algorithms)

- Graph Problems: min-cut, max flow, shortest path, matchings, maximum independent set, traveling salesman problem
- Problems with discrete constraints or outputs: bin-packing, scheduling, sequence alignment, submodular maximization
- Generally searching over a finite but exponentially large set of possible solutions. Many of these problems are NP-Hard.

Discrete (Combinatorial) Optimization: (traditional CS algorithms)

- Graph Problems: min-cut, max flow, shortest path, matchings, maximum independent set, traveling salesman problem
- Problems with discrete constraints or outputs: bin-packing, scheduling, sequence alignment, submodular maximization
- Generally searching over a finite but exponentially large set of possible solutions. Many of these problems are NP-Hard.

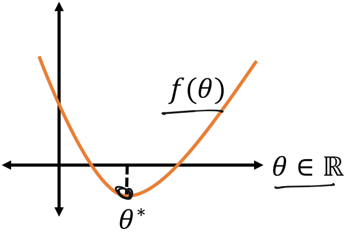
Discrete (Combinatorial) Optimization: (traditional CS algorithms)

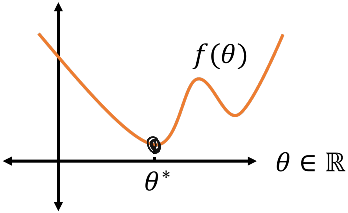
- Graph Problems: min-cut, max flow, shortest path, matchings, maximum independent set, traveling salesman problem
- Problems with discrete constraints or outputs: bin-packing, scheduling, sequence alignment, submodular maximization
- Generally searching over a finite but exponentially large set of possible solutions. Many of these problems are NP-Hard.

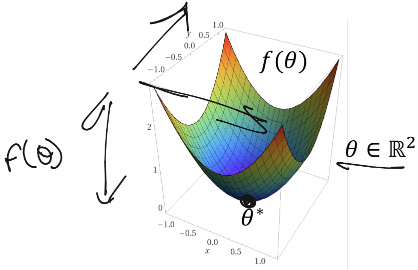
Continuous Optimization: (not covered in core CS curriculum. Touched on in ML/advanced algorithms, maybe.)

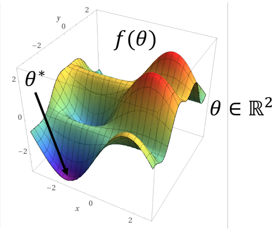
- Unconstrained convex and non-convex optimization.
- Linear programming, quadratic programming, semidefinite programming

CONTINUOUS OPTIMIZATION EXAMPLES









Given some function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, find $\vec{\theta}_\star$ with:

$$f(\vec{\theta}_\star) = \min_{\vec{\theta} \in R^d} f(\vec{\theta})$$

Given some function $f : \mathbb{R}^d \to \mathbb{R}$, find $\vec{\theta}_\star$ with:

$$f(\vec{\theta}_\star) = \min_{\vec{\theta} \in R^d} f(\vec{\theta}) + \epsilon$$

Typically up to some small approximation factor.

Given some function $f : \mathbb{R}^d \to \mathbb{R}$, find $\vec{\theta}_\star$ with:

$$f(\vec{\theta}_\star) = \min_{\vec{\theta} \in R^d} f(\vec{\theta}) + \epsilon$$

Typically up to some small approximation factor.

Often under some constraints:

- $\|\vec{\theta}\|_2 \leq 1, \quad \|\vec{\theta}\|_1 \leq 1.$
- $A\vec{\theta} \leq \vec{b}, \quad \vec{\theta}^T A \vec{\theta} \geq 0.$
- $\vec{1}^T \vec{\theta} = \sum_{i=1}^d \vec{\theta}(i) \leq c.$

v_1 top singular vector of X

$f(v) = v^T X^T X v$

s.t. $\|v\|_2 \leq 1$

WHY CONTINUOUS OPTIMIZATION?

WHY CONTINUOUS OPTIMIZATION?

Modern machine learning centers around continuous optimization.

WHY CONTINUOUS OPTIMIZATION?

Modern machine learning centers around continuous optimization.

Typical Set Up: (supervised machine learning)

- Have a model, which is a function mapping inputs to predictions (neural network, linear function, low-degree polynomial etc).
- The model is parameterized by a parameter vector (weights in a neural network, coefficients in a linear function or polynomial)
- Want to train this model on input data, by picking a parameter vector such that the model does a good job mapping inputs to predictions on your training data.

home price = $\underline{c_1}$ #bathrooms + $\underline{c_2}$ #beds

Modern machine learning centers around continuous optimization.

Typical Set Up: (supervised machine learning)

- Have a model, which is a function mapping inputs to predictions (neural network, linear function, low-degree polynomial etc).
- The model is parameterized by a parameter vector (weights in a neural network, coefficients in a linear function or polynomial)
- Want to train this model on input data, by picking a parameter vector such that the model does a good job mapping inputs to predictions on your training data.

This training step is typically formulated as a continuous optimization problem.

Example 1: Linear Regression

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle$ = θ(1) x(1) + . . . + θ(d) x(d)

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^d$ (the regression coefficients)

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^d$ (the regression coefficients)

Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

$$L(\vec{\theta}, \mathbf{X}, \vec{y}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

where ℓ is some measurement of how far $M_{\vec{\theta}}(\vec{x}_i)$ is from y_i.

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^d$ (the regression coefficients)

Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

$$L(\vec{\theta}, \mathbf{X}, \vec{y}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

where ℓ is some measurement of how far $M_{\vec{\theta}}(\vec{x}_i)$ is from y_i.

- $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \left(M_{\vec{\theta}}(\vec{x}_i) - y_i\right)^2$ (least squares regression)
- $y_i \in \{-1, 1\}$ and $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \ln\left(1 + \exp(-y_i M_{\vec{\theta}}(\vec{x}_i))\right)$ (logistic regression)

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^d$ (the regression coefficients)

Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

$$L(\vec{\theta}, \mathbf{X}, \vec{y}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i) + R(\vec{\theta})$$

where ℓ is some measurement of how far $M_{\vec{\theta}}(\vec{x}_i)$ is from y_i.

- $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \left(M_{\vec{\theta}}(\vec{x}_i) - y_i\right)^2$ (least squares regression)
- $y_i \in \{-1, 1\}$ and $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \ln\left(1 + \exp(-y_i M_{\vec{\theta}}(\vec{x}_i))\right)$ (logistic regression)

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^d$ (the regression coefficients)

Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

$$L(\vec{\theta}, \mathbf{X}, \vec{y}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i) + \lambda \|\vec{\theta}\|_2^2$$

where ℓ is some measurement of how far $M_{\vec{\theta}}(\vec{x}_i)$ is from y_i.

- $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \left(M_{\vec{\theta}}(\vec{x}_i) - y_i\right)^2$ (least squares regression)
- $y_i \in \{-1, 1\}$ and $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \ln\left(1 + \exp(-y_i M_{\vec{\theta}}(\vec{x}_i))\right)$ (logistic regression)

Example 1: Linear Regression

Model: $M_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}$ with $M_{\vec{\theta}}(\vec{x}) \stackrel{\text{def}}{=} \langle \vec{\theta}, \vec{x} \rangle = \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^d$ (the regression coefficients)

Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

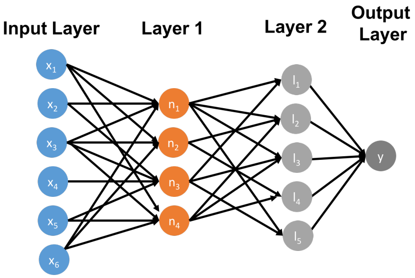
minimize w.r.t. to Θ

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = L(\vec{\theta}, \mathbf{X}, \vec{y}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i) + \lambda \|\vec{\theta}\|_2^2$$

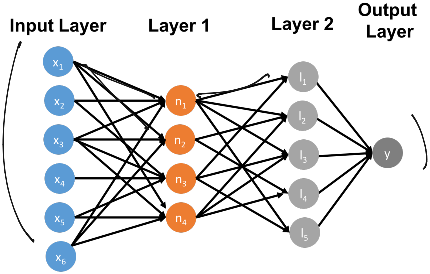
where ℓ is some measurement of how far $M_{\vec{\theta}}(\vec{x}_i)$ is from y_i.

- $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \left(M_{\vec{\theta}}(\vec{x}_i) - y_i\right)^2$ (least squares regression)
- $y_i \in \{-1, 1\}$ and $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \ln\left(1 + \exp(-y_i M_{\vec{\theta}}(\vec{x}_i))\right)$ (logistic regression)

Example 2: Neural Networks



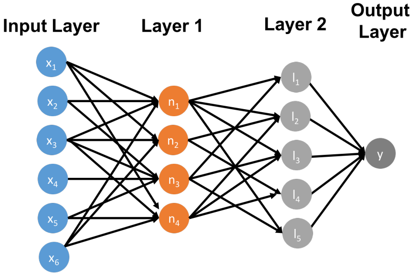
Example 2: Neural Networks



Model: $M_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^{(\# \ edges)}$ (the weights on every edge)

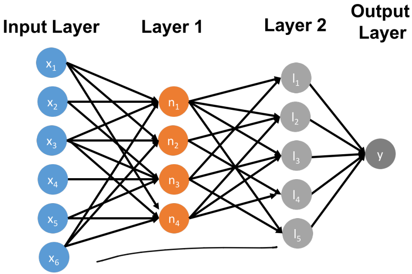
Example 2: Neural Networks



Model: $M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R}$. $M_{\vec{\theta}}(\vec{x}) = \langle \vec{w}_{out}, \sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\vec{x}))\rangle$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^{(\#\ edges)}$ (the weights on every edge)

Example 2: Neural Networks



Model: $M_{\vec{\theta}} : \mathbb{R}^d \to \mathbb{R}$. $M_{\vec{\theta}}(\vec{x}) = \langle \vec{w}_{out}, \sigma(\mathbf{W}_2 \sigma(\mathbf{W}_1 \vec{x})) \rangle$.

Parameter Vector: $\vec{\theta} \in \mathbb{R}^{(\#\ edges)}$ (the weights on every edge)

Optimization Problem: Given data points $\vec{x}_1, \ldots, \vec{x}_n$ and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

- Supervised means we have labels $y_1, \ldots, y_n$ for the training points.

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

- Supervised means we have labels $y_1, \ldots, y_n$ for the training points.
- Solving the final optimization problem has many different names: likelihood maximization, empirical risk minimization, minimizing training loss, etc.

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

- Supervised means we have labels $y_1, \ldots, y_n$ for the training points.
- Solving the final optimization problem has many different names: likelihood maximization, empirical risk minimization, minimizing training loss, etc.
- Continuous optimization is also very common in unsupervised learning.

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

- Supervised means we have labels $y_1, \ldots, y_n$ for the training points.
- Solving the final optimization problem has many different names: likelihood maximization, empirical risk minimization, minimizing training loss, etc.
- Continuous optimization is also very common in unsupervised learning. (PCA, spectral clustering, etc.)

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

- Supervised means we have labels $y_1, \ldots, y_n$ for the training points.
- Solving the final optimization problem has many different names: likelihood maximization, empirical risk minimization, minimizing training loss, etc.
- Continuous optimization is also very common in unsupervised learning. (PCA, spectral clustering, etc.)
- Generalization tries to explain why minimizing the loss $L_{\mathbf{X},\vec{y}}(\vec{\theta})$ on the *training points* minimizes the loss on future *test points*. I.e., makes us have good predictions on future inputs.

Choice of optimization algorithm for minimizing $f(\vec{\theta})$ will depend on many things:

- The form of f (in ML, depends on the model & loss function).
- Any constraints on $\vec{\theta}$ (e.g., $\|\vec{\theta}\| < c$).
- Computational constraints, such as memory constraints.

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

Choice of optimization algorithm for minimizing $f(\vec{\theta})$ will depend on many things:

- The form of f (in ML, depends on the model & loss function).
- Any constraints on $\vec{\theta}$ (e.g., $\|\vec{\theta}\| < c$).
- Computational constraints, such as memory constraints.

$$L_{\mathbf{X},\vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$

What are some popular optimization algorithms?

gradient descent (GD)

KKT

"Second Order"
- newtons method
- bfgs

linear programming.
quadratic programming.

ADMM

ADAM, Adagrad (variants on GD)
accelerated GD.