COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Cameron Musco

University of Massachusetts Amherst. Spring 2020.

Lecture 21

SUMMARY

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:

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

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.

POWER METHOD THEOREM

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 $\vec{v}^{(0)}$ then, with high probability, after $t = O\left(\frac{\log d/\epsilon}{\gamma}\right)$ steps:

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

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

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

KRYLOV SUBSPACE METHODS

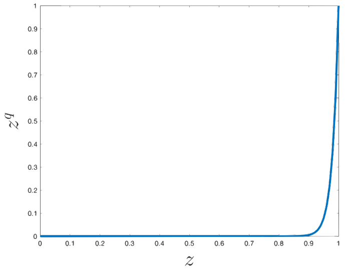
Krylov subspace methods (Lanczos method, Arnoldi method.)

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

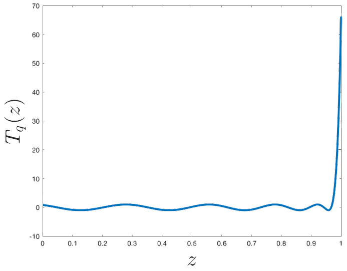
Main Idea: Need to separate σ_1 from σ_i for $i \geq 2$.

- 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?

KRYLOV SUBSPACE METHODS



vs.

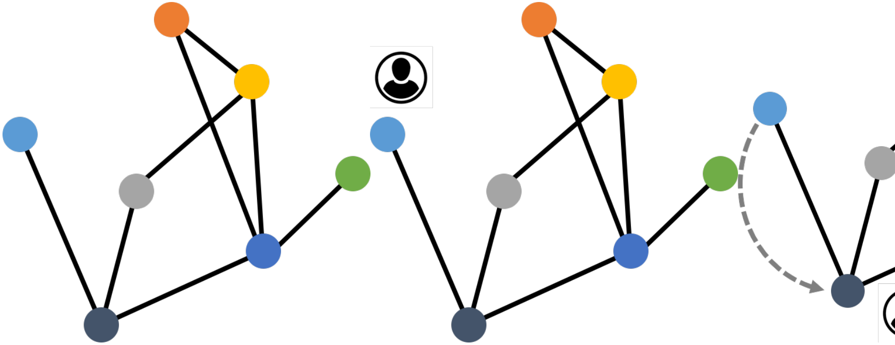


Optimal ‘jump’ polynomial in general is given by a degree t Chebyshev polynomial. Krylov methods find a polynomial tuned to the input matrix $\mathbf{X}$ that does at least as well.

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 $\mathbf{A}$.



At each step, move to a random vertex, chosen uniformly at random

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:**

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

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

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 $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. $\vec{p}^{(t)} \to$ 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 of $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. The spectral gap.

Questions on Power/Krylov Methods?

DISCRETE VS. CONTINUOUS OPTIMIZATION

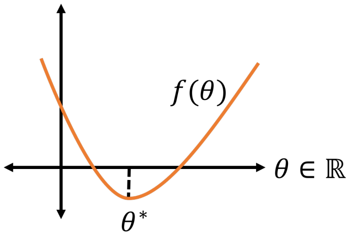
Discrete (Combinatorial) Optimization: (traditional CS algorithms)

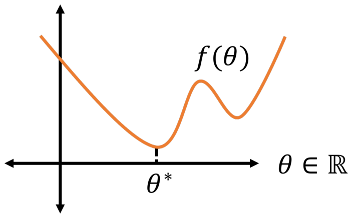
- Graph Problems: min-cut, max flow, shortest path, matchings, maximum independent set, traveling salesman problem
- Problems with discrete constraints or outputs: bin-packing, scheduling, sequence alignment, submodular maximization
- Generally searching over a finite but exponentially large set of possible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

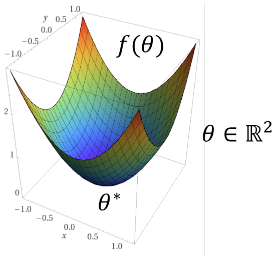
Continuous Optimization: (not covered in core CS curriculum. Touched on in ML/advanced algorithms, maybe.)

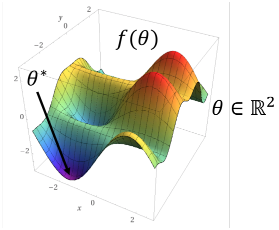
- Unconstrained convex and non-convex optimization.
- Linear programming, quadratic programming, semidefinite programming

CONTINUOUS OPTIMIZATION EXAMPLES









MATHEMATICAL SETUP

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:

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

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.

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

OPTIMIZATION IN ML

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)

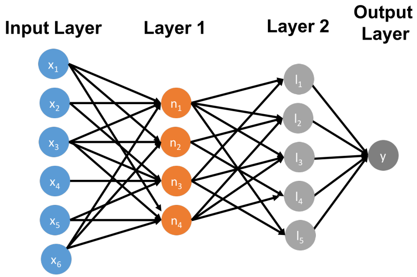
Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

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

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



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)

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:

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

OPTIMIZATION IN ML

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

- 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.

OPTIMIZATION ALGORITHMS

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?