COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Andrew McGregor

Lecture 21

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 $\max_{\vec{v}:\|\vec{v}\|_2^2=1} |\vec{v}^T \mathbf{A}\vec{v}|$

Final Two Weeks of Class:

- 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.

DISCRETE VS. CONTINUOUS OPTIMIZATION

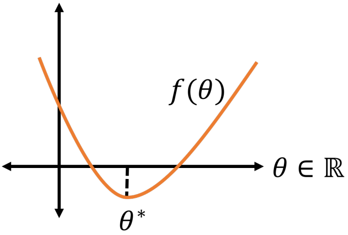
Discrete (Combinatorial) Optimization: (traditional CS algorithms)

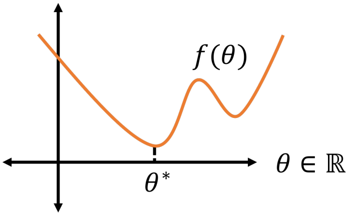
- Graph Problems: min-cut, max-cut, max flow, shortest path, matchings, maximum independent set, traveling salesman problem
- Problems with discrete constraints or outputs: bin-packing, scheduling, sequence alignment, submodular maximization
- Generally searching over a finite but exponentially large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

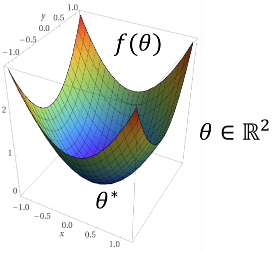
Continuous Optimization: (maybe seen in ML/advanced algorithms)

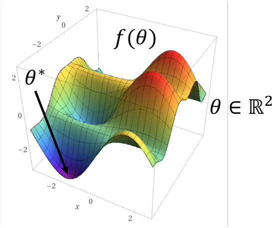
- Unconstrained convex and non-convex optimization.
- Linear programming, quadratic programming, semidefinite programming

CONTINUOUS OPTIMIZATION EXAMPLES









MATHEMATICAL SETUP

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

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

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 additive approximation term ϵ.

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

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

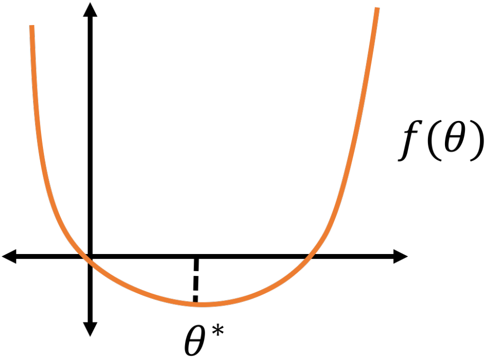
Typically up to some small additive approximation term ϵ.

Often under some constraints:

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

Definition – Convex Function: A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex iff, for any $\vec{\theta}_1, \vec{\theta}_2 \in \mathbb{R}^d$ and $\lambda \in [0, 1]$:

$$(1-\lambda) \cdot f(\vec{\theta}_1) + \lambda \cdot f(\vec{\theta}_2) \geq f\left((1-\lambda) \cdot \vec{\theta}_1 + \lambda \cdot \vec{\theta}_2\right)$$



Definition – Convex Set: A set $\mathcal{S} \subseteq \mathbb{R}^d$ is convex if and only if, for any $\vec{\theta}_1, \vec{\theta}_2 \in \mathcal{S}$ and $\lambda \in [0,1]$: $(1-\lambda)\vec{\theta}_1 + \lambda \cdot \vec{\theta}_2 \in \mathcal{S}$

Definition – Convex Set: A set $\mathcal{S} \subseteq \mathbb{R}^d$ is convex if and only if, for any $\vec{\theta}_1, \vec{\theta}_2 \in \mathcal{S}$ and $\lambda \in [0,1]$: $(1-\lambda)\vec{\theta}_1 + \lambda \cdot \vec{\theta}_2 \in \mathcal{S}$

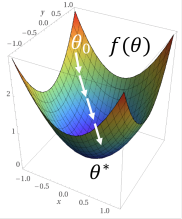
For any convex set let $P_{\mathcal{S}}(\cdot)$ denote the projection function onto $\mathcal{S}$:

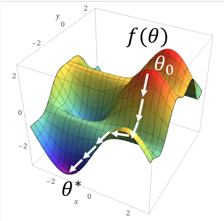
$$P_{\mathcal{S}}(\vec{y}) = \arg\min_{\vec{\theta} \in \mathcal{S}} \|\vec{\theta} - \vec{y}\|_2$$

GRADIENT DESCENT

Next few classes: Gradient descent (and some important variants)

- An extremely simple greedy iterative method, that can be applied to almost any continuous function we care about optimizing.
- Often not the 'best' choice for any given function, but it is the approach of choice in ML since it is simple, general, and often works very well.
- At each step, tries to move towards the lowest nearby point in the function that is can – in the opposite direction of the gradient.





Gradient Descent Update in 1D:

- Set θ_1 arbitrarily.
- For $i = 1$ to t:

$$\theta_{i+1} = \theta_i - \eta f'(\theta_i)$$

  i.e., increase θ if negative derivative and decrease θ if positive derivative. η is small fixed value.

- Return $\theta = \arg\min_{\theta_1, \ldots \theta_t} f(\theta_i)$.

Gradient Descent Update in 1D:

- Set θ_1 arbitrarily.
- For $i = 1$ to t:

$$\theta_{i+1} = \theta_i - \eta f'(\theta_i)$$

  i.e., increase θ if negative derivative and decrease θ if positive derivative. η is small fixed value.

- Return $\theta = \arg\min_{\theta_1,\ldots\theta_t} f(\theta_i)$.

Example: $f(x) = (x-1)^2$, $\theta_1 = 2$, and $\eta = 0.2$

BASIC IDEA OF GRADIENT DESCENT

Gradient Descent Update in 1D:

- Set θ_1 arbitrarily.
- For $i = 1$ to t:

$$\theta_{i+1} = \theta_i - \eta f'(\theta_i)$$

  i.e., increase θ if negative derivative and decrease θ if positive derivative. η is small fixed value.

- Return $\theta = \arg\min_{\theta_1, \ldots \theta_t} f(\theta_i)$.

Example: $f(x) = (x-1)^2$, $\theta_1 = 2$, and $\eta = 0.2$

- Compute derivative $f'(x) = 2(x-1)$

Gradient Descent Update in 1D:

- Set θ_1 arbitrarily.
- For $i = 1$ to t:

$$\theta_{i+1} = \theta_i - \eta f'(\theta_i)$$

  i.e., increase θ if negative derivative and decrease θ if positive derivative. η is small fixed value.
- Return $\theta = \arg\min_{\theta_1,\ldots\theta_t} f(\theta_i)$.

Example: $f(x) = (x-1)^2$, $\theta_1 = 2$, and $\eta = 0.2$

- Compute derivative $f'(x) = 2(x-1)$
- $\theta_2 = \theta_1 - \eta f'(\theta_1) = 2 - 0.2 \times f'(2) = 2 - 0.2 \times 2 = 1.6$.

Gradient Descent Update in 1D:

- Set θ_1 arbitrarily.
- For $i = 1$ to t:

$$\theta_{i+1} = \theta_i - \eta f'(\theta_i)$$

  i.e., increase θ if negative derivative and decrease θ if positive derivative. η is small fixed value.

- Return $\theta = \arg\min_{\theta_1,\ldots\theta_t} f(\theta_i)$.

Example: $f(x) = (x-1)^2$, $\theta_1 = 2$, and $\eta = 0.2$

- Compute derivative $f'(x) = 2(x-1)$
- $\theta_2 = \theta_1 - \eta f'(\theta_1) = 2 - 0.2 \times f'(2) = 2 - 0.2 \times 2 = 1.6$.
- $\theta_3 = \theta_2 - \eta f'(\theta_2) = 1.6 - 0.2 \times f'(1.6) = 1.6 - 0.2 \times 1.2 = 1.36$.

GD ANALYSIS PROOF FOR $d = 1$

Theorem: For convex function $f : \mathbb{R} \to \mathbb{R}$ where $|f'(\theta)| \leq G$ for all θ, GD run with $t \geq \frac{R^2 G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within R of θ_*, outputs $\hat{\theta}$ satisfying $f(\hat{\theta}) \leq f(\theta_*) + \epsilon$.

GD ANALYSIS PROOF FOR $d = 1$

Theorem: For convex function $f : \mathbb{R} \to \mathbb{R}$ where $|f'(\theta)| \le G$ for all θ, GD run with $t \ge \frac{R^2 G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within R of θ_*, outputs $\hat{\theta}$ satisfying $f(\hat{\theta}) \le f(\theta_*) + \epsilon$.

- Substituting $\theta_{i+1} = \theta_i - \eta f'(\theta_i)$ and letting $a_i = \theta_i - \theta_*$ gives:

$$a_{i+1}^2 = (\theta_{i+1} - \theta_*)^2 = (a_i - \eta f'(\theta_i))^2 = a_i^2 - 2\eta f'(\theta_i) a_i + (\eta f'(\theta_i))^2$$

GD analysis proof for $d = 1$

Theorem: For convex function $f : \mathbb{R} \to \mathbb{R}$ where $|f'(\theta)| \le G$ for all θ, GD run with $t \ge \frac{R^2 G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within R of θ_*, outputs $\hat{\theta}$ satisfying $f(\hat{\theta}) \le f(\theta_*) + \epsilon$.

- Substituting $\theta_{i+1} = \theta_i - \eta f'(\theta_i)$ and letting $a_i = \theta_i - \theta_*$ gives:

$$a_{i+1}^2 = (\theta_{i+1} - \theta_*)^2 = (a_i - \eta f'(\theta_i))^2 = a_i^2 - 2\eta f'(\theta_i) a_i + (\eta f'(\theta_i))^2$$

- Rearrange and use convexity to show:

$$f(\theta_i) - f(\theta_*) \le f'(\theta_i) a_i = \frac{1}{2\eta}\left(a_i^2 - a_{i+1}^2\right) + \eta (f'(\theta_i))^2/2$$

GD ANALYSIS PROOF FOR $d = 1$

Theorem: For convex function $f : \mathbb{R} \to \mathbb{R}$ where $|f'(\theta)| \leq G$ for all θ, GD run with $t \geq \frac{R^2 G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within R of θ_*, outputs $\hat{\theta}$ satisfying $f(\hat{\theta}) \leq f(\theta_*) + \epsilon$.

- Substituting $\theta_{i+1} = \theta_i - \eta f'(\theta_i)$ and letting $a_i = \theta_i - \theta_*$ gives:

$$a_{i+1}^2 = (\theta_{i+1} - \theta_*)^2 = (a_i - \eta f'(\theta_i))^2 = a_i^2 - 2\eta f'(\theta_i) a_i + (\eta f'(\theta_i))^2$$

- Rearrange and use convexity to show:

$$f(\theta_i) - f(\theta_*) \leq f'(\theta_i) a_i = \frac{1}{2\eta}\left(a_i^2 - a_{i+1}^2\right) + \eta (f'(\theta_i))^2/2$$

- Summing over i and using the fact $|f'(\theta_i)| \leq G$,

$$\frac{1}{t}\sum_{i=1}^{t}\left(f(\theta_i) - f(\theta_*)\right) \leq \left(\frac{1}{2t\eta}\sum_{i=1}^{t}(a_i^2 - a_{i+1}^2)\right) + \frac{\eta G^2}{2} \leq \frac{a_1^2}{2t\eta} + \frac{\eta G^2}{2}$$

GD ANALYSIS PROOF FOR $d = 1$

Theorem: For convex function $f : \mathbb{R} \to \mathbb{R}$ where $|f'(\theta)| \leq G$ for all θ, GD run with $t \geq \frac{R^2 G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within R of θ_*, outputs $\hat{\theta}$ satisfying $f(\hat{\theta}) \leq f(\theta_*) + \epsilon$.

- Substituting $\theta_{i+1} = \theta_i - \eta f'(\theta_i)$ and letting $a_i = \theta_i - \theta_*$ gives:

$$a_{i+1}^2 = (\theta_{i+1} - \theta_*)^2 = (a_i - \eta f'(\theta_i))^2 = a_i^2 - 2\eta f'(\theta_i) a_i + (\eta f'(\theta_i))^2$$

- Rearrange and use convexity to show:

$$f(\theta_i) - f(\theta_*) \leq f'(\theta_i) a_i = \frac{1}{2\eta}\left(a_i^2 - a_{i+1}^2\right) + \eta (f'(\theta_i))^2/2$$

- Summing over i and using the fact $|f'(\theta_i)| \leq G$,

$$\frac{1}{t}\sum_{i=1}^{t}\left(f(\theta_i) - f(\theta_*)\right) \leq \left(\frac{1}{2t\eta}\sum_{i=1}^{t}(a_i^2 - a_{i+1}^2)\right) + \frac{\eta G^2}{2} \leq \frac{a_1^2}{2t\eta} + \frac{\eta G^2}{2}$$

- Using $a_1^2 \leq R^2$ and $f(\hat{\theta}) - f(\theta^*) \leq \frac{1}{t}\sum_{i=1}^{t}\left(f(\theta_i) - f(\theta_*)\right)$

$$f(\hat{\theta}) \leq f(\theta^*) + \frac{R^2}{2t\eta} + \frac{\eta G^2}{2} \leq f(\theta^*) + \epsilon$$

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.

Example 1: Linear Regression, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood. . .)

Example 1: Linear Regression, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood...)

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

Example 1: Linear Regression, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood...)

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

Example 1: Linear Regression, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood...)

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

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

Example 1: Linear Regression, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood...)

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

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, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood...)

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

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)

OPTIMIZATION IN ML

Example 1: Linear Regression, e.g., predicting house prices based on d features (sq. footage, average price of houses in neighborhood...)

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

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

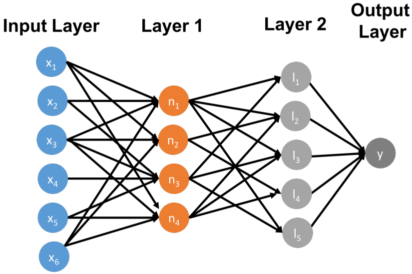
Optimization Problem: Given data points (training points) $\vec{x}_1, \ldots, \vec{x}_n$ (the rows of data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

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

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

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 $z_1, \ldots, z_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

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

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

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

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