COMPSCI 514: ALGORITHMS FOR DATA SCIENCE

Andrew McGregor
Lecture 22
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
  \[ \max_{\vec{v}: \|\vec{v}\|_2^2 = 1} |\vec{v}^T A \vec{v}| \]

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 Optimization: (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

- Continuous optimization examples
- Two-dimensional optimization examples
- Three-dimensional optimization examples

![Continuous Optimization Examples](image-url)
Given some function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, find $\vec{\theta}_*$ with:

$$f(\vec{\theta}_*) = \min_{\vec{\theta} \in \mathbb{R}^d} f(\vec{\theta})$$
Given some function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, find $\vec{\theta}^\star$ with:

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

Typically up to some small approximation factor.
Given some function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, find $\vec{\theta}_\star$ with:

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

Typically up to some small approximation factor.

Often under some constraints:

- $\|\vec{\theta}\|_2 \leq 1, \quad \|\vec{\theta}\|_1 \leq 1$.
- $A\vec{\theta} \leq \vec{b}, \quad \vec{\theta}^T A\vec{\theta} \geq 0$.
- $\sum_{i=1}^d \vec{\theta}(i) \leq c$. 
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 $M_{\vec{\theta}}(\vec{x}) \overset{\text{def}}{=} \theta(1) \cdot x(1) + \ldots + \theta(d) \cdot x(d)$. 
Example 1: Linear Regression, e.g., predicting house prices based on $d$ features (sq. footage, average price of houses in neighborhood...)

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

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

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 $X \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

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

where $\ell$ 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 $M_{\vec{\theta}}(\vec{x}) \overset{\text{def}}{=} \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

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 $X \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}^\ast$ minimizing the loss function:

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

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

- $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = (M_{\vec{\theta}}(\vec{x}_i) - y_i)^2$ (least squares regression)
- $y_i \in \{-1, 1\}$ and $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \ln (1 + \exp(-y_i M_{\vec{\theta}}(\vec{x}_i)))$ (logistic regression)
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 $M_{\vec{\theta}}(\vec{x}) \overset{\text{def}}{=} \vec{\theta}(1) \cdot \vec{x}(1) + \ldots + \vec{\theta}(d) \cdot \vec{x}(d)$.

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 $X \in \mathbb{R}^{n \times d}$) and labels $y_1, \ldots, y_n \in \mathbb{R}$, find $\vec{\theta}_*$ minimizing the loss function:

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

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

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

Model: \( M_{\vec{\theta}} : \mathbb{R}^d \rightarrow \mathbb{R} \). \( M_{\vec{\theta}}(\vec{x}) = \langle \vec{w}_{out}, \sigma(W_2 \sigma(W_1 \vec{x})) \rangle \).

Parameter Vector: \( \vec{\theta} \in \mathbb{R}^{(\# \text{ edges})} \) (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:

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

- **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 \( L_{\mathbf{X}, \mathbf{y}}(\mathbf{\theta}) \) 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.

$$L_{\vec{x}, \vec{y}}(\vec{\theta}) = \sum_{i=1}^{n} \ell(M_{\vec{\theta}}(\vec{x}_i), y_i)$$
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.
Let $\vec{e}_i \in \mathbb{R}^d$ denote the $i^{th}$ standard basis vector,

$$\vec{e}_i = [0, 0, 1, 0, 0, \ldots, 0] .$$

1 at position $i$
Let $\vec{e}_i \in \mathbb{R}^d$ denote the $i^{th}$ standard basis vector,

$$\vec{e}_i = [0, 0, 1, 0, 0, \ldots, 0]_{1 \text{ at position } i}.$$ 

**Partial Derivative:**

$$\frac{\partial f}{\partial \vec{\theta}(i)} = \lim_{\epsilon \to 0} \frac{f(\vec{\theta} + \epsilon \cdot \vec{e}_i) - f(\vec{\theta})}{\epsilon}.$$
Let $\vec{e}_i \in \mathbb{R}^d$ denote the $i^{th}$ standard basis vector,

$$\vec{e}_i = [0, 0, 1, 0, 0, \ldots, 0] \text{ with } 1 \text{ at position } i.$$

**Partial Derivative:**

$$\frac{\partial f}{\partial \theta(i)} = \lim_{\epsilon \to 0} \frac{f(\theta + \epsilon \cdot \vec{e}_i) - f(\theta)}{\epsilon}.$$

**Directional Derivative:**

$$D_{\vec{v}} f(\theta) = \lim_{\epsilon \to 0} \frac{f(\theta + \epsilon \vec{v}) - f(\theta)}{\epsilon}.$$
Gradient: Just a ‘list’ of the partial derivatives.

\[
\vec{\nabla} f(\vec{\theta}) = \begin{bmatrix}
\frac{\partial f}{\partial \theta(1)} \\
\frac{\partial f}{\partial \theta(2)} \\
\vdots \\
\frac{\partial f}{\partial \theta(d)}
\end{bmatrix}
\]
**Gradient**: Just a ‘list’ of the partial derivatives.

\[ \nabla f(\vec{\theta}) = \begin{bmatrix} \frac{\partial f}{\partial \theta(1)} \\ \frac{\partial f}{\partial \theta(2)} \\ \vdots \\ \frac{\partial f}{\partial \theta(d)} \end{bmatrix} \]

**Directional Derivative in Terms of the Gradient**:

\[ D_{\vec{v}} f(\vec{\theta}) = \langle \vec{v}, \nabla f(\vec{\theta}) \rangle. \]
Often the functions we are trying to optimize are very complex (e.g., a neural network). We will assume access to:

**Function Evaluation**: Can compute $f(\theta)$ for any $\theta$.

**Gradient Evaluation**: Can compute $\nabla f(\theta)$ for any $\theta$.
Often the functions we are trying to optimize are very complex (e.g., a neural network). We will assume access to:

**Function Evaluation:** Can compute $f(\vec{\theta})$ for any $\vec{\theta}$.

**Gradient Evaluation:** Can compute $\nabla f(\vec{\theta})$ for any $\vec{\theta}$.

In neural networks:

- Function evaluation is called a **forward pass** (propogate an input through the network).
- Gradient evaluation is called a **backward pass** (compute the gradient via chain rule, using backpropagation).
Gradient descent is a greedy iterative optimization algorithm: Starting at $\vec{\theta}^{(0)}$, in each iteration let $\vec{\theta}^{(i)} = \vec{\theta}^{(i-1)} + \eta \vec{v}$, where $\eta$ is a (small) ‘step size’ and $\vec{v}$ is a direction chosen to minimize $f(\vec{\theta}^{(i-1)} + \eta \vec{v})$. 
Gradient descent is a greedy iterative optimization algorithm: Starting at $\vec{\theta}^{(0)}$, in each iteration let $\vec{\theta}^{(i)} = \vec{\theta}^{(i-1)} + \eta \vec{v}$, where $\eta$ is a (small) ‘step size’ and $\vec{v}$ is a direction chosen to minimize $f(\vec{\theta}^{(i-1)} + \eta \vec{v})$.

$$D_{\vec{v}} f(\vec{\theta}) = \lim_{\epsilon \to 0} \frac{f(\vec{\theta} + \epsilon \vec{v}) - f(\vec{\theta})}{\epsilon}.$$
Gradient descent is a **greedy** iterative optimization algorithm: Starting at $\vec{\theta}^{(0)}$, in each iteration let $\vec{\theta}^{(i)} = \vec{\theta}^{(i-1)} + \eta \vec{v}$, where $\eta$ is a (small) ‘step size’ and $\vec{v}$ is a direction chosen to minimize $f(\vec{\theta}^{(i-1)} + \eta \vec{v})$.

$$D_{\vec{v}} f(\vec{\theta}^{(i-1)}) = \lim_{\epsilon \to 0} \frac{f(\vec{\theta}^{(i-1)} + \epsilon \vec{v}) - f(\vec{\theta}^{(i-1)})}{\epsilon}.$$
Gradient descent is a **greedy** iterative optimization algorithm: Starting at \( \theta^{(0)} \), in each iteration let \( \theta^{(i)} = \theta^{(i-1)} + \eta \vec{v} \), where \( \eta \) is a (small) ‘step size’ and \( \vec{v} \) is a direction chosen to minimize \( f(\theta^{(i-1)} + \eta \vec{v}) \).

\[
D_{\vec{v}} f(\theta^{(i-1)}) = \lim_{\epsilon \to 0} \frac{f(\theta^{(i-1)} + \epsilon \vec{v}) - f(\theta^{(i-1)})}{\epsilon}.
\]

So for small \( \eta \):

\[
f(\theta^{(i)}) - f(\theta^{(i-1)}) = f(\theta^{(i-1)} + \eta \vec{v}) - f(\theta^{(i-1)})
\]
Gradient descent is a greedy iterative optimization algorithm: Starting at \( \bar{\theta}^{(0)} \), in each iteration let \( \bar{\theta}^{(i)} = \bar{\theta}^{(i-1)} + \eta \vec{v} \), where \( \eta \) is a (small) ‘step size’ and \( \vec{v} \) is a direction chosen to minimize \( f(\bar{\theta}^{(i-1)} + \eta \vec{v}) \).

\[
D_{\vec{v}} f(\bar{\theta}^{(i-1)}) = \lim_{\epsilon \to 0} \frac{f(\bar{\theta}^{(i-1)} + \epsilon \vec{v}) - f(\bar{\theta}^{(i-1)})}{\epsilon}.
\]

So for small \( \eta \):

\[
f(\bar{\theta}^{(i)}) - f(\bar{\theta}^{(i-1)}) = f(\bar{\theta}^{(i-1)} + \eta \vec{v}) - f(\bar{\theta}^{(i-1)}) \approx \eta \cdot D_{\vec{v}} f(\bar{\theta}^{(i-1)})
\]
Gradient descent is a greedy iterative optimization algorithm: Starting at $\vec{\theta}^{(0)}$, in each iteration let $\vec{\theta}^{(i)} = \vec{\theta}^{(i-1)} + \eta \vec{v}$, where $\eta$ is a (small) ‘step size’ and $\vec{v}$ is a direction chosen to minimize $f(\vec{\theta}^{(i-1)} + \eta \vec{v})$.

$$D_{\vec{v}} f(\vec{\theta}^{(i-1)}) = \lim_{\epsilon \to 0} \frac{f(\vec{\theta}^{(i-1)} + \epsilon \vec{v}) - f(\vec{\theta}^{(i-1)})}{\epsilon}.$$  

So for small $\eta$:

$$f(\vec{\theta}^{(i)}) - f(\vec{\theta}^{(i-1)}) = f(\vec{\theta}^{(i-1)} + \eta \vec{v}) - f(\vec{\theta}^{(i-1)}) \approx \eta \cdot D_{\vec{v}} f(\vec{\theta}^{(i-1)})$$

$$= \eta \cdot \langle \vec{v}, \vec{\nabla} f(\vec{\theta}^{(i-1)}) \rangle.$$
Gradient descent is a greedy iterative optimization algorithm: Starting at $\mathbf{\theta}^{(0)}$, in each iteration let $\mathbf{\theta}^{(i)} = \mathbf{\theta}^{(i-1)} + \eta \mathbf{v}$, where $\eta$ is a (small) ‘step size’ and $\mathbf{v}$ is a direction chosen to minimize $f(\mathbf{\theta}^{(i-1)} + \eta \mathbf{v})$.

$$D_\mathbf{v} f(\mathbf{\theta}^{(i-1)}) = \lim_{\epsilon \to 0} \frac{f(\mathbf{\theta}^{(i-1)} + \epsilon \mathbf{v}) - f(\mathbf{\theta}^{(i-1)})}{\epsilon}.$$ 

So for small $\eta$:

$$f(\mathbf{\theta}^{(i)}) - f(\mathbf{\theta}^{(i-1)}) = f(\mathbf{\theta}^{(i-1)} + \eta \mathbf{v}) - f(\mathbf{\theta}^{(i-1)}) \approx \eta \cdot D_\mathbf{v} f(\mathbf{\theta}^{(i-1)})$$

$$= \eta \cdot \langle \mathbf{v}, \nabla f(\mathbf{\theta}^{(i-1)}) \rangle.$$

We want to choose $\mathbf{v}$ minimizing $\langle \mathbf{v}, \nabla f(\mathbf{\theta}^{(i-1)}) \rangle$ – i.e., pointing in the direction of $\nabla f(\mathbf{\theta}^{(i-1)})$ but with the opposite sign.
Gradient Descent

- Choose some initialization $\vec{\theta}(0)$.
- For $i = 1, \ldots, t$
  - $\vec{\theta}(i) = \vec{\theta}(i-1) - \eta \nabla f(\vec{\theta}(i-1))$
- Return $\vec{\theta}(t)$, as an approximate minimizer of $f(\vec{\theta})$.

Step size $\eta$ is chosen ahead of time or adapted during the algorithm (details to come.)
Gradient Descent

• Choose some initialization $\vec{\theta}^{(0)}$.
• For $i = 1, \ldots, t$
  • $\vec{\theta}^{(i)} = \vec{\theta}^{(i-1)} - \eta \nabla f(\vec{\theta}^{(i-1)})$
• Return $\vec{\theta}^{(t)}$, as an approximate minimizer of $f(\vec{\theta})$.

Step size $\eta$ is chosen ahead of time or adapted during the algorithm (details to come.)

• For now assume $\eta$ stays the same in each iteration.
Gradient Descent Update: \( \hat{\theta}_{i+1} = \hat{\theta}_i - \eta \nabla f(\hat{\theta}_i) \)