COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2024.

Lecture 22

Logistics

- Problem Set 5 is posted. It can be turned in up to 12/12 (next Thursday) at 11:59pm with no penalty. No extensions will be granted beyond this. The challenge problem is optional extra credit.
- The final will be on 12/18 in Totman Gym, 10:30am-12:30pm.
- Additional final review office hours will be posted soon.
- See website/Canvas for final prep material.

Summary

Last Class:

- Finish up the power method.
- Krylov subspace methods.
- Connection between random walks and power method.
- Very brief intro to continuous optimization.

This Class:

- Multivariable calculus review
- Introduction to gradient descent. Motivation as a greedy algorithm.
- Convex functions
- Analysis of gradient descent for Lipschitz, convex functions?

Mathematical Setup

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

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

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.

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:

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

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)

$\underline{\underline{\theta}}$

- 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: Linear Regression

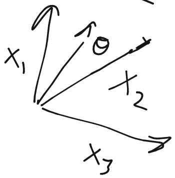
Optimization in ML

Example: Linear Regression

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

$\langle x_1, \theta \rangle$

$\langle x_2, \theta \rangle$



Optimization in ML

Example: Linear Regression

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

Optimization in ML

Example: 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 in ML

Example: 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:

distance function

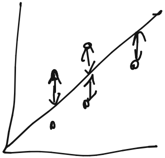
prediction

$\mathbb{R}^d \to \mathbb{R}$

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

label

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



Optimization in ML

Example: 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.

- $\ell(M_{\vec{\theta}}(\vec{x}_i), y_i) = \left(M_{\vec{\theta}}(\vec{x}_i) - y_i\right)^2$ (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: 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:

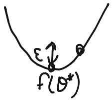
$f(\theta^*) = \min f(\theta)$

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

$f(\theta) = \min f(\theta) + \varepsilon$

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

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

- Solving the final optimization problem has many different names: likelihood maximization, empirical risk minimization, minimizing training loss, 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.
- Continuous optimization is also very common in unsupervised learning. (PCA, spectral clustering, etc.)

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.

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

Stochastic gradient descent

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?

ADAM (variant of gradient descent)

hill climbing

branch + cut

cutting plane

nelder-mead (derivative free)

0-order

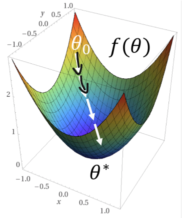
Ada-factor

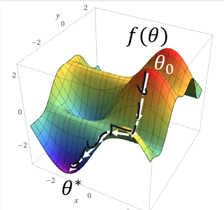
grid search

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.





Multivariate Calculus Review

Let $\vec{e}_i \in \mathbb{R}^d$ denote the i^{th} standard basis vector, $\vec{e}_i = \underbrace{[0,0,1,0,0,\ldots,0]}_{\text{1 at position } i}$.

Let $\vec{e}_i \in \mathbb{R}^d$ denote the i^{th} standard basis vector,
$\vec{e}_i = \underbrace{[0, 0, 1, 0, 0, \ldots, 0]}_{\text{1 at position } i}$.

Partial Derivative: $f: \mathbb{R}^d \to \mathbb{R}$

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

$$\begin{bmatrix} \theta(1) \\ \theta(2) \\ \vdots \\ \theta(i)+\epsilon \\ \theta(i+1) \\ \vdots \\ \vdots \end{bmatrix}$$

Let $\vec{e}_i \in \mathbb{R}^d$ denote the i^{th} standard basis vector, $\vec{e}_i = \underbrace{[0, 0, 1, 0, 0, \ldots, 0]}_{\text{1 at position } i}$.

Partial Derivative:

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

Directional Derivative:

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

Multivariate Calculus Review

$f : \mathbb{R}^d \to \mathbb{R}$ $\quad \nabla f(\theta) \in \mathbb{R}^d$

Gradient: Just a 'list' of the partial derivatives.

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

Gradient: Just a 'list' of the partial derivatives.

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

Directional Derivative in Terms of the Gradient:

$$D_{\vec{v}} f(\vec{\theta}) = \langle \vec{v}, \vec{\nabla} f(\vec{\theta}) \rangle.$$

$$V(1) \cdot \frac{\partial f}{\partial \theta(1)} + V(2) \cdot \frac{\partial f}{\partial \theta(2)} \cdots + V(d) \cdot \frac{\partial f}{\partial \theta(d)}$$

Function Access

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 $\vec{\nabla} f(\vec{\theta})$ for any $\vec{\theta}$.

Function Access

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 $\vec{\nabla} f(\vec{\theta})$ 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 Greedy Approach

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 η is a (small) 'step size' and $\vec{v}$ is a direction chosen to minimize

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

Gradient Descent Greedy Approach

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 η 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 Greedy Approach

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 η 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 Greedy Approach

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 η 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 η:

θ^{i}

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

Gradient Descent Greedy Approach

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 η 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 η:

$$\underline{f(\vec{\theta}^{(i)}) - f(\vec{\theta}^{(i-1)})} = f(\vec{\theta}^{(i-1)} + \eta\vec{v}) - f(\vec{\theta}^{(i-1)}) \approx \eta \cdot \underbrace{D_{\vec{v}} f(\vec{\theta}^{(i-1)})}_{\text{pick } v \text{ to minimize this}}$$

Gradient Descent Greedy Approach

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 η 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 η:

$$\begin{aligned} f(\vec{\theta}^{(i)}) - f(\vec{\theta}^{(i-1)}) = f(\vec{\theta}^{(i-1)} + \eta\vec{v}) - f(\vec{\theta}^{(i-1)}) &\approx \eta \cdot D_{\vec{v}} f(\vec{\theta}^{(i-1)}) \\ &= \eta \cdot \underbrace{\langle \vec{v}, \vec{\nabla} f(\vec{\theta}^{(i-1)}) \rangle}_{\text{minimize}}. \end{aligned}$$

$V = -\nabla f(\theta^{i-1})$

Gradient Descent Greedy Approach

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 η 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 η:

$$\begin{aligned} f(\vec{\theta}^{(i)}) - f(\vec{\theta}^{(i-1)}) = f(\vec{\theta}^{(i-1)} + \eta\vec{v}) - f(\vec{\theta}^{(i-1)}) &\approx \eta \cdot D_{\vec{v}} f(\vec{\theta}^{(i-1)}) \\ &= \eta \cdot \langle \vec{v}, \vec{\nabla} f(\vec{\theta}^{(i-1)}) \rangle. \end{aligned}$$

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

compute in "closed form"

$\nabla f(\theta)$

v

$$V = -\nabla f(\theta^{i-1})$$

Gradient Descent Psuedocode

Gradient Descent

$\sim \mathcal{N}(0, I_{d\times d})$

- 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 η is chosen ahead of time or adapted during the algorithm (details to come).

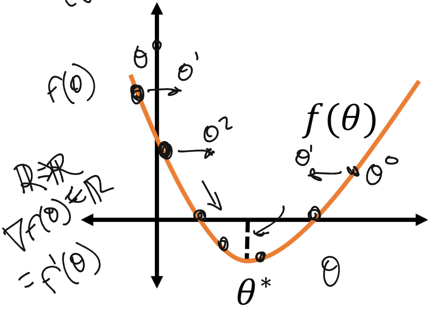
- For now assume η stays the same in each iteration.

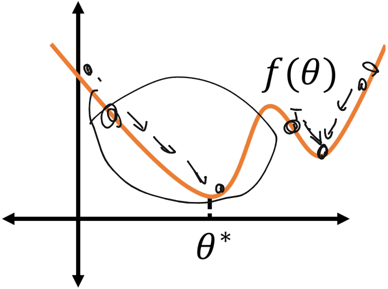
When Does Gradient Descent Work?

convex function

$\theta \in \mathbb{R} \quad \nabla f(\theta) \in \mathbb{R}$

non-convex





Gradient Descent Update: $\vec{\theta}_{i+1} = \vec{\theta}_i - \eta \nabla f(\vec{\theta}_i)$

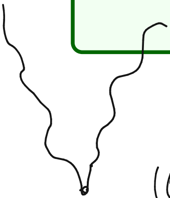
Definition – Convex Function: A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if, for any $\vec{\theta}_1, \vec{\theta}_2 \in \mathbb{R}^d$ and $\lambda \in [0, 1]$:

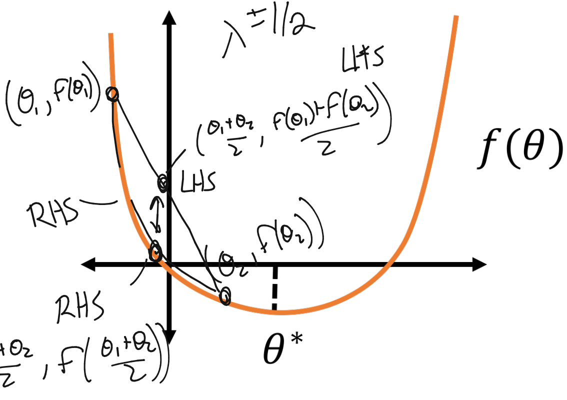
$$(1-\lambda)\cdot f(\vec{\theta}_1) + \lambda \cdot f(\vec{\theta}_2) \geq f\left((1-\lambda)\cdot\vec{\theta}_1 + \lambda\cdot\vec{\theta}_2\right)$$

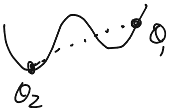
$f(x) = x$

$f(x) = x^2$

$f(x) = |x|$





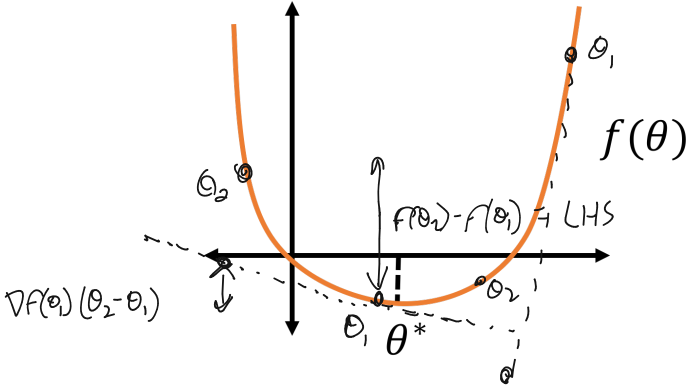


Convexity

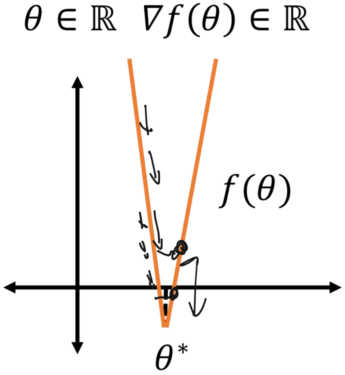
Corollary – Convex Function: A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if, for any $\vec{\theta}_1, \vec{\theta}_2 \in \mathbb{R}^d$ ~~and $\lambda \in [0,1]$~~:

$$f(\vec{\theta}_2) - f(\vec{\theta}_1) \geq \vec{\nabla} f(\vec{\theta}_1)^T \left(\vec{\theta}_2 - \vec{\theta}_1\right)$$

$\Leftrightarrow f''(\theta) \geq 0$



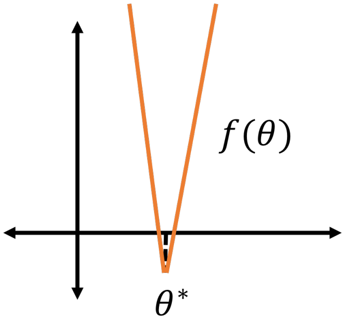
Lipschitz Functions



Gradient Descent Update:
$\vec{\theta}_{i+1} = \vec{\theta}_i - \eta \nabla f(\vec{\theta}_i)$

Lipschitz Functions

$\theta \in \mathbb{R} \quad \nabla f(\theta) \in \mathbb{R}$



$f(x) = x^2$

$f'(x) = x$

Gradient Descent Update:

$\vec{\theta}_{i+1} = \vec{\theta}_i - \eta \nabla f(\vec{\theta}_i)$

Need to assume that the function is Lipschitz (size of gradient is bounded): There is some G s.t.:

$$\forall \vec{\theta}: \quad \|\vec{\nabla} f(\vec{\theta})\|_2 \leq G \Leftrightarrow \forall \vec{\theta}_1, \vec{\theta}_2: \quad |f(\vec{\theta}_1) - f(\vec{\theta}_2)| \leq G \cdot \|\vec{\theta}_1 - \vec{\theta}_2\|_2$$

Well-Behaved Functions

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

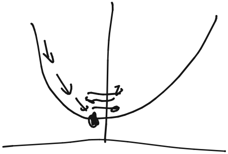
$$(1-\lambda) \cdot f(\vec{\theta}_1) + \lambda \cdot f(\vec{\theta}_2) \geq f\left((1-\lambda) \cdot \vec{\theta}_1 + \lambda \cdot \vec{\theta}_2\right)$$

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

$$f(\vec{\theta}_2) - f(\vec{\theta}_1) \geq \vec{\nabla} f(\vec{\theta}_1)^T \left(\vec{\theta}_2 - \vec{\theta}_1\right)$$

Definition – Lipschitz Function: A function $f : \mathbb{R}^d \to \mathbb{R}$ is G-Lipschitz if $\|\vec{\nabla} f(\vec{\theta})\|_2 \leq G$ for all $\vec{\theta}$.

GD Analysis – Convex Functions



Assume that:

- f is convex.
- f is G-Lipschitz.
- $\|\vec{\theta}_1 - \vec{\theta}_*\|_2 \leq R$ where $\vec{\theta}_1$ is the initialization point.

Gradient Descent

bigger steps if further away

- Choose some initialization $\vec{\theta}_1$ and set $\eta = \frac{R}{G\sqrt{t}}$.

smaller steps if steeper → more iterations

- For $i = 1, \ldots, t-1$
  - $\vec{\theta}_{i+1} = \vec{\theta}_i - \eta \vec{\nabla} f(\vec{\theta}_i)$
- Return $\hat{\theta} = \arg\min_{\vec{\theta}_1, \ldots, \vec{\theta}_t} f(\vec{\theta}_i)$.

$f(\theta_t)$

GD Analysis Proof

Theorem – GD on Convex Lipschitz Functions: For convex G-Lipschitz function f, GD run with $t \geq \frac{R^2G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within radius R of $\vec{\theta}_*$, outputs $\hat{\theta}$ satisfying:

$$f(\hat{\theta}) \leq f(\vec{\theta}_*) + \epsilon.$$

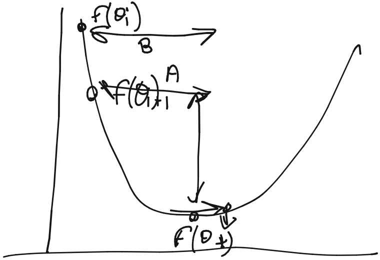
GD Analysis Proof

Theorem – GD on Convex Lipschitz Functions: For convex G-Lipschitz function f, GD run with $t \geq \frac{R^2 G^2}{\epsilon^2}$ iterations, $\eta = \frac{R}{G\sqrt{t}}$, and starting point within radius R of $\vec{\theta}_*$, outputs $\hat{\theta}$ satisfying:

$$f(\hat{\theta}) \leq f(\vec{\theta}_*) + \epsilon.$$

Step 1: For all i, $f(\vec{\theta}_i) - f(\vec{\theta}_*) \leq \frac{\|\vec{\theta}_i - \vec{\theta}_*\|_2^2 - \|\vec{\theta}_{i+1} - \vec{\theta}_*\|_2^2}{2\eta} + \frac{\eta G^2}{2}$. Visually:

small



RHS: $B^2 - A^2$