Optimization for Machine Learning: Gradient Descent and Stochastic Gradient Descent

Mathematical Foundations of Data Science

Difan Zou

Nov. 11, 2025

Recap: Optimization Basics (Previous Lecture)

Reference: Lecture_note_optimization_basics.pdf

Key concepts covered:

- **Optimization problem formulation:** $\min_\theta f(\theta)$ subject to constraints
- **Convexity:** A function is convex if $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$
- **Optimality conditions:**
    - Unconstrained: $\nabla f(\theta^*) = 0$
    - Constrained: KKT conditions
- **Smoothness (Lipschitz gradient):** $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$
- **Strong convexity:** $f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|y - x\|^2$

Today's lecture: We apply these concepts to Machine Learning optimization

- Empirical Risk Minimization (ERM)
- Gradient Descent and its convergence analysis
- Stochastic Gradient Descent for large-scale problems

Outline

Empirical Risk Minimization (ERM)

Goal: Given training data points, find parameter θ that minimizes loss

Optimization Problem:

$$\theta_S^* := \arg\min_{\theta \in \Theta} L_S(h_\theta)$$

Model Training Objective:

$$L_S(h_\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

where:

- n: training sample size
- ℓ: loss function
- f_θ: model function with parameter θ
- (x_i, y_i): training data (feature, target)

When Do Exact Solutions Exist?

Example 1: Linear Regression (Ordinary Least Squares)

- **Objective:** $\min_\theta \frac{1}{2n} \sum_{i=1}^{n} (\theta^\top x_i - y_i)^2$
- **Closed-form solution:** $\theta^* = (X^\top X)^{-1} X^\top y$

Example 2: Ridge Regression

- **Objective:** $\min_\theta \|X\theta - y\|^2 + \lambda \|\theta\|^2$
- **Closed-form solution:** $\theta^* = (X^\top X + \lambda I)^{-1} X^\top y$

Example 3: Linear Programming

- **Form:** $\min_x c^\top x$ subject to $Ax \leq b, x \geq 0$
- **Solvers:** Simplex method, interior point methods (polynomial time)

Linear Regression: Closed-Form Solution Derivation (Part 1)

Objective: $\min_\theta L(\theta) = \frac{1}{2n} \sum_{i=1}^n (\theta^\top x_i - y_i)^2$

Matrix notation: Let $X \in \mathbb{R}^{n \times d}$ be the data matrix (rows are samples), $y \in \mathbb{R}^n$ be the targets

$$L(\theta) = \frac{1}{2n} \|X\theta - y\|^2$$

Expand the squared norm:

$$\begin{aligned} L(\theta) &= \frac{1}{2n}(X\theta - y)^\top (X\theta - y) \\ &= \frac{1}{2n}(\theta^\top X^\top X\theta - 2\theta^\top X^\top y + y^\top y) \end{aligned}$$

Goal: Find θ^* such that $\nabla_\theta L(\theta^*) = 0$

Linear Regression: Closed-Form Solution Derivation (Part 2)

Compute the gradient: Starting from

$$L(\theta) = \frac{1}{2n}(\theta^\top X^\top X\theta - 2\theta^\top X^\top y + y^\top y)$$

Gradient rules:

- $\nabla_\theta(\theta^\top A\theta) = 2A\theta$ for symmetric A
- $\nabla_\theta(\theta^\top b) = b$
- $\nabla_\theta(c) = 0$ for constant c

Apply gradient:

$$\begin{aligned}\nabla_\theta L(\theta) &= \frac{1}{2n}(2X^\top X\theta - 2X^\top y) \\ &= \frac{1}{n}(X^\top X\theta - X^\top y)\end{aligned}$$

Linear Regression: Closed-Form Solution Derivation (Part 3)

Set gradient to zero:

$$\nabla_\theta L(\theta) = \frac{1}{n}(X^\top X\theta - X^\top y) = 0$$

Solve for θ:

$$\begin{aligned} X^\top X\theta - X^\top y &= 0 \\ X^\top X\theta &= X^\top y \end{aligned}$$

Multiply both sides by $(X^\top X)^{-1}$:

$$\theta^* = (X^\top X)^{-1} X^\top y$$

Note: This requires $X^\top X$ to be invertible (full column rank)

- If $d > n$ (more features than samples): not invertible
- If features are linearly dependent: not invertible
- Solution: regularization (Ridge regression)

Ridge Regression: Closed-Form Solution Derivation (Part 1)

Objective: $\min_\theta L(\theta) = \frac{1}{2n}\|X\theta - y\|^2 + \frac{\lambda}{2}\|\theta\|^2$
Expand:

$$\begin{aligned} L(\theta) &= \frac{1}{2n}(X\theta - y)^\top (X\theta - y) + \frac{\lambda}{2}\theta^\top\theta \\ &= \frac{1}{2n}(\theta^\top X^\top X\theta - 2\theta^\top X^\top y + y^\top y) + \frac{\lambda}{2}\theta^\top\theta \end{aligned}$$

Compute gradient:

$$\begin{aligned} \nabla_\theta L(\theta) &= \frac{1}{n}(X^\top X\theta - X^\top y) + \lambda\theta \\ &= \frac{1}{n}X^\top X\theta + \lambda\theta - \frac{1}{n}X^\top y \end{aligned}$$

Ridge Regression: Closed-Form Solution Derivation (Part 2)

Set gradient to zero:

$$\frac{1}{n}X^\top X\theta + \lambda\theta - \frac{1}{n}X^\top y = 0$$

Rearrange:

$$\begin{aligned} \frac{1}{n}X^\top X\theta + \lambda\theta &= \frac{1}{n}X^\top y \\ \frac{1}{n}(X^\top X + n\lambda I)\theta &= \frac{1}{n}X^\top y \\ (X^\top X + n\lambda I)\theta &= X^\top y \end{aligned}$$

Solve for θ:

$$\theta^* = (X^\top X + n\lambda I)^{-1}X^\top y$$

Key property: $X^\top X + n\lambda I$ is always invertible for $\lambda > 0$

- Works even when $d > n$
- Regularization ensures numerical stability

Why We Still Need Iterative Methods: Computational Complexity

Linear Regression Closed-Form:

- Computing $(X^\top X)^{-1}$: $O(d^3)$ time for d-dimensional features
- Matrix-vector products: $O(nd^2)$ time
- **Total:** $O(nd^2 + d^3)$

Problem: For large d (e.g., $d = 10^6$ in modern ML):

- $d^3 \approx 10^{18}$ operations $\rightarrow$ computationally infeasible!

Gradient Descent Alternative:

- Per iteration: $O(nd)$ to compute gradient
- K iterations: $O(Knd)$ total
- If $K \ll d$, much faster than $O(d^3)$

Example: $K = 100, d = 10^6, n = 10^4$

- GD: $\approx 10^{12}$ operations
- Closed-form: $\approx 10^{18}$ operations (million times slower!)

Why We Still Need Iterative Methods: Memory Constraints

Closed-form solution requires:

- Storing $X^\top X$: $d \times d$ matrix $\to O(d^2)$ memory
- For $d = 10^6$: need $\sim$8TB of memory (assuming 8 bytes per float)

Iterative methods require:

- Store parameters θ: $O(d)$ memory
- Store one mini-batch: $O(Bd)$ memory ($B \ll n$)
- **Huge difference:** GB vs TB

Why We Still Need Iterative Methods: No Closed-Form Solution

Many ML models have no closed-form solution:

Logistic Regression:

- Objective: $\min_\theta \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-y_i\theta^\top x_i))$
- Issue: No closed-form due to nonlinear log/exp
- Solution: Must use iterative methods (GD, Newton's method)

Neural Networks:

- Objective: $\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$
- f_θ: multi-layer composition with nonlinear activations
- Issue: Highly non-convex, no closed-form solution
- Solution: SGD with backpropagation is the standard

Comparison of Optimization Methods

Method	**Time**	**Memory**	**Streaming**	**General**
Closed-form (OLS)	$O(nd^2 + d^3)$	$O(d^2)$	$\times$	Linear only
Gradient Descent	$O(Knd)$	$O(d)$	$\times$	Most models
Linear Programming	Polynomial	Varies	$\times$	Linear only

K: number of iterations

Note: Other iterative methods (e.g., stochastic methods) will be discussed later

The Reality of Modern ERM Problems

High-Dimensional Parameter Spaces:

- Modern neural networks: millions to billions of parameters
  - ResNet-50: $\sim$25 million parameters
  - GPT-3: 175 billion parameters
  - BERT-Large: 340 million parameters
- Image classification: input dimension $d = 224 \times 224 \times 3 \approx 150{,}000$
- No hope for closed-form solutions or traditional methods

Extremely Complicated Objectives

Consider a deep neural network loss:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i)$$

where

$$f_\theta(x) = W^{(L)}\sigma(W^{(L-1)}\sigma(\dots\sigma(W^{(1)}x + b^{(1)})\dots) + b^{(L)})$$

Properties:

- Highly non-convex: countless local minima, saddle points, plateaus
- No analytical form: composition of many nonlinear functions
- No structure amenable to traditional solvers (LP, QP, SDP)
- Black-box nature: can only evaluate function and gradient

Numerical Methods: The ONLY Practical Choice

For modern empirical risk minimization:

- × No closed-form solutions
- × No polynomial-time exact algorithms known
- × Traditional optimization (LP, convex opt.) doesn't apply
- ✓ **Gradient-based methods** are the only scalable approach
- ✓ Can leverage **automatic differentiation** (backpropagation)
- ✓ Works despite non-convexity (empirically successful!)

Bottom Line

For deep learning and large-scale ML, gradient descent and its variants (especially SGD) are not just convenient—they are the **only viable optimization approach** we currently have.

Algorithm Basics

Core Idea: Starting from a point, move in the negative gradient direction

Update Rule:

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \cdot \nabla_\theta L(\theta^{(t)})$$

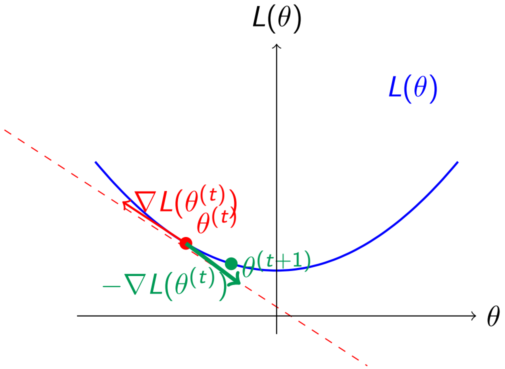
where:

- η_t: step size (learning rate)
- $\nabla_\theta L(\theta^{(t)})$: gradient at iteration t

Intuition:

- Linear approximation: $L(\theta + \Delta) \approx L(\theta) + \nabla_\theta L(\theta) \cdot \Delta$
- Best descent direction is negative gradient: $-\nabla_\theta L(\theta)$

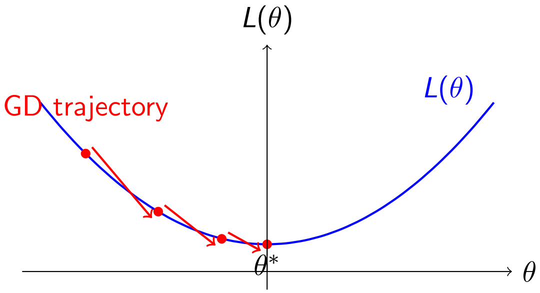
Gradient Descent: Geometric Intuition



Key insight: Negative gradient points in the direction of steepest descent

Outcome of Gradient Descent

Outcome: With proper stepsize, GD finds a point with zero gradient



Only if $\nabla_\theta L(\theta)$ approaches zero, the objective can stop decreasing

Smoothness Assumption

Definition: Function is L-smooth if $\lambda_{\max}(\nabla^2 L(\theta)) \leq L$ for all θ

Convergence with Smoothness:
By smoothness:

$$L(\theta + \Delta) \leq L(\theta) + \nabla L(\theta) \cdot \Delta + \frac{L}{2}\|\Delta\|^2$$

Setting $\Delta = -\eta \nabla L(\theta)$:

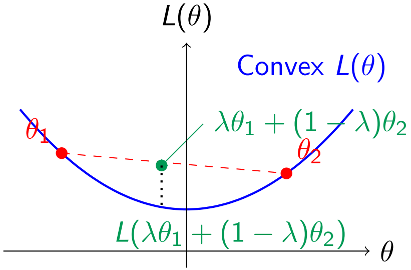
$$\begin{aligned} L(\theta^{(t+1)}) &\leq L(\theta^{(t)}) - \eta\|\nabla L(\theta^{(t)})\|^2 + \frac{L\eta^2}{2}\|\nabla L(\theta^{(t)})\|^2 \\ &= L(\theta^{(t)}) - (\eta - L\eta^2/2)\|\nabla L(\theta^{(t)})\|^2 \end{aligned}$$

Stepsize requirement: $\eta \leq 2/L$ ensures descent, then Objective keeps decreasing until zero gradient.

Convex Functions

Definition: For any θ_1, θ_2 and $\lambda \in [0, 1]$:

$$L(\lambda\theta_1 + (1-\lambda)\theta_2) \leq \lambda L(\theta_1) + (1-\lambda)L(\theta_2)$$



$L(\theta)$
Convex $L(\theta)$
θ_1
$\lambda\theta_1 + (1-\lambda)\theta_2$
θ_2
$L(\lambda\theta_1 + (1-\lambda)\theta_2)$
θ

Key Property: Zero gradient $\Rightarrow$ global minimum in convex case

Convex Functions: Equivalent Definitions

There are multiple equivalent definitions for convex functions:

1. **Monotone gradient:**
$$\langle \nabla L(\theta_1) - \nabla L(\theta_2), \theta_1 - \theta_2 \rangle \geq 0$$
2. **First-order condition:**
$$L(\theta_1) - L(\theta_2) \geq \langle \nabla L(\theta_2), \theta_1 - \theta_2 \rangle$$
3. **Hessian condition:**
$$\nabla^2 L(\theta) \succeq 0 \text{ (positive semi-definite)}$$

If θ^* satisfies $\nabla L(\theta^*) = 0$, then $L(\theta) - L(\theta^*) \geq 0$ for any $\theta \Rightarrow L(\theta^*)$ is the global minimum

Examples of Convex Loss Functions

Example 1: Linear Regression

- Loss: $\ell(f(x), y) = \frac{1}{2}(\theta^\top x - y)^2$
- Gradient: $\nabla_\theta \ell = (\theta^\top x - y)x$
- Empirical risk: $L_S(\theta) = \frac{1}{2n}\sum_{i=1}^n (\theta^\top x_i - y_i)^2$
- Gradient: $\nabla_\theta L_S = \frac{1}{n}\sum_{i=1}^n (\theta^\top x_i - y_i)x_i$
- **Property:** Convex, can be strongly convex with n data points

Example 2: Logistic Regression

- Loss: $\ell(f(x), y) = \log(1 + \exp(-y \cdot \theta^\top x))$
- Gradient: $\nabla_\theta \ell = -y \cdot x/(1 + \exp(y \cdot \theta^\top x))$
- **Property:** Convex

Convergence Rate: Smooth Objective

Theorem

Assume the objective function is L-smooth. Let $\eta = 1/L$, we have

$$\|\nabla L(\theta^{(t)})\|^2 = O(1/t)$$

Proof Sketch:

1. From smoothness with $\eta = 1/L$:
$$L(\theta^{(t+1)}) \leq L(\theta^{(t)}) - \frac{1}{2L}\|\nabla L(\theta^{(t)})\|^2$$
2. Sum over iterations:
$$\sum_{k=0}^{t} \|\nabla L(\theta^{(k)})\|^2 \leq 2L[L(\theta^{(0)}) - L(\theta^{(t+1)})]$$
3. Average $\leq O(1/t) \Rightarrow$ best $\leq O(1/t)$

Convergence Rate: Smooth + Convex

Theorem

Assume the objective function is L-smooth and convex. Let $\eta = 1/L$, we have

$$L(\theta^{(t)}) - L(\theta^*) = O(1/t)$$

Key Steps:

1. By convexity: $L(\theta^{(t+1)}) - L(\theta^*) \leq \langle \nabla L(\theta^{(t)}), \theta^{(t)} - \theta^* \rangle - \frac{\eta}{2}\|\nabla L(\theta^{(t)})\|^2$
2. Using update rule:
   $\|\theta^{(t+1)} - \theta^*\|^2 = \|\theta^{(t)} - \theta^*\|^2 - 2\eta\langle \nabla L(\theta^{(t)}), \theta^{(t)} - \theta^* \rangle + \eta^2\|\nabla L(\theta^{(t)})\|^2$
3. Combine: $L(\theta^{(t+1)}) - L(\theta^*) \leq \frac{1}{2\eta}[\|\theta^{(t)} - \theta^*\|^2 - \|\theta^{(t+1)} - \theta^*\|^2]$
4. Sum over iterations: telescoping sum gives $O(1/t)$

Strongly Convex Functions

Definition: Function is μ-strongly convex if:

- $\langle \nabla L(\theta_1) - \nabla L(\theta_2), \theta_1 - \theta_2 \rangle \geq \mu \|\theta_1 - \theta_2\|^2$
- Hessian: $\nabla^2 L(\theta) \succeq \mu I$

Theorem

Assume L-smooth and μ-strongly convex. Let $\eta = 1/L$ and $\kappa = L/\mu$, we have

$$L(\theta^{(t)}) - L(\theta^*), \|\theta^{(t)} - \theta^*\| = O(\exp(-t/\kappa))$$

- Exponential convergence (much faster than convex case!)
- κ is the condition number

Summary of Convergence Rates

Function Property	Guarantee	Rate
L-smooth	$\|\nabla L(\theta^{(t)})\|^2 \leq \epsilon$	$O(1/t)$
L-smooth + convex	$L(\theta^{(t)}) - L(\theta^*) \leq \epsilon$	$O(1/t)$
L-smooth + μ-strongly convex	$L(\theta^{(t)}) - L(\theta^*) \leq \epsilon$	$O(\exp(-t/\kappa))$

- Stronger assumptions $\Rightarrow$ faster convergence
- Strong convexity gives exponential (linear in log scale) convergence
- Condition number $\kappa = L/\mu$ determines convergence speed

Experimental Verification: Convergence Rates

Setup: Run gradient descent on different function types and observe convergence

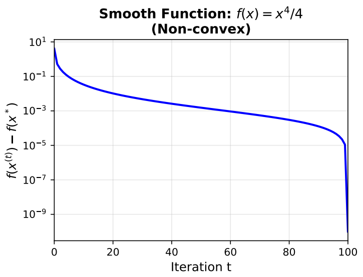
Functions tested:

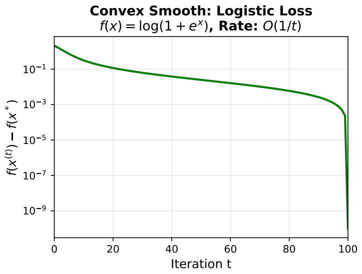
1. **Smooth (non-convex):** $f(x) = x^4/4$
2. **Convex smooth:** $f(x) = \log(1 + e^x)$ (logistic loss)
3. **Strongly convex:** $f(x) = x^2$

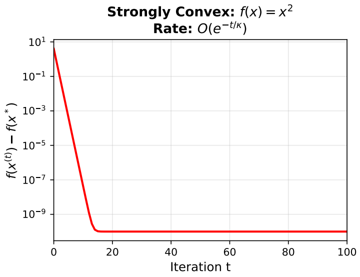
Experimental protocol:

- Initial point: $x^{(0)} = 2.0$
- Run GD for 100 iterations with appropriate learning rates
- Track $f(x^{(t)}) - f(x^*)$ (optimality gap)
- Plot on log scale to see convergence rate

Convergence Plots: Different Function Classes







Observations:

- Strongly convex: exponential decay (straight line on log plot)
- Convex smooth: $O(1/t)$ decay (slower, sublinear)
- Smooth (non-convex): Eventually reaches local minimum

Linear Regression: Detailed Gradient Derivation

Goal: Compute $\nabla_\theta \ell(f(x), y)$ where $\ell(f(x), y) = \frac{1}{2}(\theta^\top x - y)^2$
Step 1: Apply chain rule. Let $u = \theta^\top x - y$, then:

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial u} \cdot \frac{\partial u}{\partial \theta}$$

Step 2: Compute $\frac{\partial \ell}{\partial u}$:

$$\ell = \frac{1}{2}u^2 \quad \Rightarrow \quad \frac{\partial \ell}{\partial u} = u = \theta^\top x - y$$

Step 3: Compute $\frac{\partial u}{\partial \theta}$ where $u = \theta^\top x - y$:

$$\frac{\partial u}{\partial \theta} = \frac{\partial(\theta^\top x)}{\partial \theta} = x$$

Step 4: Combine using chain rule:

$$\nabla_\theta \ell = (\theta^\top x - y) \cdot x$$

Logistic Regression: Detailed Gradient Derivation

Goal: Compute $\nabla_\theta \ell(f(x), y)$ where $\ell = \log(1 + \exp(-y \cdot \theta^\top x))$
Step 1: Apply chain rule. Let $u = -y \cdot \theta^\top x$, then:

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial}{\partial \theta} \log(1 + e^u) = \frac{1}{1 + e^u} \cdot \frac{\partial}{\partial \theta}(1 + e^u)$$

Step 2: Compute $\frac{\partial}{\partial \theta}(1 + e^u)$:

$$\frac{\partial}{\partial \theta}(1 + e^u) = e^u \cdot \frac{\partial u}{\partial \theta} = e^{-y \cdot \theta^\top x} \cdot (-y \cdot x)$$

Step 3: Combine:

$$\begin{aligned}\nabla_\theta \ell &= \frac{e^{-y \cdot \theta^\top x}}{1 + e^{-y \cdot \theta^\top x}} \cdot (-y \cdot x) \\ &= \frac{-y \cdot x}{1 + e^{y \cdot \theta^\top x}}\end{aligned}$$

Logistic Regression: Alternative Form of Gradient

Rewriting the gradient: Starting from

$$\nabla_\theta \ell = \frac{-y \cdot x}{1 + e^{y \cdot \theta^\top x}}$$

Alternative formulation: Multiply numerator and denominator by $e^{-y \cdot \theta^\top x}$:

$$\begin{aligned}\nabla_\theta \ell &= \frac{-y \cdot x \cdot e^{-y \cdot \theta^\top x}}{e^{-y \cdot \theta^\top x}(1 + e^{y \cdot \theta^\top x})} \\ &= \frac{-y \cdot x \cdot e^{-y \cdot \theta^\top x}}{e^{-y \cdot \theta^\top x} + 1}\end{aligned}$$

Sigmoid interpretation: Define $\sigma(z) = \frac{1}{1+e^{-z}}$, then:

$$\nabla_\theta \ell = -y \cdot x \cdot (1 - \sigma(y \cdot \theta^\top x))$$

Intuition: Gradient proportional to prediction error weighted by input features

Proving Convexity: Linear Regression Loss

Loss function: $\ell(\theta) = \frac{1}{2}(\theta^\top x - y)^2$
Goal: Prove ℓ is convex by showing Hessian $\nabla^2 \ell(\theta) \succeq 0$
Step 1: Compute gradient (already done):

$$\nabla_\theta \ell = (\theta^\top x - y) \cdot x$$

Step 2: Compute Hessian (second derivative):

$$\begin{aligned}\nabla_\theta^2 \ell &= \frac{\partial}{\partial \theta}[(\theta^\top x - y) \cdot x] \\ &= \frac{\partial}{\partial \theta}[x \cdot \theta^\top x - y \cdot x] \\ &= x \cdot x^\top\end{aligned}$$

Step 3: Show $\nabla^2 \ell = xx^\top$ is positive semi-definite:

- For any vector v: $v^\top (xx^\top) v = (x^\top v)^2 \geq 0$ ✓
- Therefore ℓ is convex!

Proving Convexity: Logistic Regression Loss

We can similarly calcualte the Hessian:

$$\nabla^2_\theta \ell = \underbrace{\sigma(y \cdot \theta^\top x)(1 - \sigma(y \cdot \theta^\top x))}_{\alpha(\theta) > 0} \cdot xx^\top$$

Key observation: $\alpha(\theta) = \sigma(z)(1 - \sigma(z)) > 0$ for all z because:

- $\sigma(z) \in (0, 1)$ for all $z \in \mathbb{R}$
- Therefore $\sigma(z)(1 - \sigma(z)) > 0$

Positive semi-definiteness: For any $v \in \mathbb{R}^d$:

$$v^\top \nabla^2 \ell \cdot v = \alpha(\theta) \cdot v^\top (xx^\top) v = \alpha(\theta) \cdot (x^\top v)^2 \geq 0$$

Conclusion: Logistic regression loss is convex!

Non-Convex Function Examples

Example 1: Simple non-convex function

$$f(\theta) = \theta^4 - \theta^2, \quad \theta \in \mathbb{R}$$

Analysis:

- Gradient: $f'(\theta) = 4\theta^3 - 2\theta = 2\theta(2\theta^2 - 1)$
- Critical points: $\theta = 0, \pm\frac{1}{\sqrt{2}}$
- Hessian: $f''(\theta) = 12\theta^2 - 2$
- At $\theta = 0$: $f''(0) = -2 < 0$ (saddle point!)
- At $\theta = \pm\frac{1}{\sqrt{2}}$: $f''(\pm\frac{1}{\sqrt{2}}) = 4 > 0$ (local minima)

Example 2: Matrix factorization loss

$$L(W, H) = \|X - WH\|_F^2$$

- Non-convex in joint (W, H) despite being convex in each separately
- Used in collaborative filtering, topic modeling

Non-Convex Optimization: The Real Challenge

Reality Check: Most ML problems are non-convex

- Neural networks: highly non-convex loss landscapes
- Matrix factorization, dictionary learning: non-convex
- Many clustering objectives: non-convex

Previous Analysis Assumed Convexity:
Our nice convergence guarantees ($O(1/t)$, exponential) only apply to convex functions!

Challenge

In non-convex settings, gradient descent may not find the global minimum

Local Minima

Definition: Point θ^* is a local minimum if:

$$L(\theta^*) \leq L(\theta) \text{ for all } \theta \text{ in a neighborhood of } \theta^*$$

Issue:

- GD converges to a local minimum, not necessarily global
- No gradient information can distinguish local from global at stationary point
- Different initializations $\rightarrow$ different local minima

Saddle Points

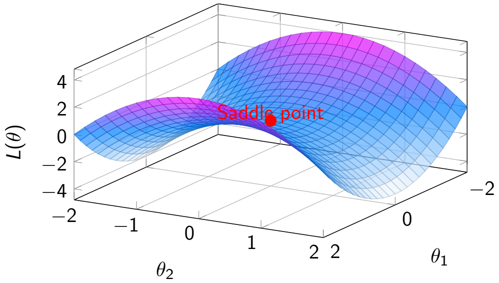
Definition: Point θ^* is a saddle point if:

- $\nabla L(\theta^*) = 0$ (gradient is zero)
- Hessian $\nabla^2 L(\theta^*)$ has both positive and negative eigenvalues
- Not a local minimum or maximum

Property:

- Function decreases in some directions, increases in others
- GD can get stuck at saddle points (especially with exact gradient)
- In high dimensions, saddle points are extremely common

Saddle Point: Visualization



Example: $L(\theta_1, \theta_2) = \theta_1^2 - \theta_2^2$ has saddle at origin

Mathematical Example: Saddle Point

Simple saddle point:

$$L(\theta_1, \theta_2) = \theta_1^2 - \theta_2^2$$

Gradient:

$$\nabla L = \begin{bmatrix} 2\theta_1 \\ -2\theta_2 \end{bmatrix} = 0 \text{ at origin}$$

Hessian at origin:

$$\nabla^2 L = \begin{bmatrix} 2 & 0 \\ 0 & -2 \end{bmatrix}$$

- Positive eigenvalue (2): upward curvature in θ_1 direction
- Negative eigenvalue (-2): downward curvature in θ_2 direction
- Origin is a saddle point, NOT a minimum

High-Dimensional Saddle Points

Key Insight (Dauphin et al., 2014): In high dimensions:

- Saddle points are exponentially more common than local minima
- Most critical points ($\nabla L = 0$) are saddle points, not local minima

Why? For random function in d dimensions:

- Probability that all d eigenvalues of Hessian are positive: $(1/2)^d \to 0$ as $d \to \infty$
- Local minimum requires ALL eigenvalues positive
- Saddle point needs just ONE negative eigenvalue: much more likely

Implication for Deep Learning:

- High-dimensional parameter spaces (millions of parameters)
- GD rarely gets trapped at *bad* local minima
- More likely to encounter saddle points
- Fortunately: SGD noise helps escape saddle points!

Plateaus and Slow Convergence

Plateau: Region where $\|\nabla L(\theta)\| \approx 0$ but not exactly zero

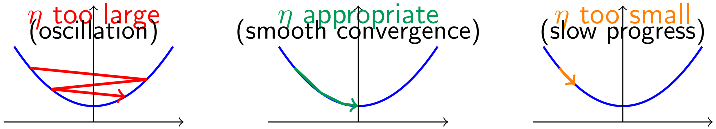
- GD makes very slow progress
- Can take many iterations to escape
- Looks like convergence, but haven't reached optimum

Example in Neural Networks:

- Saturated neurons (sigmoid/tanh): gradients near 0
- Vanishing gradients in deep networks

Sensitivity to Learning Rate

Problem: Step size η critically affects convergence



Challenge:

- Optimal η depends on local curvature (L-smoothness)
- L may vary across parameter space
- Requires tuning or adaptive methods

Summary of Limitations

Challenge	**GD Impact**	**Mitigation**
Local minima	Converges to local	Multiple inits
Saddle points	Can get stuck	SGD, add noise
Plateaus	Slow convergence	Adaptive LR
Learning rate	Critical tuning	Grid search

Despite These Limitations

GD/SGD remain the **workhorses** of modern ML

- Empirically successful for deep learning
- Many enhancements exist (momentum, adaptive methods)
- Theory still catching up to practice!

Motivation: Problem with Full-Batch GD

Gradient computation:

$$\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla L_i(\theta)$$

- Requires querying all n training data points
- Per-iteration complexity: $O(n)$
- Expensive for large datasets!

Example:

- ImageNet: $n \approx 1.2$ million images
- Each GD iteration requires forward/backward pass on all images
- For large models, this can take hours per iteration

Incremental Gradient Descent (SGD)

Algorithm:

1. Randomly permute training data $(x_1, y_1), \ldots, (x_n, y_n)$
2. In iteration t, use single data point (x_t, y_t):

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla L_t(\theta^{(t)})$$

Advantage: Per-iteration complexity $O(1)$ instead of $O(n)$

Trade-off

+ Much faster per iteration
– Noisier gradient estimates
– May need more iterations to converge

Mini-batch SGD

Algorithm:

1. Randomly sample mini-batch $\mathcal{I}^{(t)}$ with $|\mathcal{I}^{(t)}| = B$
2. Update using mini-batch gradient:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \frac{1}{B} \sum_{i \in \mathcal{I}^{(t)}} \nabla L_i(\theta^{(t)})$$

Properties:

- Expected update equals full-batch gradient:

$$\mathbb{E}[\theta^{(t+1)} \mid \theta^{(t)}] = \theta^{(t)} - \eta \cdot \nabla L(\theta^{(t)})$$

- Per-iteration complexity: $O(B)$, where $B \ll n$
- Trade-off between computation and variance

Behavior of SGD

Expected Direction: Same as GD in expectation

$$\mathbb{E}[\text{mini-batch gradient}] = \text{full gradient}$$

Variance: Stochastic gradient has variance

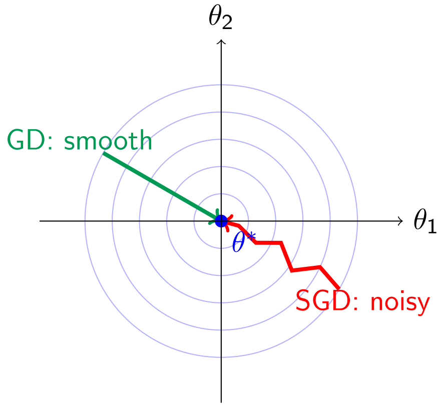
- Large variance: slower convergence, oscillation
- Small variance: behavior similar to GD

Key Insight

When data points are similar:

- Progress of one-step SGD $\approx$ progress of one-step GD
- Complexity of one-step SGD $\ll$ complexity of one-step GD

SGD vs GD: Visual Comparison



Contours of $L(\theta)$

SGD takes noisier path but can escape sharp minima

Convergence Analysis: Setup

Assume: All $L_i(\theta)$ are L-smooth
Let g_t be stochastic gradient at iteration t

By smoothness:

$$L(\theta^{(t+1)}) \leq L(\theta^{(t)}) - \eta_t \langle g_t, \nabla L(\theta^{(t)}) \rangle + \frac{L\eta_t^2}{2} \|g_t\|^2$$

Taking expectation:

$$\begin{aligned} \mathbb{E}[L(\theta^{(t+1)})] &\leq L(\theta^{(t)}) - \eta_t \|\nabla L(\theta^{(t)})\|^2 + \frac{L\eta_t^2}{2} \mathbb{E}[\|g_t\|^2] \\ &= L(\theta^{(t)}) - (\eta_t - L\eta_t^2/2) \|\nabla L(\theta^{(t)})\|^2 \\ &\quad + \frac{L\eta_t^2}{2} \mathbb{E}[\|g_t - \nabla L(\theta^{(t)})\|^2] \end{aligned}$$

Convergence Analysis: Two Terms

$$\mathbb{E}[L(\theta^{(t+1)})] \leq L(\theta^{(t)}) - \underbrace{(\eta_t - L\eta_t^2/2)\|\nabla L(\theta^{(t)})\|^2}_{\text{Decreasing term}} + \underbrace{\frac{L\eta_t^2}{2}\mathbb{E}[\|g_t - \nabla L(\theta^{(t)})\|^2]}_{\text{Variance term}}$$

Bounded Variance Assumption:

$$\mathbb{E}[\|g - \nabla L(\theta)\|^2] \leq \sigma^2$$

Key Observations:

- Need small stepsize ($\eta_t \to 0$) for convergence
- Otherwise SGD oscillates around stationary point
- Zero total gradient $\neq$ zero stochastic gradient

Convergence with Constant Stepsize

With constant stepsize $\eta \leq 1/L$:

$$\sum_{k=0}^{t} \eta_k \mathbb{E}[\|\nabla L(\theta^{(k)})\|^2] \leq 2[L(\theta^{(0)}) - \mathbb{E}[L(\theta^{(t)})]] + L \cdot \sum_{k=0}^{t} \eta_k^2 \sigma^2$$

Implications:

- Constant stepsize: SGD does not converge exactly
- Oscillates in a neighborhood of the optimum
- Neighborhood size depends on η and σ^2

For exact convergence:

- Need decaying stepsize $\eta_t \to 0$
- E.g., $\eta_t = O(1/t)$ or $\eta_t = O(1/\sqrt{t})$
- But this slows down convergence

SGD vs GD: Sample Complexity Comparison

Question: How many samples do SGD and GD need to achieve ϵ-accuracy?

Gradient Descent (GD):

- Uses all n samples per iteration
- Per-iteration cost: $O(nd)$
- Iterations needed: $O(\log(1/\epsilon))$ for strongly convex, $O(1/\epsilon)$ for convex
- **Total sample accesses:** $O(n \cdot \text{iterations})$

Stochastic Gradient Descent (SGD):

- Uses 1 sample per iteration (or mini-batch of size B)
- Per-iteration cost: $O(d)$ or $O(Bd)$
- Iterations needed: $O(1/\epsilon)$ for strongly convex, $O(1/\epsilon^2)$ for convex
- **Total sample accesses:** $O(\text{iterations})$

Sample Complexity: Detailed Analysis

For μ-strongly convex, L-smooth objectives:

Method	Iterations	Cost/Iter	Total Cost
GD	$O(\kappa \log(1/\epsilon))$	$O(nd)$	$O(\kappa nd \log(1/\epsilon))$
SGD	$O(1/(\mu\epsilon))$	$O(d)$	$O(d/(\mu\epsilon))$

where $\kappa = L/\mu$ is the condition number

Key observations:

- GD: logarithmic dependence on ϵ (fast convergence)
- SGD: linear dependence on $1/\epsilon$ (slower convergence per iteration)
- SGD wins when $n \gg 1/(\mu\epsilon)$ (large datasets!)
- Crossover point: $n \approx 1/(\mu\epsilon)$

Sample Complexity: When Does SGD Win?

Comparison for strongly convex case:

GD wins if: $\kappa n \log(1/\epsilon) < 1/(\mu\epsilon)$

- Small datasets (n small)
- Well-conditioned problems (κ small)
- High accuracy required (ϵ very small)

SGD wins if: $n \gg \frac{1}{\mu\epsilon\kappa \log(1/\epsilon)}$

- Large datasets (n large) - typical in modern ML!
- Moderate accuracy sufficient
- Can tolerate noisier updates

Example: $\epsilon = 0.01, \mu = 0.1, \kappa = 100$

- GD needs $\sim 460n$ sample accesses
- SGD needs ~ 100 sample accesses
- SGD wins decisively when $n > 100$!

Sample Complexity: Practical Implications

Modern machine learning regime:

- Large datasets: $n = 10^6$ to 10^9 (ImageNet, web-scale data)
- Moderate accuracy often sufficient: $\epsilon = 0.01$ or 0.001
- SGD-based methods dominate in practice

Additional SGD advantages:

- **Memory efficiency:** $O(d)$ vs $O(nd)$
- **Online learning:** can process streaming data
- **Generalization:** implicit regularization, better test performance
- **Hardware:** mini-batches fit in GPU memory

Trade-offs:

$+$ SGD: fewer total sample accesses, memory efficient
$-$ SGD: more iterations, noisier, harder to tune
$+$ GD: stable, predictable convergence
$-$ GD: expensive per iteration, not scalable to large n

Gradient Descent for High-Dimensional Linear Regression

Population Loss: For linear regression on data distribution

$$L(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[(y - w^\top x)^2]$$

Gradient of population loss:

$$\begin{aligned}\nabla L(w) &= \mathbb{E}_{(x,y)}[\nabla_w (y - w^\top x)^2] \\ &= \mathbb{E}_{(x,y)}[-2(y - w^\top x)x] \\ &= -2\mathbb{E}[(y - w^\top x)x]\end{aligned}$$

Gradient Descent update rule:

$$w_{t+1} = w_t - \eta \nabla L(w_t)$$

where η is the learning rate (stepsize)

GD Iterates: Exact Form

Data model: Assume $y = w^{*\top} x + \xi$ where $\mathbb{E}[\xi | x] = 0$ (noise)
Define: Covariance matrix $H = \mathbb{E}[xx^\top]$
Gradient at w**:**

$$\begin{aligned}\nabla L(w) &= -2\mathbb{E}[(y - w^\top x)x] \\ &= -2\mathbb{E}[(w^{*\top} x + \xi - w^\top x)x] \\ &= -2\mathbb{E}[((w^* - w)^\top x)x] \\ &= -2H(w^* - w) = 2H(w - w^*)\end{aligned}$$

GD update becomes:

$$w_{t+1} = w_t - 2\eta H(w_t - w^*) = (I - 2\eta H)w_t + 2\eta H w^*$$

GD Iterates: Closed Form

Recursive formula:

$$w_{t+1} - w^* = (I - 2\eta H)(w_t - w^*)$$

Unrolling the recursion:

$$\begin{aligned} w_t - w^* &= (I - 2\eta H)(w_{t-1} - w^*) \\ &= (I - 2\eta H)^2(w_{t-2} - w^*) \\ &= \dots \\ &= (I - 2\eta H)^t(w_0 - w^*) \end{aligned}$$

Exact form of iterates:

$$\boxed{w_t = w^* + (I - 2\eta H)^t(w_0 - w^*)}$$

Key insight: Convergence depends on eigenvalues of $(I - 2\eta H)$

Loss Function at Iterate w_t

Goal: Calculate $L(w_t) - L(w^*)$

Expand the loss:

$$\begin{aligned} L(w) &= \mathbb{E}[(y - w^\top x)^2] \\ &= \mathbb{E}[(w^{*\top} x + \xi - w^\top x)^2] \\ &= \mathbb{E}[((w^* - w)^\top x + \xi)^2] \\ &= \mathbb{E}[(w^* - w)^\top x x^\top (w^* - w)] + \mathbb{E}[\xi^2] \\ &= (w^* - w)^\top H (w^* - w) + \sigma^2 \end{aligned}$$

where $\sigma^2 = \mathbb{E}[\xi^2]$ is the noise variance

Optimal loss:

$$L(w^*) = \sigma^2$$

Excess Loss and Convergence Rate

Excess loss at iteration t:

$$\begin{aligned} L(w_t) - L(w^*) &= (w^* - w_t)^\top H(w^* - w_t) \\ &= \|(w^* - w_t)\|_H^2 \end{aligned}$$

where $\|v\|_H^2 = v^\top Hv$ is the Mahalanobis norm

Substitute the iterate formula:

$$\begin{aligned} L(w_t) - L(w^*) &= \|(I - 2\eta H)^t(w_0 - w^*)\|_H^2 \\ &= (w_0 - w^*)^\top[(I - 2\eta H)^t]^\top H[(I - 2\eta H)^t](w_0 - w^*) \end{aligned}$$

Eigenvalue analysis: If H has eigenvalues $\lambda_1 \geq \ldots \geq \lambda_d$:

- Convergence rate in direction v_i: $(1 - 2\eta\lambda_i)^{2t}$
- Fast convergence: large λ_i (high-variance directions)
- Slow convergence: small λ_i (low-variance directions)

Streaming SGD: Stochastic Version of GD

Constant-stepsize Streaming SGD:
In each iteration t, sample fresh (x_t, y_t) from data distribution:

$$w_t = w_{t-1} + \gamma \cdot (y_t - \langle w_{t-1}, x_t \rangle) \cdot x_t$$

- γ: constant stepsize
- $(y_t - \langle w_{t-1}, x_t \rangle) \cdot x_t$: stochastic gradient

Averaging the iterates:

- With constant stepsize, SGD oscillates around optimum
- Averaging reduces variance and gives better solution

$$\bar{w} = \frac{1}{K} \sum_{t=n-K}^{n-1} w_t.$$

High-Dimensional Linear Regression: Convergence Behavior

Key insight: In high dimensions, SGD/GD learn different features at different rates

Convergence rate depends on dimension:

- For linear regression with covariance matrix $H = \mathbb{E}[xx^\top]$
- H has eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$
- Dimensions with large eigenvalues: learn quickly
- Dimensions with small eigenvalues: learn slowly or not at all

Implications:

- **High-eigenvalue directions:** Converge in $O(\log(1/\epsilon))$ iterations
- **Low-eigenvalue directions:** May need $\gg d$ iterations
- GD/SGD provides implicit regularization by focusing on high-variance directions
- Similar to Ridge regression but without explicit λ tuning

Practical takeaway: GD/SGD naturally prioritizes important features in data

Simulation: Eigenvalue Decay $\lambda_k = 1/k$

Setup:

- Dimension $d = 20$
- Covariance matrix H with eigenvalues $\lambda_k = 1/k$ for $k = 1, \ldots, d$
- Learning rate $\eta = 0.05$
- Run GD: $w_{t+1} = w_t - 2\eta H(w_t - w^*)$

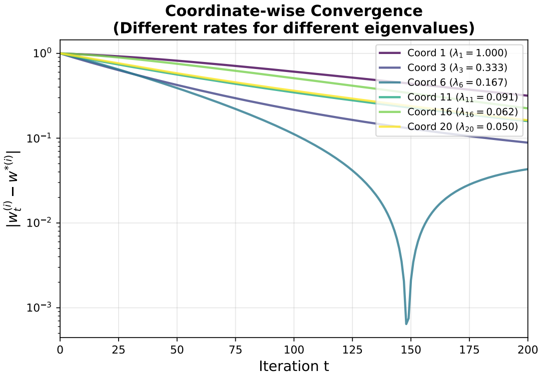
Theory predicts:

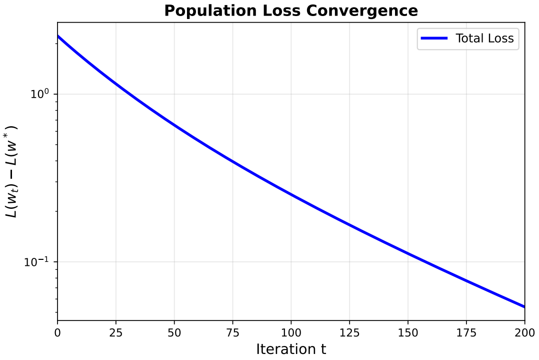
- Coordinate i converges at rate $(1 - 2\eta\lambda_i)^t$
- Large λ_i (first coordinates): fast convergence
- Small λ_i (last coordinates): slow convergence

Observations:

- First coordinate ($\lambda_1 = 1.0$): converges in $\sim$20 iterations
- Middle coordinates ($\lambda_{10} \approx 0.1$): converges in $\sim$100 iterations
- Last coordinates ($\lambda_{20} = 0.05$): still converging after 200 iterations

Coordinate-wise Convergence Dynamics





Key observations:

- Different coordinates converge at vastly different rates
- Convergence rate proportional to eigenvalue magnitude
- High-eigenvalue directions dominate early iterations

Neural Network Structure

Multi-layer network:

$$\text{Input } x \to \text{Layer } 1 \to \text{Layer } 2 \to \dots \to \text{Layer } L \to \text{Output } f(x;\theta)$$

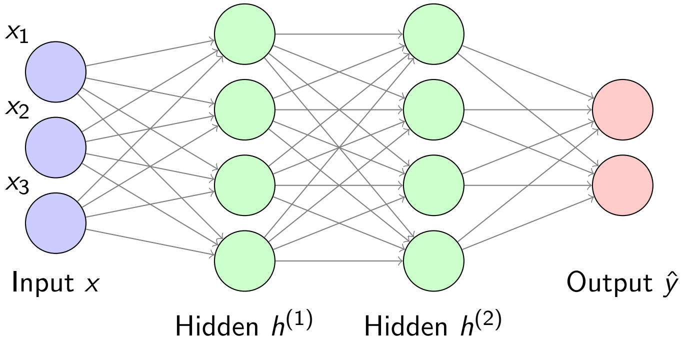
Layer computation:

$$h^{(\ell)} = \sigma(W^{(\ell)} h^{(\ell-1)} + b^{(\ell)})$$

where:

- $W^{(\ell)}$: weight matrix at layer ℓ
- $b^{(\ell)}$: bias vector
- σ: activation function (ReLU, sigmoid, tanh)
- $h^{(0)} = x$ (input)

Neural Network Visualization



Chain Rule for Gradients

Loss function: $L(\theta) = \ell(f(x;\theta), y)$
Goal: Compute $\nabla_\theta L$ for gradient descent

Chain rule: For nested functions $f(g(h(x)))$:

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{dx}$$

For neural networks:

- Forward pass: compute activations layer by layer
- Backward pass: compute gradients using chain rule
- Gradients flow backward through the network

Backpropagation Algorithm

Forward Pass (compute activations):

```
1: for $\ell = 1$ to $L$ do
2:     $z^{(\ell)} = W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}$
3:     $h^{(\ell)} = \sigma(z^{(\ell)})$
4: end for
5: Output: $f(x;\theta) = h^{(L)}$
```

Backward Pass (compute gradients):

```
1: Compute output gradient: $\delta^{(L)} = \nabla_{h^{(L)}} \ell(f(x;\theta), y)$
2: for $\ell = L$ down to 1 do
3:     $\delta^{(\ell)} = \delta^{(\ell+1)} \odot \sigma'(z^{(\ell)})$     (element-wise)
4:     $\nabla_{W^{(\ell)}} L = \delta^{(\ell)} (h^{(\ell-1)})^\top$
5:     $\nabla_{b^{(\ell)}} L = \delta^{(\ell)}$
6:     $\delta^{(\ell-1)} = (W^{(\ell)})^\top \delta^{(\ell)}$
7: end for
```

Backpropagation: Key Formula

Chain rule applied:

$$\delta^{(\ell)} = \frac{\partial L}{\partial z^{(\ell)}} = \frac{\partial L}{\partial z^{(\ell+1)}} \cdot \frac{\partial z^{(\ell+1)}}{\partial h^{(\ell)}} \cdot \frac{\partial h^{(\ell)}}{\partial z^{(\ell)}}$$

Intuition:

- $\delta^{(\ell)}$: sensitivity of loss to pre-activation at layer ℓ
- Propagates backward using chain rule
- Each layer: multiply by local derivatives

Efficiency:

- Reuses intermediate activations from forward pass
- Avoids recomputing derivatives multiple times
- Critical for training deep networks

Example: 2-Layer Network

Network:

$$h^{(1)} = \sigma(W^{(1)}x + b^{(1)})$$
$$f(x) = W^{(2)}h^{(1)} + b^{(2)}$$

Loss: $L = \frac{1}{2}\|f(x) - y\|^2$

Gradients:

Output layer:

$$\delta^{(2)} = f(x) - y$$
$$\nabla_{W^{(2)}} L = \delta^{(2)}(h^{(1)})^\top, \quad \nabla_{b^{(2)}} L = \delta^{(2)}$$

Hidden layer:

$$\delta^{(1)} = [(W^{(2)})^\top \delta^{(2)}] \odot \sigma'(W^{(1)}x + b^{(1)})$$
$$\nabla_{W^{(1)}} L = \delta^{(1)}x^\top, \quad \nabla_{b^{(1)}} L = \delta^{(1)}$$

SGD for Neural Network Training

Mini-batch SGD with backpropagation:

```
1: Sample mini-batch $\{(x_i, y_i)\}_{i\in\mathcal{I}}$
2: for each $(x_i, y_i)$ do
3:     Forward pass: compute $f(x_i; \theta)$
4:     Backward pass: compute $\nabla_\theta \ell(f(x_i; \theta), y_i)$
5: end for
6: Average gradients: $g = \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \nabla_\theta \ell(f(x_i; \theta), y_i)$
7: Update parameters: $\theta \leftarrow \theta - \eta \cdot g$
```

Computational Efficiency:

- Backpropagation: $O(\#\text{parameters})$ per sample
- Without backpropagation: would need $O(\#\text{parameters}^2)$
- Critical for training large networks

Summary: Gradient Descent

- ✓ Fundamental optimization algorithm for ML
- ✓ Convergence depends on function properties
  - Smooth: $O(1/t)$ for gradient norm
  - Smooth + Convex: $O(1/t)$ for function value
  - Smooth + Strongly convex: $O(\exp(-t/\kappa))$ exponential
- ✓ Requires computing full gradient: $O(n)$ per iteration
- × Can get trapped at local minima or saddle points (non-convex)
- × Expensive for large datasets

Summary: Stochastic Gradient Descent

- ✓ Uses mini-batches or single samples: $O(B)$ or $O(1)$ per iteration
- ✓ Expected update direction $=$ gradient direction
- ✓ Can process streaming data
- ✓ Noise helps escape saddle points
- × Has variance $\rightarrow$ needs careful stepsize selection
- × Constant stepsize: oscillates around optimum
- × Decaying stepsize: converges but slower per iteration

Summary: Applications

Linear Models:

- Regression, logistic: convex, well-understood
- SGD for linear regression: implicit regularization
- Competitive with ridge regression in many cases

Neural Networks:

- Backpropagation for efficient gradient computation
- SGD is the workhorse for deep learning
- Works empirically despite non-convexity

Practical Considerations:

- Stepsize selection is crucial (typically $\eta \leq 1/L$)
- Mini-batch size: trade-off between computation and variance
- For large datasets, SGD variants are essential

Key Takeaways

Why Numerical Optimization?

Modern ML problems are high-dimensional with complicated objectives. Gradient-based methods are the **only viable approach**.

Gradient Descent

Fundamental algorithm with strong theoretical guarantees for convex problems. Limitations in non-convex settings (local minima, saddle points).

Stochastic Gradient Descent

Scales to large datasets via mini-batching. Trade-off: faster iterations but noisier updates. Essential for modern deep learning.

Backpropagation

Efficient computation of gradients using chain rule. Enables training of deep neural networks.

Looking Forward

Topics beyond this lecture:

- **Momentum methods:** Accelerate convergence by using historical gradients
- **Adaptive learning rates:** AdaGrad, RMSProp, Adam
- **Variance reduction:** SVRG, SAGA for combining GD and SGD benefits
- **Second-order methods:** Newton's method, L-BFGS
- **Non-convex optimization theory:** Understanding why SGD works for deep learning

Thank you!