

Data Science for Mathematicians

Lesson 5: Gradient Descent

Department of Mathematics and Computer Science

Outline

- 1 The Engine of Learning
- 2 The Calculus of Loss Landscapes
- 3 Batch Gradient Descent
- 4 Convexity and Convergence Guarantees
- 5 Scaling to Large Data

From Models to Optimization

A supervised learning system has two components:

- **Model:** a parameterized function $f_{\theta}(\mathbf{x})$ with parameter vector $\theta \in \mathbb{R}^p$
- **Loss function:** $L(\theta)$ measures the total prediction error over training data

For ordinary least squares (OLS):

$$L(\beta) = \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|_2^2$$

From Models to Optimization

A supervised learning system has two components:

- **Model:** a parameterized function $f_{\theta}(\mathbf{x})$ with parameter vector $\theta \in \mathbb{R}^p$
- **Loss function:** $L(\theta)$ measures the total prediction error over training data

For ordinary least squares (OLS):

$$L(\beta) = \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|_2^2$$

The Central Problem

Training a model means solving the optimization problem

$$\theta^* = \arg \min_{\theta \in \mathbb{R}^p} L(\theta).$$

The loss function $L(\theta)$ defines a **loss landscape**—a high-dimensional surface whose lowest point is the optimal model.

Why Analytical Solutions Fail

OLS has a rare closed-form solution: $\beta^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$.

This requires setting $\nabla L(\beta) = \mathbf{0}$ and solving the system. Two conditions make this infeasible in general:

Analytical Intractability

For logistic regression, neural networks, etc., $\nabla L(\theta) = \mathbf{0}$ is a system of nonlinear equations with no closed form.

Computational Infeasibility

Inverting $\mathbf{X}^T \mathbf{X}$ (a $p \times p$ matrix) costs $O(p^3)$. For $p = 10^5$, this is 10^{15} operations—infeasible.

Why Analytical Solutions Fail

OLS has a rare closed-form solution: $\beta^* = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$.

This requires setting $\nabla L(\beta) = \mathbf{0}$ and solving the system. Two conditions make this infeasible in general:

Analytical Intractability

For logistic regression, neural networks, etc., $\nabla L(\theta) = \mathbf{0}$ is a system of nonlinear equations with no closed form.

Computational Infeasibility

Inverting $\mathbf{X}^T \mathbf{X}$ (a $p \times p$ matrix) costs $O(p^3)$. For $p = 10^5$, this is 10^{15} operations—infeasible.

The Solution: Iterative Methods

Instead of a perfect global answer in one step, iterative methods make *local, incremental* progress—far more broadly applicable.

The Loss Landscape

Definition: Loss Landscape

The **loss landscape** of a model with parameters $\theta \in \mathbb{R}^p$ is the scalar field $L: \mathbb{R}^p \rightarrow \mathbb{R}$. Each point represents a parameterization; the value at that point is the model error.

The Loss Landscape

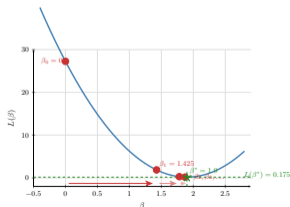
Definition: Loss Landscape

The **loss landscape** of a model with parameters $\theta \in \mathbb{R}^p$ is the scalar field $L: \mathbb{R}^p \rightarrow \mathbb{R}$. Each point represents a parameterization; the value at that point is the model error.

$p = 1: L: \mathbb{R} \rightarrow \mathbb{R}$
A parabola

$p = 2: L: \mathbb{R}^2 \rightarrow \mathbb{R}$
A 3D surface (paraboloid for OLS)

$p \gg 2: L: \mathbb{R}^p \rightarrow \mathbb{R}$
Impossible to visualize—rely on gradients and Hessians



The Gradient: Direction of Steepest Ascent

Definition: Gradient

The **gradient** of $L(\boldsymbol{\theta})$ at $\boldsymbol{\theta} \in \mathbb{R}^p$ is the vector of first-order partial derivatives:

$$\nabla L(\boldsymbol{\theta}) = \left[\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \dots, \frac{\partial L}{\partial \theta_p} \right]^T \in \mathbb{R}^p.$$

The Gradient: Direction of Steepest Ascent

Definition: Gradient

The **gradient** of $L(\boldsymbol{\theta})$ at $\boldsymbol{\theta} \in \mathbb{R}^p$ is the vector of first-order partial derivatives:

$$\nabla L(\boldsymbol{\theta}) = \left[\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \dots, \frac{\partial L}{\partial \theta_p} \right]^T \in \mathbb{R}^p.$$

Three key geometric facts:

- $\nabla L(\boldsymbol{\theta})$ points in the direction of **steepest ascent**
- $\|\nabla L(\boldsymbol{\theta})\|_2$ measures the rate of ascent (large = steep)
- $\nabla L(\boldsymbol{\theta}) = \mathbf{0}$ at a **critical point** (locally flat landscape)

The Gradient: Direction of Steepest Ascent

Definition: Gradient

The **gradient** of $L(\theta)$ at $\theta \in \mathbb{R}^p$ is the vector of first-order partial derivatives:

$$\nabla L(\theta) = \left[\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \dots, \frac{\partial L}{\partial \theta_p} \right]^T \in \mathbb{R}^p.$$

Three key geometric facts:

- $\nabla L(\theta)$ points in the direction of **steepest ascent**
- $\|\nabla L(\theta)\|_2$ measures the rate of ascent (large = steep)
- $\nabla L(\theta) = \mathbf{0}$ at a **critical point** (locally flat landscape)

Key Insight for Optimization

The **negative gradient** $-\nabla L(\theta)$ points in the direction of **steepest descent**. Taking a step in this direction reduces the loss.

Example: Gradient of a Quadratic

Example: Gradient in \mathbb{R}^2

Let $L(\boldsymbol{\theta}) = \theta_1^2 + 4\theta_2^2$ for $\boldsymbol{\theta} = [\theta_1, \theta_2]^T \in \mathbb{R}^2$.

$$\nabla L(\boldsymbol{\theta}) = \begin{pmatrix} 2\theta_1 \\ 8\theta_2 \end{pmatrix}$$

Example: Gradient of a Quadratic

Example: Gradient in \mathbb{R}^2

Let $L(\boldsymbol{\theta}) = \theta_1^2 + 4\theta_2^2$ for $\boldsymbol{\theta} = [\theta_1, \theta_2]^T \in \mathbb{R}^2$.

$$\nabla L(\boldsymbol{\theta}) = \begin{pmatrix} 2\theta_1 \\ 8\theta_2 \end{pmatrix}$$

At $\boldsymbol{\theta} = [1, 1]^T$: $\nabla L = [2, 8]^T$, with $\|\nabla L\|_2 = \sqrt{68} \approx 8.25$.

The gradient points predominantly in the θ_2 direction—the curvature is 4 times steeper along θ_2 than θ_1 .

At $\boldsymbol{\theta} = \mathbf{0}$: $\nabla L = \mathbf{0}$ —the origin is a critical point (the global minimum).

The Hessian: Local Curvature

Definition: Hessian Matrix

Let $L: \mathbb{R}^p \rightarrow \mathbb{R}$ be twice continuously differentiable. The **Hessian** $\mathbf{H}_L(\boldsymbol{\theta})$ is the $p \times p$ matrix of second-order partial derivatives:

$$[\mathbf{H}_L(\boldsymbol{\theta})]_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j}, \quad i, j = 1, \dots, p.$$

The Hessian: Local Curvature

Definition: Hessian Matrix

Let $L: \mathbb{R}^p \rightarrow \mathbb{R}$ be twice continuously differentiable. The **Hessian** $\mathbf{H}_L(\boldsymbol{\theta})$ is the $p \times p$ matrix of second-order partial derivatives:

$$[\mathbf{H}_L(\boldsymbol{\theta})]_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j}, \quad i, j = 1, \dots, p.$$

Theorem: Symmetry of the Hessian

If L is twice continuously differentiable, then $\mathbf{H}_L(\boldsymbol{\theta}) = \mathbf{H}_L(\boldsymbol{\theta})^T$ everywhere. (Clairaut's theorem: mixed partials commute.)

Example

For $L(\boldsymbol{\theta}) = \theta_1^2 + 4\theta_2^2$: $\mathbf{H}_L = \begin{pmatrix} 2 & 0 \\ 0 & 8 \end{pmatrix}$. Eigenvalues 2 and 8 reveal that curvature along θ_2 is four times steeper than θ_1 .

Classifying Critical Points

At a critical point θ_c (where $\nabla L(\theta_c) = \mathbf{0}$), the **eigenvalues** of $\mathbf{H}_L(\theta_c)$ determine its type:

Theorem: Second Derivative Test

- ① All eigenvalues $> 0 \Rightarrow$ **positive definite** \Rightarrow local minimum
- ② All eigenvalues $< 0 \Rightarrow$ **negative definite** \Rightarrow local maximum
- ③ Mixed signs \Rightarrow **indefinite** \Rightarrow saddle point
- ④ Any zero eigenvalue \Rightarrow test is **inconclusive**

Classifying Critical Points

At a critical point θ_c (where $\nabla L(\theta_c) = \mathbf{0}$), the **eigenvalues** of $\mathbf{H}_L(\theta_c)$ determine its type:

Theorem: Second Derivative Test

- ① All eigenvalues $> 0 \Rightarrow$ **positive definite** \Rightarrow local minimum
- ② All eigenvalues $< 0 \Rightarrow$ **negative definite** \Rightarrow local maximum
- ③ Mixed signs \Rightarrow **indefinite** \Rightarrow saddle point
- ④ Any zero eigenvalue \Rightarrow test is **inconclusive**

Example: Saddle Point

$f(x_1, x_2) = x_1^2 - x_2^2$ has $\nabla f = \mathbf{0}$ at the origin. Hessian: $\mathbf{H}_f = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$ — eigenvalues ± 2 , so the origin is a saddle point. The surface curves *up* along x_1 and *down* along x_2 .

Taylor Expansions and the Optimization Hierarchy

Expanding $L(\boldsymbol{\theta})$ around a point $\boldsymbol{\theta}_k$:

First-order (linear) approximation:

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}_k) + \nabla L(\boldsymbol{\theta}_k)^T (\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Second-order (quadratic) approximation:

$$L(\boldsymbol{\theta}) \approx L(\boldsymbol{\theta}_k) + \nabla L(\boldsymbol{\theta}_k)^T (\boldsymbol{\theta} - \boldsymbol{\theta}_k) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_k)^T \mathbf{H}_L(\boldsymbol{\theta}_k)(\boldsymbol{\theta} - \boldsymbol{\theta}_k)$$

Taylor Expansions and the Optimization Hierarchy

Expanding $L(\theta)$ around a point θ_k :

First-order (linear) approximation:

$$L(\theta) \approx L(\theta_k) + \nabla L(\theta_k)^T (\theta - \theta_k)$$

Second-order (quadratic) approximation:

$$L(\theta) \approx L(\theta_k) + \nabla L(\theta_k)^T (\theta - \theta_k) + \frac{1}{2}(\theta - \theta_k)^T \mathbf{H}_L(\theta_k)(\theta - \theta_k)$$

Hierarchy of Optimization Methods

- **First-order methods** (Gradient Descent): use only ∇L — cheap per step, broadly applicable
- **Second-order methods** (Newton's method): use ∇L and \mathbf{H}_L — faster convergence, but computing the Hessian costs $O(p^2)$ memory and $O(p^3)$ operations

Deriving the Update Rule

We seek a sequence $\{\boldsymbol{\theta}_k\}$ with $L(\boldsymbol{\theta}_{k+1}) < L(\boldsymbol{\theta}_k)$.

Write $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}$. First-order Taylor expansion:

$$L(\boldsymbol{\theta}_{k+1}) \approx L(\boldsymbol{\theta}_k) + \nabla L(\boldsymbol{\theta}_k)^T \Delta\boldsymbol{\theta}$$

Deriving the Update Rule

We seek a sequence $\{\boldsymbol{\theta}_k\}$ with $L(\boldsymbol{\theta}_{k+1}) < L(\boldsymbol{\theta}_k)$.

Write $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}$. First-order Taylor expansion:

$$L(\boldsymbol{\theta}_{k+1}) \approx L(\boldsymbol{\theta}_k) + \nabla L(\boldsymbol{\theta}_k)^T \Delta\boldsymbol{\theta}$$

To guarantee descent, we need $\nabla L(\boldsymbol{\theta}_k)^T \Delta\boldsymbol{\theta} < 0$. By Cauchy–Schwarz, this inner product is minimized when $\Delta\boldsymbol{\theta}$ is *antiparallel* to $\nabla L(\boldsymbol{\theta}_k)$:

$$\Delta\boldsymbol{\theta} = -\eta \nabla L(\boldsymbol{\theta}_k), \quad \eta > 0.$$

Deriving the Update Rule

We seek a sequence $\{\boldsymbol{\theta}_k\}$ with $L(\boldsymbol{\theta}_{k+1}) < L(\boldsymbol{\theta}_k)$.

Write $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}$. First-order Taylor expansion:

$$L(\boldsymbol{\theta}_{k+1}) \approx L(\boldsymbol{\theta}_k) + \nabla L(\boldsymbol{\theta}_k)^T \Delta\boldsymbol{\theta}$$

To guarantee descent, we need $\nabla L(\boldsymbol{\theta}_k)^T \Delta\boldsymbol{\theta} < 0$. By Cauchy–Schwarz, this inner product is minimized when $\Delta\boldsymbol{\theta}$ is *antiparallel* to $\nabla L(\boldsymbol{\theta}_k)$:

$$\Delta\boldsymbol{\theta} = -\eta \nabla L(\boldsymbol{\theta}_k), \quad \eta > 0.$$

Then:

$$L(\boldsymbol{\theta}_{k+1}) \approx L(\boldsymbol{\theta}_k) - \eta \|\nabla L(\boldsymbol{\theta}_k)\|_2^2 \leq L(\boldsymbol{\theta}_k).$$

Gradient Descent Update Rule

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla L(\boldsymbol{\theta}_k)$$

Algorithm: Batch Gradient Descent

Batch Gradient Descent

Input: θ_0 , learning rate $\eta > 0$, tolerance $\epsilon > 0$

While $\|\nabla L(\theta_k)\|_2 > \epsilon$:

① Compute gradient over *all* n samples:

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

② Update: $\theta_{k+1} \leftarrow \theta_k - \eta \nabla L(\theta_k)$

Return θ_k

Algorithm: Batch Gradient Descent

Batch Gradient Descent

Input: θ_0 , learning rate $\eta > 0$, tolerance $\epsilon > 0$

While $\|\nabla L(\theta_k)\|_2 > \epsilon$:

① Compute gradient over *all* n samples:

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

② Update: $\theta_{k+1} \leftarrow \theta_k - \eta \nabla L(\theta_k)$

Return θ_k

Named “batch” GD because the gradient is computed over the entire *batch* of training data. For convex loss functions, this is guaranteed to converge to the global minimum (Section 4).

Matrix Calculus for OLS

Theorem: Matrix Calculus Identities

For $\beta \in \mathbb{R}^p$, $\mathbf{c} \in \mathbb{R}^p$, symmetric $\mathbf{A} \in \mathbb{R}^{p \times p}$:

- ① $\nabla_{\beta}(\mathbf{c}^T \beta) = \mathbf{c}$
- ② $\nabla_{\beta}(\beta^T \mathbf{A} \beta) = 2\mathbf{A}\beta$ (when $\mathbf{A} = \mathbf{A}^T$)

Matrix Calculus for OLS

Theorem: Matrix Calculus Identities

For $\beta \in \mathbb{R}^p$, $\mathbf{c} \in \mathbb{R}^p$, symmetric $\mathbf{A} \in \mathbb{R}^{p \times p}$:

- 1 $\nabla_{\beta}(\mathbf{c}^T \beta) = \mathbf{c}$
- 2 $\nabla_{\beta}(\beta^T \mathbf{A} \beta) = 2\mathbf{A}\beta$ (when $\mathbf{A} = \mathbf{A}^T$)

The OLS loss expands as a quadratic form:

$$L(\beta) = \frac{1}{n}(\beta^T (\mathbf{X}^T \mathbf{X}) \beta - 2\beta^T (\mathbf{X}^T \mathbf{y}) + \mathbf{y}^T \mathbf{y})$$

Applying the identities term by term (note $\mathbf{X}^T \mathbf{X}$ is symmetric):

$$\nabla L(\beta) = \frac{1}{n}(2(\mathbf{X}^T \mathbf{X})\beta - 2\mathbf{X}^T \mathbf{y}) = \frac{2}{n}\mathbf{X}^T (\mathbf{X}\beta - \mathbf{y})$$

OLS Update Rule

Gradient Descent Update for OLS

$$\beta_{k+1} = \beta_k - \frac{2\eta}{n} \mathbf{X}^T (\mathbf{X}\beta_k - \mathbf{y})$$

Interpretation of $\mathbf{X}^T (\mathbf{X}\beta_k - \mathbf{y})$:

- $(\mathbf{X}\beta_k - \mathbf{y}) \in \mathbb{R}^n$ is the **residual vector**—the prediction error for each data point
- \mathbf{X}^T projects these residuals back into the **parameter space** to determine the update direction

OLS Update Rule

Gradient Descent Update for OLS

$$\beta_{k+1} = \beta_k - \frac{2\eta}{n} \mathbf{X}^T (\mathbf{X}\beta_k - \mathbf{y})$$

Interpretation of $\mathbf{X}^T (\mathbf{X}\beta_k - \mathbf{y})$:

- $(\mathbf{X}\beta_k - \mathbf{y}) \in \mathbb{R}^n$ is the **residual vector**—the prediction error for each data point
- \mathbf{X}^T projects these residuals back into the **parameter space** to determine the update direction

Each iteration requires one matrix-vector product: cost $O(np)$ per update.

The Critical Role of the Learning Rate

The **learning rate** η measures how much we trust the local linear (Taylor) approximation.

η too small

- Loss reliably decreases
- Extremely slow convergence

η too large

- Step overshoots minimum
- Loss oscillates or diverges

η well-chosen

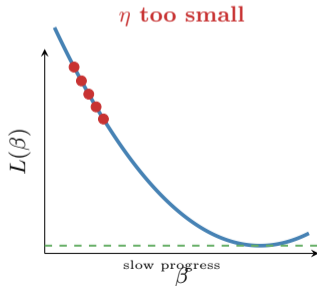
- Balances speed with stability
- Smooth convergence

The Critical Role of the Learning Rate

The **learning rate** η measures how much we trust the local linear (Taylor) approximation.

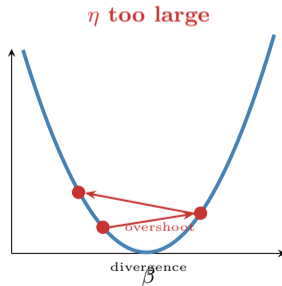
η too small

- Loss reliably decreases
- Extremely slow convergence



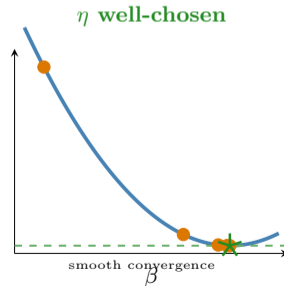
η too large

- Step overshoots minimum
- Loss oscillates or diverges



η well-chosen

- Balances speed with stability
- Smooth convergence



Convergence Condition for OLS

For OLS, the Hessian is constant: $\mathbf{H}_L = \frac{2}{n} \mathbf{X}^T \mathbf{X}$.

Let λ_{\max} be the largest eigenvalue of \mathbf{H}_L . Batch GD converges if and only if

$$\eta < \frac{2}{\lambda_{\max}}.$$

Example: Instability from Excessive η

For $n = 4$, $\mathbf{x} = [1, 2, 3, 4]^T$, $\mathbf{y} = [2, 4, 5, 8]^T$: the Hessian is $H_L = 15$, so the stability bound is $\eta < 2/15 \approx 0.133$.

With $\eta = 0.15$ (violating the bound), the contraction factor $|1 - 0.15 \times 15| = 1.25 > 1$:

$$\beta_0 = 0 \rightarrow \beta_1 = 4.275 \rightarrow \beta_2 = -1.069 \rightarrow \beta_3 = 5.611 \rightarrow \dots$$

The loss grows by a factor of ≈ 1.56 at each step—geometric divergence.

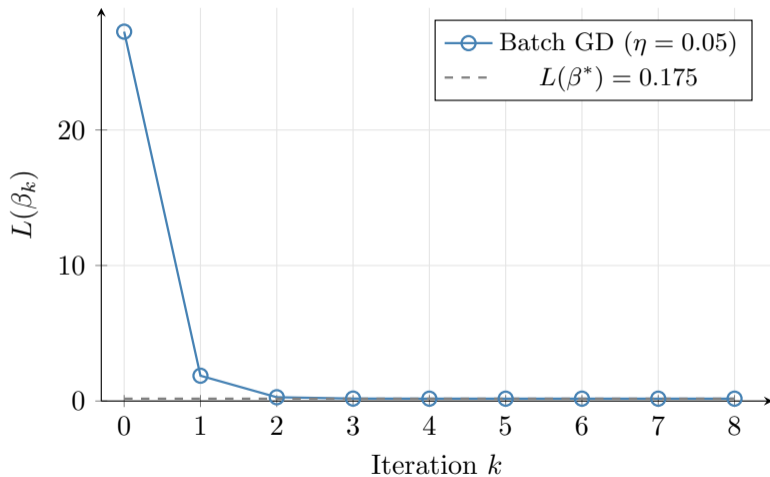
Example: Batch GD Convergence Trace

Dataset: $y = \beta x$, $n = 4$, $\mathbf{x} = [1, 2, 3, 4]^T$, $\mathbf{y} = [2, 4, 5, 8]^T$, $\eta = 0.05$, $\beta_0 = 0$. Optimal: $\beta^* = 1.9$, $L(\beta^*) = 0.175$.

k	β_k	$\nabla L(\beta_k)$	$L(\beta_k)$
0	0.0000	-28.500	27.2500
1	1.4250	-7.125	1.8672
2	1.7813	-1.781	0.2808
3	1.8703	-0.445	0.1816
4	1.8926	-0.111	0.1754
5	1.8981	-0.028	0.1750

The loss decreases **monotonically**—a hallmark of batch GD with a well-chosen η . By iteration 3, β has captured 98% of the distance to β^* .

Batch GD: Smooth Monotonic Convergence



Computational Cost of Batch GD

Each gradient evaluation requires a full pass over the entire dataset—cost $O(np)$ per update.

Example: Cost at Scale

With $n = 10^6$ samples and $p = 10^3$ features:

- Cost per gradient: $n \times p = 10^9$ operations
- Convergence requires $\sim 10^3$ iterations
- Total: 10^{12} operations ≈ 17 minutes on a desktop CPU
- A 10-point hyperparameter search: ≈ 170 minutes *per model*

Computational Cost of Batch GD

Each gradient evaluation requires a full pass over the entire dataset—cost $O(np)$ per update.

Example: Cost at Scale

With $n = 10^6$ samples and $p = 10^3$ features:

- Cost per gradient: $n \times p = 10^9$ operations
- Convergence requires $\sim 10^3$ iterations
- Total: 10^{12} operations ≈ 17 minutes on a desktop CPU
- A 10-point hyperparameter search: ≈ 170 minutes *per model*

Key Weakness

Batch GD computes an exact gradient at every step, but exactness is expensive. Every update requires a full dataset pass regardless of gradient estimation quality.

Why Convexity Matters

Gradient descent is guaranteed to take steps **downhill**—but not necessarily toward the *global* minimum.

A non-convex landscape can contain:

- **Local minima:** small valleys that trap gradient descent
- **Saddle points:** flat regions that slow convergence
- **Plateaus:** nearly zero gradient, making progress undetectable

Why Convexity Matters

Gradient descent is guaranteed to take steps **downhill**—but not necessarily toward the *global* minimum.

A non-convex landscape can contain:

- **Local minima:** small valleys that trap gradient descent
- **Saddle points:** flat regions that slow convergence
- **Plateaus:** nearly zero gradient, making progress undetectable

The Good News for OLS

The OLS loss function is **convex**—its landscape is a single bowl with no local minima. This guarantees gradient descent converges to the *unique* global minimum.

Convex Sets and Convex Functions

Definition: Convex Set

$C \subseteq \mathbb{R}^p$ is **convex** if for any $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$: $\lambda \mathbf{x} + (1 - \lambda) \mathbf{y} \in C$. The line segment connecting any two points stays in C .

Convex Sets and Convex Functions

Definition: Convex Set

$C \subseteq \mathbb{R}^p$ is **convex** if for any $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$: $\lambda \mathbf{x} + (1 - \lambda) \mathbf{y} \in C$. The line segment connecting any two points stays in C .

Definition: Convex Function

$f: C \rightarrow \mathbb{R}$ is **convex** if for any $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$:

$$f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}).$$

The chord connecting two points on the graph lies *above* the graph.

Convex Sets and Convex Functions

Definition: Convex Set

$C \subseteq \mathbb{R}^p$ is **convex** if for any $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$: $\lambda \mathbf{x} + (1 - \lambda) \mathbf{y} \in C$. The line segment connecting any two points stays in C .

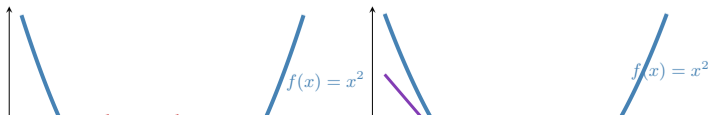
Definition: Convex Function

$f: C \rightarrow \mathbb{R}$ is **convex** if for any $\mathbf{x}, \mathbf{y} \in C$ and $\lambda \in [0, 1]$:

$$f(\lambda \mathbf{x} + (1 - \lambda) \mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}).$$

The chord connecting two points on the graph lies *above* the graph.

Convex function: chord lies above graph First-order condition: tangent lies below graph



Characterizing Convexity

Lemma: First-Order Condition

f is convex on an open convex C if and only if for all $\mathbf{x}, \mathbf{y} \in C$:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}).$$

The tangent hyperplane at any point is a **global underestimator** of f .

Characterizing Convexity

Lemma: First-Order Condition

f is convex on an open convex C if and only if for all $\mathbf{x}, \mathbf{y} \in C$:

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}).$$

The tangent hyperplane at any point is a **global underestimator** of f .

Lemma: Second-Order Condition

f twice continuously differentiable on open convex C is convex if and only if:

$$\mathbf{H}_f(\mathbf{x}) \succeq \mathbf{0} \quad \forall \mathbf{x} \in C.$$

Non-negative curvature in all directions at every point. For univariate f : reduces to $f''(x) \geq 0$.

The Cornerstone Theorem of Optimization

Theorem: Global Optimality for Convex Functions

Let $f: C \rightarrow \mathbb{R}$ be convex on a convex set C . If $\mathbf{x}^* \in C$ is a **local minimum** of f , then \mathbf{x}^* is also the **global minimum**.

The Cornerstone Theorem of Optimization

Theorem: Global Optimality for Convex Functions

Let $f: C \rightarrow \mathbb{R}$ be convex on a convex set C . If $\mathbf{x}^* \in C$ is a **local minimum** of f , then \mathbf{x}^* is also the **global minimum**.

Proof Sketch.

Suppose \mathbf{x}^* is local but not global, so $\exists \mathbf{y}$ with $f(\mathbf{y}) < f(\mathbf{x}^*)$.

By convexity, for any $\lambda \in (0, 1]$: $\mathbf{z}_\lambda = \lambda \mathbf{y} + (1 - \lambda) \mathbf{x}^* \in C$ and

$$f(\mathbf{z}_\lambda) \leq \lambda f(\mathbf{y}) + (1 - \lambda) f(\mathbf{x}^*) < f(\mathbf{x}^*).$$

Choosing λ small enough, \mathbf{z}_λ falls within the local neighborhood of \mathbf{x}^* yet has lower loss—**contradiction**. □

The Cornerstone Theorem of Optimization

Theorem: Global Optimality for Convex Functions

Let $f: C \rightarrow \mathbb{R}$ be convex on a convex set C . If $\mathbf{x}^* \in C$ is a **local minimum** of f , then \mathbf{x}^* is also the **global minimum**.

Proof Sketch.

Suppose \mathbf{x}^* is local but not global, so $\exists \mathbf{y}$ with $f(\mathbf{y}) < f(\mathbf{x}^*)$.

By convexity, for any $\lambda \in (0, 1]$: $\mathbf{z}_\lambda = \lambda \mathbf{y} + (1 - \lambda) \mathbf{x}^* \in C$ and

$$f(\mathbf{z}_\lambda) \leq \lambda f(\mathbf{y}) + (1 - \lambda) f(\mathbf{x}^*) < f(\mathbf{x}^*).$$

Choosing λ small enough, \mathbf{z}_λ falls within the local neighborhood of \mathbf{x}^* yet has lower loss—**contradiction**. □

Consequence

For a convex loss function, there are *no misleading local minima*. Gradient descent can only get stuck at the global optimum.

Convexity of the OLS Loss

Theorem: Convexity of OLS Loss

$L(\beta) = \frac{1}{n} \|\mathbf{X}\beta - \mathbf{y}\|_2^2$ is convex (strictly convex when $\mathbf{X}^T \mathbf{X}$ is invertible).

Proof via Second-Order Condition.

The Hessian is constant:

$$\mathbf{H}_L(\beta) = \frac{2}{n} \mathbf{X}^T \mathbf{X}.$$

For any $\mathbf{z} \in \mathbb{R}^p$:

$$\mathbf{z}^T \mathbf{H}_L \mathbf{z} = \frac{2}{n} \mathbf{z}^T \mathbf{X}^T \mathbf{X} \mathbf{z} = \frac{2}{n} \|\mathbf{X}\mathbf{z}\|_2^2 \geq 0.$$

So $\mathbf{H}_L \succeq \mathbf{0}$ everywhere $\Rightarrow L$ is convex. When \mathbf{X} has full column rank, $\|\mathbf{X}\mathbf{z}\|_2^2 > 0$ for all $\mathbf{z} \neq \mathbf{0}$
 $\Rightarrow \mathbf{H}_L \succ \mathbf{0} \Rightarrow$ strictly convex with a *unique* global minimum. □ □

The Computational Bottleneck

For a general loss $L(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n L_i(\boldsymbol{\theta})$, the batch gradient is:

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

One parameter update = one full pass over all n data points.

The Computational Bottleneck

For a general loss $L(\theta) = \frac{1}{n} \sum_{i=1}^n L_i(\theta)$, the batch gradient is:

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

One parameter update = one full pass over all n data points.

The Insight: Data Redundancy

Large datasets are **highly redundant**. The gradient from one sample is very similar to the gradient from another similar sample. Batch GD wastefully re-computes and averages nearly identical gradient information.

Solution: Replace the expensive exact gradient with a cheap, noisy, but *unbiased* estimate—**stochastic approximation**.

Epochs and Update Steps

Definition: Update Step / Iteration

One step of $\theta_{k+1} = f_k(\theta_k)$ applying the update rule once.

Definition: Epoch

One complete sequential pass through all n training samples. Data is reshuffled between epochs.

Epochs and Update Steps

Definition: Update Step / Iteration

One step of $\theta_{k+1} = f_k(\theta_k)$ applying the update rule once.

Definition: Epoch

One complete sequential pass through all n training samples. Data is reshuffled between epochs.

Method	Samples per update	Updates per epoch
Batch GD	n (all)	1
SGD	1	n
Mini-Batch GD	b	n/b

Algorithm: Stochastic Gradient Descent (SGD)

SGD

Input: θ_0 , learning rate schedule $\{\eta_k\}$, epochs T

For $t = 1, \dots, T$:

- ① Randomly shuffle $\{1, \dots, n\}$ into permutation π
- ② **For** $j = 1, \dots, n$: pick $i = \pi(j)$, then

$$\theta_{k+1} \leftarrow \theta_k - \eta_k \nabla L_i(\theta_k)$$

Return θ_k

Algorithm: Stochastic Gradient Descent (SGD)

SGD

Input: θ_0 , learning rate schedule $\{\eta_k\}$, epochs T

For $t = 1, \dots, T$:

- ① Randomly shuffle $\{1, \dots, n\}$ into permutation π
- ② **For** $j = 1, \dots, n$: pick $i = \pi(j)$, then

$$\theta_{k+1} \leftarrow \theta_k - \eta_k \nabla L_i(\theta_k)$$

Return θ_k

Lemma: Unbiasedness of the Stochastic Gradient

The stochastic gradient is an unbiased estimator of the true gradient:

$$\mathbb{E}_i[\nabla L_i(\theta)] = \frac{1}{n} \sum^n \nabla L_j(\theta) = \nabla L(\theta).$$

Example: SGD Convergence Trace (Epoch 1)

Same dataset: $y = \beta x$, $n = 4$, $\beta^* = 1.9$, $\eta = 0.02$, $\beta_0 = 0$. Per-sample gradient:
 $\nabla L_i(\beta) = 2x_i(\beta x_i - y_i)$.

Epoch 1, shuffled order $i = 2, 4, 1, 3$:

Step k	Sample	$\nabla L_i(\beta_k)$	β_{k+1}
0	$i = 2$ ($x = 2$)	-16.000	0.320
1	$i = 4$ ($x = 4$)	-53.760	1.395
2	$i = 1$ ($x = 1$)	-1.210	1.419
3	$i = 3$ ($x = 3$)	-4.450	1.508

Example: SGD Convergence Trace (Epoch 1)

Same dataset: $y = \beta x$, $n = 4$, $\beta^* = 1.9$, $\eta = 0.02$, $\beta_0 = 0$. Per-sample gradient:
 $\nabla L_i(\beta) = 2x_i(\beta x_i - y_i)$.

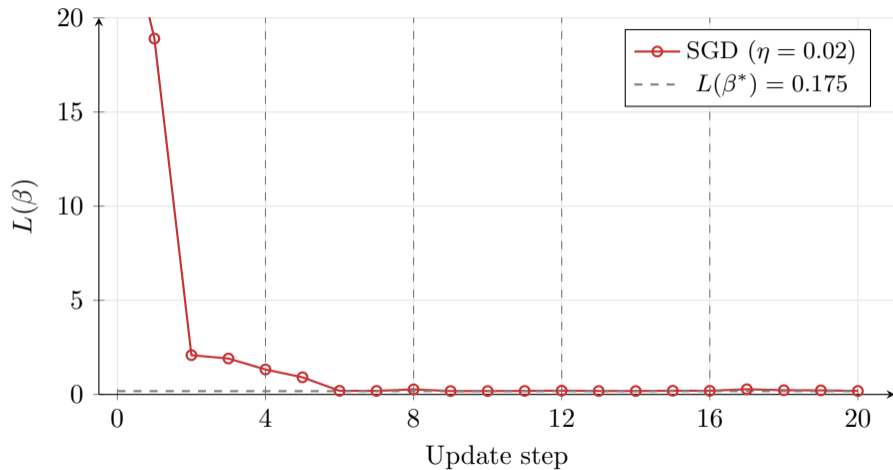
Epoch 1, shuffled order $i = 2, 4, 1, 3$:

Step k	Sample	$\nabla L_i(\beta_k)$	β_{k+1}
0	$i = 2$ ($x = 2$)	-16.000	0.320
1	$i = 4$ ($x = 4$)	-53.760	1.395
2	$i = 1$ ($x = 1$)	-1.210	1.419
3	$i = 3$ ($x = 3$)	-4.450	1.508

After just **one epoch**, SGD reaches $\beta \approx 1.508$ (79% of the way to β^*).

Steps vary wildly: sample 4 moved β by 1.075, sample 1 by only 0.024. This **high variance** is the signature weakness of SGD.

SGD: Noisy but Fast Convergence



Properties of SGD

- **Erratic trajectory:** High gradient variance causes oscillation around the descent path; the algorithm never truly settles at the optimum under a constant η .
- **Learning rate schedule:** Convergence requires a decaying schedule satisfying

$$\sum_{k=1}^{\infty} \eta_k = \infty \quad \text{and} \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty.$$

Common choices: $\eta_k \propto 1/k$ or $\eta_k \propto 1/\sqrt{k}$.

- **Escaping local minima:** In non-convex landscapes (e.g., neural networks), SGD noise can help the algorithm escape shallow local minima—a critical advantage over batch GD.

Algorithm: Mini-Batch Gradient Descent

Mini-Batch GD

Input: θ_0 , learning rate η , mini-batch size b , epochs T

For $t = 1, \dots, T$: shuffle and partition $\{1, \dots, n\}$ into batches $\mathcal{B}_1, \dots, \mathcal{B}_{\lceil n/b \rceil}$

For each mini-batch \mathcal{B}_m :

$$\mathbf{g}_k = \frac{1}{b} \sum_{i \in \mathcal{B}_m} \nabla L_i(\theta_k), \quad \theta_{k+1} \leftarrow \theta_k - \eta \mathbf{g}_k$$

Algorithm: Mini-Batch Gradient Descent

Mini-Batch GD

Input: θ_0 , learning rate η , mini-batch size b , epochs T

For $t = 1, \dots, T$: shuffle and partition $\{1, \dots, n\}$ into batches $\mathcal{B}_1, \dots, \mathcal{B}_{\lceil n/b \rceil}$

For each mini-batch \mathcal{B}_m :

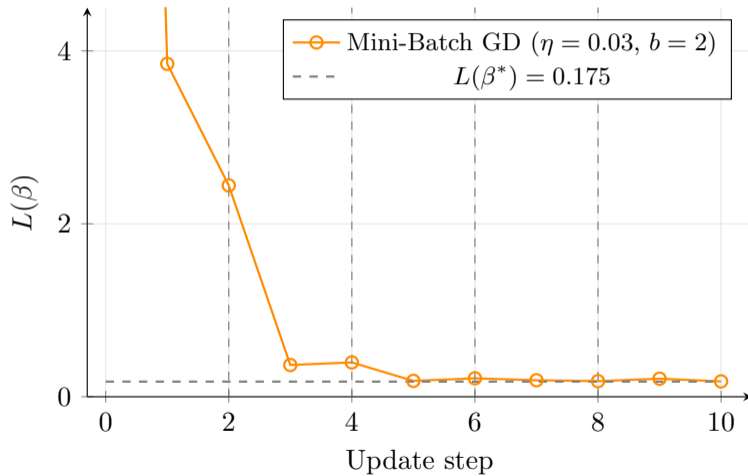
$$\mathbf{g}_k = \frac{1}{b} \sum_{i \in \mathcal{B}_m} \nabla L_i(\theta_k), \quad \theta_{k+1} \leftarrow \theta_k - \eta \mathbf{g}_k$$

Interpolation Between Extremes

- $b = 1$: recovers SGD (maximum variance, minimum cost)
- $b = n$: recovers batch GD (zero variance, maximum cost)
- $b \in \{32, 64, 128\}$: the practical sweet spot

Variance reduction: $\text{Var}(\mathbf{g}_k) = \frac{1}{b} \text{Var}(\nabla L_i(\theta))$ —averaging over b samples reduces variance by factor b .

Mini-Batch GD: Smoother than SGD, Faster than Batch



Why Mini-Batch GD Dominates in Practice

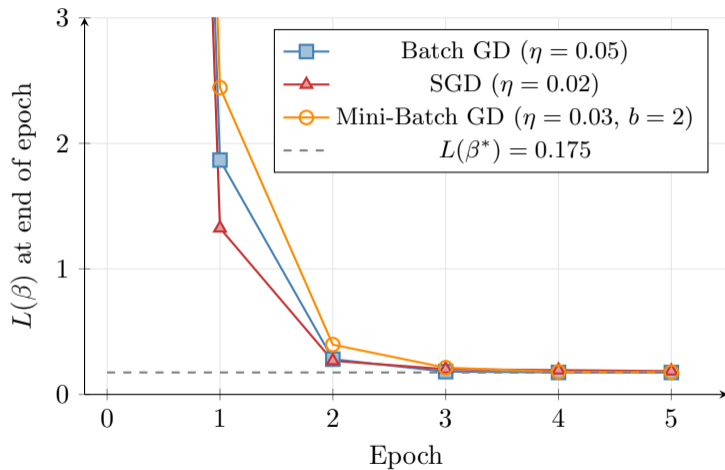
- 1 **Variance reduction:** Averaging over b samples gives a gradient estimate b times less noisy than SGD, enabling a larger stable learning rate.
- 2 **Frequent updates:** n/b updates per epoch instead of 1—much faster early progress than batch GD.
- 3 **GPU parallelism:** Modern GPUs have thousands of cores optimized for SIMD operations. A mini-batch of size $b = 32$ or 64 takes roughly the *same wall-clock time* as a single sample, since all b forward/backward passes run in parallel.
 - SGD: cannot saturate GPU parallelism (1 sample at a time)
 - Batch GD: often exceeds GPU memory
 - Mini-batch: the optimal fit for GPU hardware

Why Mini-Batch GD Dominates in Practice

- 1 **Variance reduction:** Averaging over b samples gives a gradient estimate b times less noisy than SGD, enabling a larger stable learning rate.
- 2 **Frequent updates:** n/b updates per epoch instead of 1—much faster early progress than batch GD.
- 3 **GPU parallelism:** Modern GPUs have thousands of cores optimized for SIMD operations. A mini-batch of size $b = 32$ or 64 takes roughly the *same wall-clock time* as a single sample, since all b forward/backward passes run in parallel.
 - SGD: cannot saturate GPU parallelism (1 sample at a time)
 - Batch GD: often exceeds GPU memory
 - Mini-batch: the optimal fit for GPU hardware

The co-evolution of mini-batch methods and GPU computing is central to the rise of deep learning.

Comparing the Three Variants



Summary: Gradient Descent Variants

Property	Batch GD	SGD	Mini-Batch GD
Gradient uses	All n samples	1 random sample	b random samples
Updates per epoch	1	n	n/b
Cost per update	$O(np)$	$O(p)$	$O(bp)$
Memory	High	Low	Medium
Convergence path	Smooth (convex)	Noisy, erratic	Smoother than SGD
GPU utilization	Poor (memory)	Poor (no vectorization)	Excellent
Key advantage	Convergence guarantee	Speed; escapes local minima	Efficiency–stability balance

Key Takeaways

- 1 **Training as optimization:** Learning means solving $\theta^* = \arg \min L(\theta)$. The gradient and Hessian describe the local geometry of the loss landscape.
- 2 **Gradient descent:** Derived from the first-order Taylor expansion, the update rule $\theta_{k+1} = \theta_k - \eta \nabla L(\theta_k)$ guarantees local descent. The learning rate η is the critical hyperparameter.
- 3 **Convexity guarantees global convergence:** Any local minimum of a convex function is a global minimum. The OLS loss is convex (Hessian $= \frac{2}{n} \mathbf{X}^T \mathbf{X} \succeq \mathbf{0}$), guaranteeing convergence.
- 4 **Stochastic approximation:** The stochastic gradient is an unbiased estimator of the true gradient. SGD and mini-batch GD trade gradient exactness for computational efficiency.
- 5 **Mini-batch GD is the practical standard:** It balances variance reduction, update frequency, and GPU parallelism—the reason deep learning is computationally feasible.