Gradient Descent: The Foundation of Machine Learning

From Taylor Series to Modern Deep Learning

Nipun Batra and the teaching staff

IIT Gandhinagar

August 28, 2025

Table of Contents

Introduction: Why Optimization Matters

The Core Machine Learning Challenge

Key Points:

Central Problem: Find the best parameters $\boldsymbol{\theta}^*$ for our model

The Core Machine Learning Challenge

Key Points:

Central Problem: Find the best parameters $\boldsymbol{\theta}^*$ for our model

Examples everywhere in ML:

- **Linear regression:** $\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$

The Core Machine Learning Challenge

Key Points:

Central Problem: Find the best parameters $\boldsymbol{\theta}^*$ for our model

Examples everywhere in ML:

- **Linear regression:** $\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$
- **Logistic regression:** $\min_{\boldsymbol{\theta}} -\sum \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$

The Core Machine Learning Challenge

Key Points:

Central Problem: Find the best parameters $\boldsymbol{\theta}^*$ for our model

Examples everywhere in ML:

- **Linear regression:** $\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$
- **Logistic regression:** $\min_{\boldsymbol{\theta}} -\sum \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$
- **Neural networks:** $\min_{\boldsymbol{\theta}} \sum \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$

The Core Machine Learning Challenge

Key Points:

Central Problem: Find the best parameters $\boldsymbol{\theta}^*$ for our model

Examples everywhere in ML:

- **Linear regression:** $\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$
- **Logistic regression:** $\min_{\boldsymbol{\theta}} -\sum \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$
- **Neural networks:** $\min_{\boldsymbol{\theta}} \sum \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$

The Core Machine Learning Challenge

Key Points:

Central Problem: Find the best parameters $\boldsymbol{\theta}^*$ for our model

Examples everywhere in ML:

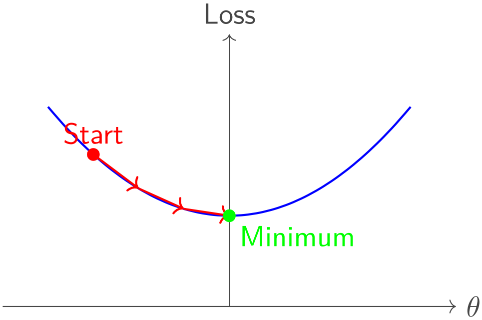
- **Linear regression:** $\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$
- **Logistic regression:** $\min_{\boldsymbol{\theta}} -\sum \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta})$
- **Neural networks:** $\min_{\boldsymbol{\theta}} \sum \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$

Important: The Challenge

Most ML problems have **no closed-form solution!**

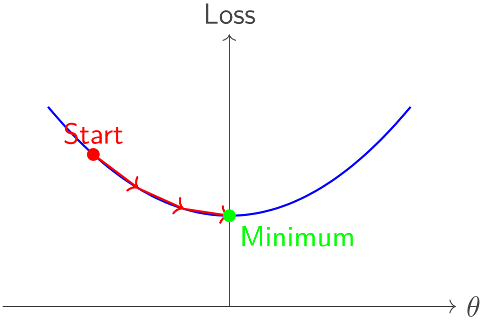
Enter: Iterative Optimization

Since we can't solve directly, we use iterative methods:



Enter: Iterative Optimization

Since we can't solve directly, we use iterative methods:



Definition: Gradient Descent

The workhorse algorithm that powers modern machine learning

Intuition: Following the Steepest Path

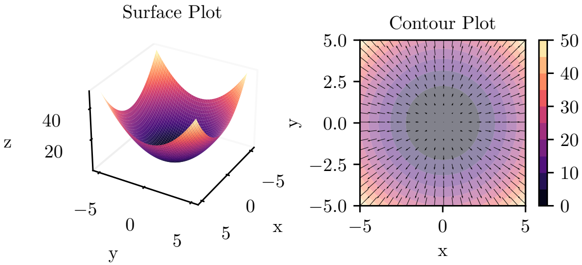
The Mountain Climbing Analogy

Imagine: You're lost in fog and want to reach the valley

The Mountain Climbing Analogy

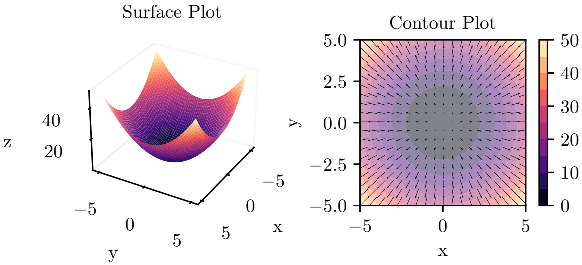
Imagine: You're lost in fog and want to reach the valley

Mathematical Definition of Gradient



For function $f(x,y) = x^2 + y^2$**:**

Mathematical Definition of Gradient

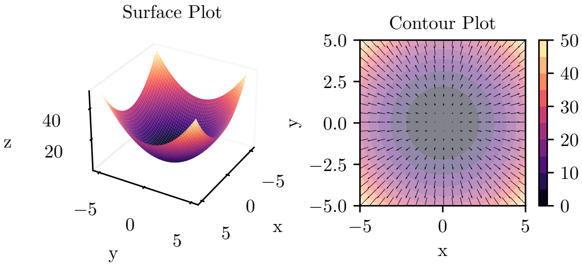


For function $f(x, y) = x^2 + y^2$**:**

Definition: Gradient

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Mathematical Definition of Gradient



For function $f(x, y) = x^2 + y^2$**:**

Definition: Gradient

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Key Points:

Mathematical Foundation: Taylor Series

Why Taylor Series?

Example: The Core Idea

If we can't solve $\min f(\mathbf{x})$ exactly, let's approximate $f(\mathbf{x})$ locally!

Why Taylor Series?

Example: The Core Idea

If we can't solve $\min f(\mathbf{x})$ exactly, let's approximate $f(\mathbf{x})$ locally!

Taylor series expansion around point x_0:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \tag{1}$$

Why Taylor Series?

Example: The Core Idea

If we can't solve $\min f(\mathbf{x})$ exactly, let's approximate $f(\mathbf{x})$ locally!

Taylor series expansion around point $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \tag{1}$$

Different orders of approximation:

- **0th order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0)$ (constant)

Why Taylor Series?

Example: The Core Idea

If we can't solve $\min f(\mathbf{x})$ exactly, let's approximate $f(\mathbf{x})$ locally!

Taylor series expansion around point $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \tag{1}$$

Different orders of approximation:

- **0th order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0)$ (constant)
- **1st order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ (linear)

Why Taylor Series?

Example: The Core Idea

If we can't solve $\min f(\mathbf{x})$ exactly, let's approximate $f(\mathbf{x})$ locally!

Taylor series expansion around point $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \ldots \tag{1}$$

Different orders of approximation:

- **0th order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0)$ (constant)
- **1st order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ (linear)
- **2nd order:** Includes curvature via Hessian

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$
- $f'(0) = -\sin(0) = 0$

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$
- $f'(0) = -\sin(0) = 0$
- $f''(0) = -\cos(0) = -1$

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$
- $f'(0) = -\sin(0) = 0$
- $f''(0) = -\cos(0) = -1$
- $f'''(0) = \sin(0) = 0$

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$
- $f'(0) = -\sin(0) = 0$
- $f''(0) = -\cos(0) = -1$
- $f'''(0) = \sin(0) = 0$
- $f^{(4)}(0) = \cos(0) = 1$

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$
- $f'(0) = -\sin(0) = 0$
- $f''(0) = -\cos(0) = -1$
- $f'''(0) = \sin(0) = 0$
- $f^{(4)}(0) = \cos(0) = 1$

Taylor Series: Concrete Example

Let's approximate $f(x) = \cos(x)$ **around** $x_0 = 0$:

- $f(0) = \cos(0) = 1$
- $f'(0) = -\sin(0) = 0$
- $f''(0) = -\cos(0) = -1$
- $f'''(0) = \sin(0) = 0$
- $f^{(4)}(0) = \cos(0) = 1$

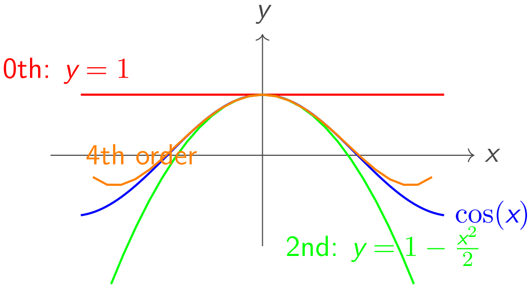
Taylor approximations:

$$\text{0th:} \quad f(x) \approx 1 \tag{2}$$

$$\text{2nd:} \quad f(x) \approx 1 - \frac{x^2}{2} \tag{3}$$

$$\text{4th:} \quad f(x) \approx 1 - \frac{x^2}{2} + \frac{x^4}{24} \tag{4}$$

Visual: Taylor Approximations



Key Points: H

igher-order = better approximation, but 1st-order is often sufficient!

Derivation: From Taylor Series to Gradient Descent

The Key Question

Goal: Find $\Delta\mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta\mathbf{x}) < f(\mathbf{x}_0)$

The Key Question

Goal: Find $\Delta\mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta\mathbf{x}) < f(\mathbf{x}_0)$
Using 1st-order Taylor approximation:

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} \tag{5}$$

The Key Question

Goal: Find $\Delta\mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta\mathbf{x}) < f(\mathbf{x}_0)$
Using 1st-order Taylor approximation:

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} \tag{5}$$

For the function to decrease:

$$\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$$

The Key Question

Goal: Find $\Delta\mathbf{x}$ such that $f(\mathbf{x}_0 + \Delta\mathbf{x}) < f(\mathbf{x}_0)$
Using 1st-order Taylor approximation:

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} \tag{5}$$

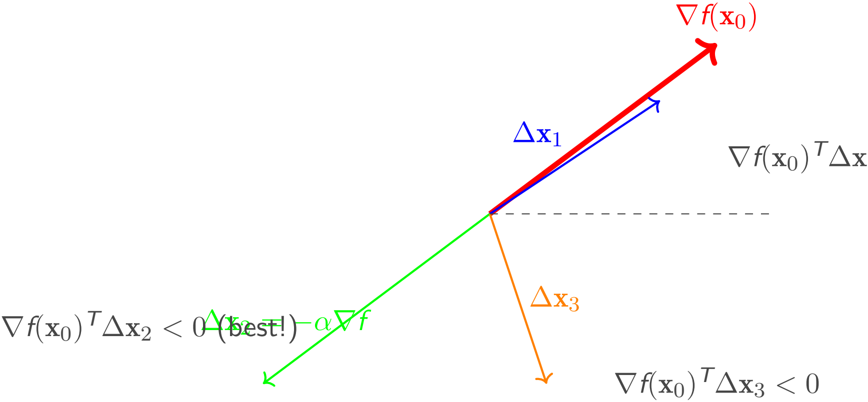
For the function to decrease:

$$\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$$

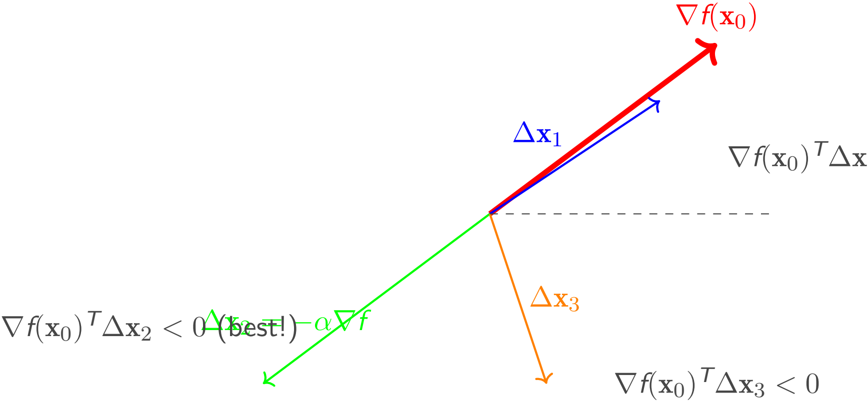
Important: Vector Geometry Reminder

For vectors $\mathbf{a}, \mathbf{b}$: $\mathbf{a}^T\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos(\theta)$
Most negative when: $\cos(\theta) = -1$ (opposite directions!)

Visual Derivation with TikZ



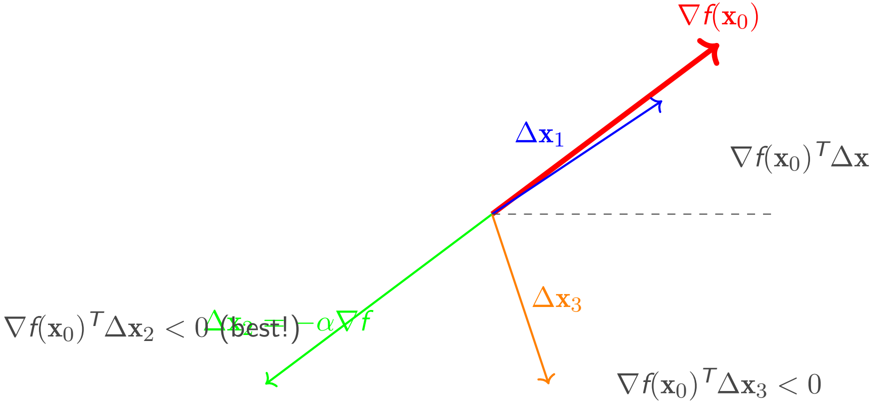
Visual Derivation with TikZ



Definition: Optimal Choice

$$\Delta \mathbf{x} = -\alpha \nabla f(\mathbf{x}_0), \quad \alpha > 0$$

Visual Derivation with TikZ



Definition: Optimal Choice

$$\Delta \mathbf{x} = -\alpha \nabla f(\mathbf{x}_0), \quad \alpha > 0$$

Pop Quiz #1: Understanding the Derivation

Answer this!

Consider $f(x) = x^2 + 2$ at point $x_0 = 2$.
Questions:

1. What is $f(x_0)$ and $f'(x_0)$?
2. Write the 1st-order Taylor approximation
3. If we take step $\Delta x = -0.1 \cdot f'(x_0)$, what is our new x?
4. Will the function value decrease?

The Gradient Descent Algorithm

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point θ_0
2. **Repeat until convergence:**

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$
2. **Repeat until convergence:**
   - Compute gradient: $\mathbf{g}_t = \nabla f(\boldsymbol{\theta}_t)$

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$
2. **Repeat until convergence:**
   - Compute gradient: $\mathbf{g}_t = \nabla f(\boldsymbol{\theta}_t)$
   - Update parameters: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \mathbf{g}_t$

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$
2. **Repeat until convergence:**
   - Compute gradient: $\mathbf{g}_t = \nabla f(\boldsymbol{\theta}_t)$
   - Update parameters: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \mathbf{g}_t$
   - Check stopping criterion

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$
2. **Repeat until convergence:**
   - Compute gradient: $\mathbf{g}_t = \nabla f(\boldsymbol{\theta}_t)$
   - Update parameters: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \mathbf{g}_t$
   - Check stopping criterion

The Complete Algorithm

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$
2. **Repeat until convergence:**
   - Compute gradient: $\mathbf{g}_t = \nabla f(\boldsymbol{\theta}_t)$
   - Update parameters: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \mathbf{g}_t$
   - Check stopping criterion

Key hyperparameter: Learning rate α

Learning Rate: The Step Size

The learning rate α controls how big steps we take:

- **Too small α:** Slow convergence

Learning Rate: The Step Size

The learning rate α controls how big steps we take:

- **Too small α:** Slow convergence
- **Good α:** Fast, stable convergence

Learning Rate: The Step Size

The learning rate α controls how big steps we take:

- **Too small α:** Slow convergence
- **Good α:** Fast, stable convergence
- **Too large α:** Overshooting, instability

Learning Rate: The Step Size

The learning rate α controls how big steps we take:

- **Too small α:** Slow convergence
- **Good α:** Fast, stable convergence
- **Too large α:** Overshooting, instability
- **Way too large α:** Divergence!

Learning Rate: The Step Size

The learning rate α controls how big steps we take:

- **Too small α:** Slow convergence
- **Good α:** Fast, stable convergence
- **Too large α:** Overshooting, instability
- **Way too large α:** Divergence!

Learning Rate: The Step Size

The learning rate α controls how big steps we take:

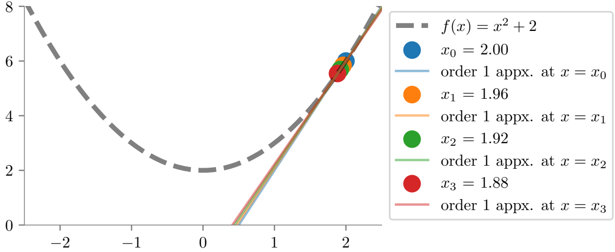
- **Too small α:** Slow convergence
- **Good α:** Fast, stable convergence
- **Too large α:** Overshooting, instability
- **Way too large α:** Divergence!

Key Points: L

earning rate selection is crucial for success!

Learning Rate Visualization: Too Small

$\alpha = 0.01$ - **Slow but steady**

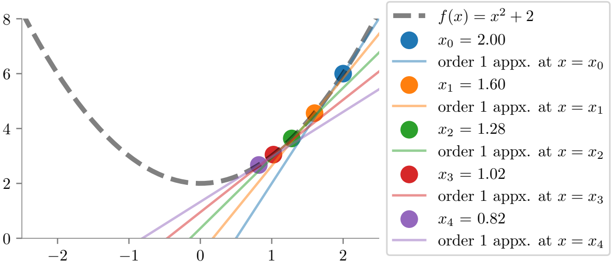


Important: Issue

Many iterations needed $\rightarrow$ Computationally expensive

Learning Rate Visualization: Just Right

$\alpha = 0.1$ - **The sweet spot**

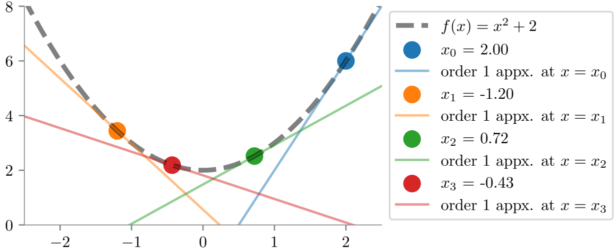


Key Points: P

erfect balance: Fast convergence + Stability

Learning Rate Visualization: Too Large

$\alpha = 0.8$ - **Getting risky**

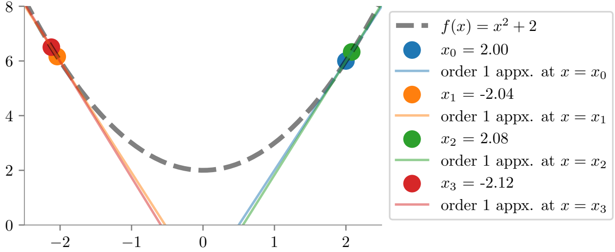


Important: Warning

Fast but oscillatory - watch for instability!

Learning Rate Visualization: Disaster

$\alpha = 1.01$ - **Complete failure**



Important: Disaster Zone

Function values explode! Always monitor your loss curves.

Application: Linear Regression

Our First Real Example

Problem: Learn $y = \theta_0 + \theta_1 x$ from data

x	y
1	1
2	2
3	3

Our First Real Example

Problem: Learn $y = \theta_0 + \theta_1 x$ from data

x	y
1	1
2	2
3	3

Cost function (Mean Squared Error):

$$\mathrm{MSE}(\theta_0, \theta_1) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)^2$$

Computing the Gradients

We need: $\nabla \mathrm{MSE} = \begin{bmatrix} \frac{\partial \mathrm{MSE}}{\partial \theta_0} \\ \frac{\partial \mathrm{MSE}}{\partial \theta_1} \end{bmatrix}$

Computing the Gradients

We need: $\nabla \mathrm{MSE} = \begin{bmatrix} \frac{\partial \mathrm{MSE}}{\partial \theta_0} \\ \frac{\partial \mathrm{MSE}}{\partial \theta_1} \end{bmatrix}$

Partial derivatives:

$$\frac{\partial \mathrm{MSE}}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)(-1) \tag{6}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} \epsilon_i \tag{7}$$

Computing the Gradients

We need: $\nabla \mathrm{MSE} = \begin{bmatrix} \frac{\partial \mathrm{MSE}}{\partial \theta_0} \\ \frac{\partial \mathrm{MSE}}{\partial \theta_1} \end{bmatrix}$

Partial derivatives:

$$\frac{\partial \mathrm{MSE}}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)(-1) \tag{6}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} \epsilon_i \tag{7}$$

$$\frac{\partial \mathrm{MSE}}{\partial \theta_1} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)(-x_i) \tag{8}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} \epsilon_i x_i \tag{9}$$

where $\epsilon_i = y_i - \hat{y}_i$ is the residual.

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = 4 + 0 \cdot 2 = 4$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = 4 + 0 \cdot 3 = 4$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = 4 + 0 \cdot 3 = 4$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = 4 + 0 \cdot 3 = 4$

Compute errors:

- $\epsilon_1 = 1 - 4 = -3$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = 4 + 0 \cdot 3 = 4$

Compute errors:

- $\epsilon_1 = 1 - 4 = -3$
- $\epsilon_2 = 2 - 4 = -2$

Step-by-Step Example: Setup

Initial values: $\theta_0 = 4, \theta_1 = 0$
Learning rate: $\alpha = 0.1$
Iteration 1 - Compute predictions:

- $\hat{y}_1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = 4 + 0 \cdot 3 = 4$

Compute errors:

- $\epsilon_1 = 1 - 4 = -3$
- $\epsilon_2 = 2 - 4 = -2$
- $\epsilon_3 = 3 - 4 = -1$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial\, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial\, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$

Parameter updates:

- $\theta_0^{(1)} = 4 - 0.1 \times 4 = 3.6$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$

Parameter updates:

- $\theta_0^{(1)} = 4 - 0.1 \times 4 = 3.6$
- $\theta_1^{(1)} = 0 - 0.1 \times 6.67 = -0.67$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3-2-1) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3\cdot 1-2\cdot 2-1\cdot 3) = 6.67$

Parameter updates:

- $\theta_0^{(1)} = 4 - 0.1 \times 4 = 3.6$
- $\theta_1^{(1)} = 0 - 0.1 \times 6.67 = -0.67$

Step-by-Step Example: Updates

Compute gradients:

- $\frac{\partial \,\mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \,\mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$

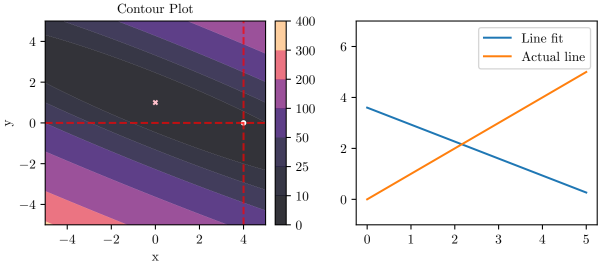
Parameter updates:

- $\theta_0^{(1)} = 4 - 0.1 \times 4 = 3.6$
- $\theta_1^{(1)} = 0 - 0.1 \times 6.67 = -0.67$

Key Points: N

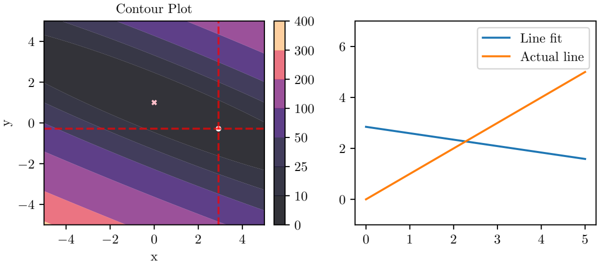
ew parameters: $(\theta_0, \theta_1) = (3.6, -0.67)$
We moved closer to the true solution $(0, 1)$!

Visual Journey: GD in Action



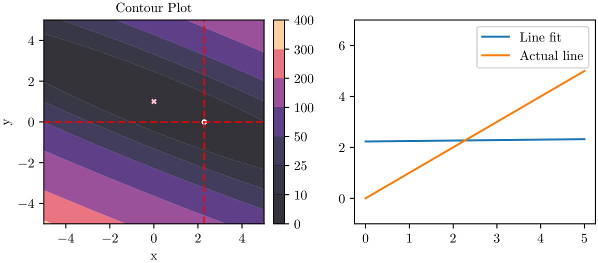
Notice: Steps get smaller as we approach the minimum!

Visual Journey: GD in Action



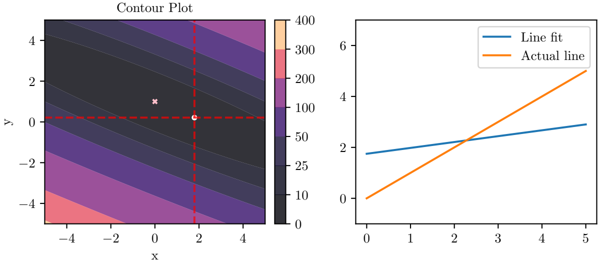
Notice: Steps get smaller as we approach the minimum!

Visual Journey: GD in Action



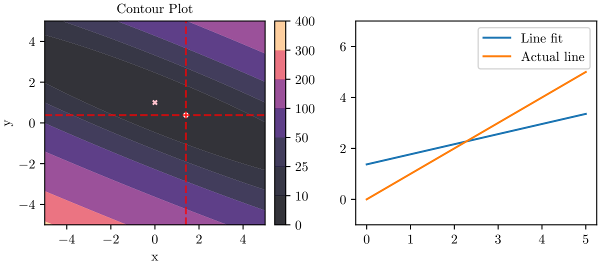
Notice: Steps get smaller as we approach the minimum!

Visual Journey: GD in Action



Notice: Steps get smaller as we approach the minimum!

Visual Journey: GD in Action



Notice: Steps get smaller as we approach the minimum!

Variants: Batch vs Stochastic vs Mini-batch

The Gradient Descent Family

Three variants based on data usage per update:

Definition: Batch Gradient Descent

Use ALL training data for each gradient computation

Definition: Stochastic Gradient Descent (SGD)

Use ONE sample for each gradient computation

Definition: Mini-batch Gradient Descent

Use a SMALL BATCH of samples for each gradient computation

Comparison: The Trade-offs

Method	**Data/update**	**Updates/epoch**	**Convergence**	**Memory**
Batch GD	n (all)	1	Smooth	High
SGD	1	n	Noisy	Low
Mini-batch	b (batch)	n/b	Balanced	Medium

Comparison: The Trade-offs

Method	Data/update	Updates/epoch	Convergence	Memory
Batch GD	n (all)	1	Smooth	High
SGD	1	n	Noisy	Low
Mini-batch	b (batch)	n/b	Balanced	Medium

Key Points:

Modern ML Standard: Mini-batch GD with batch sizes 32-256

- Good balance of stability and efficiency
- Enables GPU parallelization
- Better gradient estimates than pure SGD

Epochs vs Iterations

Definition: Iteration

One parameter update step

Definition: Epoch

One complete pass through the entire training dataset

Epochs vs Iterations

Definition: Iteration

One parameter update step

Definition: Epoch

One complete pass through the entire training dataset

For dataset with 1000 samples:

- **Batch GD:** 1 iteration = 1 epoch

Epochs vs Iterations

Definition: Iteration

One parameter update step

Definition: Epoch

One complete pass through the entire training dataset

For dataset with 1000 samples:

- **Batch GD:** 1 iteration = 1 epoch
- **SGD:** 1000 iterations = 1 epoch

Epochs vs Iterations

Definition: Iteration

One parameter update step

Definition: Epoch

One complete pass through the entire training dataset

For dataset with 1000 samples:

- **Batch GD:** 1 iteration = 1 epoch
- **SGD:** 1000 iterations = 1 epoch
- **Mini-batch (size 100):** 10 iterations = 1 epoch

Epochs vs Iterations

Definition: Iteration

One parameter update step

Definition: Epoch

One complete pass through the entire training dataset

For dataset with 1000 samples:

- **Batch GD:** 1 iteration = 1 epoch
- **SGD:** 1000 iterations = 1 epoch
- **Mini-batch (size 100):** 10 iterations = 1 epoch

Epochs vs Iterations

Definition: Iteration

One parameter update step

Definition: Epoch

One complete pass through the entire training dataset

For dataset with 1000 samples:

- **Batch GD:** 1 iteration = 1 epoch
- **SGD:** 1000 iterations = 1 epoch
- **Mini-batch (size 100):** 10 iterations = 1 epoch

Important: Important

Always specify which metric when discussing convergence

Mathematical Properties

SGD as Unbiased Estimator

True gradient: $\nabla L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$

SGD as Unbiased Estimator

True gradient: $\nabla L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$
SGD estimate: $\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)$ where $(\mathbf{x}, y)$ is randomly sampled.

SGD as Unbiased Estimator

True gradient: $\nabla L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$
SGD estimate: $\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)$ where $(\mathbf{x}, y)$ is randomly sampled.

Theorem: Unbiased Property

$$\mathbb{E}[\nabla \tilde{L}(\boldsymbol{\theta})] = \nabla L(\boldsymbol{\theta})$$

SGD as Unbiased Estimator

True gradient: $\nabla L(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i)$
SGD estimate: $\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)$ where $(\mathbf{x}, y)$ is randomly sampled.

Theorem: Unbiased Property

$$\mathbb{E}[\nabla \tilde{L}(\boldsymbol{\theta})] = \nabla L(\boldsymbol{\theta})$$

Proof:

$$\begin{aligned}\mathbb{E}[\nabla \tilde{L}(\boldsymbol{\theta})] &= \mathbb{E}[\nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)] \\ &= \sum_{i=1}^{n} \frac{1}{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) = \nabla L(\boldsymbol{\theta})\end{aligned}$$

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"
- But they average to the correct direction

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"
- But they average to the correct direction
- Noise can help escape local minima

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"
- But they average to the correct direction
- Noise can help escape local minima
- Theoretical justification for SGD's success

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"
- But they average to the correct direction
- Noise can help escape local minima
- Theoretical justification for SGD's success

Why Unbiasedness Matters

Key Points:

Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"
- But they average to the correct direction
- Noise can help escape local minima
- Theoretical justification for SGD's success

Example: Intuition

Like asking random people for directions:

- Each answer might be slightly off
- But with no systematic bias, the average is correct

Advanced SGD Theory

For detailed mathematical analysis, see:

Computational Complexity

GD vs Normal Equation: When to Use What?

For linear regression, we have two options:

Important: Normal Equation

$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
Time: $\mathcal{O}(d^2 n + d^3)$
Space: $\mathcal{O}(d^2)$

GD vs Normal Equation: When to Use What?

For linear regression, we have two options:

Important: Normal Equation

$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
Time: $\mathcal{O}(d^2 n + d^3)$
Space: $\mathcal{O}(d^2)$

Key Points: Gradient Descent

$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha\mathbf{X}^T(\mathbf{X}\boldsymbol{\theta}_t - \mathbf{y})$
Time: $\mathcal{O}(T \cdot nd)$ for T iterations
Space: $\mathcal{O}(nd)$

When to Choose Which Method

Scenario	**Normal Eq.**	**Gradient Desc.**
Few features ($d < 1000$)	Yes	Yes
Many features ($d > 10000$)	No	Yes
Non-linear models	No	Yes
Large datasets	No	Yes
Need exact solution	Yes	No
Online learning	No	Yes

When to Choose Which Method

Scenario	**Normal Eq.**	**Gradient Desc.**
Few features ($d < 1000$)	Yes	Yes
Many features ($d > 10000$)	No	Yes
Non-linear models	No	Yes
Large datasets	No	Yes
Need exact solution	Yes	No
Online learning	No	Yes

Key Points:

Modern ML: Gradient descent dominates due to:

- High-dimensional problems (d very large)
- Non-linear models (neural networks)
- Large datasets (n very large)

Pop Quiz #2: Complexity Analysis

Answer this!

Dataset: $n = 10^6$ samples, $d = 10^3$ features
Questions:

1. Normal equation complexity?
2. GD complexity for 100 iterations?
3. Which would you choose?
4. What if $d = 10^6$?

Modern Extensions

Beyond Basic Gradient Descent

Modern optimizers address GD limitations:

- **Momentum:** Accelerates in consistent directions

Beyond Basic Gradient Descent

Modern optimizers address GD limitations:

- **Momentum:** Accelerates in consistent directions
- **AdaGrad:** Adaptive per-parameter learning rates

Beyond Basic Gradient Descent

Modern optimizers address GD limitations:

- **Momentum:** Accelerates in consistent directions
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates

Beyond Basic Gradient Descent

Modern optimizers address GD limitations:

- **Momentum:** Accelerates in consistent directions
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Handles non-stationary objectives

Beyond Basic Gradient Descent

Modern optimizers address GD limitations:

- **Momentum:** Accelerates in consistent directions
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Handles non-stationary objectives

Beyond Basic Gradient Descent

Modern optimizers address GD limitations:

- **Momentum:** Accelerates in consistent directions
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Handles non-stationary objectives

Example: Why These Improvements?

- Handle different parameter scales
- Accelerate convergence
- Reduce oscillations
- Better for non-convex landscapes

Gradient Descent in Deep Learning

Key Points: E

very deep learning framework uses gradient descent variants!

Gradient Descent in Deep Learning

Key Points: E

very deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradients for neural networks

Gradient Descent in Deep Learning

Key Points: E

very deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradients for neural networks
- **Automatic differentiation:** PyTorch/TensorFlow magic

Gradient Descent in Deep Learning

Key Points: E

very deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradients for neural networks
- **Automatic differentiation:** PyTorch/TensorFlow magic
- **GPU acceleration:** Parallel mini-batch processing

Gradient Descent in Deep Learning

Key Points: E

very deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradients for neural networks
- **Automatic differentiation:** PyTorch/TensorFlow magic
- **GPU acceleration:** Parallel mini-batch processing
- **Mixed precision:** 16-bit + 32-bit arithmetic

Practical Considerations

Learning Rate Selection Strategies

Common approaches:

- **Grid search:** Try $\{0.001, 0.01, 0.1, 1.0\}$

Learning Rate Selection Strategies

Common approaches:

- **Grid search:** Try $\{0.001, 0.01, 0.1, 1.0\}$
- **Learning rate schedules:** Start high, decay over time

Learning Rate Selection Strategies

Common approaches:

- **Grid search:** Try $\{0.001, 0.01, 0.1, 1.0\}$
- **Learning rate schedules:** Start high, decay over time
- **Adaptive methods:** Let algorithm adjust automatically

Learning Rate Selection Strategies

Common approaches:

- **Grid search:** Try $\{0.001, 0.01, 0.1, 1.0\}$
- **Learning rate schedules:** Start high, decay over time
- **Adaptive methods:** Let algorithm adjust automatically
- **Learning rate finder:** Gradually increase and monitor loss

Learning Rate Selection Strategies

Common approaches:

- **Grid search:** Try $\{0.001, 0.01, 0.1, 1.0\}$
- **Learning rate schedules:** Start high, decay over time
- **Adaptive methods:** Let algorithm adjust automatically
- **Learning rate finder:** Gradually increase and monitor loss

Learning Rate Selection Strategies

Common approaches:

- **Grid search:** Try $\{0.001, 0.01, 0.1, 1.0\}$
- **Learning rate schedules:** Start high, decay over time
- **Adaptive methods:** Let algorithm adjust automatically
- **Learning rate finder:** Gradually increase and monitor loss

Important: Warning Signs

- Loss exploding $\rightarrow$ α too high
- Very slow progress $\rightarrow$ α too low
- Oscillating loss $\rightarrow$ Try smaller α or momentum

Convergence Criteria

When to stop optimization:

- **Gradient norm:** $\|\nabla f(\boldsymbol{\theta})\| < \epsilon$

Convergence Criteria

When to stop optimization:

- **Gradient norm:** $\|\nabla f(\boldsymbol{\theta})\| < \epsilon$
- **Function change:** $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$

Convergence Criteria

When to stop optimization:

- **Gradient norm:** $\|\nabla f(\boldsymbol{\theta})\| < \epsilon$
- **Function change:** $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$
- **Parameter change:** $\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\| < \epsilon$

Convergence Criteria

When to stop optimization:

- **Gradient norm:** $\|\nabla f(\boldsymbol{\theta})\| < \epsilon$
- **Function change:** $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$
- **Parameter change:** $\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\| < \epsilon$
- **Maximum iterations:** Safety upper bound

Convergence Criteria

When to stop optimization:

- **Gradient norm:** $\|\nabla f(\boldsymbol{\theta})\| < \epsilon$
- **Function change:** $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$
- **Parameter change:** $\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\| < \epsilon$
- **Maximum iterations:** Safety upper bound

Convergence Criteria

When to stop optimization:

- **Gradient norm:** $\|\nabla f(\boldsymbol{\theta})\| < \epsilon$
- **Function change:** $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$
- **Parameter change:** $\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\| < \epsilon$
- **Maximum iterations:** Safety upper bound

Key Points:

Best practice: Use multiple criteria + validation performance

Common Pitfalls

Important: Pitfall 1: Poor Initialization

Problem: Starting at bad points
Solution: Xavier/He initialization

Common Pitfalls

Important: Pitfall 1: Poor Initialization

Problem: Starting at bad points
Solution: Xavier/He initialization

Important: Pitfall 2: Wrong Learning Rate

Problem: Divergence or slow convergence
Solution: Learning rate schedules, adaptive optimizers

Common Pitfalls

Important: Pitfall 1: Poor Initialization

Problem: Starting at bad points
Solution: Xavier/He initialization

Important: Pitfall 2: Wrong Learning Rate

Problem: Divergence or slow convergence
Solution: Learning rate schedules, adaptive optimizers

Important: Pitfall 3: Poor Feature Scaling

Problem: Different scales cause poor convergence
Solution: Standardization: $(x - \mu)/\sigma$

Summary and Takeaways

What We've Learned

Key Points: G

radient descent is the backbone of modern machine learning!

What We've Learned

Key Points: G

radient descent is the backbone of modern machine learning!

Key concepts covered:

- **Mathematical foundation:** Taylor series derivation

What We've Learned

Key Points: G

radient descent is the backbone of modern machine learning!

Key concepts covered:

- **Mathematical foundation:** Taylor series derivation
- **Geometric intuition:** Steepest descent direction

What We've Learned

Key Points: G

radient descent is the backbone of modern machine learning!

Key concepts covered:

- **Mathematical foundation:** Taylor series derivation
- **Geometric intuition:** Steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch

What We've Learned

Key Points: G

radient descent is the backbone of modern machine learning!

Key concepts covered:

- **Mathematical foundation:** Taylor series derivation
- **Geometric intuition:** Steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch
- **Theoretical properties:** Unbiased estimation

What We've Learned

Key Points: G

radient descent is the backbone of modern machine learning!

Key concepts covered:

- **Mathematical foundation:** Taylor series derivation
- **Geometric intuition:** Steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch
- **Theoretical properties:** Unbiased estimation
- **Practical aspects:** Learning rates, convergence

Looking Ahead

Advanced optimization topics:

- **Second-order methods:** Newton's method, L-BFGS

Looking Ahead

Advanced optimization topics:

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers

Looking Ahead

Advanced optimization topics:

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers
- **Global optimization:** Simulated annealing

Looking Ahead

Advanced optimization topics:

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers
- **Global optimization:** Simulated annealing
- **Distributed optimization:** Federated learning

Looking Ahead

Advanced optimization topics:

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers
- **Global optimization:** Simulated annealing
- **Distributed optimization:** Federated learning

Looking Ahead

Advanced optimization topics:

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers
- **Global optimization:** Simulated annealing
- **Distributed optimization:** Federated learning

Key Points: M

aster gradient descent first - it's the foundation for everything else!

Final Pop Quiz #3

Answer this!

True or False?

1. SGD always converges faster than batch GD
2. Learning rates should decrease during training
3. SGD gradient estimates are unbiased
4. Normal equation is always better than GD
5. GD can find global minima for any function

Thank You!

Questions?

Next lecture: Advanced Optimization Techniques
Practice: Implement GD for your favorite ML model!