Gradient Descent: The Foundation of Machine Learning Optimization

From Taylor Series to Modern Deep Learning

Nipun Batra and the teaching staff

IIT Gandhinagar

August 28, 2025

Table of Contents

1. Mathematical Foundations
2. Taylor Series: The Mathematical Foundation
   2.1 Univariate Taylor Series
   2.2 Multivariate Taylor Series
3. From Taylor Series to Gradient Descent
4. The Gradient Descent Algorithm
5. Gradient Descent for Linear Regression
6. Variants of Gradient Descent
7. Mathematical Properties
8. Computational Complexity
9. Advanced Topics and Extensions
10. Practical Considerations
11. Summary and Key Takeaways

Mathematical Foundations

The Big Picture: Why Optimization Matters

Key Points:

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

The Big Picture: Why Optimization Matters

Key Points:

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

Examples everywhere:

- Linear regression: Minimize $(y - \mathbf{X}\boldsymbol{\theta})^2$

The Big Picture: Why Optimization Matters

Key Points:

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

Examples everywhere:

- Linear regression: Minimize $(y - \mathbf{X}\boldsymbol{\theta})^2$
- Neural networks: Minimize classification/regression loss

The Big Picture: Why Optimization Matters

Key Points:

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

Examples everywhere:

- Linear regression: Minimize $(y - \mathbf{X}\boldsymbol{\theta})^2$
- Neural networks: Minimize classification/regression loss
- Logistic regression: Minimize cross-entropy loss

The Big Picture: Why Optimization Matters

Key Points:

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

Examples everywhere:

- Linear regression: Minimize $(y - \mathbf{X}\boldsymbol{\theta})^2$
- Neural networks: Minimize classification/regression loss
- Logistic regression: Minimize cross-entropy loss

The Big Picture: Why Optimization Matters

Key Points:

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

Examples everywhere:

- Linear regression: Minimize $(y - \mathbf{X}\boldsymbol{\theta})^2$
- Neural networks: Minimize classification/regression loss
- Logistic regression: Minimize cross-entropy loss

Important: The Challenge

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

Gradient Intuition: Climbing Mountains

Imagine you're hiking in dense fog and want to reach the valley:

- You can only feel the slope beneath your feet

Gradient Intuition: Climbing Mountains

Imagine you're hiking in dense fog and want to reach the valley:

- You can only feel the slope beneath your feet
- **Strategy:** Always step in the steepest downhill direction

Gradient Intuition: Climbing Mountains

Imagine you're hiking in dense fog and want to reach the valley:

- You can only feel the slope beneath your feet
- **Strategy:** Always step in the steepest downhill direction
- **Gradient** = Direction of steepest uphill (ascent)

Gradient Intuition: Climbing Mountains

Imagine you're hiking in dense fog and want to reach the valley:

- You can only feel the slope beneath your feet
- **Strategy:** Always step in the steepest downhill direction
- **Gradient** = Direction of steepest uphill (ascent)
- **Negative gradient** = Direction of steepest downhill (descent)

Gradient Intuition: Climbing Mountains

Imagine you're hiking in dense fog and want to reach the valley:

- You can only feel the slope beneath your feet
- **Strategy:** Always step in the steepest downhill direction
- **Gradient** = Direction of steepest uphill (ascent)
- **Negative gradient** = Direction of steepest downhill (descent)

Gradient Intuition: Climbing Mountains

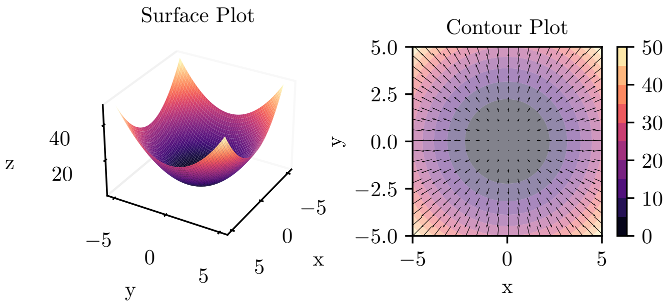
Imagine you're hiking in dense fog and want to reach the valley:

- You can only feel the slope beneath your feet
- **Strategy:** Always step in the steepest downhill direction
- **Gradient** = Direction of steepest uphill (ascent)
- **Negative gradient** = Direction of steepest downhill (descent)

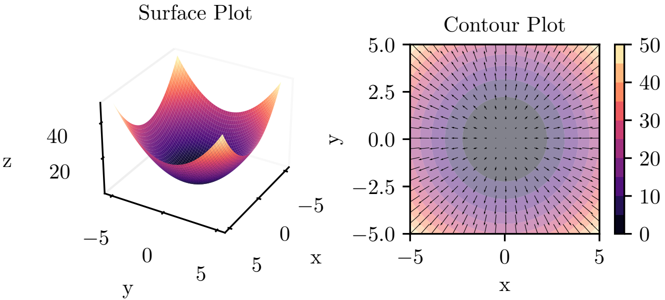
Key Points:

Key insight: Gradient points in direction of steepest ascent
So $-\nabla f$ points in direction of steepest descent!

Geometric Intuition with Level Sets



Geometric Intuition with Level Sets



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

Taylor Series: The Mathematical Foundation

Why Taylor Series? The Key Insight

Example: The Core Idea

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

Why Taylor Series? The Key Insight

Example: The Core Idea

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

Strategy:

- Replace complicated function with simpler approximation

Why Taylor Series? The Key Insight

Example: The Core Idea

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

Strategy:

- Replace complicated function with simpler approximation
- Optimize the approximation instead

Why Taylor Series? The Key Insight

Example: The Core Idea

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

Strategy:

- Replace complicated function with simpler approximation
- Optimize the approximation instead
- Move to new point and repeat

Why Taylor Series? The Key Insight

Example: The Core Idea

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

Strategy:

- Replace complicated function with simpler approximation
- Optimize the approximation instead
- Move to new point and repeat

Why Taylor Series? The Key Insight

Example: The Core Idea

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

Strategy:

- Replace complicated function with simpler approximation
- Optimize the approximation instead
- Move to new point and repeat

Important: Taylor Series Power

Any smooth function can be approximated by polynomials!

Taylor Series: Starting with 1D

Taylor series expansion around point x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{6}f'''(x_0)(x - x_0)^3 + \dots \tag{1}$$

Taylor Series: Starting with 1D

Taylor series expansion around point x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{6}f'''(x_0)(x - x_0)^3 + \ldots \tag{1}$$

Different orders of approximation:

- **Zero-order:** $f(x) \approx f(x_0)$ (constant)

Taylor Series: Starting with 1D

Taylor series expansion around point x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{6}f'''(x_0)(x - x_0)^3 + \ldots \tag{1}$$

Different orders of approximation:

- **Zero-order:** $f(x) \approx f(x_0)$ (constant)
- **First-order:** $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$ (linear)

Taylor Series: Starting with 1D

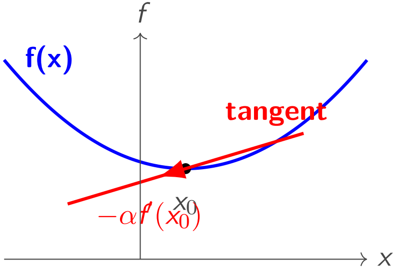
Taylor series expansion around point x_0:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{6}f'''(x_0)(x - x_0)^3 + \ldots \tag{1}$$

Different orders of approximation:

- **Zero-order:** $f(x) \approx f(x_0)$ (constant)
- **First-order:** $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$ (linear)
- **Second-order:** adds $\frac{1}{2}f''(x_0)(x - x_0)^2$ (quadratic)

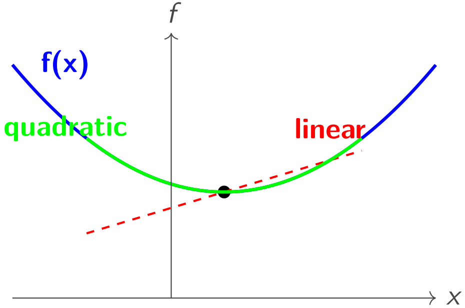
Visual: Tangent Line Approximation



Key Points:

Linear approximation gives us the direction to move!

Adding Quadratic Term



Key Points:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Concrete Example: $f(x) = \cos(x)$ at $x_0 = 0$

Let's compute the derivatives:

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

Taylor approximations:

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

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

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

Extension to Multiple Variables

For function $f(\mathrm{x})$ around point x_0:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \tag{5}$$

Extension to Multiple Variables

For function $f(\mathbf{x})$ around point $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T\nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \tag{5}$$

Where:

- $\nabla f(\mathbf{x}_0)$ is the **gradient** (vector of partial derivatives)

Extension to Multiple Variables

For function $f(\mathbf{x})$ around point $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \ldots \tag{5}$$

Where:

- $\nabla f(\mathbf{x}_0)$ is the **gradient** (vector of partial derivatives)
- $\nabla^2 f(\mathbf{x}_0)$ is the **Hessian** (matrix of second derivatives)

Extension to Multiple Variables

For function $f(\mathbf{x})$ around point $\mathbf{x}_0$:

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \ldots \tag{5}$$

Where:

- $\nabla f(\mathbf{x}_0)$ is the **gradient** (vector of partial derivatives)
- $\nabla^2 f(\mathbf{x}_0)$ is the **Hessian** (matrix of second derivatives)
- $(\mathbf{x} - \mathbf{x}_0) = \Delta\mathbf{x}$ is the step vector

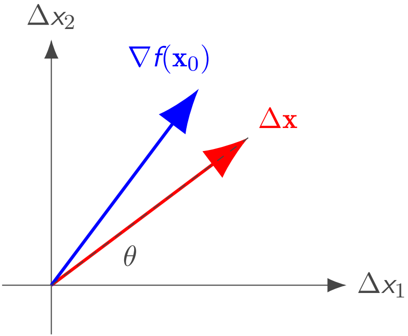
Understanding the Linear Term

The first-order term: $\nabla f(\mathbf{x}_0)^T \Delta \mathbf{x}$ where $\Delta \mathbf{x} = \mathbf{x} - \mathbf{x}_0$

Understanding the Linear Term

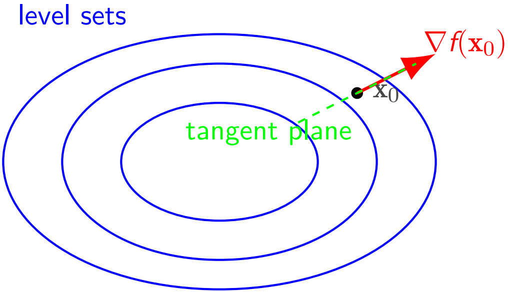
The first-order term: $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x}$ where $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$

For 2D case: $\Delta\mathbf{x} = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \begin{bmatrix} x_1 - x_{0,1} \\ x_2 - x_{0,2} \end{bmatrix}$



Geometric interpretation: $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} = |\nabla f||\Delta\mathbf{x}| \cos\theta$

Visual: Multivariate Case with Level Sets



Key Points:

Gradient $\perp$ level sets, tangent plane $\perp$ gradient

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 first-order Taylor approximation:

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} \tag{6}$$

The Key Question

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

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} \tag{6}$$

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 first-order Taylor approximation:

$$f(\mathbf{x}_0 + \Delta\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} \tag{6}$$

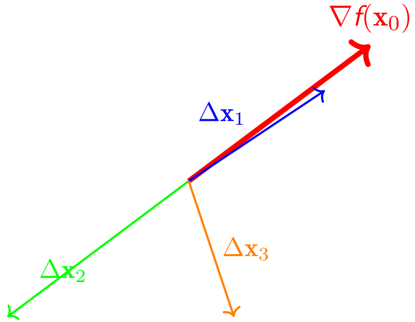
For the function to decrease:

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

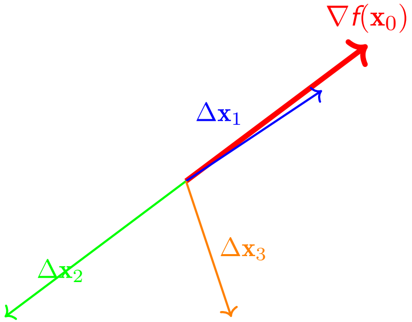
Important: Vector Geometry Reminder

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: Finding the Best Direction



Visual Derivation: Finding the Best Direction



Dot products tell us the direction:

- $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x}_1 > 0$ (increases function)
- $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x}_2 < 0$ (decreases function - good!)
- $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x}_3 < 0$ (decreases function)

The Optimal Choice: Direction of Steepest Descent

Definition: Optimal Choice

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

The Optimal Choice: Direction of Steepest Descent

Definition: Optimal Choice

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

Why this choice?

- $-\nabla f(\mathbf{x}_0)$ points in direction of steepest descent

The Optimal Choice: Direction of Steepest Descent

Definition: Optimal Choice

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

Why this choice?

- $-\nabla f(\mathbf{x}_0)$ points in direction of steepest descent
- $\alpha > 0$ controls the step size

The Optimal Choice: Direction of Steepest Descent

Definition: Optimal Choice

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

Why this choice?

- $-\nabla f(\mathbf{x}_0)$ points in direction of steepest descent
- $\alpha > 0$ controls the step size
- Guarantees $\nabla f(\mathbf{x}_0)^T\Delta\mathbf{x} < 0$ (function decrease)

The Optimal Choice: Direction of Steepest Descent

Definition: Optimal Choice

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

Why this choice?

- $-\nabla f(\mathbf{x}_0)$ points in direction of steepest descent
- $\alpha > 0$ controls the step size
- Guarantees $\nabla f(\mathbf{x}_0)^T \Delta \mathbf{x} < 0$ (function decrease)

The Optimal Choice: Direction of Steepest Descent

Definition: Optimal Choice

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

Why this choice?

- $-\nabla f(\mathbf{x}_0)$ points in direction of steepest descent
- $\alpha > 0$ controls the step size
- Guarantees $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$ (function decrease)

Key Points:

This gives us the fundamental gradient descent step!

The Gradient Descent Update Rule

This gives us the gradient descent update:

$$\mathbf{x}_{\mathsf{new}} = \mathbf{x}_{\mathsf{old}} - \alpha \nabla f(\mathbf{x}_{\mathsf{old}})$$

The Gradient Descent Update Rule

This gives us the gradient descent update:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - \alpha \nabla f(\mathbf{x}_{\text{old}})$$

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

The Gradient Descent Update Rule

This gives us the gradient descent update:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - \alpha \nabla f(\mathbf{x}_{\text{old}})$$

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Key properties:

- Uses only first derivatives (gradients)

The Gradient Descent Update Rule

This gives us the gradient descent update:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - \alpha \nabla f(\mathbf{x}_{\text{old}})$$

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Key properties:

- Uses only first derivatives (gradients)
- Greedy local search

The Gradient Descent Update Rule

This gives us the gradient descent update:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - \alpha \nabla f(\mathbf{x}_{\text{old}})$$

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Key properties:

- Uses only first derivatives (gradients)
- Greedy local search
- Guaranteed convergence for convex functions

The Gradient Descent Update Rule

This gives us the gradient descent update:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{old}} - \alpha \nabla f(\mathbf{x}_{\text{old}})$$

Definition: Gradient Descent Algorithm

An iterative first-order optimization method for finding local minima

Key properties:

- Uses only first derivatives (gradients)
- Greedy local search
- Guaranteed convergence for convex functions
- Foundation of modern machine learning

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

Algorithm Steps:

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

The Complete Algorithm

Algorithm Steps:

1. **Initialize:** Choose starting point $\boldsymbol{\theta}_0$
2. **Repeat until convergence:**

The Complete Algorithm

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

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

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

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

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

The Complete Algorithm

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

Key Points:

Learning rate selection is crucial for success!

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

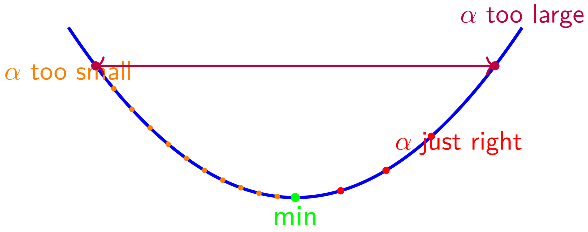
The learning rate α controls how big steps we take:

- **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

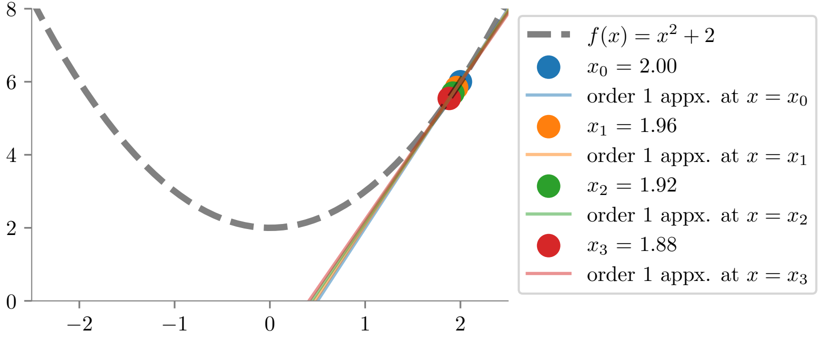
The learning rate α controls how big steps we take:

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



Learning Rate Visualization: Too Small

$\alpha = 0.01$: **Convergence is slow but stable**

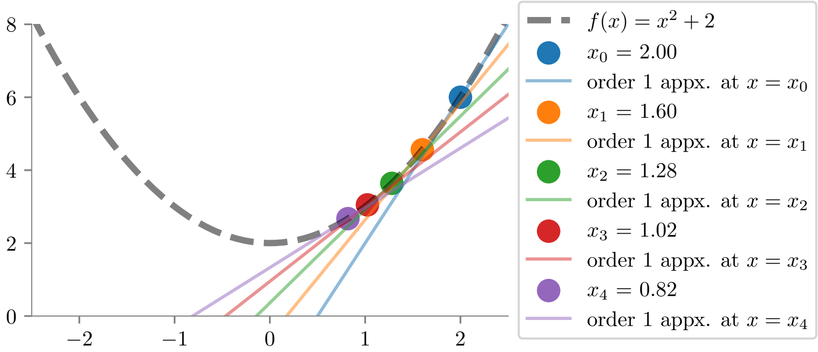


Important: Problem

Takes many iterations to reach the minimum. Computationally expensive!

Learning Rate: Just Right

$\alpha = 0.1$: **Good balance: Fast and stable convergence**

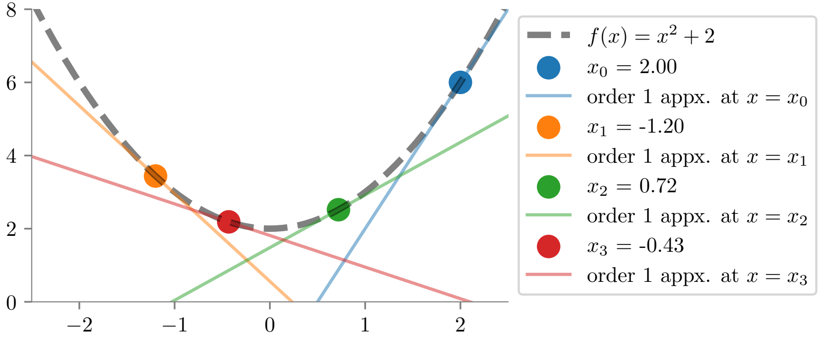


Key Points:

Perfect balance: Fast convergence + Stability

Learning Rate: Too Large

$\alpha = 0.8$: **Fast but may overshoot**

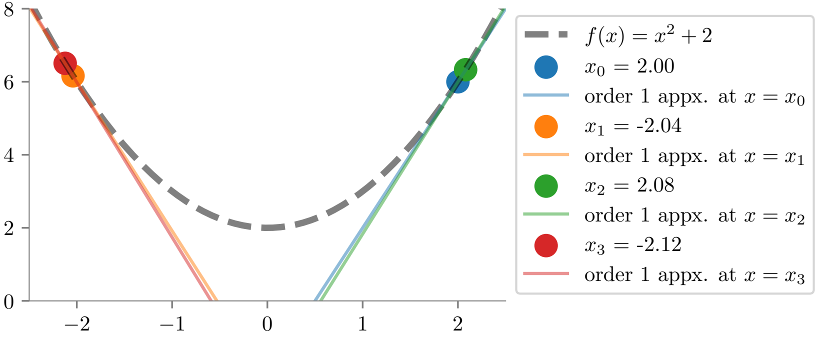


Important: Warning

Quick convergence but risk of instability. Watch out for oscillations!

Learning Rate: Disaster

$\alpha = 1.01$**: Divergence! Function values explode**



Important: Disaster Zone

The algorithm diverges. Always monitor your loss curves!

Gradient Descent for Linear Regression

Linear Regression: Our First Application

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

x	**y**
1	1
2	2
3	3

Linear Regression: Our First Application

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

x	**y**
1	1
2	2
3	3

Cost Function (Mean Squared Error):

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

Linear Regression: Our First Application

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

x	**y**
1	1
2	2
3	3

Cost Function (Mean Squared Error):

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

Goal: $(\theta_0^*, \theta_1^*) = \arg\min_{\theta_0, \theta_1} \mathrm{MSE}(\theta_0, \theta_1)$

Computing Gradients for Linear Regression

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

Computing Gradients for Linear Regression

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

Let's compute each partial derivative:

$$\frac{\partial \mathrm{MSE}}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)(-1) \tag{7}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} \epsilon_i \tag{8}$$

Computing Gradients for Linear Regression

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

Let's compute each partial derivative:

$$\frac{\partial \mathrm{MSE}}{\partial \theta_0} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)(-1) \tag{7}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} \epsilon_i \tag{8}$$

$$\frac{\partial \mathrm{MSE}}{\partial \theta_1} = \frac{2}{n} \sum_{i=1}^{n} (y_i - \theta_0 - \theta_1 x_i)(-x_i) \tag{9}$$

$$= -\frac{2}{n} \sum_{i=1}^{n} \epsilon_i x_i \tag{10}$$

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 - Predictions:

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$

Step-by-Step Example: Setup

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

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = \theta_0 + \theta_1 \cdot 2 = 4 + 0 \cdot 2 = 4$

Step-by-Step Example: Setup

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

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = \theta_0 + \theta_1 \cdot 2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = \theta_0 + \theta_1 \cdot 3 = 4 + 0 \cdot 3 = 4$

Step-by-Step Example: Setup

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

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = \theta_0 + \theta_1 \cdot 2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = \theta_0 + \theta_1 \cdot 3 = 4 + 0 \cdot 3 = 4$

Step-by-Step Example: Setup

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

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = \theta_0 + \theta_1 \cdot 2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = \theta_0 + \theta_1 \cdot 3 = 4 + 0 \cdot 3 = 4$

Errors (residuals):

- $\epsilon_1 = y_1 - \hat{y}_1 = 1 - 4 = -3$

Step-by-Step Example: Setup

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

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = \theta_0 + \theta_1 \cdot 2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = \theta_0 + \theta_1 \cdot 3 = 4 + 0 \cdot 3 = 4$

Errors (residuals):

- $\epsilon_1 = y_1 - \hat{y}_1 = 1 - 4 = -3$
- $\epsilon_2 = y_2 - \hat{y}_2 = 2 - 4 = -2$

Step-by-Step Example: Setup

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

- $\hat{y}_1 = \theta_0 + \theta_1 \cdot 1 = 4 + 0 \cdot 1 = 4$
- $\hat{y}_2 = \theta_0 + \theta_1 \cdot 2 = 4 + 0 \cdot 2 = 4$
- $\hat{y}_3 = \theta_0 + \theta_1 \cdot 3 = 4 + 0 \cdot 3 = 4$

Errors (residuals):

- $\epsilon_1 = y_1 - \hat{y}_1 = 1 - 4 = -3$
- $\epsilon_2 = y_2 - \hat{y}_2 = 2 - 4 = -2$
- $\epsilon_3 = y_3 - \hat{y}_3 = 3 - 4 = -1$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = -\frac{2}{3}(-6) = 4$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = -\frac{2}{3}(-6) = 4$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = -\frac{2}{3}(-10) = 6.67$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3 - 2 - 1) = -\frac{2}{3}(-6) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = -\frac{2}{3}(-10) = 6.67$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3 - 2 - 1) = -\frac{2}{3}(-6) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = -\frac{2}{3}(-10) = 6.67$

Parameter updates:

- $\theta_0 = 4 - 0.1 \times 4 = 3.6$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3 - 2 - 1) = -\frac{2}{3}(-6) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = -\frac{2}{3}(-10) = 6.67$

Parameter updates:

- $\theta_0 = 4 - 0.1 \times 4 = 3.6$
- $\theta_1 = 0 - 0.1 \times 6.67 = -0.67$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = -\frac{2}{3}(-6) = 4$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = -\frac{2}{3}(-10) = 6.67$

Parameter updates:

- $\theta_0 = 4 - 0.1 \times 4 = 3.6$
- $\theta_1 = 0 - 0.1 \times 6.67 = -0.67$

Step-by-Step Example: Gradients

Compute gradients:

- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_0} = -\frac{2}{3}(-3-2-1) = -\frac{2}{3}(-6) = 4$
- $\frac{\partial\,\mathrm{MSE}}{\partial\theta_1} = -\frac{2}{3}(-3\cdot 1 - 2\cdot 2 - 1\cdot 3) = -\frac{2}{3}(-10) = 6.67$

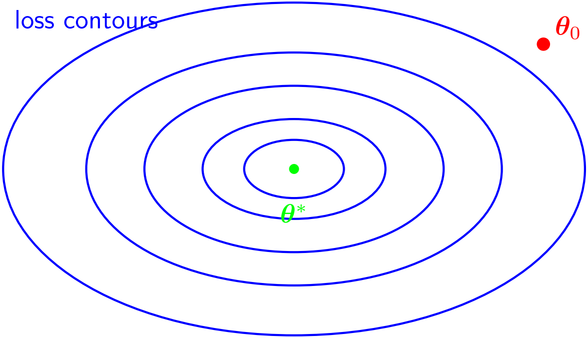
Parameter updates:

- $\theta_0 = 4 - 0.1 \times 4 = 3.6$
- $\theta_1 = 0 - 0.1 \times 6.67 = -0.67$

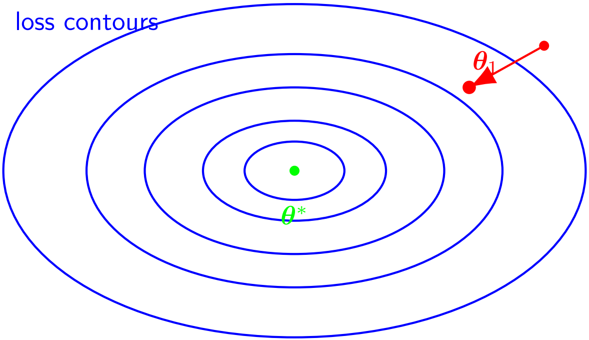
Key Points:

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

Visual Journey: Gradient Descent in Action



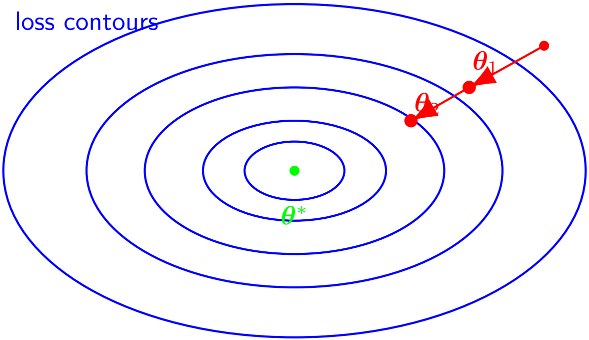
Visual Journey: Gradient Descent in Action



Key Points:

Steps get smaller as we approach minimum (gradient magnitude decreases)!

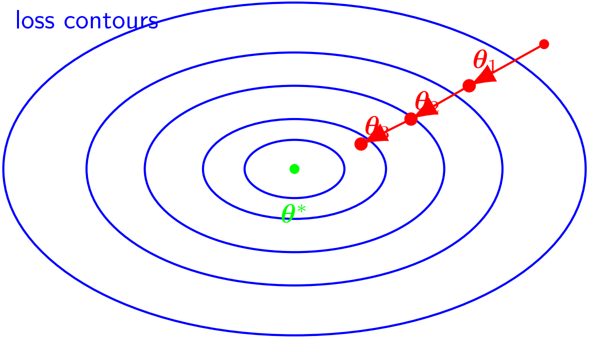
Visual Journey: Gradient Descent in Action



Key Points:

Steps get smaller as we approach minimum (gradient magnitude decreases)!

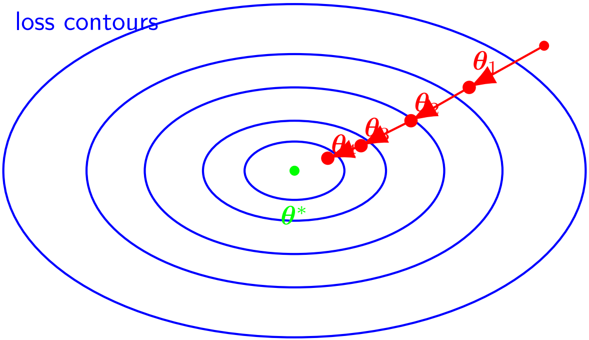
Visual Journey: Gradient Descent in Action



Key Points:

Steps get smaller as we approach minimum (gradient magnitude decreases)!

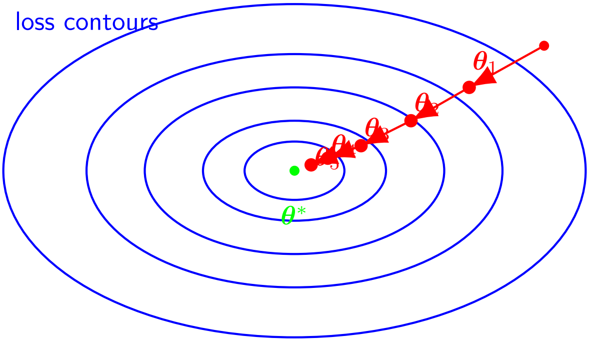
Visual Journey: Gradient Descent in Action



Key Points:

Steps get smaller as we approach minimum (gradient magnitude decreases)!

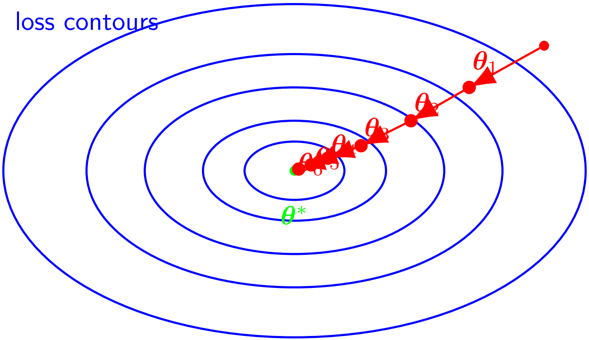
Visual Journey: Gradient Descent in Action



Key Points:

Steps get smaller as we approach minimum (gradient magnitude decreases)!

Visual Journey: Gradient Descent in Action



Key Points:

Steps get smaller as we approach minimum (gradient magnitude decreases)!

Variants of Gradient Descent

The Gradient Descent Family

Three main variants based on data usage:

Definition: Batch Gradient Descent

Use all training data to compute each gradient

Definition: Stochastic Gradient Descent (SGD)

Use one sample to compute each gradient

Definition: Mini-batch Gradient Descent

Use a small batch of samples to compute each gradient

Comparison: Batch vs SGD vs Mini-batch

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

Comparison: Batch vs SGD vs Mini-batch

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

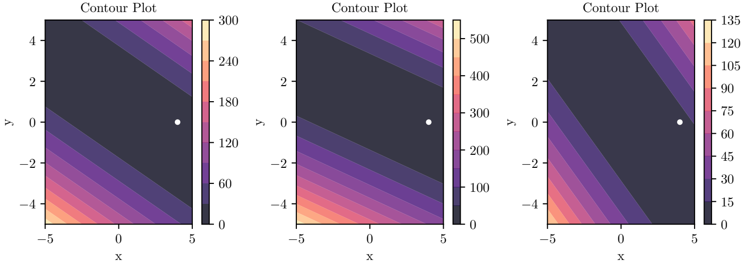
Key Points:

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

- Good balance of stability and efficiency
- Enables parallel computation (GPUs!)
- Better gradient estimates than pure SGD

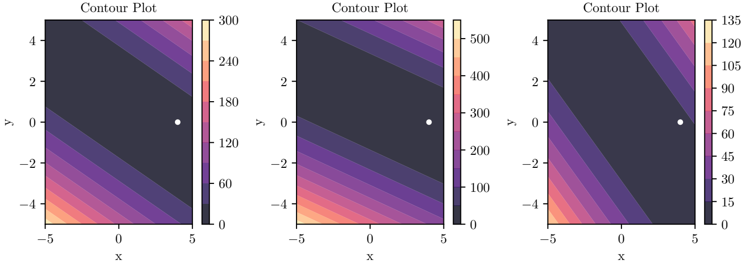
SGD: The Noisy Path

SGD uses one sample at a time for updates



SGD: The Noisy Path

SGD uses one sample at a time for updates

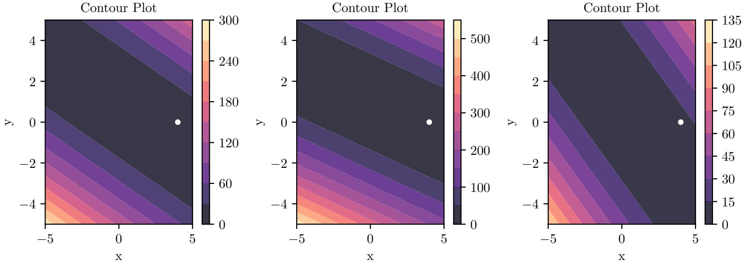


Trade-offs:

- **Pro:** Fast updates, can escape local minima

SGD: The Noisy Path

SGD uses one sample at a time for updates

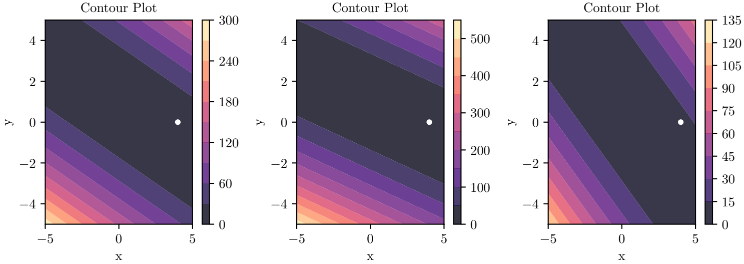


Trade-offs:

- **Pro:** Fast updates, can escape local minima
- **Con:** Noisy convergence, may not reach exact minimum

SGD: The Noisy Path

SGD uses one sample at a time for updates



Trade-offs:

- **Pro:** Fast updates, can escape local minima
- **Con:** Noisy convergence, may not reach exact minimum
- **Key insight:** Noise can be beneficial for non-convex problems!

Mathematical Properties

SGD as an Unbiased Estimator

True gradient (what we want):

$$\nabla L(\boldsymbol{\theta}) = \nabla \left(\frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) \right) \tag{11}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) \qquad \text{(linearity of gradient)} \tag{12}$$

SGD as an Unbiased Estimator

True gradient (what we want):

$$\nabla L(\boldsymbol{\theta}) = \nabla \left(\frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) \right) \tag{11}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) \qquad \text{(linearity of gradient)} \tag{12}$$

SGD gradient estimate (what we compute):

$$\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}_j; \boldsymbol{\theta}), y_j)$$

where $(\mathbf{x}_j, y_j)$ is sampled uniformly from $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$

The Unbiased Property: Mathematical Proof

Theorem: SGD Unbiased Estimator Property

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

The Unbiased Property: Mathematical Proof

Theorem: SGD Unbiased Estimator Property

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

Detailed Proof:

$$\begin{aligned}
\mathbb{E}[\nabla \tilde{L}(\boldsymbol{\theta})] &= \mathbb{E}\left[\nabla \ell(f(\mathbf{x}_j; \boldsymbol{\theta}), y_j)\right] && (13)\\
&= \sum_{i=1}^{n} P(\text{sample } i) \cdot \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) && (14)\\
&= \sum_{i=1}^{n} \frac{1}{n} \cdot \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) && (15)\\
&= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) \qquad \text{(linearity of expectation)} && (16)
\end{aligned}$$

The Unbiased Property: Mathematical Proof

Theorem: SGD Unbiased Estimator Property

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

Detailed Proof:

$$\begin{aligned}
\mathbb{E}[\nabla \tilde{L}(\boldsymbol{\theta})] &= \mathbb{E}\left[\nabla \ell(f(\mathbf{x}_j; \boldsymbol{\theta}), y_j)\right] && (13)\\
&= \sum_{i=1}^{n} P(\text{sample } i) \cdot \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) && (14)\\
&= \sum_{i=1}^{n} \frac{1}{n} \cdot \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) && (15)\\
&= \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(f(\mathbf{x}_i; \boldsymbol{\theta}), y_i) \qquad \text{(linearity of expectation)} && (16)
\end{aligned}$$

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 over time

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 over time
- Theoretical guarantee that justifies SGD's effectiveness

Why Unbiasedness Matters

Key Points:

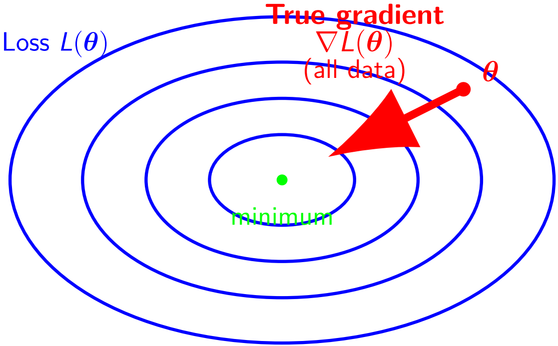
Key insight: On average, SGD points in the correct direction!

Practical implications:

- Individual SGD steps may be "wrong"
- But they average to the correct direction over time
- Theoretical guarantee that justifies SGD's effectiveness
- The "noise" helps escape local minima in non-convex problems

Visual Intuition 1: Overall Loss Surface

True loss function using all data points:

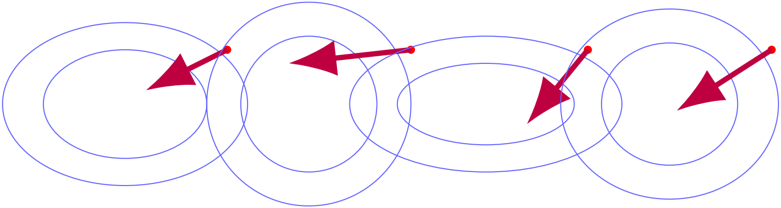


Key Points:

This is what we want: gradient computed using ALL data points

Visual Intuition 2: Individual Sample Loss Surfaces

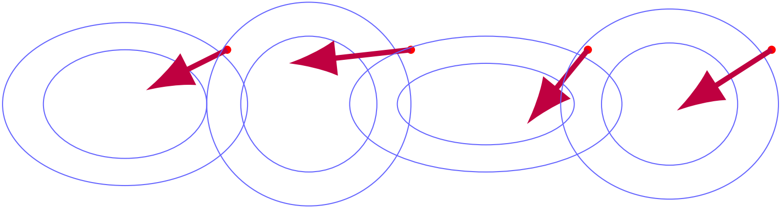
Loss for individual data points (different shapes):



Sample 1: $\nabla \ell_1$ Sample 2: $\nabla \ell_2$ Sample 3: $\nabla \ell_3$ Sample n: $\nabla \ell_n$

Visual Intuition 2: Individual Sample Loss Surfaces

Loss for individual data points (different shapes):



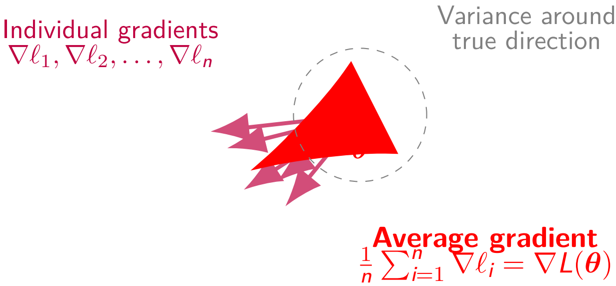
Sample 1: $\nabla \ell_1$ Sample 2: $\nabla \ell_2$ Sample 3: $\nabla \ell_3$ Sample n: $\nabla \ell_n$

Important: Key Observation

Each individual gradient points in a **different direction** - some variation!

Visual Intuition 3: Averaging Individual Gradients

The magic: Average of individual gradients = True gradient

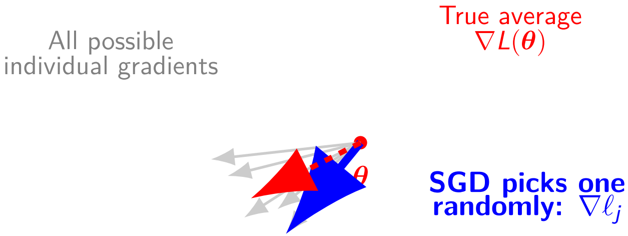


Theorem: Visual Proof of Unbiasedness

Even though individual gradients vary, their average equals the true gradient!

Visual Intuition 4: SGD Sampling Process

SGD randomly picks one gradient at a time:



Key Points:

Key insight: Sometimes SGD goes "wrong" direction, but on average it's correct!

Why Unbiasedness Matters in Practice

Why Unbiasedness Matters in Practice

Example: Intuitive Analogy

Like asking random people for directions:

- Each person's answer might be slightly off
- But if there's no systematic bias, the average is correct
- SGD does the same with gradient estimates!

Computational Complexity

GD vs Normal Equation: Complexity

For linear regression:

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: Complexity

For linear regression:

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 Use Which Method

Key Points:

Modern ML: Gradient descent dominates due to:

- High-dimensional data (d very large)
- Non-linear models (no normal equation exists)
- Large datasets (n very large)

When to Use Which Method

Key Points:

Modern ML: Gradient descent dominates due to:

- High-dimensional data (d very large)
- Non-linear models (no normal equation exists)
- Large datasets (n very large)

Decision criteria:

- **Few features ($d < 1000$):** Consider normal equation

When to Use Which Method

Key Points:

Modern ML: Gradient descent dominates due to:

- High-dimensional data (d very large)
- Non-linear models (no normal equation exists)
- Large datasets (n very large)

Decision criteria:

- **Few features ($d < 1000$):** Consider normal equation
- **Many features ($d > 10000$):** Gradient descent

When to Use Which Method

Key Points:

Modern ML: Gradient descent dominates due to:

- High-dimensional data (d very large)
- Non-linear models (no normal equation exists)
- Large datasets (n very large)

Decision criteria:

- **Few features ($d < 1000$):** Consider normal equation
- **Many features ($d > 10000$):** Gradient descent
- **Non-linear models:** Only gradient descent works

When to Use Which Method

Key Points:

Modern ML: Gradient descent dominates due to:

- High-dimensional data (d very large)
- Non-linear models (no normal equation exists)
- Large datasets (n very large)

Decision criteria:

- **Few features ($d < 1000$):** Consider normal equation
- **Many features ($d > 10000$):** Gradient descent
- **Non-linear models:** Only gradient descent works
- **Online learning:** Only gradient descent works

Advanced Topics and Extensions

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1-\beta)\mathbf{g}_t$

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Exponential moving average of squared gradients

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Exponential moving average of squared gradients

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Exponential moving average of squared gradients

Why these improvements?

- Handle different parameter scales automatically

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Exponential moving average of squared gradients

Why these improvements?

- Handle different parameter scales automatically
- Accelerate convergence in relevant directions

Beyond Basic Gradient Descent

Modern optimizers improve upon vanilla GD:

- **Momentum:** $\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t$
- **AdaGrad:** Adaptive per-parameter learning rates
- **Adam:** Combines momentum + adaptive rates
- **RMSprop:** Exponential moving average of squared gradients

Why these improvements?

- Handle different parameter scales automatically
- Accelerate convergence in relevant directions
- Reduce oscillations in narrow valleys

Gradient Descent in Deep Learning

Key Points:

Every deep learning framework uses gradient descent variants!

Gradient Descent in Deep Learning

Key Points:

Every deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradient computation

Gradient Descent in Deep Learning

Key Points:

Every deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradient computation
- **Automatic differentiation:** PyTorch/TensorFlow magic

Gradient Descent in Deep Learning

Key Points:

Every deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradient computation
- **Automatic differentiation:** PyTorch/TensorFlow magic
- **GPU acceleration:** Parallel mini-batch processing

Gradient Descent in Deep Learning

Key Points:

Every deep learning framework uses gradient descent variants!

Key modern extensions:

- **Backpropagation:** Efficient gradient computation
- **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

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

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

Warning signs:

- Loss exploding $\rightarrow \alpha$ too high

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

Warning signs:

- Loss exploding $\rightarrow$ α too high
- Very slow progress $\rightarrow$ α too low

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

Warning signs:

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

Convergence Criteria

When to stop training?

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

Convergence Criteria

When to stop training?

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

Convergence Criteria

When to stop training?

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

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

Convergence Criteria

When to stop training?

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

Convergence Criteria

When to stop training?

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

Key Points:

Best practice: Use multiple criteria + validation performance

Common Pitfalls

Important: Pitfall 1: Poor Initialization

Problem: Bad starting points
Solution: Xavier/He initialization

Common Pitfalls

Important: Pitfall 1: Poor Initialization

Problem: Bad starting 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: Bad starting 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: Standardize features: $(x - \mu)/\sigma$

Summary and Key Takeaways

What We've Learned

Key Points:

Gradient descent is the backbone of modern machine learning!

What We've Learned

Key Points:

Gradient descent is the backbone of modern machine learning!

Journey recap:

- **Mathematical foundation:** Taylor series derivation

What We've Learned

Key Points:

Gradient descent is the backbone of modern machine learning!

Journey recap:

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

What We've Learned

Key Points:

Gradient descent is the backbone of modern machine learning!

Journey recap:

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

What We've Learned

Key Points:

Gradient descent is the backbone of modern machine learning!

Journey recap:

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

What We've Learned

Key Points:

Gradient descent is the backbone of modern machine learning!

Journey recap:

- **Mathematical foundation:** Taylor series derivation
- **Geometric intuition:** Steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch
- **Theoretical properties:** Unbiased estimator guarantees
- **Practical wisdom:** Learning rates, scaling, diagnostics

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch
- Experiment with different learning rates

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Compare batch vs SGD vs mini-batch

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Compare batch vs SGD vs mini-batch
- Try advanced optimizers (Adam, momentum)

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Compare batch vs SGD vs mini-batch
- Try advanced optimizers (Adam, momentum)
- Apply to real datasets

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Compare batch vs SGD vs mini-batch
- Try advanced optimizers (Adam, momentum)
- Apply to real datasets

From Theory to Practice

Next steps for mastery:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Compare batch vs SGD vs mini-batch
- Try advanced optimizers (Adam, momentum)
- Apply to real datasets

Key Points:

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

Final Pop Quiz #2

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 always beats gradient descent
5. GD guarantees global minimum for any function

Deep Dive: Advanced Theory

For comprehensive mathematical analysis:

Important: Reference Materials

- SGD.pdf: Detailed convergence proofs
- Florian's estimators: https://florian.github.io/estimators/
- Interactive notebooks for hands-on practice

Pop Quiz Solutions

Quiz #1 Solutions:

1. $f(2) = 6$, $f'(2) = 4$
2. $f(x) \approx 6 + 4(x - 2)$
3. New $x = 2 - 0.1 \times 4 = 1.6$
4. Yes, function decreases!

Pop Quiz Solutions

Quiz #1 Solutions:

1. $f(2) = 6$, $f'(2) = 4$
2. $f(x) \approx 6 + 4(x - 2)$
3. New $x = 2 - 0.1 \times 4 = 1.6$
4. Yes, function decreases!

Quiz #2 Solutions:

1. False - SGD faster per epoch, may need more epochs
2. True - schedules often improve convergence
3. True - key theoretical property
4. False - only for linear problems, small d
5. False - only local minima; global for convex only

Thank You!

Questions?

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