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

Mathematical Foundations

The Big Picture: Why Optimization Matters

- **Core ML Problem:** Find best parameters θ^* for our model

The Big Picture: Why Optimization Matters

- **Core ML Problem:** Find best parameters θ^* for our model
- **Examples everywhere:**

The Big Picture: Why Optimization Matters

- **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

- **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

- **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

- **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
- **Challenge:** Most ML problems have no closed-form solution

The Big Picture: Why Optimization Matters

- **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
- **Challenge:** Most ML problems have no closed-form solution
- **Solution:** Iterative optimization algorithms

The Big Picture: Why Optimization Matters

- **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
- **Challenge:** Most ML problems have no closed-form solution
- **Solution:** Iterative optimization algorithms

The Big Picture: Why Optimization Matters

- **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
- **Challenge:** Most ML problems have no closed-form solution
- **Solution:** Iterative optimization algorithms

Key Points: G

radient descent is the workhorse of modern machine learning!

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

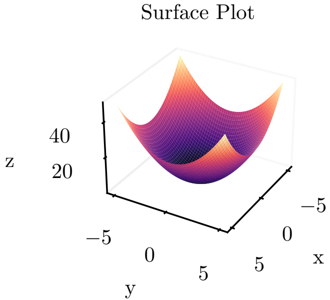
Imagine you're hiking in dense fog and want to reach the valley:

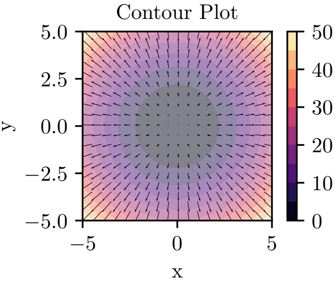
- You can only feel the slope beneath your feet
- **Strategy:** Always step in the steepest downhill direction
- **Gradient** = Direction of steepest uphill (ascent)
- **Negative gradient** = Direction of steepest downhill (descent)

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)

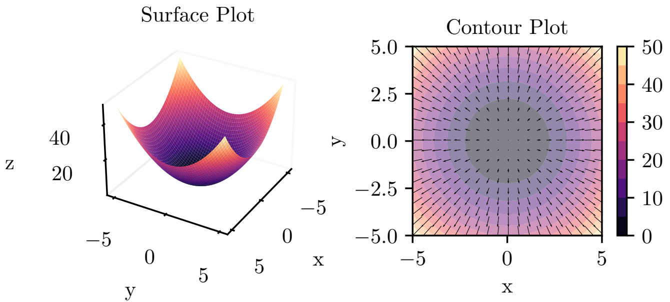




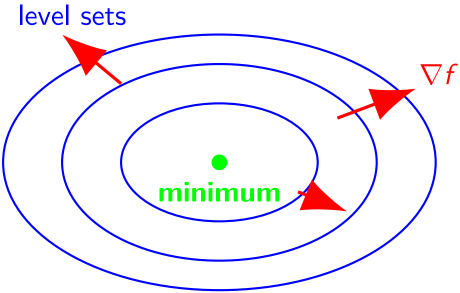
Gradient Intuition: Climbing Mountains

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)



Geometric Intuition with Level Sets



Key Points: K

ey insight: Gradient $\perp$ level sets, points toward steepest ascent

Taylor Series: The Mathematical Foundation

Why Taylor Series? The Key Insight

Definition: The Core Idea

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

Why Taylor Series? The Key Insight

Definition: The Core Idea

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

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

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

Why Taylor Series? The Key Insight

Definition: The Core Idea

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

Taylor series expansion around point x_0:

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

- **Zero-order:** $f(\mathrm{x}) \approx f(\mathrm{x}_0)$ (just the function value)

Why Taylor Series? The Key Insight

Definition: The Core Idea

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

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

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

- **Zero-order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0)$ (just the function value)
- **First-order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ (linear approximation)

Why Taylor Series? The Key Insight

Definition: The Core Idea

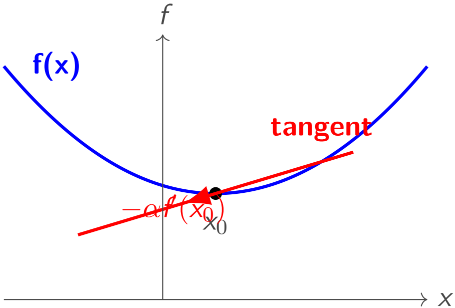
If we can't solve $\min f(\mathbf{x})$ exactly, let's approximate $f(\mathbf{x})$ locally and optimize that!

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

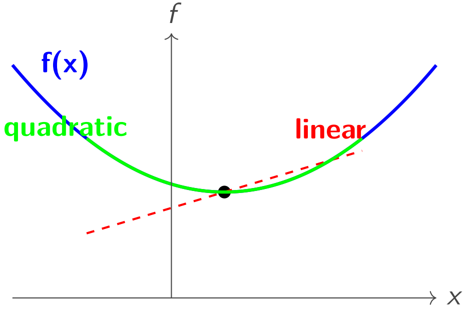
$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \dots \tag{1}$$

- **Zero-order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0)$ (just the function value)
- **First-order:** $f(\mathbf{x}) \approx f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0)$ (linear approximation)
- **Second-order:** Includes curvature via Hessian $\nabla^2 f(\mathbf{x}_0)$

Univariate Taylor: Visual Understanding



Adding Quadratic Term



Important: Key Insight

Higher-order terms give better approximations, but first-order is often sufficient for optimization!

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:

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

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

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

Multivariate Taylor Series

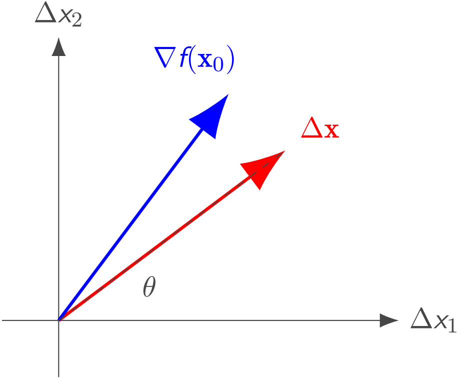
Extension to multiple variables:

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

Multivariate Taylor Series

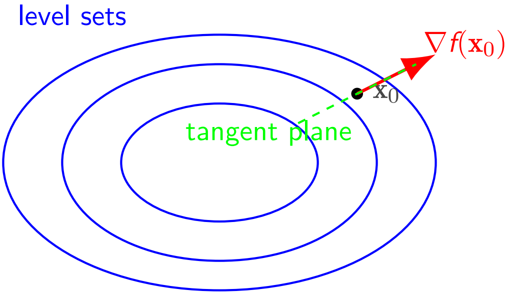
Extension to multiple variables:

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



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

Visual: Multivariate Case with Level Sets



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

From Taylor Series to Gradient Descent

The Derivation: From Theory to Algorithm

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

The Derivation: From Theory to Algorithm

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 Derivation: From Theory to Algorithm

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

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

To minimize, we need: $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$

The Derivation: From Theory to Algorithm

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

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

To minimize, we need: $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$

Example: Vector Geometry Insight

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

The Derivation: From Theory to Algorithm

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

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

To minimize, we need: $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$

Example: Vector Geometry Insight

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

Optimal choice: $\Delta\mathbf{x} = -\alpha\nabla f(\mathbf{x}_0)$ where $\alpha > 0$

The Derivation: From Theory to Algorithm

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

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

To minimize, we need: $\nabla f(\mathbf{x}_0)^T \Delta\mathbf{x} < 0$

Example: Vector Geometry Insight

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

Optimal choice: $\Delta\mathbf{x} = -\alpha\nabla f(\mathbf{x}_0)$ where $\alpha > 0$

Definition: Gradient Descent Update Rule

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

Pop Quiz #1: Taylor Series Understanding

Answer this!

Given $f(x) = x^2 + 2$ and expansion point $x_0 = 2$:
Questions:

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

Pop Quiz #1: Solutions

Example: Solutions

Given $f(x) = x^2 + 2$ and $x_0 = 2$:

1. $f(2) = 4 + 2 = 6$
2. $f'(x) = 2x$, so $f'(2) = 4$
3. $f(x) \approx 6 + 4(x - 2) = 4x - 2$
4. $\Delta x = -0.1 \times 4 = -0.4$, so $x_{new} = 2 - 0.4 = 1.6$

Gradient Descent Algorithm

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** θ_0 (random or educated guess)

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** $\boldsymbol{\theta}_0$ (random or educated guess)
2. **For** $t = 0, 1, 2, \ldots$ until convergence:

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** $\boldsymbol{\theta}_0$ (random or educated guess)
2. **For** $t = 0, 1, 2, \ldots$ until convergence:
   - Compute gradient: $\mathbf{g}_t = \nabla f(\boldsymbol{\theta}_t)$

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** $\boldsymbol{\theta}_0$ (random or educated guess)
2. **For** $t = 0, 1, 2, \ldots$ 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 Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** $\boldsymbol{\theta}_0$ (random or educated guess)
2. **For** $t = 0, 1, 2, \ldots$ 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 convergence: $|\mathbf{g}_t| < \epsilon$ or $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** $\boldsymbol{\theta}_0$ (random or educated guess)
2. **For** $t = 0, 1, 2, \ldots$ 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 convergence: $|\mathbf{g}_t| < \epsilon$ or $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$

The Gradient Descent Algorithm

Definition: Gradient Descent

An iterative first-order optimization algorithm for finding local minima of differentiable functions

Algorithm:

1. **Initialize:** $\boldsymbol{\theta}_0$ (random or educated guess)
2. **For** $t = 0, 1, 2, \ldots$ 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 convergence: $|\mathbf{g}_t| < \epsilon$ or $|f(\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{\theta}_t)| < \epsilon$

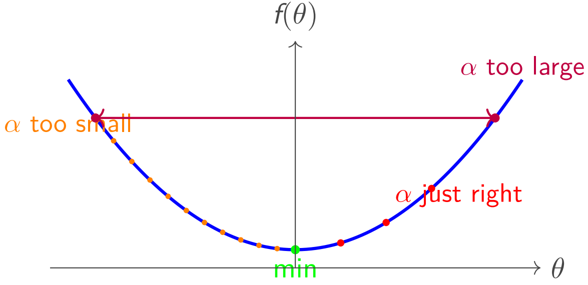
Key Points:

Key Properties:

- First-order method (uses gradients, not Hessians)

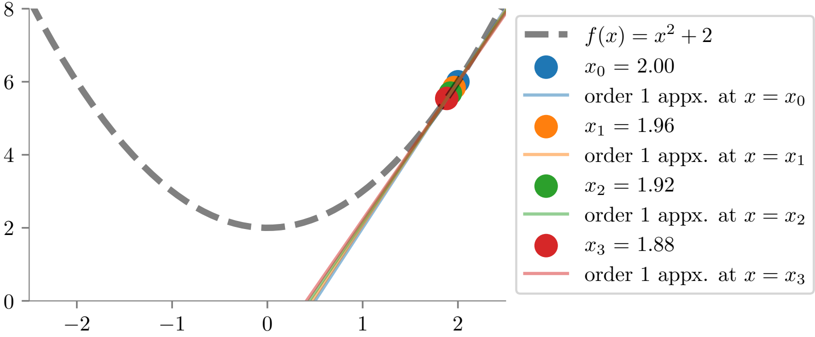
The Learning Rate: Your Step Size

The learning rate α controls how big steps we take



Learning Rate: Too Small ($\alpha = 0.01$)

Convergence is slow but stable

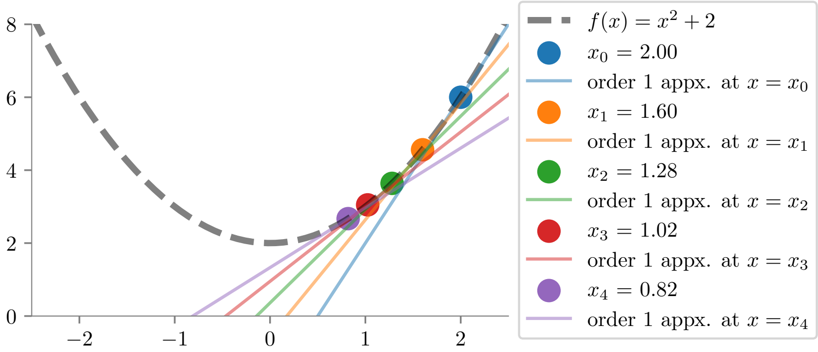


Important: Problem

Takes many iterations to reach the minimum. Computationally expensive!

Learning Rate: Just Right ($\alpha = 0.1$)

Good balance: Fast and stable convergence

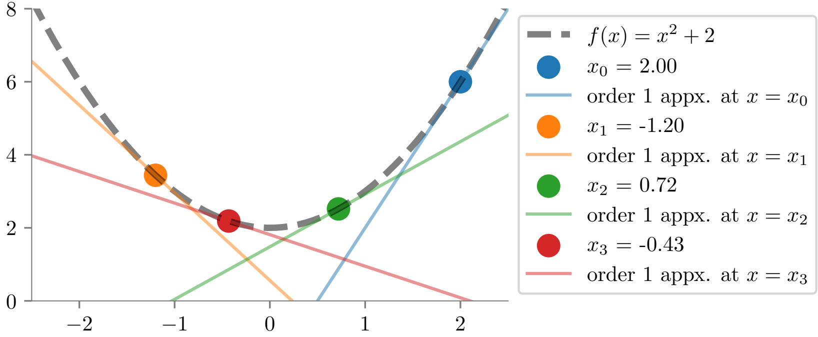


Key Points: T

his is often the sweet spot for many problems!

Learning Rate: Too Large ($\alpha = 0.8$)

Fast but may overshoot

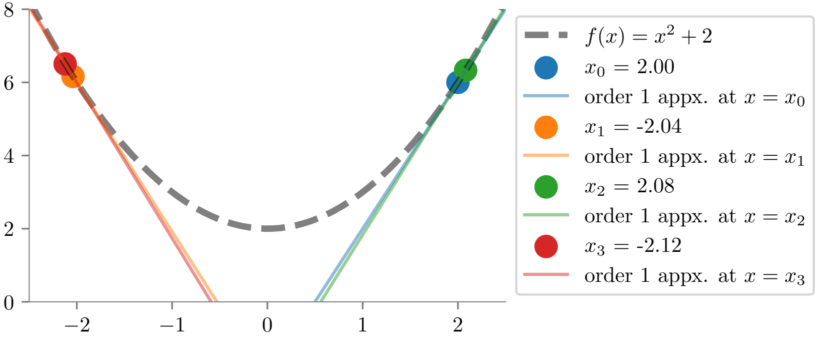


Important: Warning

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 Real Application

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

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

Linear Regression: Our First Real 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 Real 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.

Gradient Descent: Step-by-Step Example

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

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$
- $\frac{\partial \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$
- $\theta_0 = 4 - 0.1 \times 4 = 3.6$

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$
- $\frac{\partial \,\mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \,\mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$
- $\theta_0 = 4 - 0.1 \times 4 = 3.6$
- $\theta_1 = 0 - 0.1 \times 6.67 = -0.67$

Gradient Descent: Step-by-Step Example

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

- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$
- $\theta_0 = 4 - 0.1 \times 4 = 3.6$
- $\theta_1 = 0 - 0.1 \times 6.67 = -0.67$

Gradient Descent: Step-by-Step Example

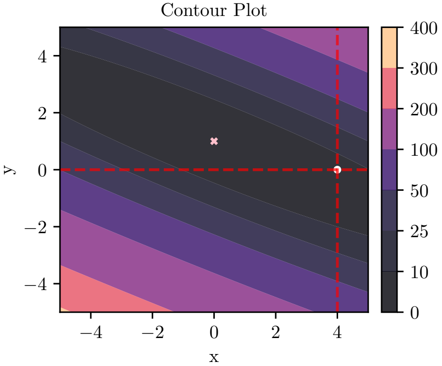
Initial values: $\theta_0 = 4, \theta_1 = 0$, **Learning rate:** $\alpha = 0.1$
Iteration 1:

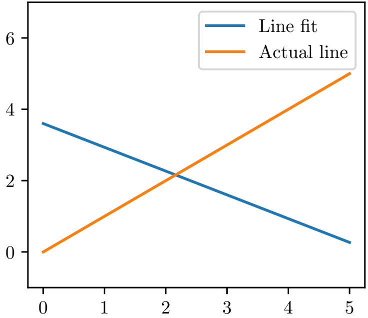
- Predictions: $\hat{y}_1 = 4, \hat{y}_2 = 4, \hat{y}_3 = 4$
- Errors: $\epsilon_1 = 1 - 4 = -3, \epsilon_2 = 2 - 4 = -2, \epsilon_3 = 3 - 4 = -1$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_0} = -\frac{2}{3}(-3 - 2 - 1) = 4$
- $\frac{\partial \, \mathrm{MSE}}{\partial \theta_1} = -\frac{2}{3}(-3 \cdot 1 - 2 \cdot 2 - 1 \cdot 3) = 6.67$
- $\theta_0 = 4 - 0.1 \times 4 = 3.6$
- $\theta_1 = 0 - 0.1 \times 6.67 = -0.67$

New parameters: $(\theta_0, \theta_1) = (3.6, -0.67)$

Visual Journey: Gradient Descent in Action

Let's watch gradient descent navigate the loss landscape:

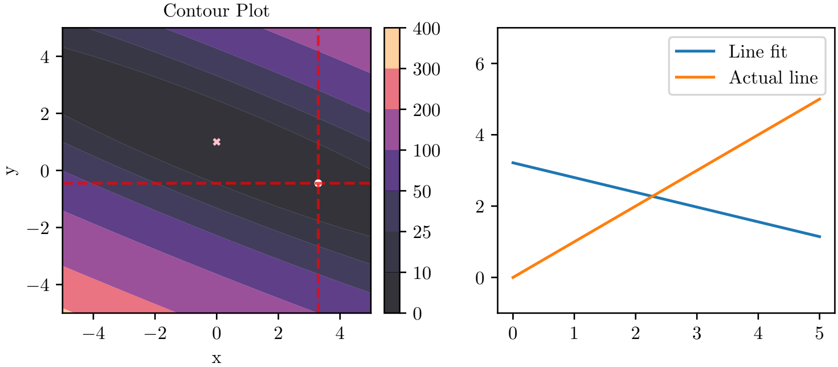




Notice: The algorithm takes larger steps when gradient is large, smaller steps as it approaches the minimum!

Visual Journey: Gradient Descent in Action

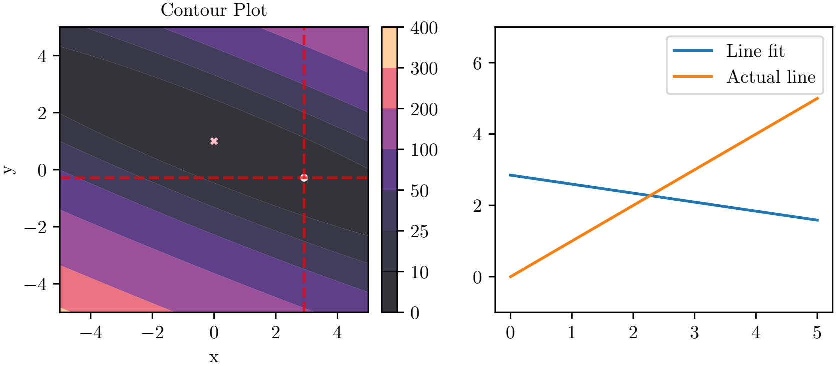
Let's watch gradient descent navigate the loss landscape:



Notice: The algorithm takes larger steps when gradient is large, smaller steps as it approaches the minimum!

Visual Journey: Gradient Descent in Action

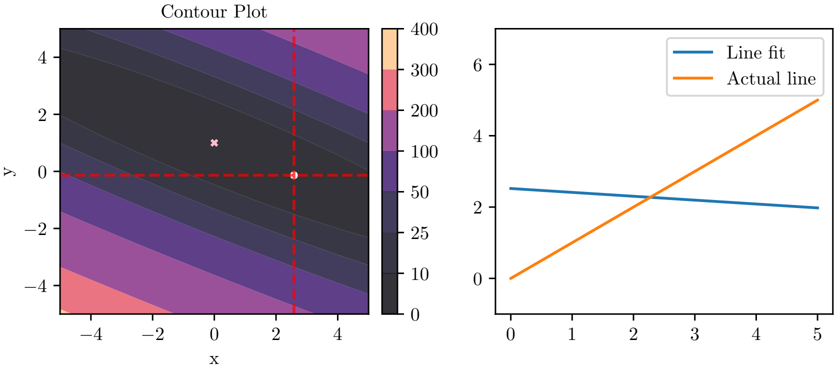
Let's watch gradient descent navigate the loss landscape:



Notice: The algorithm takes larger steps when gradient is large, smaller steps as it approaches the minimum!

Visual Journey: Gradient Descent in Action

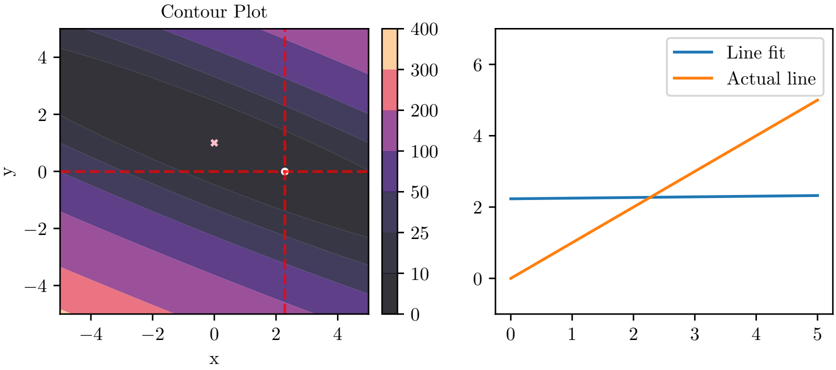
Let's watch gradient descent navigate the loss landscape:



Notice: The algorithm takes larger steps when gradient is large, smaller steps as it approaches the minimum!

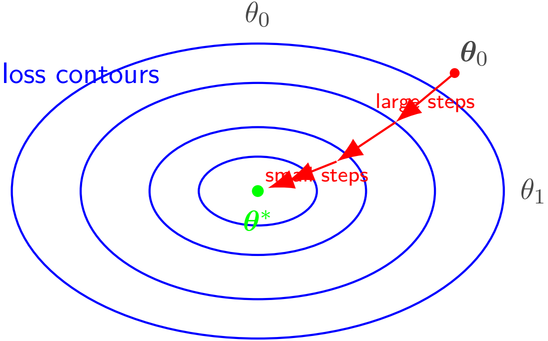
Visual Journey: Gradient Descent in Action

Let's watch gradient descent navigate the loss landscape:



Notice: The algorithm takes larger steps when gradient is large, smaller steps as it approaches the minimum!

Visual: GD Path on Loss Surface (TikZ Version)



Notice: Algorithm takes larger steps when gradient is large!

Variants of Gradient Descent

The Gradient Descent Family

Three main variants based on how much data we use per update:

Definition: Batch Gradient Descent (GD)

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 (MBGD)

Use a small batch of samples to compute each gradient

The Gradient Descent Family

Three main variants based on how much data we use per update:

Definition: Batch Gradient Descent (GD)

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 (MBGD)

Use a small batch of samples to compute each gradient

Trade-offs: Computational cost vs. convergence stability vs. memory usage

Batch vs Stochastic vs Mini-batch

Method	**Data per update**	**Updates per epoch**	**Converg**
Batch GD	n (all)	1	Smoo
SGD	1	n	Nois
Mini-batch GD	b (batch size)	n/b	Balanc

Batch vs Stochastic vs Mini-batch

Method	Data per update	Updates per epoch	Converg
Batch GD	n (all)	1	Smoo
SGD	1	n	Nois
Mini-batch GD	b (batch size)	n/b	Balanc

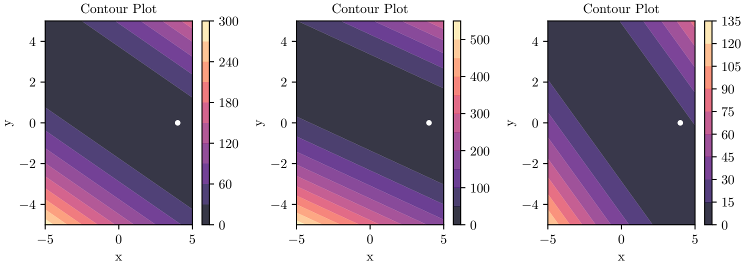
Key Points:

Modern ML: Mini-batch GD with batch sizes 32-256 is most common

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

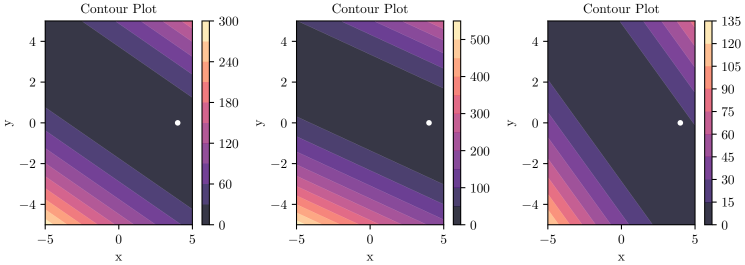
Stochastic Gradient Descent: The Noisy Path

SGD uses one sample at a time for updates



Stochastic Gradient Descent: The Noisy Path

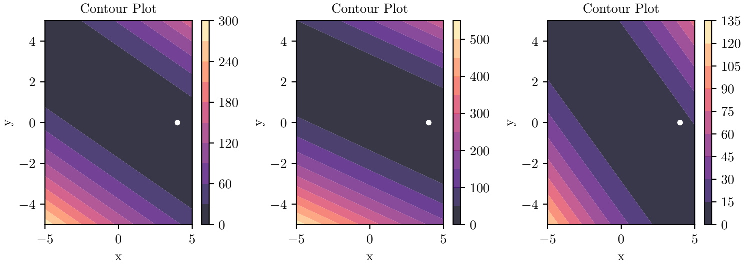
SGD uses one sample at a time for updates



- **Pro:** Fast updates, can escape local minima due to noise

Stochastic Gradient Descent: The Noisy Path

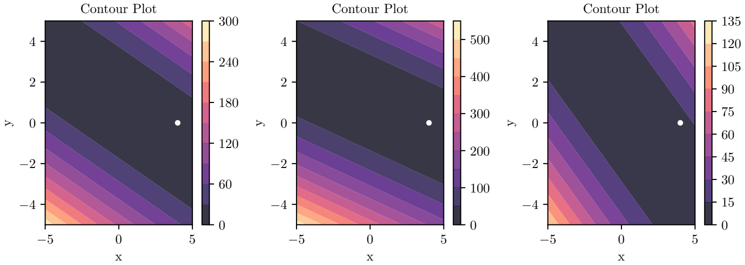
SGD uses one sample at a time for updates



- **Pro:** Fast updates, can escape local minima due to noise
- **Con:** Noisy convergence, may never reach exact minimum

Stochastic Gradient Descent: The Noisy Path

SGD uses one sample at a time for updates



- **Pro:** Fast updates, can escape local minima due to noise
- **Con:** Noisy convergence, may never reach exact minimum
- **Key insight:** The noise can be beneficial for non-convex problems!

Epochs vs Iterations: Important Distinction

Definition: Iteration

One parameter update step (one gradient computation and update)

Definition: Epoch

One complete pass through the entire training dataset

Epochs vs Iterations: Important Distinction

Definition: Iteration

One parameter update step (one gradient computation and update)

Definition: Epoch

One complete pass through the entire training dataset

For dataset with 1000 samples:

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

Epochs vs Iterations: Important Distinction

Definition: Iteration

One parameter update step (one gradient computation and update)

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: Important Distinction

Definition: Iteration

One parameter update step (one gradient computation and update)

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 (batch size 100):** 10 iterations = 1 epoch

Epochs vs Iterations: Important Distinction

Definition: Iteration

One parameter update step (one gradient computation and update)

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 (batch size 100):** 10 iterations = 1 epoch

Epochs vs Iterations: Important Distinction

Definition: Iteration

One parameter update step (one gradient computation and update)

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 (batch size 100):** 10 iterations = 1 epoch

Important: Important

Mathematical Properties

SGD as an 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 an 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 gradient estimate: $\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)$, where $(\mathbf{x}, y)$ is sampled uniformly from $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$.

SGD as an 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 gradient estimate: $\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)$, where $(\mathbf{x}, y)$ is sampled uniformly from $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$.

Theorem: Unbiased Estimator Property

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

SGD as an 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 gradient estimate: $\nabla \tilde{L}(\boldsymbol{\theta}) = \nabla \ell(f(\mathbf{x}; \boldsymbol{\theta}), y)$, where $(\mathbf{x}, y)$ is sampled uniformly from $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$.

Theorem: Unbiased Estimator Property

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

Proof:

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

Why Unbiasedness Matters

Key Points:

Unbiased means: On average, SGD points in the right direction!

Why Unbiasedness Matters

Key Points:

Unbiased means: On average, SGD points in the right direction!

Implications:

- Individual SGD steps might be "wrong", but they average to the correct direction

Why Unbiasedness Matters

Key Points:

Unbiased means: On average, SGD points in the right direction!

Implications:

- Individual SGD steps might be "wrong", but they average to the correct direction
- This theoretical guarantee justifies why SGD works in practice

Why Unbiasedness Matters

Key Points:

Unbiased means: On average, SGD points in the right direction!

Implications:

- Individual SGD steps might be “wrong”, but they average to the correct direction
- This theoretical guarantee justifies why SGD works in practice
- The noise in SGD can actually help escape local minima

Why Unbiasedness Matters

Key Points:

Unbiased means: On average, SGD points in the right direction!

Implications:

- Individual SGD steps might be “wrong”, but they average to the correct direction
- This theoretical guarantee justifies why SGD works in practice
- The noise in SGD can actually help escape local minima

Why Unbiasedness Matters

Key Points:

Unbiased means: On average, SGD points in the right direction!

Implications:

- Individual SGD steps might be "wrong", but they average to the correct direction
- This theoretical guarantee justifies why SGD works in practice
- The noise in SGD can actually help escape local minima

Example: Intuitive Analogy

Imagine asking random people for directions to a destination:

- Individual answers might be slightly off

Computational Complexity

Computational Complexity: GD vs Normal Equation

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 complexity: $\mathcal{O}(d^2 n + d^3)$

Key Points: Gradient Descent

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

Computational Complexity: GD vs Normal Equation

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 complexity: $\mathcal{O}(d^2n + d^3)$

Key Points: Gradient Descent

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

When to use which?

- **Few features (d small):** Normal equation

Computational Complexity: GD vs Normal Equation

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 complexity: $\mathcal{O}(d^2 n + d^3)$

Key Points: Gradient Descent

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

When to use which?

- **Few features (d small):** Normal equation
- **Many features (d large):** Gradient descent

Computational Complexity: GD vs Normal Equation

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 complexity: $\mathcal{O}(d^2n + d^3)$

Key Points: Gradient Descent

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

When to use which?

- **Few features (d small):** Normal equation
- **Many features (d large):** Gradient descent
- **Non-linear models:** Only gradient descent works

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Normal Equation (one-time):

- Compute $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^2 n)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Normal Equation (one-time):

- Compute $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^2 n)$
- Invert $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^3)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Normal Equation (one-time):

- Compute $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^2 n)$
- Invert $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^3)$
- Compute $\mathbf{X}^T\mathbf{y}$: $\mathcal{O}(dn)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Normal Equation (one-time):

- Compute $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^2 n)$
- Invert $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^3)$
- Compute $\mathbf{X}^T\mathbf{y}$: $\mathcal{O}(dn)$
- Final multiplication: $\mathcal{O}(d^2)$

Complexity Analysis: Breaking It Down

Gradient Descent per iteration:

- Compute $\mathbf{X}\boldsymbol{\theta}$: $\mathcal{O}(nd)$
- Compute residual $\mathbf{X}\boldsymbol{\theta} - \mathbf{y}$: $\mathcal{O}(n)$
- Compute $\mathbf{X}^T$ (residual): $\mathcal{O}(nd)$
- Update $\boldsymbol{\theta}$: $\mathcal{O}(d)$
- **Total per iteration:** $\mathcal{O}(nd)$

Normal Equation (one-time):

- Compute $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^2 n)$
- Invert $\mathbf{X}^T\mathbf{X}$: $\mathcal{O}(d^3)$
- Compute $\mathbf{X}^T\mathbf{y}$: $\mathcal{O}(dn)$
- Final multiplication: $\mathcal{O}(d^2)$
- **Total:** $\mathcal{O}(d^2 n + d^3)$

Pop Quiz #2: Complexity Comparison

Answer this!

You have a dataset with $n = 10^6$ samples and $d = 10^3$ features.

Questions:

1. What's the complexity of normal equation?
2. What's the complexity of 100 GD iterations?
3. Which method would you choose and why?
4. What if $d = 10^6$ instead?

Pop Quiz #2: Solutions

Example: Solutions

For $n = 10^6$, $d = 10^3$:

1. Normal equation:
   $O(d^2 n + d^3) = O(10^{12} + 10^9) = O(10^{12})$
2. 100 GD iterations: $O(100 \cdot dn) = O(10^{11})$
3. Choose GD ($10\times$ faster)
4. If $d = 10^6$: Normal equation becomes $O(10^{18})$, GD becomes $O(10^{14})$ – definitely choose GD!

Advanced Topics and Extensions

Beyond Basic Gradient Descent

Modern deep learning uses advanced optimizers:

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

Beyond Basic Gradient Descent

Modern deep learning uses advanced optimizers:

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

Beyond Basic Gradient Descent

Modern deep learning uses advanced optimizers:

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

Beyond Basic Gradient Descent

Modern deep learning uses advanced optimizers:

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

Beyond Basic Gradient Descent

Modern deep learning uses advanced optimizers:

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

Beyond Basic Gradient Descent

Modern deep learning uses advanced optimizers:

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

Example: Why These Improvements?

- Handle different parameter scales automatically
- Accelerate convergence in relevant directions
- Reduce oscillations in narrow valleys
- Better performance on non-convex landscapes

Gradient Descent in Deep Learning

Key Points: E

very modern deep learning framework uses gradient descent variants!

Gradient Descent in Deep Learning

Key Points: E

very modern deep learning framework uses gradient descent variants!

Key extensions:

- **Backpropagation:** Efficient gradient computation for neural networks

Gradient Descent in Deep Learning

Key Points: E

very modern deep learning framework uses gradient descent variants!

Key extensions:

- **Backpropagation:** Efficient gradient computation for neural networks
- **Automatic differentiation:** PyTorch, TensorFlow handle gradients automatically

Gradient Descent in Deep Learning

Key Points: E

very modern deep learning framework uses gradient descent variants!

Key extensions:

- **Backpropagation:** Efficient gradient computation for neural networks
- **Automatic differentiation:** PyTorch, TensorFlow handle gradients automatically
- **GPU acceleration:** Parallel computation of mini-batch gradients

Gradient Descent in Deep Learning

Key Points: E

very modern deep learning framework uses gradient descent variants!

Key extensions:

- **Backpropagation:** Efficient gradient computation for neural networks
- **Automatic differentiation:** PyTorch, TensorFlow handle gradients automatically
- **GPU acceleration:** Parallel computation of mini-batch gradients
- **Mixed precision:** Use both 16-bit and 32-bit arithmetic

Practical Considerations

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Common strategies:

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

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Common strategies:

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

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Common strategies:

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

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Common strategies:

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

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Common strategies:

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

Choosing Learning Rates: Practical Tips

Key Points: L

earning rate selection is more art than science!

Common strategies:

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

Important: Warning Signs

- Loss exploding $\rightarrow$ Learning rate too high
- Very slow convergence $\rightarrow$ Learning rate too low
- Oscillating loss $\rightarrow$ Try smaller learning rate or

Convergence Criteria: When to Stop?

Common stopping criteria:

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

Convergence Criteria: When to Stop?

Common stopping criteria:

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

Convergence Criteria: When to Stop?

Common stopping criteria:

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

Common stopping criteria:

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

Convergence Criteria: When to Stop?

Common stopping criteria:

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

Convergence Criteria: When to Stop?

Common stopping criteria:

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

Example: Practical Advice

- Always set a maximum iteration limit
- Monitor multiple criteria simultaneously
- Use validation set performance in practice
- Early stopping prevents overfitting

Common Pitfalls and How to Avoid Them

Important: Pitfall 1: Poor Initialization

Problem: Starting at bad points (e.g., all zeros)
Solution: Use Xavier/He initialization for neural networks

Common Pitfalls and How to Avoid Them

Important: Pitfall 1: Poor Initialization

Problem: Starting at bad points (e.g., all zeros)
Solution: Use Xavier/He initialization for neural networks

Important: Pitfall 2: Learning Rate Too High/Low

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

Common Pitfalls and How to Avoid Them

Important: Pitfall 1: Poor Initialization

Problem: Starting at bad points (e.g., all zeros)
Solution: Use Xavier/He initialization for neural networks

Important: Pitfall 2: Learning Rate Too High/Low

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

Important: Pitfall 3: Poor Feature Scaling

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

Summary and Key Takeaways

What We've Learned: The Big Picture

Key Points: G

radient descent is the foundation of modern machine learning optimization!

What We've Learned: The Big Picture

Key Points: G

radient descent is the foundation of modern machine learning optimization!

Core concepts:

- **Mathematical foundation:** Taylor series approximation

What We've Learned: The Big Picture

Key Points: G

radient descent is the foundation of modern machine learning optimization!

Core concepts:

- **Mathematical foundation:** Taylor series approximation
- **Geometric intuition:** Follow steepest descent direction

What We've Learned: The Big Picture

Key Points: G

radient descent is the foundation of modern machine learning optimization!

Core concepts:

- **Mathematical foundation:** Taylor series approximation
- **Geometric intuition:** Follow steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch

What We've Learned: The Big Picture

Key Points: G

radient descent is the foundation of modern machine learning optimization!

Core concepts:

- **Mathematical foundation:** Taylor series approximation
- **Geometric intuition:** Follow steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch
- **Theoretical properties:** SGD is unbiased estimator

What We've Learned: The Big Picture

Key Points: G

radient descent is the foundation of modern machine learning optimization!

Core concepts:

- **Mathematical foundation:** Taylor series approximation
- **Geometric intuition:** Follow steepest descent direction
- **Algorithm variants:** Batch, SGD, mini-batch
- **Theoretical properties:** SGD is unbiased estimator
- **Practical considerations:** Learning rates, convergence criteria

From Theory to Practice: Next Steps

Practice opportunities:

- Implement gradient descent from scratch

From Theory to Practice: Next Steps

Practice opportunities:

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

From Theory to Practice: Next Steps

Practice opportunities:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Try different optimization functions

From Theory to Practice: Next Steps

Practice opportunities:

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

From Theory to Practice: Next Steps

Practice opportunities:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Try different optimization functions
- Compare batch vs SGD vs mini-batch
- Visualize convergence paths

From Theory to Practice: Next Steps

Practice opportunities:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Try different optimization functions
- Compare batch vs SGD vs mini-batch
- Visualize convergence paths

From Theory to Practice: Next Steps

Practice opportunities:

- Implement gradient descent from scratch
- Experiment with different learning rates
- Try different optimization functions
- Compare batch vs SGD vs mini-batch
- Visualize convergence paths

Key Points: M

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

Pop Quiz #3: Comprehensive Review

Answer this!

True or False?

1. SGD always converges faster than batch gradient descent
2. The learning rate should decrease as training progresses
3. SGD gradient estimates are unbiased
4. Normal equation is always better than gradient descent
5. Gradient descent can only find global minima

Pop Quiz #3: Solutions

Example: Solutions

1. **False** - SGD converges faster per epoch but may need more epochs
2. **True** - Learning rate schedules often improve convergence
3. **True** - This is the key theoretical property of SGD
4. **False** - Normal equation only works for linear problems and small d
5. **False** - GD finds local minima; global minima only guaranteed for convex functions

Looking Ahead: Advanced Optimization

What's next in optimization?

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

Looking Ahead: Advanced Optimization

What's next in optimization?

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

Looking Ahead: Advanced Optimization

What's next in optimization?

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

Looking Ahead: Advanced Optimization

What's next in optimization?

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers, KKT conditions
- **Global optimization:** Simulated annealing, genetic algorithms
- **Distributed optimization:** Federated learning, parameter servers

Looking Ahead: Advanced Optimization

What's next in optimization?

- **Second-order methods:** Newton's method, L-BFGS
- **Constrained optimization:** Lagrange multipliers, KKT conditions
- **Global optimization:** Simulated annealing, genetic algorithms
- **Distributed optimization:** Federated learning, parameter servers
- **Meta-learning:** Learning to optimize

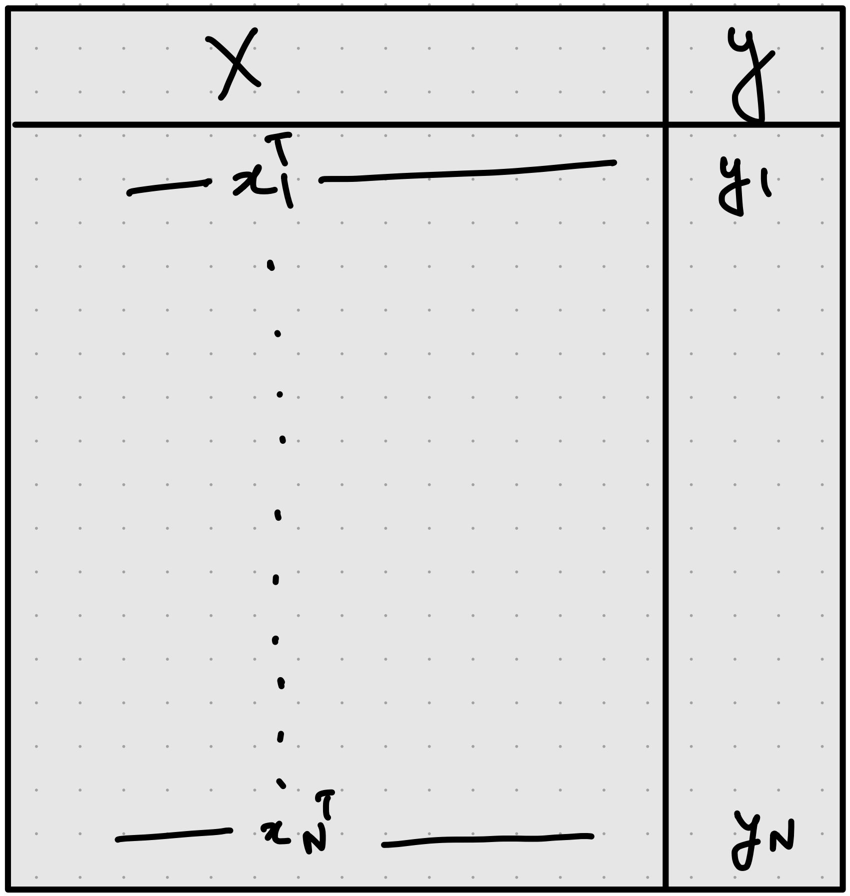
Additional Resources: SGD Deep Dive

For detailed mathematical analysis and proofs:

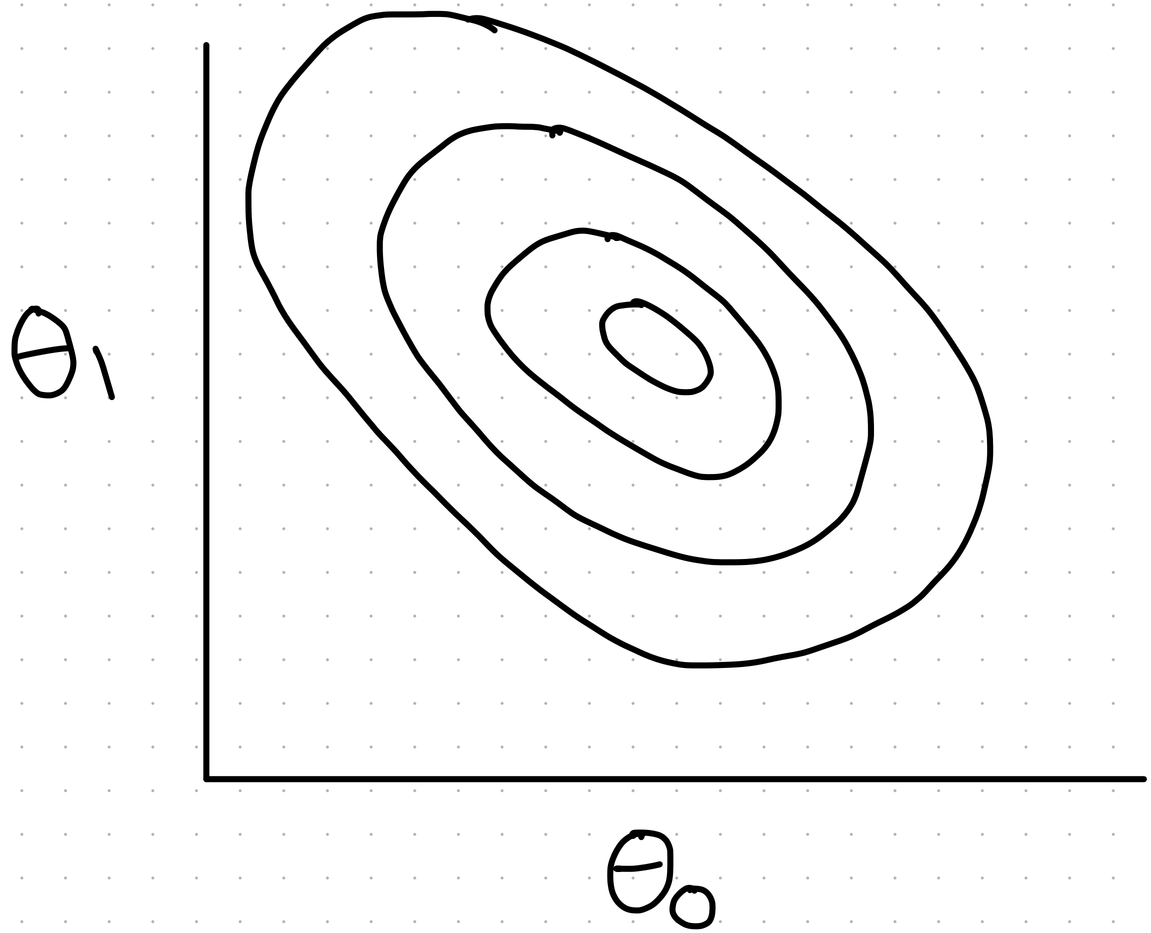
Important: Reference Material

See SGD.pdf in the assets folder for:

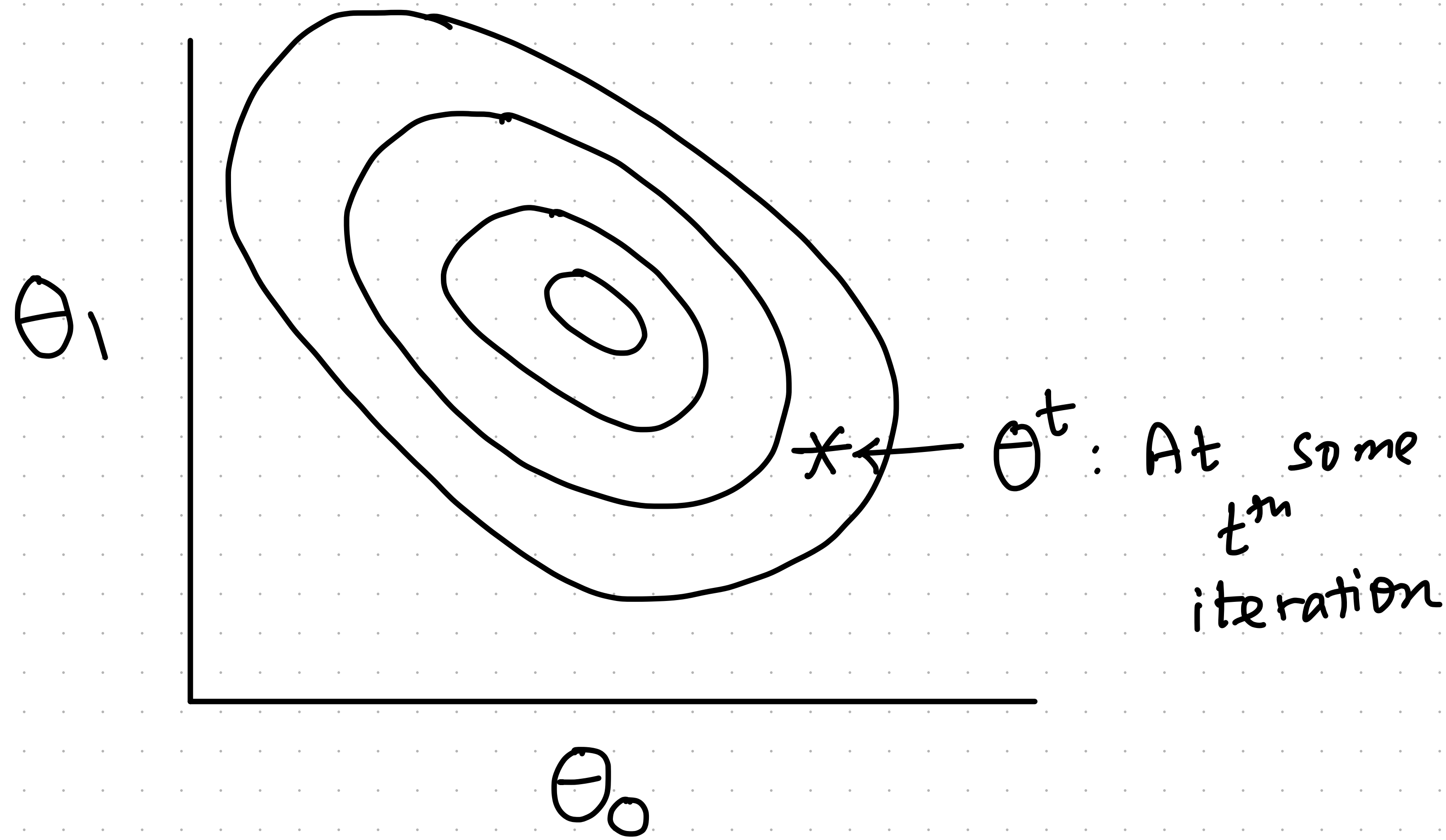
- Formal convergence proofs
- Variance analysis of SGD
- Advanced theoretical properties
- Comparison with other optimization methods


X
y
x_1^T
y_1
x_N^T
y_N

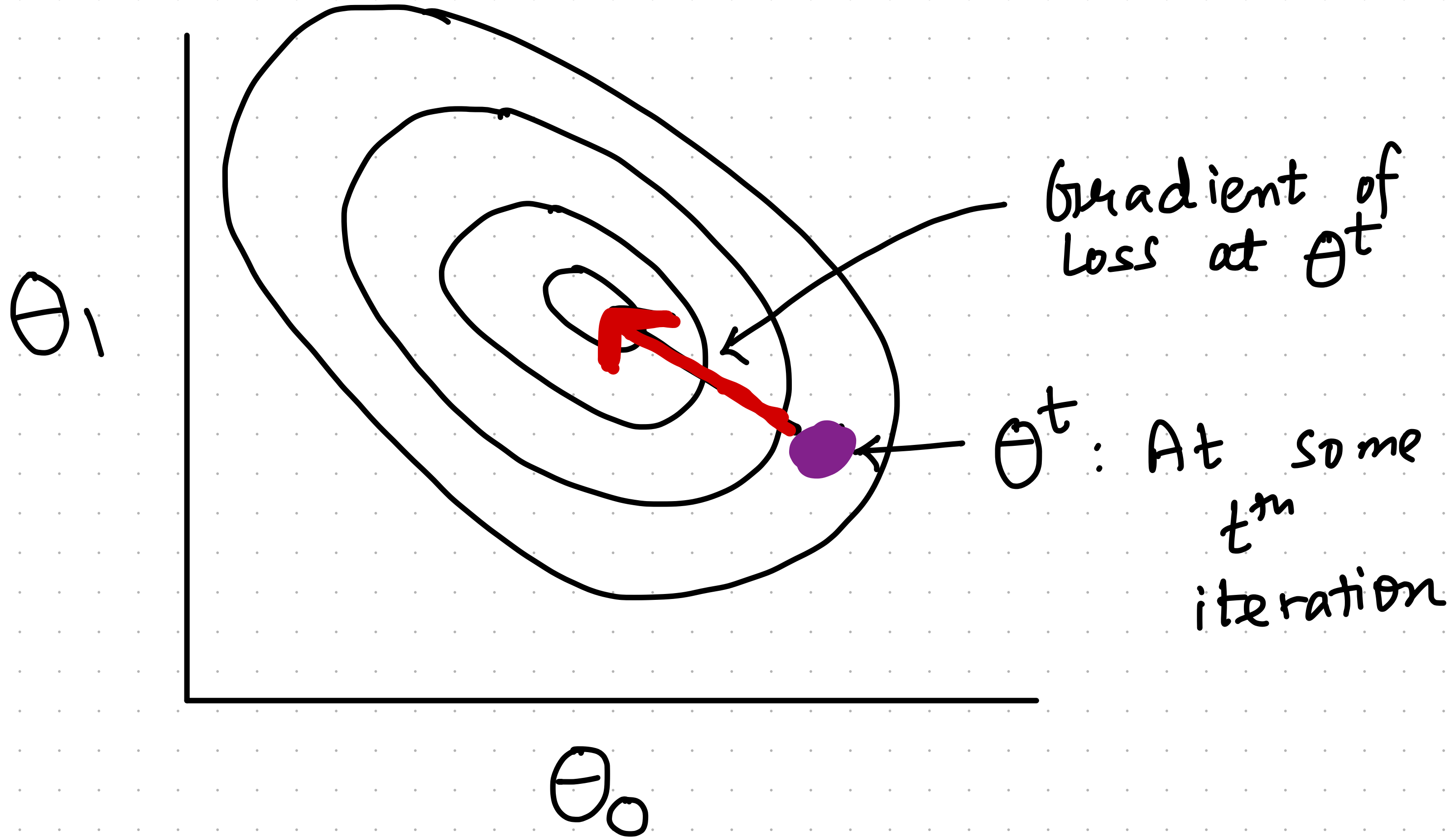
X	y	$\hat{y} = f(x, \theta)$
— x_1^T —	y_1	$\hat{y}_1$
$\vdots$		$\vdots$
— x_N^T —	y_N	$\hat{y}_N$



LOSS SURFACE OVER
'N' EXAMPLES



LOSS SURFACE OVER
"N" EXAMPLES



LOSS SURFACE OVER
6N EXAMPLES

Thank You!

Questions?

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