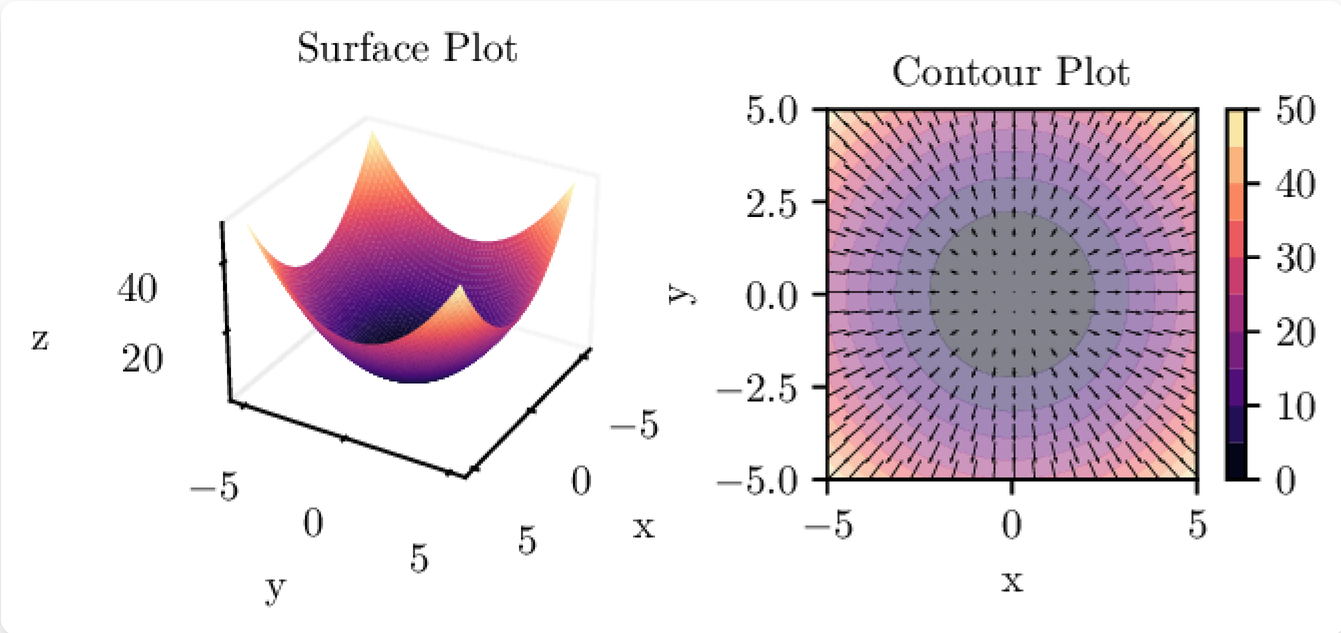
Gradient Descent

Nipun Batra, IIT Gandhinagar

Revision: Contour Plot And Gradients

$z = f(x, y) = x^2 + y^2$



Gradient denotes the direction of steepest ascent or the direction in which there is a maximum increase in fx, y

Optimization Algorithms

Core Concepts

- We often want to minimize/maximize a function
- We wanted to minimize the cost function:

$$f(\theta) = (y - X\theta)^T (y - X\theta)$$

- Note: here θ is the parameter vector

General Components

- **Maximize or Minimize** a function subject to some constraints
- Today, we focus on **unconstrained optimization** $noconstraints$
- We focus on **minimization**
- **Goal**:

Introduction to Gradient Descent

Key Properties

- Gradient descent is an **optimization algorithm**
- Used to find the minimum of a function in unconstrained settings
- It is an **iterative algorithm**
- It is a **first order** optimization algorithm
- It is a **local search algorithm/greedy**

Algorithm Steps

1. **Initialize** θ to some random value
2. **Compute** the gradient of the cost function at θ: $\nabla f(\theta)$
3. **For Iteration** $i \, where\$i = 1, 2, \ldots \$$ or until convergence:
4. $\theta_i \leftarrow \theta_{i-1} - \alpha \nabla f(\theta_{i-1})$

Taylor's Series Foundation

Basic Form

Taylor's series approximates a function $f(x)$ around point x_0 using a polynomial:

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

Vector Form

$$f(\vec{x}) = f(\vec{x_0}) + \nabla f(\vec{x_0})^T(\vec{x} - \vec{x_0}) + \frac{1}{2}(\vec{x} - \vec{x_0})^T \nabla^2 f(\vec{x_0})(\vec{x} - \vec{x_0}) + \ldots$$

where $\nabla^2 f(\vec{x_0})$ is the **Hessian matrix** and $\nabla f(\vec{x_0})$ is the **gradient vector**

बोध
KNOWLEDGE

From Taylor's Series to Gradient Descent

Minimization Logic

- **Goal**: Find $\Delta\vec{x}$ such that $f(\vec{x}_0 + \Delta\vec{x})$ is minimized
- This is equivalent to minimizing $f(\vec{x}_0) + \nabla f(\vec{x}_0)^T \Delta\vec{x}$
- This happens when vectors $\nabla f(\vec{x}_0)$ and $\Delta\vec{x}$ are at phase angle of 180°
- **Solution**: $\Delta\vec{x} = -\alpha\nabla f(\vec{x}_0)$ where α is a scalar

The Gradient Descent Update Rule

$$\vec{x}_1 = \vec{x}_0 - \alpha\nabla f(\vec{x}_0)$$

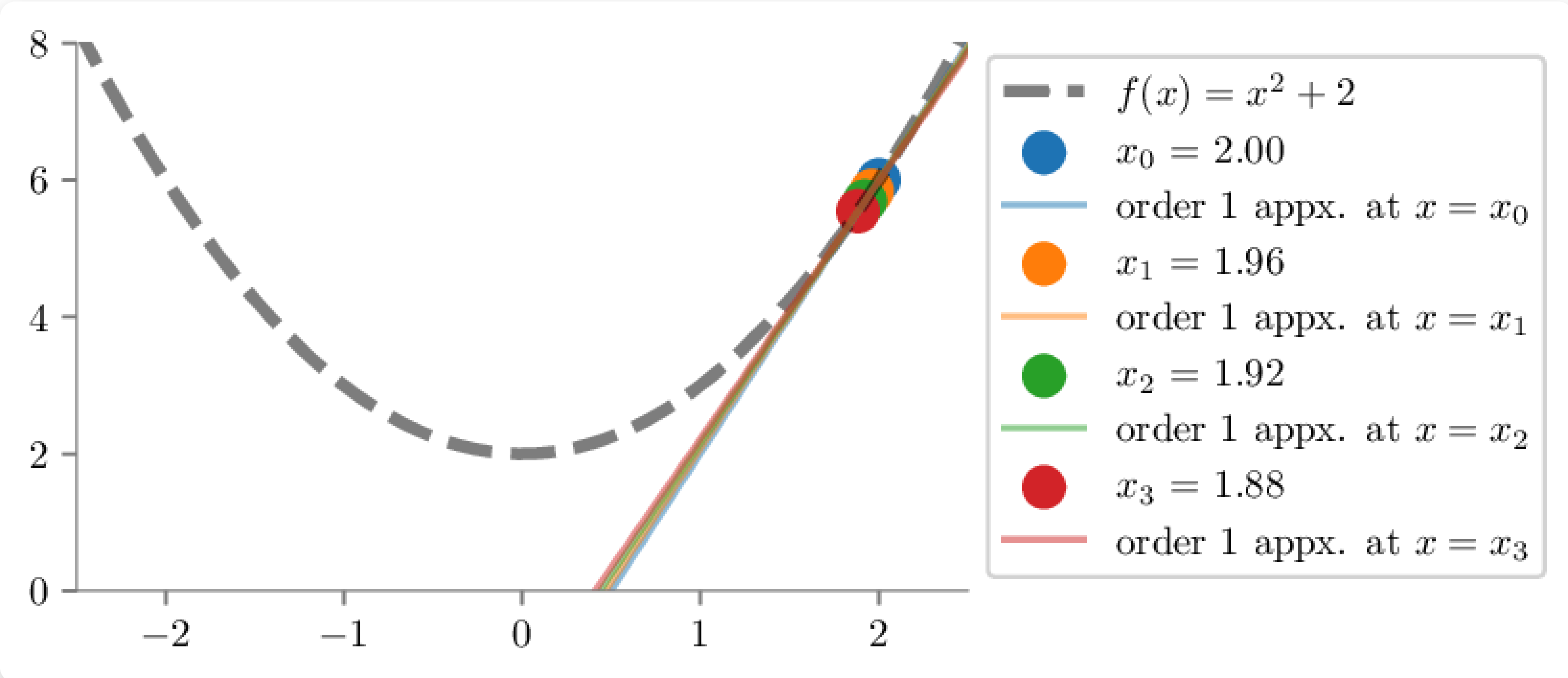
बोध

KNOWLEDGE

Effect of Learning Rate

Low Learning Rate $\alpha = 0.01$

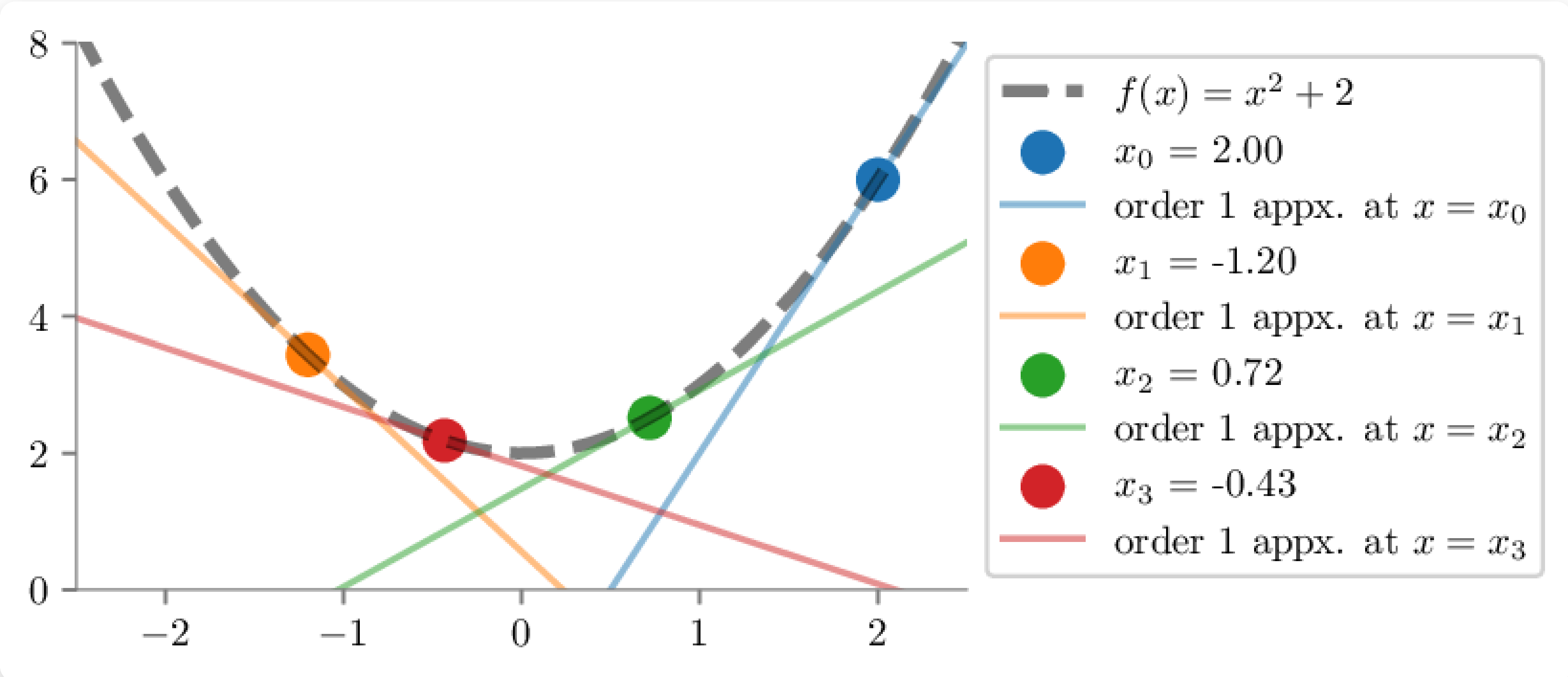
Converges slowly



Effect of Learning Rate

High Learning Rate $\alpha = 0.8$

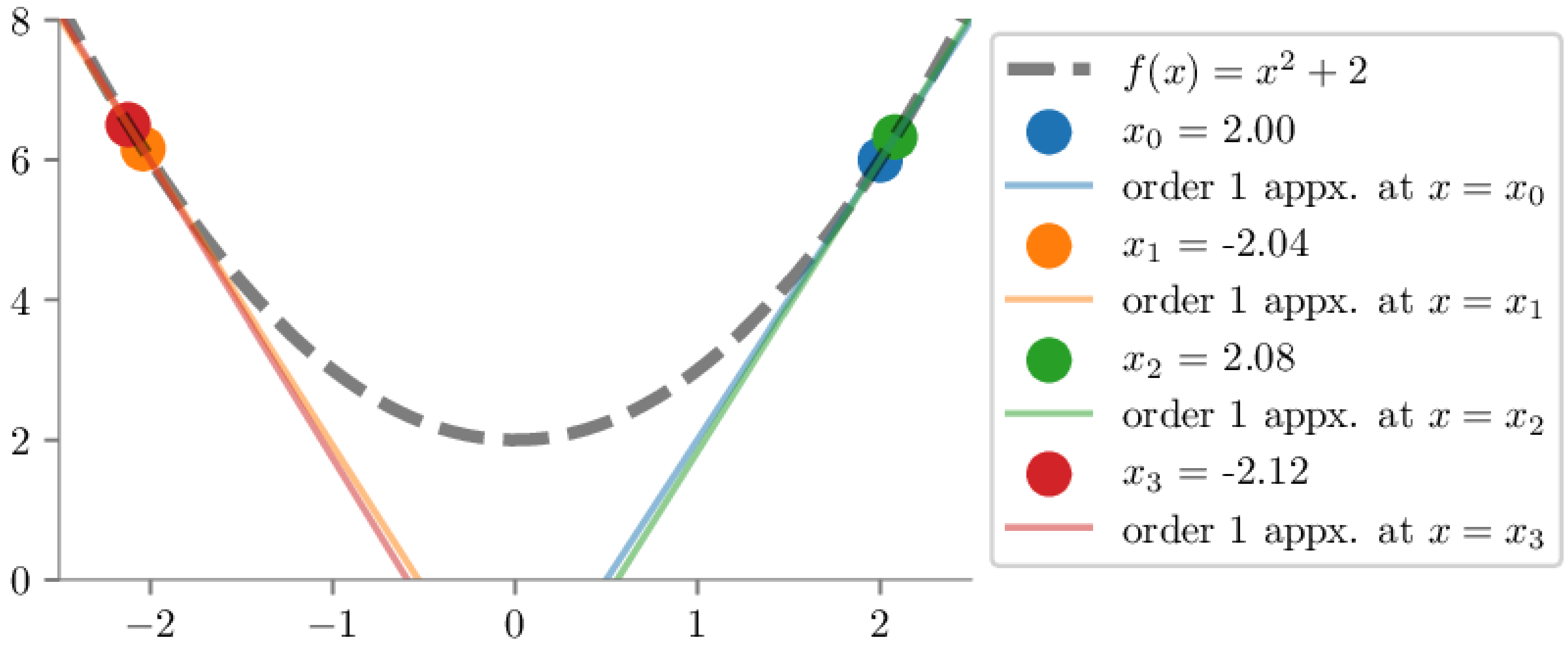
Converges quickly, but might overshoot



Effect of Learning Rate

Very High Learning Rate $\alpha = 1.01$

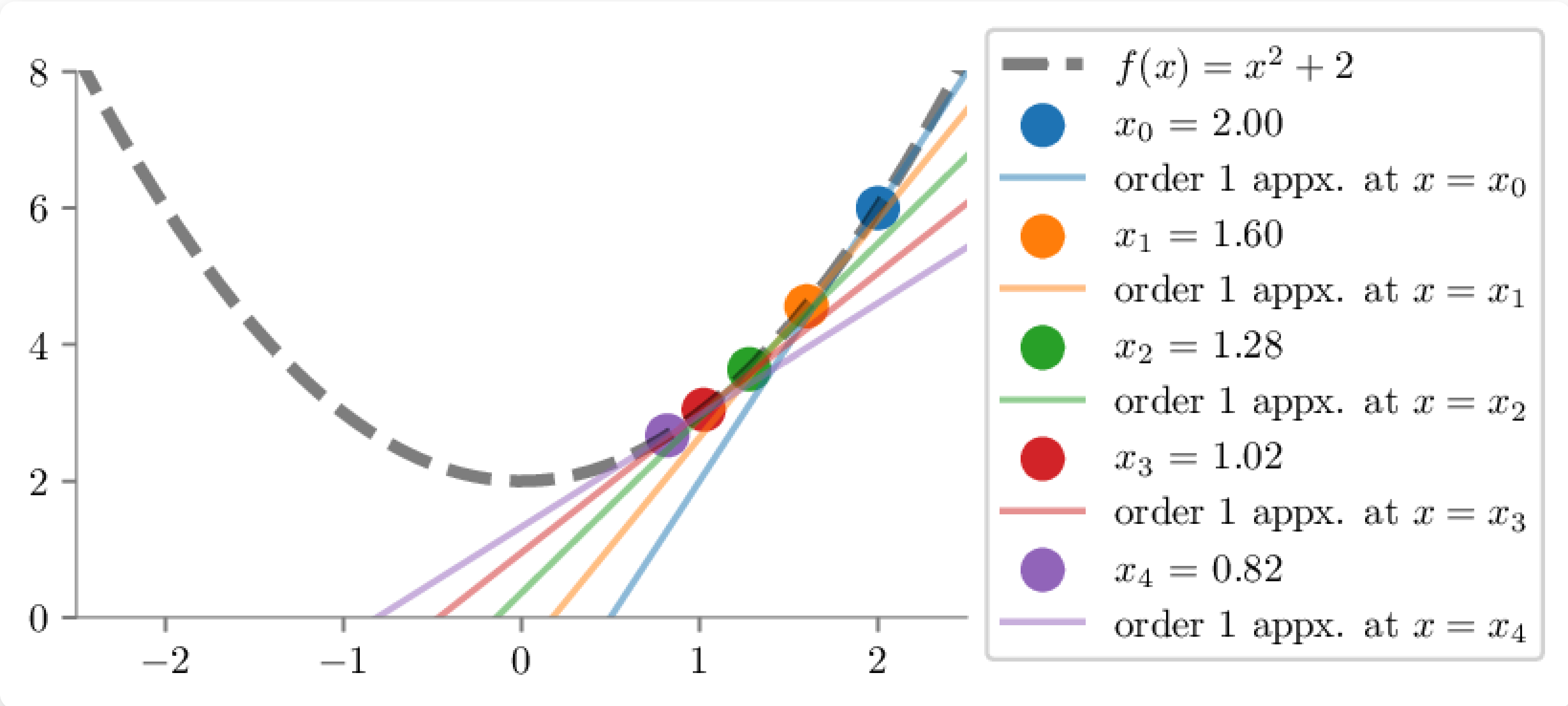
Diverges



Effect of Learning Rate

Appropriate Learning Rate $\alpha = 0.1$

Just right



Terminology: Loss vs Cost vs Objective

Loss Function

- Usually defined on a **data point, prediction and label**
- Measures the penalty
- Example: Square loss $l(f(x_i|\theta), y_i) = (f(x_i|\theta) - y_i)^2$

Cost Function

- More general: **sum of loss functions** over training set plus **model complexity penalty**
- Example: Mean Squared Error $MSE(\theta) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i|\theta) - y_i)^2$

Objective Function

- **Most general term** for any function optimized during training

Gradient Descent Example

Learn $y = \theta_0 + \theta_1 x$ using gradient descent: - Initial: $(\theta_0, \theta_1) = (4, 0)$ - Step-size: $\alpha = 0.1$ - Dataset:

x	y
1	1
2	2
3	3

Error Calculation

- Predictor: $\hat{y} = \theta_0 + \theta_1 x$
- Error for i^{th} datapoint: $\epsilon_i = y_i - \hat{y}_i$

Gradient Computation

Partial Derivatives

$$\frac{\partial MSE}{\partial \theta_0} = \frac{2\sum_i (y_i - \theta_0 - \theta_1 x_i)(-1)}{N} = \frac{2\sum_i \epsilon_i (-1)}{N}$$

$$\frac{\partial MSE}{\partial \theta_1} = \frac{2\sum_i (y_i - \theta_0 - \theta_1 x_i)(-x_i)}{N} = \frac{2\sum_i \epsilon_i (-x_i)}{N}$$

Update Rules

$$\theta_0 = \theta_0 - \alpha \frac{\partial MSE}{\partial \theta_0}$$

$$\theta_1 = \theta_1 - \alpha \frac{\partial MSE}{\partial \theta_1}$$

Algorithm Variants

Gradient Descent GD

- Dataset: $D = (X, y)$ of size N
- **For each epoch:**
- Predict $\hat{y} = pred(X, \theta)$
- Compute loss: $J(\theta) = loss(y, \hat{y})$
- Compute gradient: $\nabla J(\theta) = grad(J)(\theta)$
- Update: $\theta = \theta - \alpha \nabla J(\theta)$

Stochastic Gradient Descent SGD

- **For each epoch:**
- Shuffle D
- **For each sample** i in $[1, N]$:

SGD vs Gradient Descent

Vanilla Gradient Descent

- Updates parameters **after going through all data**
- **Smooth curve** for Iteration vs Cost
- Takes **more time** per update $computesgradientoverallsamples$

Stochastic Gradient Descent

- Updates parameters **after seeing each point**
- **Noisier curve** for iteration vs cost
- **Less time** per update $gradientoveroneexample$

SGD Contour Visualization

Mathematical Foundation: Unbiased Estimator

True Gradient

For dataset $\mathcal{D} = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} loss(f(x_i, \theta), y_i)$$

True gradient:

$$\nabla L = \frac{1}{n} \sum_{i=1}^{n} \nabla \operatorname{loss}(f(x_i), y_i)$$

SGD Estimator

Computational Complexity Analysis

Normal Equation: $\hat{\theta} = (X^T X)^{-1} X^T y$

For $X \in \mathbb{R}^{N \times D}$: - $X^T X$: $\mathcal{O}(D^2 N)$ - Matrix inversion: $\mathcal{O}(D^3)$ - $X^T y$: $\mathcal{O}(DN)$ - Final multiplication: $\mathcal{O}(D^2)$

Total complexity: $\mathcal{O}(D^2 N + D^3)$

Gradient Descent Complexity

Vectorized update: $\theta = \theta - \alpha X^T(X\theta - y)$

Efficient form: $\theta = \theta - \alpha X^T X\theta + \alpha X^T y$

- Pre-compute $X^T X$ and $X^T y$: $\mathcal{O}(D^2 N)$
- Per iteration: $\mathcal{O}(D^2)$
- For t iterations: $\mathcal{O}(D^2 N + tD^2) = \mathcal{O}((N+t)D^2)$

Alternative form: $\mathcal{O}(NDt)$ per iteration

बोध
KNOWLEDGE

When to Use Which Algorithm?

Normal Equation

- **Good when**: D is small
- **Advantages**: Direct solution, no iterations
- **Disadvantages**: $\mathcal{O}(D^3)$ matrix inversion

Gradient Descent

- **Good when**: D is large or N is large
- **Advantages**: Scales well, iterative improvement
- **Disadvantages**: Requires tuning, local minima

Summary

Key Takeaways

1. **Gradient Descent** is a fundamental optimization algorithm
2. **Learning rate** α is crucial - too small $slow$, too large $divergence$
3. **SGD** provides unbiased estimates with faster per-iteration updates
4. **Computational complexity** depends on problem dimensions
5. **Taylor series** provides theoretical foundation

Applications

- Linear regression
- Logistic regression
- Neural networks
- Any differentiable optimization problem