NUS | Computing
National University of Singapore

Eric Han
eric_han@nus.edu.sg
https://eric-han.com

Computer Science

L01 – 8 May 2025

Introduction to Gradient Descent

PDP – Micro-teaching Component

Content

By the end of this activity, you should be able to:

- **Recognize** why/how gradient descent is essential in ML training
- **Follow** and apply the gradient descent manually on simple functions
- **Adjust** the learning rate to observe and achieve convergence
- **Implement** gradient descent in code and verify it works correctly

Recap

1. Compute Complexity is the measure of the computational time needed to execute an algorithm as a function of the input size.
2. Matrix multiplication: if A is $m \times k$ and B is $k \times m$, then AB is $m \times m$.
3. Matrix inversion: inverting a $m \times m$ matrix takes $O(m^3)$ time.
4. Residual Sum of Squares: $\mathrm{RSS}(X) = \sum_{i=1}^{n} (e_i)^2 = \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i)\right)^2$

Why study Gradient Descent?

Gradient descent and its extensions/variants are used across Machine Learning:

- **Line search**: Adaptive step size based on function values
- **Momentum**: Combines past updates to smooth oscillations
- **Adaptive methods**: Algorithms like Adagrad, Adam improve learning in practice
- **Distributed algorithms**: Enable training at scale across multiple machines
- **Second-order methods**: Use curvature information (e.g., Newton's method), though expensive for large d
- **Zero-th order methods**: Optimize without gradients – useful for black-box or simulation-based problems (e.g., Bayesian optimization)

Takeaway: Gradient descent builds the foundation for understanding modern optimization.

Linear Models

Model between a single 1D input x and output y by learning parameters $\theta = \begin{bmatrix}\beta_0 \\ \beta_1\end{bmatrix}$:

$$y \sim \beta_0 + \beta_1 x \tag{1}$$

We define $x = \begin{bmatrix}1 & x\end{bmatrix}$ and express the function $\hat{f}_\theta(x)$ as:

$$\hat{f}_\theta(x) = \beta_0 + \beta_1 x = \begin{bmatrix}1 & x\end{bmatrix}\begin{bmatrix}\beta_0 \\ \beta_1\end{bmatrix} = x\theta \tag{2}$$

Here, β_1 is the slope and β_0 is the intercept; Here generalized to m-dimensions:

$$\hat{f}_\theta(x) = \beta_0 + \beta_1 x^{(1)} + \dots + \beta_m x^{(m)} = \begin{bmatrix}1 & x^{(1)} & \dots & x^{(m)}\end{bmatrix}\begin{bmatrix}\beta_0 \\ \beta_1 \\ \dots \\ \beta_m\end{bmatrix} = x\theta \tag{3}$$

The simplest way is to consider perfect conditions, then we can use $\hat{\theta} = X^{-1}Y$.

Matrix Inversion Example (1D)

Given $n = 2$ points $(1,\ 3), (4,\ 9)$, we convert to matrix form:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix}, \quad y = \begin{bmatrix} 3 \\ 9 \end{bmatrix}$$

Solve using: $\theta = X^{-1}y$, via calculation of inverse:

$$X^{-1} = \frac{1}{(1)(4) - (1)(1)} \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix}$$

Then, we apply the inverse:

$$\theta = \frac{1}{3} \begin{bmatrix} 4 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 9 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 12 - 9 \\ -3 + 9 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Resulting line: $\hat{y} = 1 + 2x$.

Problems with Matrix Inversion

Noise exists, causing the X^{-1} to be non-invertible.

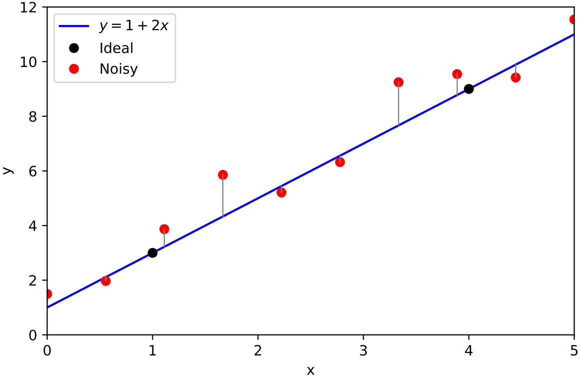

Figure 1: Estimated/Actual line (blue); Error e_i (vertical line).

Estimate $\hat{\theta}$ by minimizing the loss function L: $\mathrm{RSS}(X) = \underbrace{\mathrm{RSS}(\theta)}_{\text{fix } X\text{, find } \theta} = L_{\text{RSS}}(\theta) = L(\theta)$

Normal Equation (RSS)

We want to find the estimate $\hat{\theta} = \arg\min_{\theta \in \Theta} L(\theta)$ where it minimizes the loss.

$$
\begin{aligned}
L(\theta) &= L_{\text{RSS}}(\theta) = \sum_{i=1}^{n} (y_i - x_i\theta)^2 && \text{(expand the loss)} \\
&= \|Y - X\theta\|^2 = Y^\top Y - 2Y^\top X\theta + \theta^\top X^\top X\theta && \text{(convert to matrices)} \\
\nabla L(\theta) &= \frac{\partial L_{\text{RSS}}}{\partial \theta} = -\left(2X^\top (Y - X\theta)\right) && \text{(compute gradient)} \\
\nabla L(\theta) = 0 &\implies \hat{\theta} = \underbrace{(X^\top X)^{-1} X^\top}_{\text{Pseudo-inverse } X^\dagger} Y && \text{(Solve for } \hat{\theta}\text{)}
\end{aligned}
$$

Problems with Normal Equation

- **Require** closed-form gradient ∇L
- **Compute Complexity** of $X^\dagger$ — takes very long to compute $(X^\top X)^{-1}$: $O(m^3)$
- **Accuracy** — Invertiblity of $(X^\top X)^{-1}$
- **Optimality** of $\hat{\theta}$ — $L(\theta)$ is convex.

Convex Function (1D)

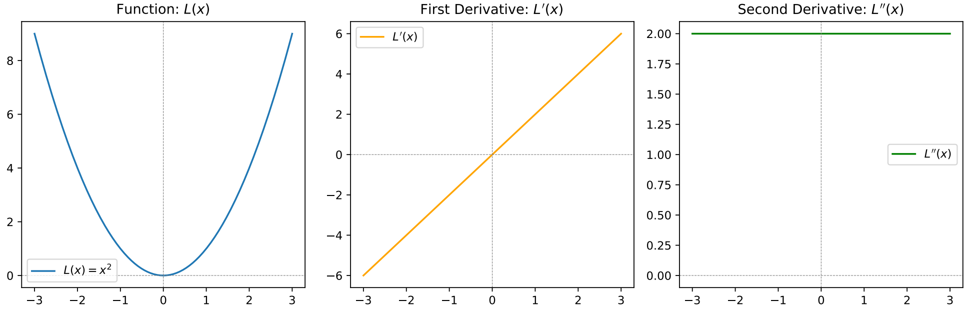

Figure 2: A function is **convex** if it curves upwards — like a bowl — and has no dips.

Definition: A twice-differentiable function $L : \mathbb{R} \to \mathbb{R}$ is convex if: $L''(\theta) \geq 0 \quad \forall \theta \in \mathbb{R}$

Implication: If $L''(\theta) > 0$, L is **strictly convex**, and any local minimizer is the unique global minimizer.

Gradient Descent

Given a convex loss function $L : \mathbb{R}^m \to \mathbb{R}$ with a minimizer, an initial point $\theta^{(0)} \in \mathbb{R}^m$, step size $\gamma > 0$, and number of iterations $T > 0$, we iteratively update:

$$\theta^{(t+1)} = \theta^{(t)} - \gamma \nabla L(\theta^{(t)}). \tag{4}$$

Each step moves in the direction of steepest descent scaled by γ;

Algorithm

1. Start with initial θ_0, step size γ, and total steps T
2. Run for each step:
   - Update $\theta^{(t+1)} = \theta^{(t)} - \gamma \nabla L(\theta^{(t)})$
3. Return at step T: $\theta^{(T)}$

5-minute Activity [L00.1]

Work in groups of 2s or 3s (split the work):

1. Given a simple function $L(x) = x^2$,
   a. What is its first-order derivative L'?
   b. Solve for the minimum x
   c. Compute step-by-step over 3 iterations of x-values for $\gamma \in \{10, 1, 0.1, 0.01\}$.
   d. What did you notice?
2. [Extra] What is the Compute Complexity of Gradient Descent for RSS?

Gradient Descent Algorithm

1. Start with initial θ_0, step size γ, and total steps T
2. Run for each step:
   - Update $\theta^{(t+1)} = \theta^{(t)} - \gamma \nabla L(\theta^{(t)})$
3. Return at step T: $\theta^{(T)}$

We say Gradient Descent converges if it finds the global minimizer.

Answer

t	10.0	1.0	0.1	0.01
0	5	5	5	5
1	-95	-5	4	4.9
2	1805	5	3.2	4.802
3	-34295	-5	2.56	4.70596

1. Given a simple function $L(x) = x^2$,
   a. First-order derivative $L'(x) = 2x$
   b. Minimum is $x = 0$
   c. See above, step-by-step over 3 iterations of x-values.
   d. What did we notice:
      1. Gradient Descent does not find 0 within $T = 3$, requiring more iterations.
      2. For bad values of γ, Gradient Descent does not converge.
      3. Speed of convergence depends on γ.
2. Compute Complexity is $O(nmT)$.

Illustration

See the animation for how gradient descent evolves over time.

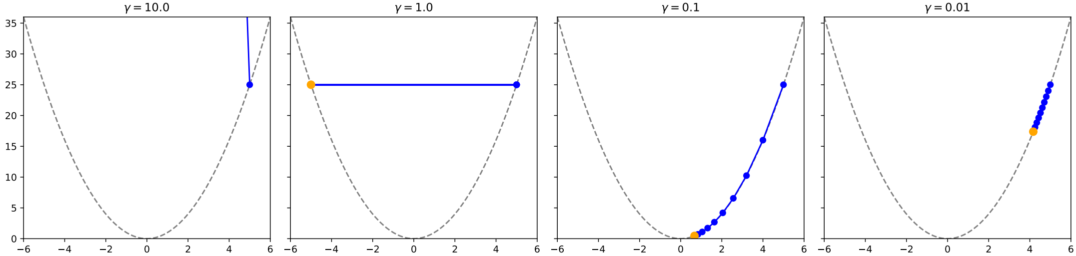

Figure 3: x-values after 10 steps of gradient descent with various learning rates on $L(x) = x^2$.

Summary

Takeaway: Gradient descent is the foundation of modern optimization in machine learning.

- **Linear models** can be solved via matrix inversion only under ideal conditions.
- **Normal equation** avoids direct inversion of X, but still incurs $O(m^3)$ compute cost.
- **Convex functions** ensure any local min is global min, enabling efficient optimization.
- **Gradient descent** is a scalable, general-purpose method:
  - Works even when inversion fails.
  - Faster in most situations to compute $O(nmT)$.
  - Depends critically on the learning rate γ.
  - Converges if convex and γ is not bad.
- Key trade-off: **simplicity vs. scalability vs. convergence speed**.

Assignment 1 [Due in 7 days]

Implement the function `gradient_descent(grad_L, theta_0, gamma, T)` to minimize $L(\theta)$ using T steps of gradient descent; Submit on Coursemology.