CS-470: Machine Learning

Week 2 — Linear Regression and Gradient Descent

Instructor: Dr. Sajjad Hussain

Department of Electrical and Computer Engineering
SEECS, NUST

September 18, 2025

Outline

Recap of Week 1

- Defined Machine Learning and its importance.
- Discussed types of ML: supervised, unsupervised, semi-supervised, reinforcement.
- Highlighted applications of ML in real-world problems.
- Covered challenges in ML: poor quality data, irrelevant features, overfitting, underfitting, validation, hyperparameters.
- Concluded with the No Free Lunch Theorem.

Introduction to Linear Regression

- Linear Regression is one of the simplest and most widely used supervised learning algorithms.
- **Goal**: Model the relationship between input features (x) and a continuous output variable (y).
- Hypothesis function assumes a linear relationship:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

- **Applications**: Predicting house prices, forecasting sales, estimating risk scores.

Linear Regression with One Variable

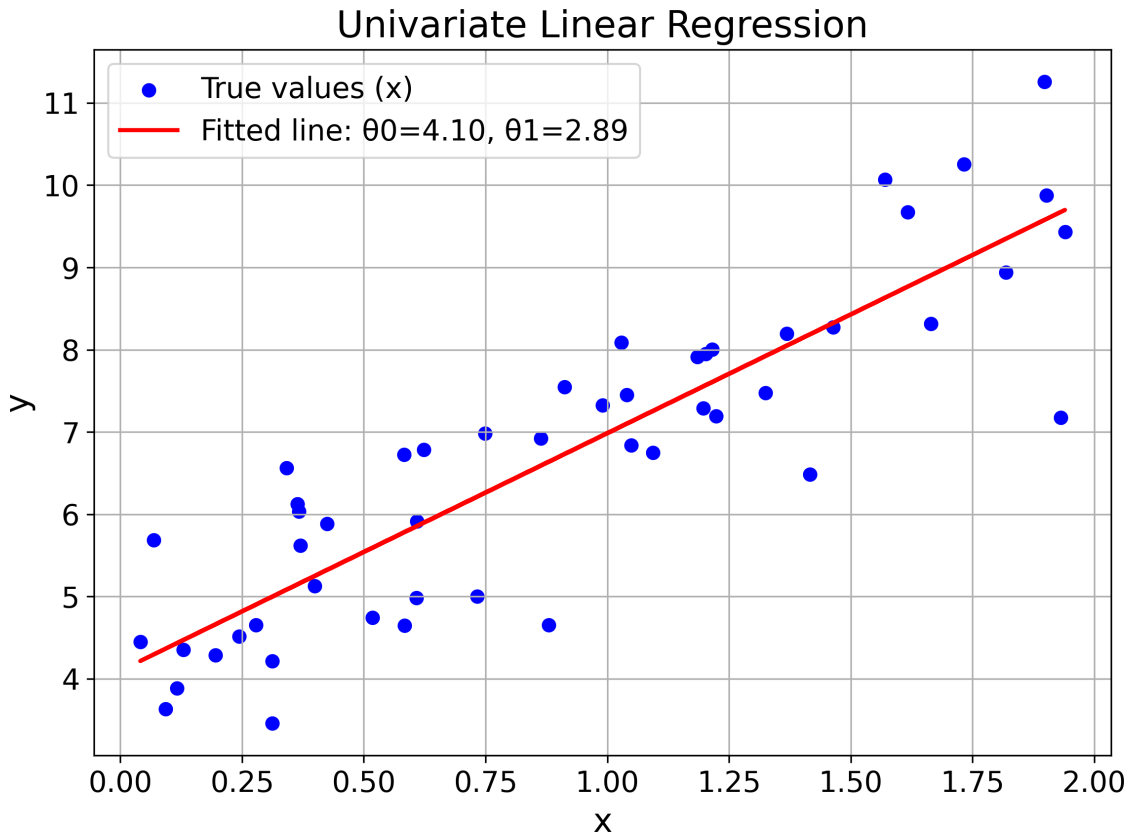
- **Simplest case**: one input feature (x) and one output variable (y).
- **Hypothesis function**:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

- **Parameters**:
  - θ_0: intercept (value of y when $x = 0$)
  - θ_1: slope (change in y for unit change in x)
- **Example**: Predicting house price (y) based on size (x).

Univariate Linear Regression

- We generate a synthetic dataset with a linear relationship and some noise.
- The red line shows the regression line that best fits the data points in blue.



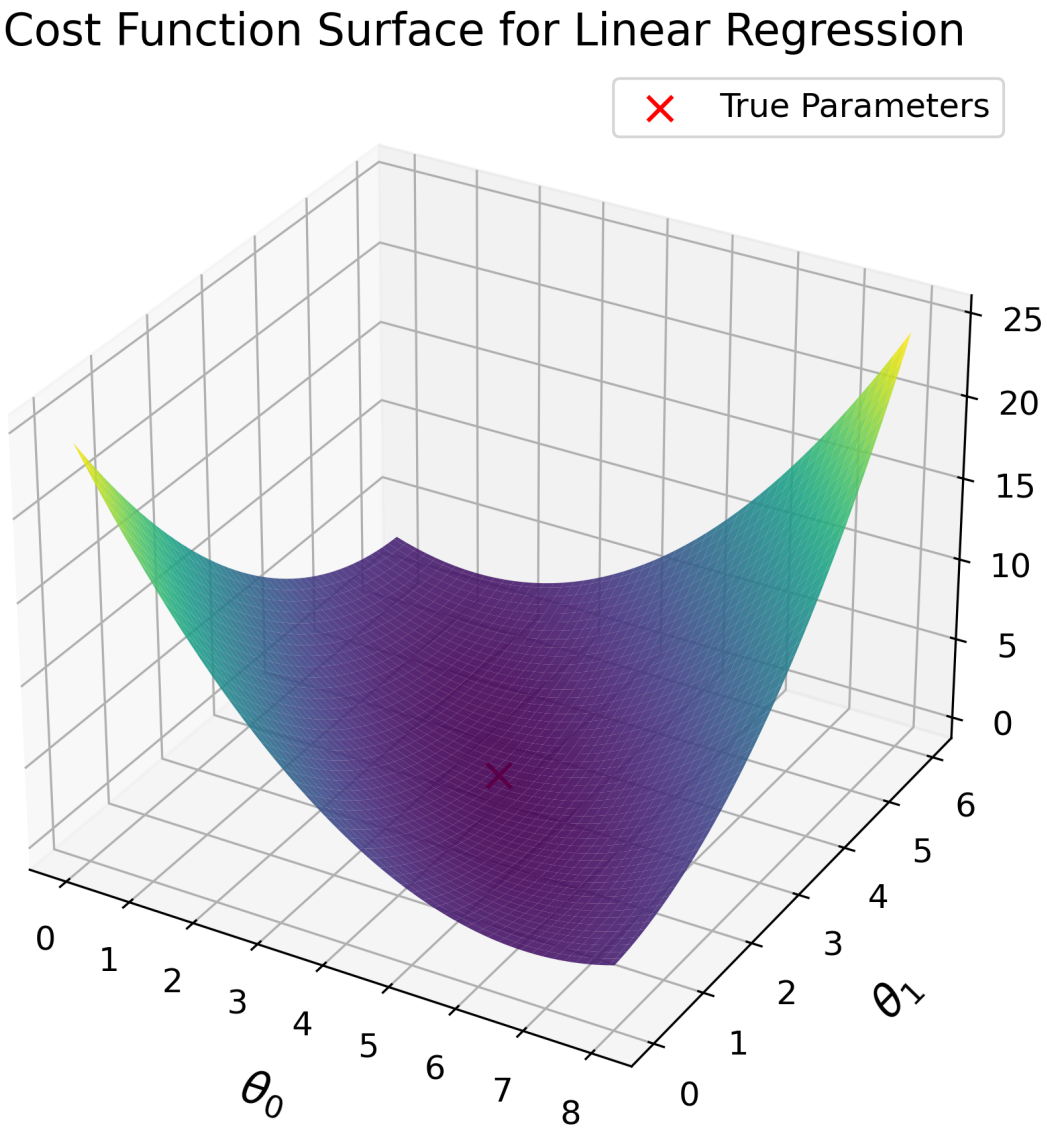
Cost Function

- To check how well our proposed regression line fits the training data, we need a measure of error.
- The most common measure is the **Mean Squared Error (MSE)**.
- For each training example, we look at the difference between the predicted value $h_\theta(x^{(i)})$ and the actual value $y^{(i)}$.
- We square this difference (so that negative and positive errors do not cancel out).
- Then, we take the average of these squared errors across all m training examples.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- **Goal:** Find values of θ_0, θ_1 that make this cost $J(\theta_0, \theta_1)$ as small as possible.

Cost Function Visualization



Linear Regression with Two Variables

- When we have two input features, the hypothesis extends naturally:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- Example: Predicting house price based on
  - x_1: size of the house (in square feet)
  - x_2: number of bedrooms
- The model represents a plane in 3D space that best fits the training data.

Multiple Linear Regression

- With n input features, the hypothesis becomes:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- Compact notation:

$$h_\theta(x) = \theta^T x$$

- Here:
  - $x = [1, x_1, x_2, \ldots, x_n]^T$ is the feature vector (with 1 for intercept).
  - $\theta = [\theta_0, \theta_1, \ldots, \theta_n]^T$ is the parameter vector.
- Applications: price prediction, risk analysis, forecasting with many factors.

Matrix Notation for Linear Regression

- Training set with m examples and n features:
  - Feature matrix:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix}$$

  - Parameter vector: $\theta = [\theta_0, \theta_1, \dots, \theta_n]^T$
  - Output vector: $y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}]^T$
- Hypothesis for all examples:

$$h_\theta(X) = X\theta$$

Cost Function in Matrix Form

- Recall cost function (Mean Squared Error):

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- In matrix notation:

$$J(\theta) = \frac{1}{2m} (X\theta - y)^T (X\theta - y)$$

- Advantages of matrix notation:
  - Compact and elegant representation
  - Efficient computation using linear algebra libraries

Normal Equation Solution

- To minimize the cost function $J(\theta)$, we can solve analytically:

$$\theta = (X^T X)^{-1} X^T y$$

- This is called the **Normal Equation**.
- Advantages:
  - No need to choose a learning rate
  - No iteration required
- Works well when the number of features n is small to moderate.

Example: Housing Data

- Suppose we want to predict house prices based on:
  - x_1: size (sq. ft.)
  - x_2: number of bedrooms
- Sample dataset:

$$X = \begin{bmatrix} 1 & 2104 & 3 \\ 1 & 1600 & 3 \\ 1 & 2400 & 3 \\ 1 & 1416 & 2 \\ 1 & 3000 & 4 \end{bmatrix}, \quad y = \begin{bmatrix} 399900 \\ 329900 \\ 369000 \\ 232000 \\ 539900 \end{bmatrix}$$

- Using the Normal Equation, we can compute θ directly.

Limitations of Normal Equation

- Computing $(X^T X)^{-1}$ becomes expensive when:
  - Number of features n is very large
  - Matrix $X^T X$ is ill-conditioned (nearly singular)
- Computational complexity:

$$O(n^3) \quad \text{(due to matrix inversion)}$$

- For large-scale problems (e.g., millions of features), the Normal Equation is not practical.
- This motivates the use of **iterative methods** such as Gradient Descent.

Gradient Descent: Introduction

- For large datasets or many features, solving the Normal Equation is too expensive.
- **Gradient Descent** is an iterative optimization algorithm to minimize the cost function.
- Idea: start with some initial guess for parameters θ, then repeatedly adjust them to reduce the cost.
- Eventually, parameters converge to values that minimize $J(\theta)$.

Gradient Descent: Intuition

- Imagine standing on a hill and wanting to reach the lowest point (valley).
- At each step, you look at the slope of the ground (the gradient).
- You then take a small step in the opposite direction of the slope.
- Repeat this process until you reach the bottom.

Gradient Descent: Update Rule

- The parameters are updated using:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- Here:
  - α: learning rate (step size)
  - $\frac{\partial}{\partial \theta_j} J(\theta)$: derivative (slope) of cost function w.r.t. parameter θ_j
- Update is applied simultaneously for all parameters.

Example: Parameter Update

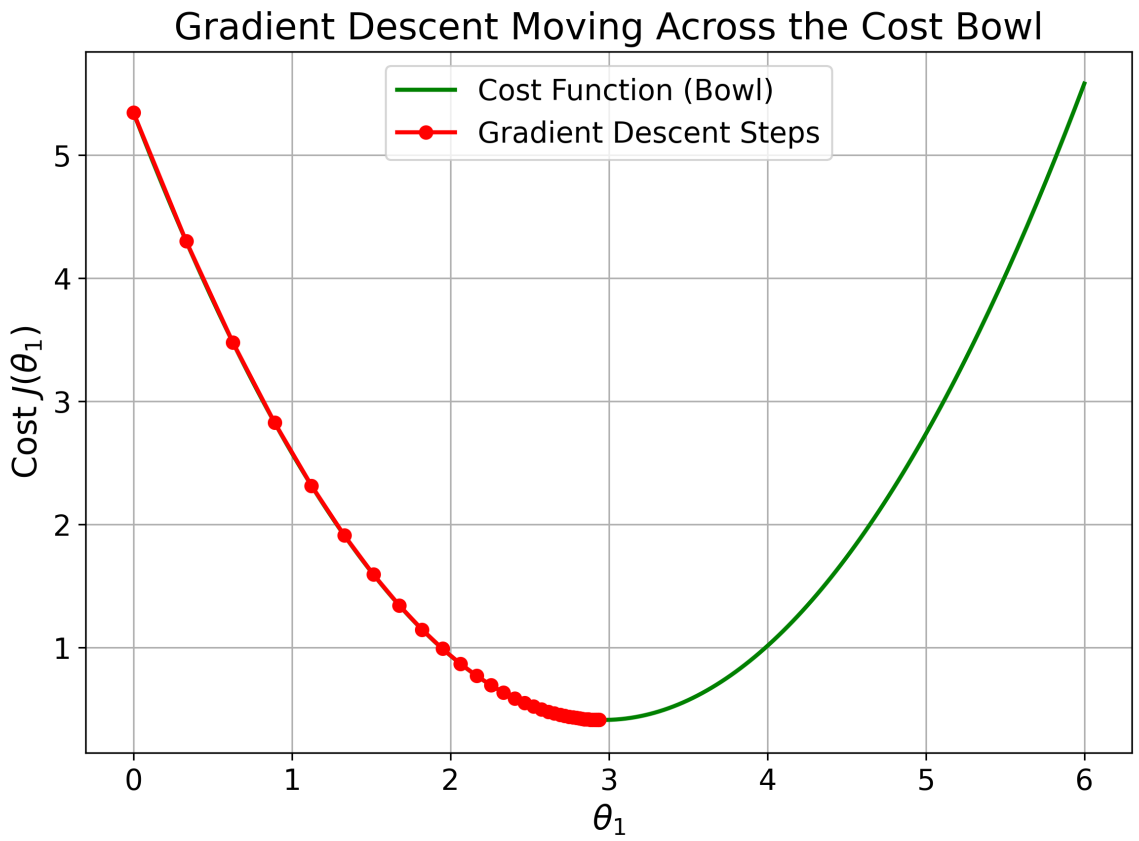
- Suppose current parameters are $\theta_0 = 0, \theta_1 = 0$.
- Compute gradient of cost function at these values.
- Update using:

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta), \quad \theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta)$$

- After several iterations, parameters gradually move closer to the optimal solution.

Gradient Descent Across the Cost Bowl

- The green curve shows the cost function $J(\theta_1)$ as a function of θ_1.
- The red points and line illustrate the gradient descent parameter updates starting from a zero initial value of θ_1.

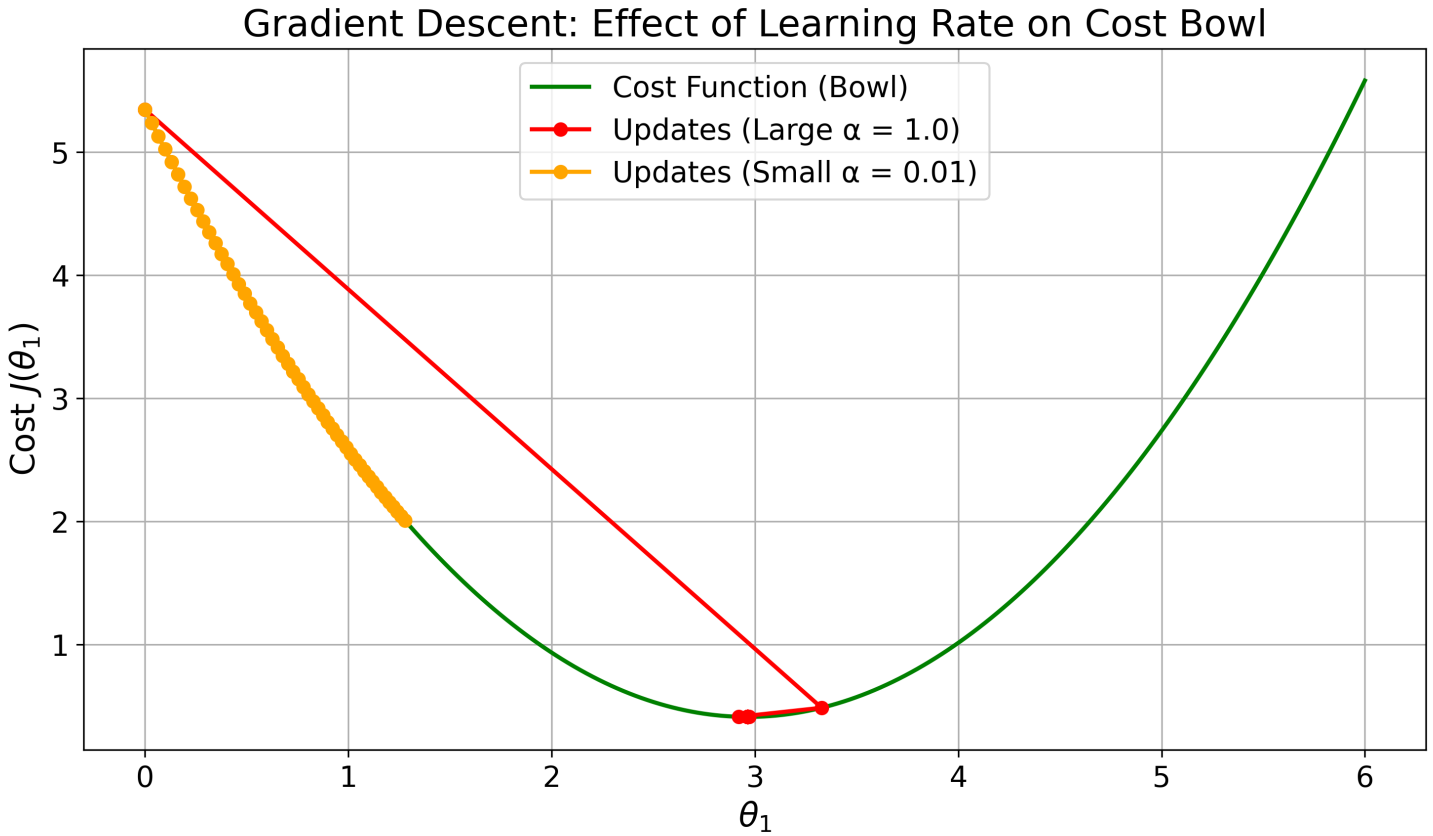


Role of Learning Rate (α)

- α controls how big a step we take each iteration.
- If α is too small:
  - Convergence is very slow.
- If α is too large:
  - Algorithm may overshoot the minimum.
  - May fail to converge and even diverge.
- Choosing the right learning rate is crucial for efficiency and stability.

Effect of Learning Rate on Gradient Descent

- Red points and line: parameter updates with a **large learning rate** $\alpha = 1.0$; notice overshooting and oscillations.
- Orange points and line: parameter updates with a **small learning rate** $\alpha = 0.01$; notice slow convergence toward the minimum.



Gradient Descent: Key Points

- Start with initial guess for parameters.
- Repeatedly update parameters by subtracting gradient.
- Continue until parameters converge (or cost function stops decreasing).
- Works well for large-scale problems where the Normal Equation is not practical.

Gradient Descent Variants

- Gradient descent can be applied in different ways depending on how many training examples are used to compute the gradient at each step.
- Choosing the right variant affects **convergence speed, stability, and computational efficiency**.
- Three common variants:
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent

Batch Gradient Descent

- Uses the **entire training dataset** to compute the gradient for each parameter update.
- Pros:
  - Stable updates
  - Moves directly toward global minimum in convex problems
- Cons:
  - Very slow for large datasets
  - Requires large memory to store all gradients at once
- Update rule for parameter θ_j:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} J(\theta; x^{(i)}, y^{(i)})$$

Stochastic Gradient Descent

- Uses **one random training example** to compute the gradient for each update.
- Pros:
  - Much faster per update
  - Can escape shallow local minima
- Cons:
  - Updates are noisy and fluctuate heavily
  - Convergence is less smooth
- Update rule for parameter θ_j:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta; x^{(i)}, y^{(i)})$$

- Each iteration uses a different randomly selected training example

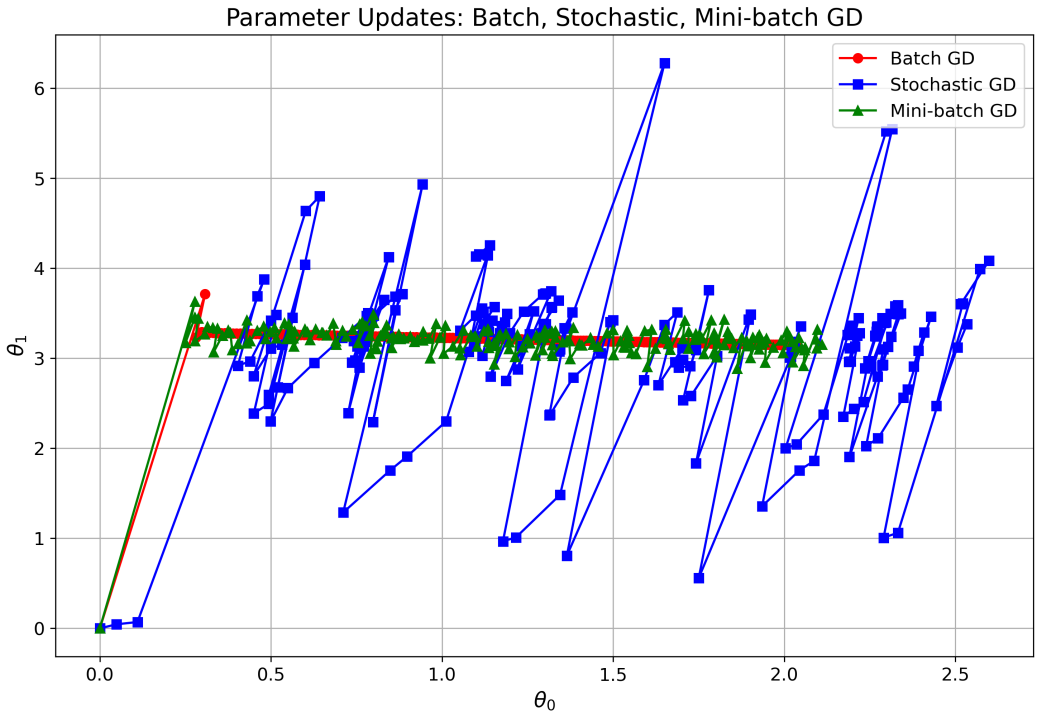
Mini-Batch Gradient Descent

- Uses a **subset of training examples** (mini-batch) to compute the gradient for each update.
- Pros:
  - Combines benefits of batch and stochastic gradient descent
  - Can use vectorized operations for efficiency
  - Reduces memory requirements
- Cons:
  - Mini-batch size needs to be chosen carefully
- Update rule for parameter θ_j with mini-batch size b:

$$\theta_j := \theta_j - \alpha \frac{1}{b} \sum_{i \in \text{mini-batch}} \frac{\partial}{\partial \theta_j} J(\theta; x^{(i)}, y^{(i)})$$

Gradient Descent Variants: Parameter Updates

- **Red line:** Batch Gradient Descent – smooth, stable convergence using the full dataset.
- **Blue line:** Stochastic Gradient Descent – noisy updates, fluctuates around the optimum.
- **Green line:** Mini-batch Gradient Descent – balanced updates using small subsets of data.



Summary of Week 2

- Learned Linear Regression:
  - Single-variable, two-variable, and multiple-variable regression
  - Multiple linear regression equation and matrix notation
  - Solving via linear algebra and limitations for large datasets
- Introduced Gradient Descent:
  - Iterative method to minimize the cost function
  - Parameter update rule and role of learning rate (α)
  - Visualization of parameter updates and convergence behavior
- Gradient Descent Variants:
  - Batch GD: uses full dataset, smooth convergence
  - Stochastic GD: uses one example per update, noisy but faster
  - Mini-batch GD: uses small subset, balanced between speed and stability