CSE 403: Machine Learning

Chapter 3: Linear Regression

Md Atikuzzaman
Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

Outline

Why Linear Regression?

- Many real-world tasks require predicting a **continuous value**.
- Examples: house price, salary, temperature, demand forecasting.
- Linear regression provides a **simple, interpretable baseline**.

Linear regression is often the first model to try: fast to train, easy to debug, easy to explain.

Supervised Learning View

Data

We are given labeled examples:

$$\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$$

where $y^{(i)} \in \mathbb{R}$ is continuous.

Goal

Learn a function $f_\theta(x)$ that predicts $\hat{y} \approx y$ for unseen inputs.

Simple Linear Regression (1 Feature)

Hypothesis (Model)

$$\hat{y} = f_\theta(x) = \theta_0 + \theta_1 x$$

- θ_0: intercept (bias)
- θ_1: slope (effect of x on y)

The model assumes a linear relationship between input x and target y.

Example Dataset: House Prices

Rooms	Price ($1000)
2	18
3	22
4	26
5	28
6	30

$x = \text{Rooms}, \qquad y = \text{Price}$

How Do We Measure Error?

Residual (Prediction Error)

For one example $(x^{(i)}, y^{(i)})$:

$$e^{(i)} = \hat{y}^{(i)} - y^{(i)}$$

- If $e^{(i)} > 0$: model overestimates
- If $e^{(i)} < 0$: model underestimates

Squared Loss (MSE)

Loss for One Example

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Empirical Risk (Mean Squared Error)

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

Squaring penalizes large errors more strongly and gives a smooth objective for optimization.

Learning Objective

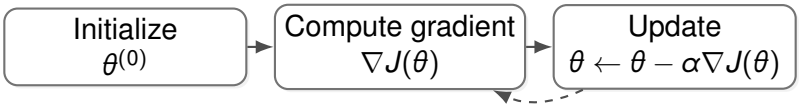
Optimization Problem

$$\hat{\theta} = \arg\min_{\theta \in \Theta} J(\theta)$$

- Choose parameters θ that minimize the average squared error.
- Two common training approaches:
  - Closed-form solution (Normal Equation)
  - Iterative optimization (Gradient Descent)

Gradient Descent: Idea

- Start with an initial guess $\theta^{(0)}$.
- Compute gradient of $J(\theta)$.
- Update parameters in the direction that reduces the loss.



Gradient Descent Update Rule

Vector Update

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta J(\theta^{(t)})$$

For Linear Regression (MSE)

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

- α: learning rate
- Repeat until convergence or early stopping

Goal: Fit a Line to the Data

Given Dataset

We model **Price** (y) from **Rooms** (x):

$$(x, y) \in \{(2, 18), (3, 22), (4, 26), (5, 28), (6, 30)\}, \quad y \text{ in } \$1000$$

Hypothesis (Simple Linear Regression)

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

Loss Function and Objective

Squared Loss (per example)

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

Mean Squared Error (MSE) Objective

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

- $m = 5$ (number of training examples)
- Training means: **minimize** $J(\theta_0, \theta_1)$

Gradient Descent: Core Idea

We update parameters iteratively

$$\theta \leftarrow \theta - \alpha \nabla J(\theta)$$

- Start with an initial guess (θ_0, θ_1)
- Compute gradients (direction of steepest increase)
- Move **opposite** direction to reduce loss
- Repeat until the loss stops improving

Gradient descent is an optimization method, not a machine learning model.

Gradients for Simple Linear Regression

Define the error (residual)

$$e^{(i)} = \hat{y}^{(i)} - y^{(i)} = \left(\theta_0 + \theta_1 x^{(i)}\right) - y^{(i)}$$

Partial derivatives of MSE

$$\frac{\partial J}{\partial \theta_0} = \frac{2}{m}\sum_{i=1}^{m} e^{(i)} \qquad \frac{\partial J}{\partial \theta_1} = \frac{2}{m}\sum_{i=1}^{m} e^{(i)} x^{(i)}$$

Update rules

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}, \qquad \theta_1 \leftarrow \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$$

Gradient Descent Algorithm (Step-by-Step)

Algorithm (Batch Gradient Descent)

1. Initialize θ_0, θ_1 (e.g., 0) and choose learning rate α
2. Repeat for $t = 0, 1, 2, \ldots$ until convergence:
   1. Compute predictions: $\hat{y}^{(i)} = \theta_0 + \theta_1 x^{(i)}$
   2. Compute errors: $e^{(i)} = \hat{y}^{(i)} - y^{(i)}$
   3. Compute gradients:

$$g_0 = \frac{2}{m} \sum_{i=1}^{m} e^{(i)}, \qquad g_1 = \frac{2}{m} \sum_{i=1}^{m} e^{(i)} x^{(i)}$$

   4. Update:

$$\theta_0 \leftarrow \theta_0 - \alpha g_0, \qquad \theta_1 \leftarrow \theta_1 - \alpha g_1$$

Note: Batch GD uses all m examples in every iteration.

Worked Example: Initialization

Dataset

$$(2, 18), (3, 22), (4, 26), (5, 28), (6, 30)$$

Choose

$$\theta_0 = 0, \quad \theta_1 = 0, \quad \alpha = 0.01, \quad m = 5$$

- We will do **one full batch update** to show the process.

Iteration 1: Predictions and Errors

Predictions with $\theta_0 = 0, \theta_1 = 0$

$$\hat{y}^{(i)} = \theta_0 + \theta_1 x^{(i)} = 0 \quad \Rightarrow \quad \hat{y} = [0, 0, 0, 0, 0]$$

Errors $e^{(i)} = \hat{y}^{(i)} - y^{(i)}$

$$e = [-18, -22, -26, -28, -30]$$

Iteration 1: Compute Gradients

Gradient w.r.t. θ_0

$$g_0 = \frac{2}{m}\sum_{i=1}^{m} e^{(i)} = \frac{2}{5}\big(-18-22-26-28-30\big) = \frac{2}{5}(-124) = -49.6$$

Gradient w.r.t. θ_1

$$g_1 = \frac{2}{m}\sum_{i=1}^{m} e^{(i)}x^{(i)} = \frac{2}{5}\big((-18)2+(-22)3+(-26)4+(-28)5+(-30)6\big)$$

$$= \frac{2}{5}\big(-36-66-104-140-180\big) = \frac{2}{5}(-526) = -210.4$$

Iteration 1: Parameter Update

Update with $\alpha = 0.01$

$$\theta_0 \leftarrow 0 - 0.01(-49.6) = 0.496$$

$$\theta_1 \leftarrow 0 - 0.01(-210.4) = 2.104$$

New model after 1 iteration

$$\hat{y} = 0.496 + 2.104x$$

After the first step, the line moves upward and starts fitting the data trend.

Check: New Predictions (After 1 Update)

Compute $\hat{y} = 0.496 + 2.104x$

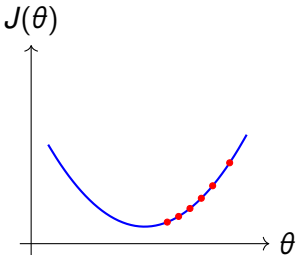
$$\hat{y}(2) = 4.704, \quad \hat{y}(3) = 6.808, \quad \hat{y}(4) = 8.912, \quad \hat{y}(5) = 11.016, \quad \hat{y}(6) = 13.120$$

- True outputs are $[18, 22, 26, 28, 30]$.
- Predictions are still low, but **closer than 0**.
- Repeating iterations will keep reducing the loss.

Effect of Learning Rate α

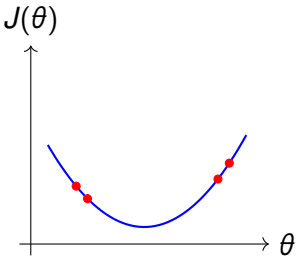
Too Small α

- Very slow convergence
- Many iterations required



Too Large α

- Oscillation
- May diverge



Choose α carefully for stable and fast convergence.

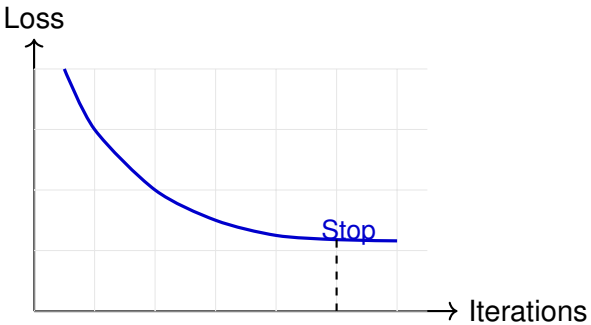
Stopping Criteria

Common Stopping Rules

- Loss improvement is very small:

$$|J^{(t+1)} - J^{(t)}| < \epsilon$$

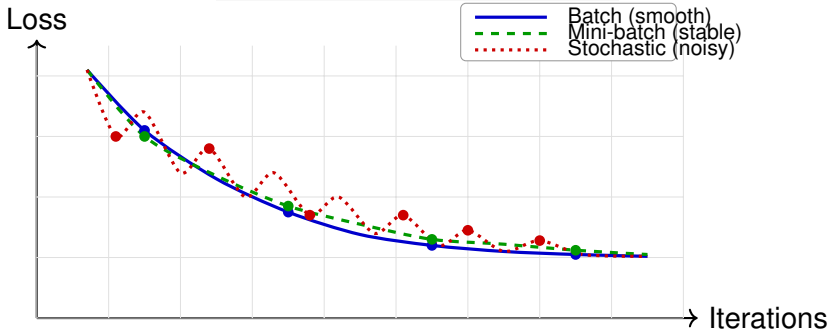
- Maximum iterations reached
- Validation loss stops improving **(Early Stopping)**



Goal: Stop training when further updates provide negligible improvement.

Optimization Methods: Gradient Descent Variants

Batch GD	Stochastic GD	Mini-batch GD
Uses all m samples	1 sample at a time	Small batches (e.g., 32)
Smooth convergence	Noisy updates	Stable + Efficient
Computationally heavy	Fast for large data	Most commonly used



How Do We Evaluate a Regression Model?

- **MSE:** $\frac{1}{m}\sum(\hat{y} - y)^2$
- **RMSE:** $\sqrt{\text{MSE}}$ (same unit as y)
- **MAE:** $\frac{1}{m}\sum|\hat{y} - y|$
- R^2**:** proportion of variance explained

Always report metrics on a held-out test set to estimate generalization performance.

Interpretation: Coefficients

Meaning of Parameters

In $\hat{y} = \theta_0 + \theta^\top x$:

- θ_0 shifts predictions up/down (baseline)
- θ_j is the expected change in y when x_j increases by 1 unit, **holding other features fixed**

Multiple Linear Regression (d Features)

Vector Form

Let $x \in \mathbb{R}^d$ and $\theta \in \mathbb{R}^d$.

$$\hat{y} = f_\theta(x) = \theta_0 + \theta^\top x$$

- $x = [x_1, \ldots, x_d]^\top$
- $\theta = [\theta_1, \ldots, \theta_d]^\top$
- Each coefficient θ_j measures the contribution of feature x_j

Closed-Form Solution (Normal Equation)

Design Matrix

Let $X \in \mathbb{R}^{m \times (d+1)}$ include a column of ones:

$$X = \begin{bmatrix} 1 & (x^{(1)})^\top \\ 1 & (x^{(2)})^\top \\ \vdots & \vdots \\ 1 & (x^{(m)})^\top \end{bmatrix}, \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Normal Equation

Solution

If $X^\top X$ is invertible:

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

- Fast and exact for small to medium feature sizes.
- Can be expensive when $(d + 1)$ is very large.
- If $X^\top X$ is not invertible, use:

$$\hat{\theta} = X^\dagger y$$

where $X^\dagger$ is the pseudoinverse.

Why Use Different Loss Functions?

- Different loss functions measure prediction error differently.
- Choice of loss function affects:
  - Sensitivity to outliers
  - Model robustness
  - Optimization behavior
- Mean Squared Error (MSE) is common, but not always the best choice.

Selecting an appropriate loss function can significantly improve model performance.

Mean Absolute Error (MAE)

Definition

$$L(y, \hat{y}) = |y - \hat{y}|$$

- Measures the absolute difference between prediction and true value.
- Less sensitive to outliers than MSE.
- All errors contribute linearly.

Empirical Loss

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} |y^{(i)} - \hat{y}^{(i)}|$$

Huber Loss

Definition

$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ \delta |y - \hat{y}| - \frac{\delta^2}{2}, & |y - \hat{y}| > \delta \end{cases}$$

- Combines advantages of MSE and MAE.
- Quadratic for small errors (smooth optimization).
- Linear for large errors (robust to outliers).

Quantile Loss (Pinball Loss)

Definition

For quantile $\tau \in (0, 1)$:

$$L_\tau(y, \hat{y}) = \begin{cases} \tau(y - \hat{y}), & y \geq \hat{y} \\ (1 - \tau)(\hat{y} - y), & y < \hat{y} \end{cases}$$

- Used in **quantile regression**.
- Estimates conditional quantiles instead of the mean.
- Useful when prediction uncertainty is important.

Comparison of Common Loss Functions

Loss Function	Formula Type	Outlier Sensitivity
MSE	Squared Error	High
MAE	Absolute Error	Medium
Huber	Hybrid	Low
Quantile	Asymmetric Error	Task-dependent

- MSE penalizes large errors heavily.
- MAE is more robust to extreme values.
- Huber provides a balance between both.

Key Takeaways

- **Problem:** Linear regression models a continuous target using labeled data $(x^{(i)}, y^{(i)})$.
- **Model:** Predictions are linear in features:

$$\hat{y} = \theta_0 + \theta^\top x$$

- **Training objective:** Learn parameters by minimizing an empirical loss (commonly MSE, but alternatives exist for robustness).
- **Optimization choices:**
  - **Gradient Descent** (batch/SGD/mini-batch): scalable and widely used
  - **Normal Equation** / **Pseudoinverse**: exact for small to medium feature sizes
- **Evaluation:** Use a held-out test set and report metrics such as MSE/RMSE, MAE, and R^2.
- **Interpretation:** Each coefficient θ_j represents the expected change in y per one-unit increase in x_j, holding other features fixed.

A good workflow: choose a loss, train with a suitable optimizer, validate on test data, and interpret coefficients.

References

1. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer.
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer.
3. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press.