Machine Learning Basics

December 3, 2025

Linear Algebra: Concepts & Examples Comprehensive Guide

Contents

1 Vectors **4**

1.1 Definition and Operations 4

1.2 Example 4

2 Matrices **5**

2.1 Matrix Multiplication 5

2.2 Example 5

3 3. Determinants **5**

3.1 Definitions 5

3.2 Example 5

4 4. Matrix Inversion **5**

4.1 Formula (2x2) 5

4.2 Example 6

5 Gaussian Elimination **6**

5.1 Example System 6

6 6. Eigenvalues and Eigenvectors **6**

6.1 Algorithm 6

6.2 Example 7

7 Random Variables and Distributions **7**

7.1 Discrete vs. Continuous 7

7.2 Probability Functions 8

7.3 Expected Value (Mean) 8

8 Common Distributions **8**

8.1 Uniform Distribution 8

8.2 Normal (Gaussian) Distribution 9

9 Normal Equation (Normál Egyenlet) **9**

10 1. Feature Normalization (Leírók Normalizálása) 10
10.1 Standardization (Z-score Normalization) 10
10.2 Min-Max Scaling 11
10.3 Normal Equation (Normál Egyenlet) 11

11 Probability Theory 12
11.1 Univariate Normal Distribution (Egyváltozós Normális Eloszlás) 12
11.2 Multivariate Normal Distribution (Többváltozós Normális Eloszlás) 12

12 Dimensionality Reduction 12
12.1 Principal Component Analysis (PCA) 12

13 Dimensionality Reduction 13
13.1 Principal Component Analysis (PCA) 13
13.1.1 Geometric Intuition 13
13.1.2 The Algorithm Steps 13
13.1.3 Choosing k (Explained Variance) 14

14 Regression Analysis (Regresszió) 14
14.1 Linear Regression (Lineáris Regresszió) 14
14.1.1 Univariate (Egyváltozós) 14
14.1.2 Multivariate (Többváltozós) 15
14.1.3 Cost Function: Mean Squared Error (MSE) 15
14.2 Polynomial Regression (Polinomiális Regresszió) 15
14.3 Logistic Regression (Logisztikus Regresszió) 15
14.3.1 Cost Function: Binary Cross-Entropy (Log Loss) 15

15 Classification (Osztályozás) 16
15.1 Binary Classification (Bináris Osztályozás) 16
15.2 Multi-class Classification (Többosztályos Osztályozás) 16

16 Regularization (Regularizáció) 16
16.1 Overfitting vs. Underfitting (Túlillesztés és Alulillesztés) 16
16.2 Regularized Regression (Regularizált Regresszió) 17

17 Neural Networks (Neurális Hálók) 17
17.1 Architecture 17
17.2 Cost Functions in Neural Networks 18
17.3 Optimization with Backpropagation (Optimalizáció Hiba-visszaterjesztéssel) 18

18 Model Selection & Diagnostics 19
18.1 Dataset Splitting (Adatbázisok) 19
18.2 Bias vs. Variance (Diagnosztika) 19
18.3 Learning Curves (Tanulógörbék) 20

19 Error Metrics for Skewed Classes (Egyenlőtlen osztályok) 20
19.1 Metrics 20

20 Recommender Systems (Ajánlórendszerek) 21
20.1 Problem Formulation . . . 21
20.2 Collaborative Filtering (Low Rank Matrix Factorization) . . . 21

21 Large Scale Machine Learning 21
21.1 Batch Gradient Descent (Standard) . . . 22
21.2 Stochastic Gradient Descent (SGD) . . . 22
21.3 Mini-batch Gradient Descent . . . 23
21.4 Online Learning . . . 23
21.5 Distributed Learning Architectures . . . 24
21.5.1 Data Parallelism (Map-Reduce) . . . 24
21.5.2 Model Parallelism . . . 24
21.6 Pipeline Ceiling Analysis . . . 24

1 Vectors

A vector is an ordered list of numbers, geometrically representing a magnitude and direction.

1.1 Definition and Operations

Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$.

- **Column Vector:** $\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$.
- **Addition:** $\mathbf{u} + \mathbf{v}$ adds components element-wise.
- **Dot Product:** $\mathbf{u} \cdot \mathbf{v} = \sum u_i v_i$.
- **Norm (Length):** $\|\mathbf{u}\| = \sqrt{\mathbf{u} \cdot \mathbf{u}}$.

1.2 Example

Let $\mathbf{u} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$.

$$\mathbf{u} + \mathbf{v} = \begin{bmatrix} 2+1 \\ 1+3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$
$$\mathbf{u} \cdot \mathbf{v} = (2)(1) + (1)(3) = 5$$
$$\|\mathbf{u}\| = \sqrt{2^2 + 1^2} = \sqrt{5} \approx 2.236$$

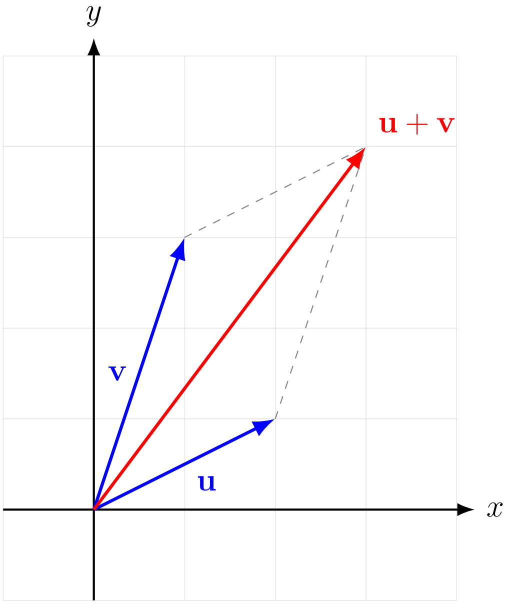


Figure 1: Vector Addition (Parallelogram Rule).

2 Matrices

A matrix is a rectangular array of numbers. An $m \times n$ matrix has m rows and n columns.

2.1 Matrix Multiplication

If A is $m \times n$ and B is $n \times p$, then $C = AB$ is $m \times p$.

$$c_{ij} = \text{Row}_i(A) \cdot \text{Col}_j(B)$$

2.2 Example

Let $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$.

$$AB = \begin{bmatrix} (1)(2) + (2)(1) & (1)(0) + (2)(2) \\ (3)(2) + (4)(1) & (3)(0) + (4)(2) \end{bmatrix} = \begin{bmatrix} 4 & 4 \\ 10 & 8 \end{bmatrix}$$

3 3. Determinants

The determinant is a scalar value associated with a square matrix that encodes scaling properties (like area or volume). If $\det(A) = 0$, the matrix is singular (not invertible).

3.1 Definitions

- **2x2 Matrix:** $\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$.
- **3x3 Matrix:** Uses cofactor expansion (e.g., across top row).

3.2 Example

Let $A = \begin{bmatrix} 2 & 5 \\ 1 & 3 \end{bmatrix}$.

$$\det(A) = (2)(3) - (5)(1) = 6 - 5 = 1$$

Since $\det(A) \neq 0$, A is invertible.

4 4. Matrix Inversion

The inverse A^{-1} of a square matrix A satisfies $AA^{-1} = A^{-1}A = I$ (Identity Matrix).

4.1 Formula (2x2)

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

4.2 Example

Using A from the section above ($\det(A) = 1$):

$$A^{-1} = \frac{1}{1}\begin{bmatrix} 3 & -5 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 3 & -5 \\ -1 & 2 \end{bmatrix}$$

Check:

$$AA^{-1} = \begin{bmatrix} 2 & 5 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 3 & -5 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 6-5 & -10+10 \\ 3-3 & -5+6 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

5 Gaussian Elimination

An algorithm to solve systems of linear equations of the form $A\mathbf{x} = \mathbf{b}$. We form the **Augmented Matrix** $[A|\mathbf{b}]$ and perform row operations to reach **Row Echelon Form (REF)**.

5.1 Example System

$$\begin{cases} x + 2y + z = 8 \\ 2x + 5y + 3z = 19 \\ -x + y + 2z = 5 \end{cases} \implies \left[\begin{array}{ccc|c} 1 & 2 & 1 & 8 \\ 2 & 5 & 3 & 19 \\ -1 & 1 & 2 & 5 \end{array}\right]$$

Step 1: Eliminate x from rows 2 and 3.

- $R_2 \leftarrow R_2 - 2R_1$
- $R_3 \leftarrow R_3 + R_1$

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 8 \\ 0 & 1 & 1 & 3 \\ 0 & 3 & 3 & 13 \end{array}\right]$$

Step 2: Eliminate y from row 3.

- $R_3 \leftarrow R_3 - 3R_2$

$$\left[\begin{array}{ccc|c} 1 & 2 & 1 & 8 \\ 0 & 1 & 1 & 3 \\ 0 & 0 & 0 & 4 \end{array}\right]$$

Analysis: The last row says $0x + 0y + 0z = 4$, which is impossible ($0 \neq 4$). **Conclusion:** This system has **No Solution** (Inconsistent).

6 6. Eigenvalues and Eigenvectors

An eigenvector of a square matrix A is a non-zero vector $\mathbf{v}$ such that $A\mathbf{v} = \lambda\mathbf{v}$, where λ is a scalar (eigenvalue).

6.1 Algorithm

1. Solve the characteristic equation $\det(A - \lambda I) = 0$ to find λ.
2. For each λ, solve $(A - \lambda I)\mathbf{v} = \mathbf{0}$ to find $\mathbf{v}$.

6.2 Example

Let $A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$.

1. Find Eigenvalues:

$$\det\left(\begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) = \det \begin{bmatrix} 4-\lambda & 1 \\ 2 & 3-\lambda \end{bmatrix}$$

$$(4-\lambda)(3-\lambda) - 2 = \lambda^2 - 7\lambda + 12 - 2 = \lambda^2 - 7\lambda + 10 = 0$$

Factoring $(\lambda - 5)(\lambda - 2) = 0$ gives eigenvalues $\lambda_1 = 5, \lambda_2 = 2$.

2. Find Eigenvector for $\lambda_1 = 5$: Solve $(A - 5I)\mathbf{v} = \mathbf{0}$:

$$\begin{bmatrix} 4-5 & 1 \\ 2 & 3-5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$-x + y = 0 \implies x = y$. Let $x = 1$, then $y = 1$. Eigenvector $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

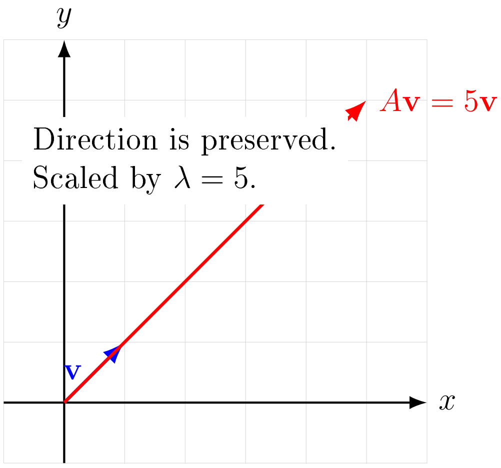


Figure 2: Visualizing the Eigenvector transformation.

7 Random Variables and Distributions

A **Random Variable** (RV) is a function that maps outcomes of a random experiment to real numbers. It translates experimental outcomes into numerical data.

7.1 Discrete vs. Continuous

- **Discrete Random Variable:** Can only take on a countable number of distinct values (e.g., outcome of a dice roll, number of heads in 10 tosses).
- **Continuous Random Variable:** Can take on any value within an interval (e.g., height, temperature, time).

7.2 Probability Functions

- **Discrete:** Described by a **Probability Mass Function (PMF)**, denoted $P(X = x)$.
- **Continuous:** Described by a **Probability Density Function (PDF)**, denoted $f(x)$. Probabilities are areas under the curve:

$$P(a \leq X \leq b) = \int_a^b f(x)\,dx$$

Note: For continuous variables, the probability of any single exact point is zero ($P(X = c) = 0$).

7.3 Expected Value (Mean)

The expected value $E[X]$ (often denoted μ) is the theoretical long-run average of the random variable. It is a measure of the "center" of the distribution.

- **Discrete Case:** Weighted sum of values.

$$E[X] = \sum_i x_i \cdot P(X = x_i)$$

Example (Fair Die): $E[X] = 1(\frac{1}{6}) + \cdots + 6(\frac{1}{6}) = 3.5$.

- **Continuous Case:** Integral of value weighted by density.

$$E[X] = \int_{-\infty}^{\infty} x \cdot f(x)\,dx$$

8 Common Distributions

8.1 Uniform Distribution

In a uniform distribution, all outcomes in the sample space are equally likely.

1. Discrete Uniform ($U\{a, b\}$): A finite set of values $\{a, a+1, \ldots, b\}$ where each outcome has probability $\frac{1}{n}$.

$$E[X] = \frac{a+b}{2}$$

2. Continuous Uniform ($U[a, b]$): The probability density is constant over the interval $[a, b]$.

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

$$E[X] = \frac{a+b}{2}$$

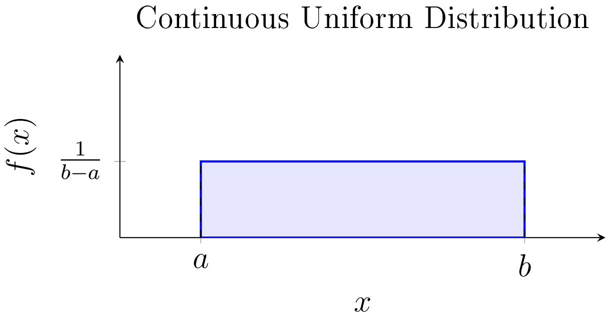


Figure 3: The PDF of a Continuous Uniform Distribution.

8.2 Normal (Gaussian) Distribution

The Normal distribution is the most important continuous distribution in statistics. It is symmetric, bell-shaped, and defined entirely by its mean (μ) and variance (σ^2).

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

PDF Formula:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

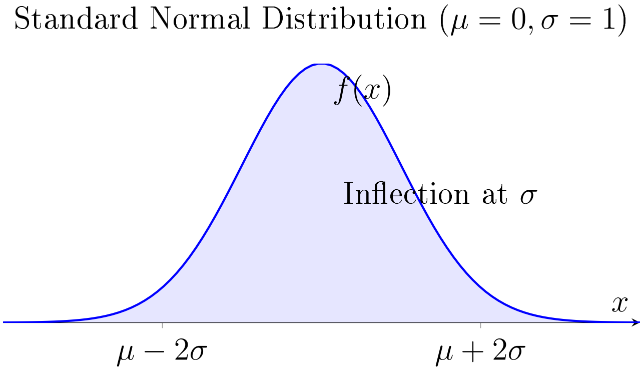


Figure 4: The Normal Distribution (Bell Curve). 68% of data falls within $\mu \pm \sigma$.

9 Normal Equation (Normál Egyenlet)

For Linear Regression, the minimum of the cost function can be found analytically in a closed form without iteration. By setting the gradient to zero, we get:

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

- **Pros:** No need to choose a learning rate; gives an exact solution immediately.
- **Cons:** Computationally expensive for large datasets because inverting the matrix $(X^T X)$ has a complexity of $O(n^3)$ (where n is the number of features).

Example Calculation

Suppose we have a tiny dataset with 3 points: $(1,1),(2,2),(3,2)$. We want to fit a line $y = \theta_0 + \theta_1 x$.

1. **Construct Matrices:** Add a column of 1s for the bias term θ_0.

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

2. **Compute X^TX:**

$$X^TX = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

3. **Compute Inverse $(X^TX)^{-1}$:** Using the formula for 2x2 inverse:

$$\det = 3(14) - 6(6) = 42 - 36 = 6$$

$$(X^TX)^{-1} = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix}$$

4. **Compute $X^T\mathbf{y}$:**

$$X^T\mathbf{y} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1+2+2 \\ 1+4+6 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$$

5. **Solve for θ:**

$$\theta = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 11 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 70 - 66 \\ -30 + 33 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1/2 \end{bmatrix}$$

Result: $\theta_0 \approx 0.67$ (Intercept), $\theta_1 = 0.5$ (Slope).

10 1. Feature Normalization (Leírók Normalizálása)

Machine learning algorithms (e.g., Gradient Descent, k-NN) often perform poorly if the input features have vastly different scales (e.g., Age [0-100] vs. Salary [1000-10000]). Normalization aims to bring features to a common scale.

10.1 Standardization (Z-score Normalization)

Data is transformed so that the mean (μ) is 0 and the standard deviation (σ) is 1. This is the most common method (used in PCA, Logistic Regression, etc.).

$$x' = \frac{x - \mu}{\sigma}$$

10.2 Min-Max Scaling

Maps the data to a fixed interval, typically $[0, 1]$.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Disadvantage: It is highly sensitive to outliers.

10.3 Normal Equation (Normál Egyenlet)

For Linear Regression, the minimum of the cost function can be found analytically in a closed form without iteration. By setting the gradient to zero, we get:

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

- **Pros:** No need to choose a learning rate; gives an exact solution immediately.
- **Cons:** Computationally expensive for large datasets because inverting the matrix $(X^T X)$ has a complexity of $O(n^3)$ (where n is the number of features).

Example Calculation

Suppose we have a tiny dataset with 3 points: $(1, 1), (2, 2), (3, 2)$. We want to fit a line $y = \theta_0 + \theta_1 x$.

1. **Construct Matrices:** Add a column of 1s for the bias term θ_0.

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$$

2. **Compute $X^T X$:**

$$X^T X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

3. **Compute Inverse $(X^T X)^{-1}$:** Using the formula for 2x2 inverse:

$$\det = 3(14) - 6(6) = 42 - 36 = 6$$

$$(X^T X)^{-1} = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix}$$

4. **Compute $X^T \mathbf{y}$:**

$$X^T \mathbf{y} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 + 2 + 2 \\ 1 + 4 + 6 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$$

5. **Solve for θ:**

$$\theta = \frac{1}{6} \begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 11 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 70 - 66 \\ -30 + 33 \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 1/2 \end{bmatrix}$$

Result: $\theta_0 \approx 0.67$ (Intercept), $\theta_1 = 0.5$ (Slope).

11 Probability Theory

11.1 Univariate Normal Distribution (Egyváltozós Normális Eloszlás)

The most common distribution in nature (Gaussian). Defined by mean μ and variance σ^2:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

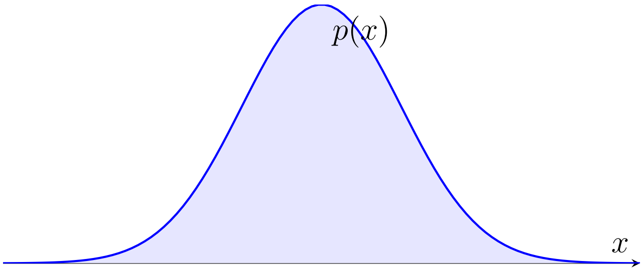


Figure 5: Standard Normal Distribution.

11.2 Multivariate Normal Distribution (Többváltozós Normális Eloszlás)

Generalization to k dimensions. Defined by the mean vector $\boldsymbol{\mu}$ and the covariance matrix Σ:

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

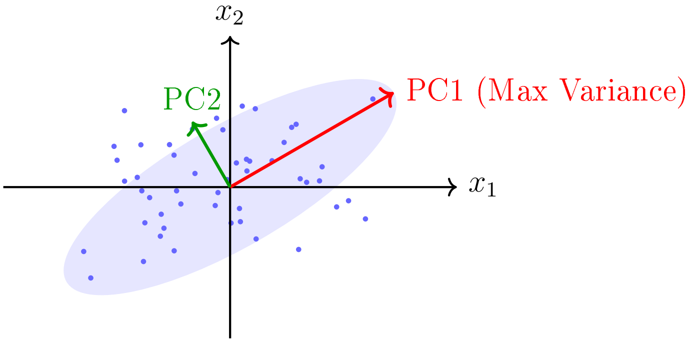
Important for algorithms like Gaussian Mixture Models (GMM) and Linear Discriminant Analysis (LDA).

12 Dimensionality Reduction

12.1 Principal Component Analysis (PCA)

PCA reduces the dimensionality of the data while preserving as much variance (information) as possible. **Steps:**

1. **Standardize** the data.
2. Compute the **Covariance Matrix** Σ.
3. Compute the **Eigenvectors** and **Eigenvalues** of Σ.
4. Keep the top k eigenvectors corresponding to the largest eigenvalues (Principal Components).
5. Project the original data onto these components.



PCA identifies the axes of maximum variance.

Figure 6: Concept of PCA: Finding the intrinsic axes of the data.

13 Dimensionality Reduction

13.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is an unsupervised learning algorithm used for dimensionality reduction. It identifies the "principal components"—linear combinations of the original features—that capture the maximum variance in the data.

13.1.1 Geometric Intuition

Imagine a cloud of data points in 2D. PCA finds a new coordinate system such that:

1. The first axis (Principal Component 1 or PC1) aligns with the direction of greatest spread (variance) in the data.

2. The second axis (PC2) is orthogonal (perpendicular) to the first and captures the remaining variance.

By projecting data onto just PC1, we reduce the data from 2D to 1D while losing the least amount of information.

13.1.2 The Algorithm Steps

1. **Data Preprocessing:** Standardize the data (as described in Section 1) so that each feature has mean 0 and variance 1. Let this matrix be X (size $m \times n$).

2. **Compute Covariance Matrix:** The covariance matrix Σ expresses how features vary together.

$$\Sigma = \frac{1}{m} X^T X$$

   Σ is an $n \times n$ symmetric matrix.

3. **Eigenvalue Decomposition:** Compute the eigenvectors and eigenvalues of Σ.

$$\Sigma U = \lambda U$$

   - **Eigenvectors (U):** Unit vectors representing the direction of the new axes.

- **Eigenvalues (λ):** Scalars representing the amount of variance captured by each eigenvector.

  In practice, we often use Singular Value Decomposition (SVD) on X for numerical stability.

4. **Select Principal Components:** Sort eigenvectors by descending eigenvalues. Choose the top k eigenvectors to form a transformation matrix U_{reduce} $(n \times k)$.

5. **Project Data:** Transform the original data X into the new lower-dimensional subspace Z.

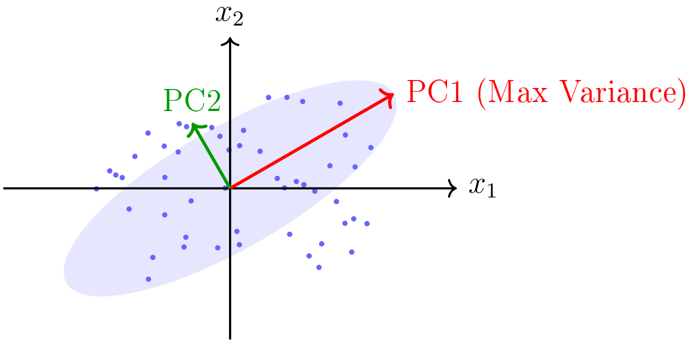
$$Z = X \cdot U_{reduce}$$

   The resulting Z is size $m \times k$.

13.1.3 Choosing k (Explained Variance)

How many components should we keep? We look at the *Explained Variance Ratio*. The variance explained by the i-th component is proportional to its eigenvalue λ_i.

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j} \geq 0.95 \quad \text{(e.g., 95\% variance retained)}$$



PCA identifies the axes of maximum variance.

Figure 7: Concept of PCA: Finding the intrinsic axes of the data.

14 Regression Analysis (Regresszió)

14.1 Linear Regression (Lineáris Regresszió)

Linear regression attempts to model the relationship between variables by fitting a linear equation to observed data.

14.1.1 Univariate (Egyváltozós)

Predicting a target y based on a single feature x.

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

14.1.2 Multivariate (Többváltozós)

Predicting y based on multiple features $x_1, x_2, \ldots, x_n$. We define $x_0 = 1$ (bias term).

$$h_\theta(\mathbf{x}) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \boldsymbol{\theta}^T \mathbf{x}$$

14.1.3 Cost Function: Mean Squared Error (MSE)

The objective is to minimize the average squared difference between predictions and actual values:

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

14.2 Polynomial Regression (Polinomiális Regresszió)

Used when data is non-linear. We create new features by raising original features to powers (e.g., x^2, x^3).

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d$$

Although the curve is non-linear, the problem remains *linear in the parameters* θ, so the normal equation or gradient descent can still be used.

14.3 Logistic Regression (Logisztikus Regresszió)

Despite the name, this is a **classification** algorithm for predicting probabilities (values between 0 and 1). It uses the **Sigmoid (Logistic) Function**:

$$g(z) = \frac{1}{1 + e^{-z}}$$

The hypothesis becomes:

$$h_\theta(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

This represents the probability that the output is 1: $P(y = 1 | \mathbf{x}; \boldsymbol{\theta})$.

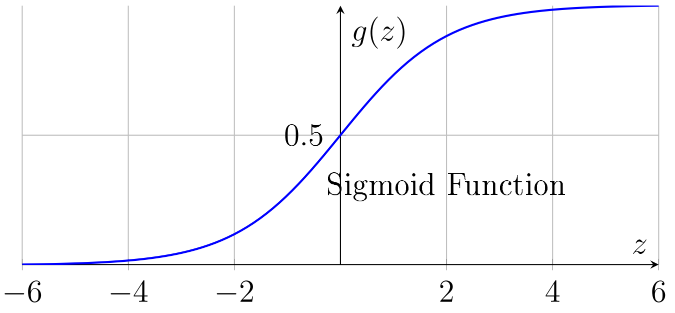


Figure 8: The Sigmoid function maps any real number to the $(0, 1)$ interval.

14.3.1 Cost Function: Binary Cross-Entropy (Log Loss)

We cannot use MSE for Logistic Regression because the Sigmoid function makes the cost non-convex (multiple local minima). Instead, we use the Log Loss:

$$\text{Cost}(h_\theta(\mathbf{x}), y) = \begin{cases} -\log(h_\theta(\mathbf{x})) & \text{if } y = 1 \\ -\log(1 - h_\theta(\mathbf{x})) & \text{if } y = 0 \end{cases}$$

Combined into a single equation:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\theta(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(\mathbf{x}^{(i)})) \right]$$

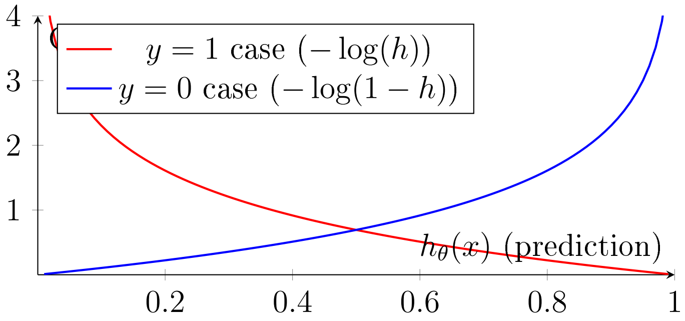


Figure 9: Log Loss penalties. If $y = 1$ and prediction ≈ 0, cost $\to \infty$.

15 Classification (Osztályozás)

15.1 Binary Classification (Bináris Osztályozás)

The target variable y has only two possible values: 0 (Negative Class) or 1 (Positive Class). Logistic regression uses a threshold (usually 0.5) to decide the class:

$$\text{Predict } y = 1 \text{ if } h_\theta(\mathbf{x}) \geq 0.5 \implies \boldsymbol{\theta}^T \mathbf{x} \geq 0$$

15.2 Multi-class Classification (Többosztályos Osztályozás)

When y can take on $k > 2$ values (e.g., classifying digits 0-9).

One-vs-All (One-vs-Rest): Train k separate binary classifiers.

- Classifier i: Distinguish class i from all other classes.
- Prediction: Choose the class i that maximizes $h_\theta^{(i)}(\mathbf{x})$.

16 Regularization (Regularizáció)

16.1 Overfitting vs. Underfitting (Túlillesztés és Alulillesztés)

- **Underfitting (High Bias):** The model is too simple to capture the underlying trend (e.g., fitting a line to a quadratic curve). Training error is high.
- **Overfitting (High Variance):** The model is too complex and fits the noise in the training data rather than the signal. Training error is low, but test error is high.

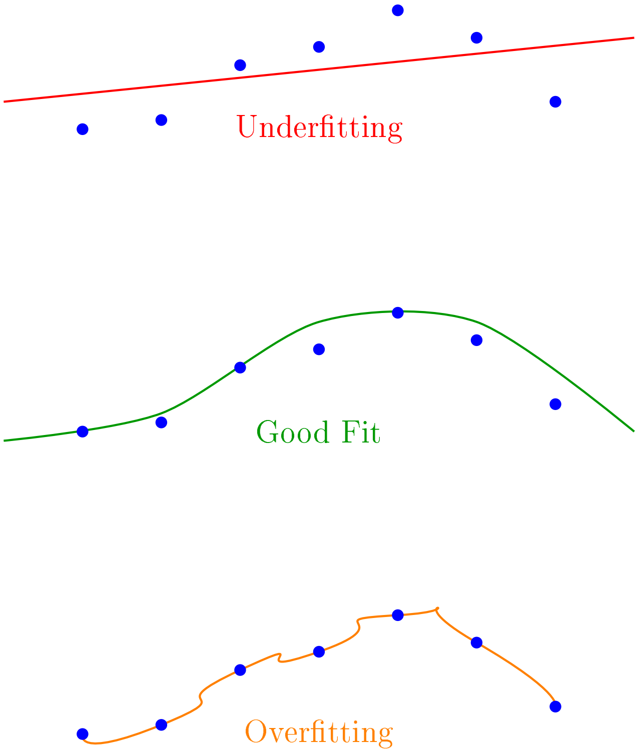


Figure 10: Visualizing Bias vs. Variance.

16.2 Regularized Regression (Regularizált Regresszió)

To prevent overfitting, we penalize large weights (θ) by adding a regularization term to the cost function.

Ridge Regression (L2 Regularization):

$$J(\theta) = \text{MSE}(\theta) + \lambda \sum_{j=1}^{n} \theta_j^2$$

This shrinks coefficients toward zero but rarely makes them exactly zero.

Lasso Regression (L1 Regularization):

$$J(\theta) = \text{MSE}(\theta) + \lambda \sum_{j=1}^{n} |\theta_j|$$

This can lead to sparse models where some coefficients become exactly zero (feature selection).

17 Neural Networks (Neurális Hálók)

17.1 Architecture

A Neural Network consists of layers of neurons (units).

- **Input Layer:** Features (x).
- **Hidden Layers:** Compute intermediate representations using activation functions (Sigmoid, ReLU, Tanh).
- **Output Layer:** Final prediction ($\hat{y}$).

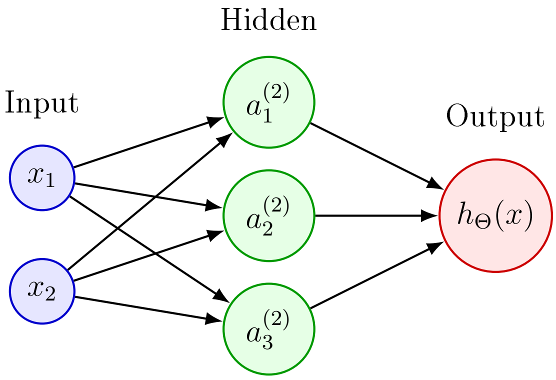


Figure 11: A simple 2-layer Neural Network architecture.

17.2 Cost Functions in Neural Networks

The choice of cost function $J(\Theta)$ depends on the problem type:

1. Regression Problems: Used when predicting a continuous value (e.g., house price).

- **Output Layer Activation:** Linear ($g(z) = z$).
- **Cost Function:** Mean Squared Error (MSE).

$$J(\Theta) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

2. Classification Problems: Used when predicting classes (e.g., Cat vs. Dog, Digits 0-9).

- **Output Layer Activation:** Sigmoid (Binary) or Softmax (Multi-class).
- **Cost Function:** Cross-Entropy Loss (Generalization of Log Loss).

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} \log(\hat{y}_k^{(i)})$$

Where K is the number of classes.

17.3 Optimization with Backpropagation (Optimalizáció Hibavisszaterjesztéssel)

Backpropagation efficiently computes the gradient of the cost function $\nabla_\Theta J(\Theta)$ for optimization algorithms like Gradient Descent. The specific formulas for "error terms" (δ) change slightly depending on the chosen cost function.

Steps:

1. **Forward Propagation:** Compute activations $a^{(l)}$ layer by layer to get the output $\hat{y}$.

2. **Compute Cost:** Calculate $J(\Theta)$ using MSE (Regression) or Cross-Entropy (Classification).

3. **Backward Propagation:** Compute "error terms" $\delta^{(l)}$ starting from the output layer.

   - **Output Layer (L):** $\delta^{(L)} = \hat{y} - y$ (This holds true for both MSE with Linear output AND Cross-Entropy with Sigmoid/Softmax output).
   - **Hidden Layers (l):** Propagate error backwards.

     $$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \cdot g'(z^{(l)})$$

     where g' is the derivative of the activation function (e.g., for Sigmoid: $a(1-a)$).

4. **Compute Gradients:** The partial derivative for weights is the product of the error term and the activation from the previous layer.

   $$\frac{\partial J}{\partial \Theta_{ij}^{(l)}} = a_j^{(l)} \delta_i^{(l+1)}$$

5. **Update Weights:** $\Theta := \Theta - \alpha \nabla J(\Theta)$.

18 Model Selection & Diagnostics

18.1 Dataset Splitting (Adatbázisok)

To properly evaluate a model and select hyperparameters (like polynomial degree d or regularization parameter λ), data is typically split into three sets:

1. **Training Set ($\approx$ 60%):** Used to minimize the cost function $J(\theta)$ and fit parameters.

2. **Cross Validation Set ($\approx$ 20%):** Used to evaluate model performance on unseen data during model selection. We choose the model with the lowest $J_{CV}(\theta)$.

3. **Test Set ($\approx$ 20%):** Used only once at the very end to estimate the generalization error.

18.2 Bias vs. Variance (Diagnosztika)

Diagnosing whether a model is underfitting (High Bias) or overfitting (High Variance) is crucial for improvement.

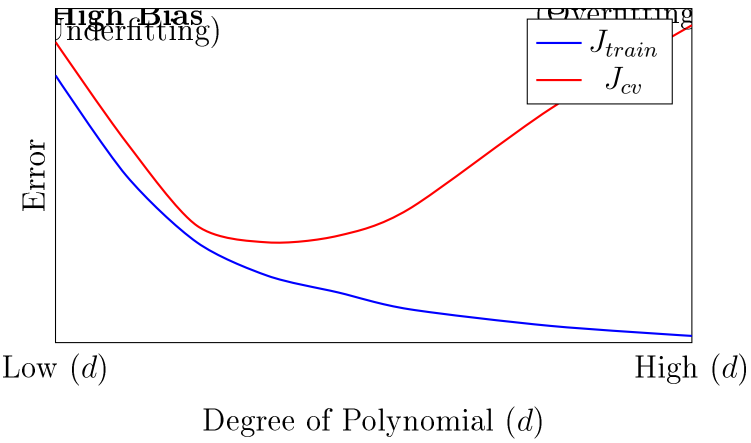


Figure 12: Model Complexity vs. Error. Optimal complexity is where J_{cv} is minimized.

18.3 Learning Curves (Tanulógörbék)

Plotting error against the number of training examples (m).

- **High Bias:** J_{train} and J_{cv} converge quickly but to a high error value. Adding more data **does not** help.
- **High Variance:** There is a large gap between J_{train} and J_{cv}. J_{train} is low, J_{cv} is high. Adding more data **likely helps**.

19 Error Metrics for Skewed Classes (Egyenlőtlen osztályok)

When classes are imbalanced (e.g., 99.5% of patients are healthy), Accuracy is a misleading metric. We use Precision and Recall based on the Confusion Matrix.

	Predicted 1	Predicted 0
Actual 1	True Positive (TP)	False Positive (FP)
Actual 0	False Negative (FN)	True Negative (TN)

Figure 13: Confusion Matrix Structure.

19.1 Metrics

- **Precision (P):** Of all patients we predicted as sick, what fraction actually has the disease?

$$P = \frac{TP}{TP + FP}$$

- **Recall (R):** Of all patients that actually have the disease, what fraction did we correctly detect?
$$R = \frac{TP}{TP + FN}$$
- F_1 **Score:** The harmonic mean of Precision and Recall. Used to compare models.
$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

20 Recommender Systems (Ajánlórendszerek)

20.1 Problem Formulation

Given n_u users and n_m items (movies), we want to predict missing ratings.

- $r(i, j) = 1$ if user j has rated movie i.
- $y^{(i,j)}$ is the rating given by user j to movie i.

20.2 Collaborative Filtering (Low Rank Matrix Factorization)

We learn both feature vectors $\mathbf{x}^{(i)}$ for movies and parameter vectors $\boldsymbol{\theta}^{(j)}$ for users simultaneously. **Objective Function:** Minimize squared error over all rated entries + Regularization.

$$J(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n_m)}, \boldsymbol{\theta}^{(1)}, \dots, \boldsymbol{\theta}^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left((\boldsymbol{\theta}^{(j)})^T \mathbf{x}^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \|\mathbf{x}^{(i)}\|^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \|\boldsymbol{\theta}^{(j)}\|^2$$

Gradient Descent Updates:

- For movie features $x_k^{(i)}$:
$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\boldsymbol{\theta}^{(j)})^T \mathbf{x}^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$
- For user preferences $\theta_k^{(j)}$:
$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\boldsymbol{\theta}^{(j)})^T \mathbf{x}^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

21 Large Scale Machine Learning

Scaling to massive datasets ($m > 10^7$) requires specialized algorithms and system architectures. We examine three variations of Gradient Descent that manage the trade-off between accuracy and computational speed.

21.1 Batch Gradient Descent (Standard)

In Batch Gradient Descent, we calculate the gradient of the cost function using the **entire training dataset** before making a single update to the parameters.

Cost Function:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Update Rule:

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- **Pros:**
  - Guaranteed to converge to the global minimum for convex error surfaces (and a local minimum for non-convex surfaces).
  - The trajectory is smooth and direct.
- **Cons:**
  - Extremely slow on very large datasets because it sums over m examples for every single step.
  - Requires loading the whole dataset into memory (RAM).
  - Cannot learn "online" (requires retraining from scratch for new data).

21.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent approximates the gradient using a **single training example** $(x^{(i)}, y^{(i)})$ at a time. It updates the parameters immediately after looking at just one sample.

Algorithm:

1. Randomly shuffle the dataset.
2. Repeat for $1 \ldots$ epochs:
   - For $i = 1 \ldots m$:

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

- **Pros:**
  - Much faster iterations; often converges (to a "good enough" solution) much faster than Batch GD for large m.
  - Can handle datasets larger than memory.
  - Adds noise which can help escape shallow local minima.
- **Cons:**
  - The path to the minimum is "noisy" and oscillates (wanders) around the optimal value rather than settling perfectly.
  - Loses vectorization benefits (processing 1 item is inefficient for SIMD/GPUs).

21.3 Mini-batch Gradient Descent

Mini-batch GD is the standard compromise used in modern Deep Learning. It computes the gradient using a small random subset of data (a **batch**) of size b (typically $b = 32, 64, 128, \dots$).

Update Rule: For a batch $B = \{(x^{(k)}, y^{(k)}) \mid k = i, \dots, i+b-1\}$:

$$\theta_j := \theta_j - \alpha \frac{1}{b} \sum_{k \in B} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

- **Advantages:**
  - **Vectorization:** Modern hardware (GPUs) can parallelize the matrix operations for 64 examples almost as fast as for 1 example.
  - **Stability:** The gradient approximation is smoother than SGD (less noise), allowing for larger learning rates.
  - **Speed:** Much faster than Batch GD; does not require full dataset in memory.

Convergence Paths

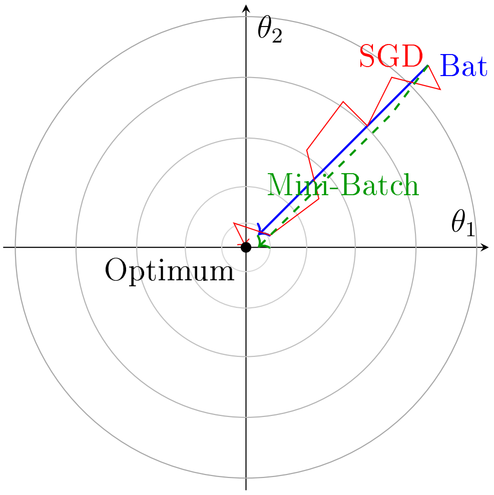


Figure 14: Visual comparison of convergence paths. Batch GD takes a direct path but steps are slow. SGD zig-zags significantly. Mini-batch offers a balanced path.

21.4 Online Learning

Used when data arrives in a continuous stream (e.g., clickstream data, search queries) and we do not store the dataset.

- **Process:** Receive $(x, y) \to$ Update $\theta \to$ Discard (x, y).
- **Adaptability:** The model can adapt to changing user preferences (concept drift) over time.
- **Algorithm:** Essentially Stochastic Gradient Descent (SGD) running indefinitely.

21.5 Distributed Learning Architectures

When data or models are too large for a single machine, we parallelize.

21.5.1 Data Parallelism (Map-Reduce)

The dataset is split into chunks. Each machine computes gradients for its chunk, and a central server aggregates them.

$$\theta_{new} = \theta_{old} - \alpha \sum_{k=1}^{N_{machines}} (\text{Partial Gradient}_k)$$

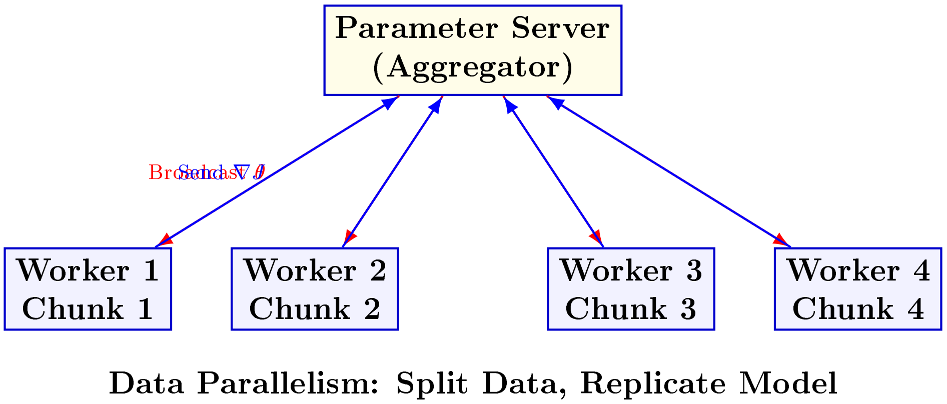


Figure 15: Map-Reduce / Data Parallelism architecture.

21.5.2 Model Parallelism

Used when the **model itself** (e.g., a massive Neural Network) is too large to fit in the memory of one GPU/machine.

- Different layers or different neurons of the same layer are placed on different machines.
- Requires high-bandwidth communication between machines to transfer activations and gradients.

21.6 Pipeline Ceiling Analysis

When building complex ML systems (pipelines), it is crucial to identify which component limits performance.

- **Method:** Manually set the output of a specific component to "perfect" (ground truth) and measure the overall system accuracy improvement.
- **Goal:** Identify the "bottleneck" where engineering effort yields the highest ROI.

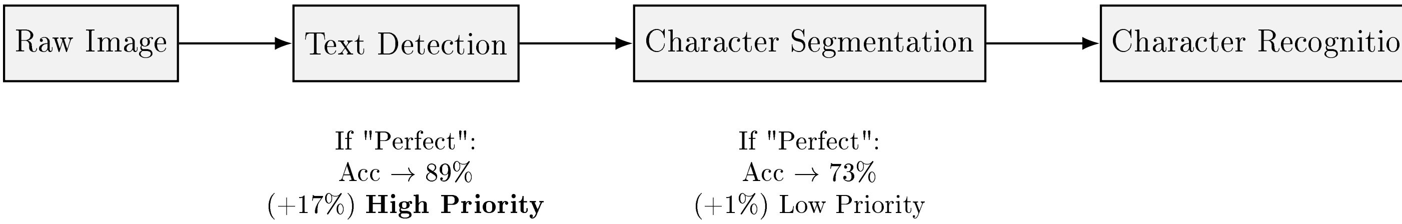


Figure 16: Ceiling Analysis Example (OCR Pipeline).