AI Applications Lecture 2
Machine Learning Models and Parametric Functions

SUZUKI, Atsushi
Jing WANG

2025-09-01

Contents

1 Introduction

1.1 Review of the Previous Lecture

In the previous lecture, we re-examined the diverse tasks that AI should solve from a unified perspective.

- The tasks we want computers to solve can be formulated as an **input-output relationship**, where an appropriate output is returned for a given input.
- Real-world entities such as text, images, and audio can be converted into numerical data (vectors or tensors) based on well-defined rules.
- From the above two points, we confirmed that solving a task with AI boils down to a **function determination problem**: finding an appropriate **function** from a set of input numerical data (input space) to a set of output numerical data (output space).

1.2 Learning Outcomes of This Lecture

Through this lecture, students will aim to be able to perform the following tasks.

- Formulate a function determination problem as a parameter determination problem for a **parametric function**.
- Explain what parameters and functions constitute basic AI/machine learning models such as linear regression, logistic regression, decision trees, and k-NN.
- Formulate machine learning as a problem of determining the parameters of a parametric function using data, and explain the difference between **training** and **inference** in machine learning.

1.3 Policy of This Lecture

The field of AI and machine learning is developing very rapidly, and specific models widely known today are likely to be outdated by the time you graduate. Therefore, this lecture emphasizes understanding more general and universal concepts that will be applicable in the future and will not become obsolete, rather than learning individual technologies.

As a first step, we will learn about ”parametric functions,” a mathematical framework common to many machine learning models, including neural networks. Understanding this concept will allow you to view various models from a unified perspective and clearly distinguish between the two major phases of AI development: training and inference. This distinction is an essential foundation for understanding the applications of neural networks, which we will learn about in subsequent lectures, as well as practical issues such as licensing.

2 Preparation: Mathematical Notations

Here is a list of the basic mathematical notations used in this lecture.

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a \coloneqq b$ indicates that a is defined as b.
- **Set:**
  - Sets are often denoted by uppercase calligraphic letters. Example: $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that the element x belongs to the set $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (set-builder notation).
  - $\{x \in \mathcal{A} | P(x)\}$: The set of elements in $\mathcal{A}$ for which the proposition $P(x)$ is true (set-builder notation).
  - $\mathbb{R}$: The set of all real numbers.
  - $\mathbb{R}_{>0}$: The set of all positive real numbers.
  - $\mathbb{R}_{\geq 0}$: The set of all non-negative real numbers.
  - $\mathbb{Z}$: The set of all integers.
  - $\mathbb{Z}_{>0}$: The set of all positive integers.
  - $\mathbb{Z}_{\geq 0}$: The set of all non-negative integers.
- **Function:**
  - $f : X \to Y$: Indicates that the function f is a map that takes an element of set X as input and outputs an element of set Y.
  - $y = f(x)$: Indicates that the output of the function f for an input $x \in X$ is $y \in Y$.
- **Vector:**
  - In this course, a vector refers to a column of numbers.
  - Vectors are denoted by bold italic lowercase letters. Example: $\boldsymbol{v}$.
  - $\boldsymbol{v} \in \mathbb{R}^n$: Indicates that the vector $\boldsymbol{v}$ is an n-dimensional real vector.
  - The i-th element of a vector $\boldsymbol{v}$ is denoted as v_i. $\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$.
- **Matrix:**

- Matrices are denoted by bold italic uppercase letters. Example: $\boldsymbol{A}$.
- $\boldsymbol{A} \in \mathbb{R}^{m,n}$: Indicates that the matrix $\boldsymbol{A}$ is an $m \times n$ real matrix.
- The element in the i-th row and j-th column of a matrix $\boldsymbol{A}$ is denoted as $a_{i,j}$.

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}.$$

- The transpose of a matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^\top$. If $\boldsymbol{A} \in \mathbb{R}^{m,n}$, then $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$, and its elements are given by $(\boldsymbol{A}^\top)_{ij} = a_{ji}$.
- A vector is also a matrix with 1 column, and its transpose can be defined. $\boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \in \mathbb{R}^{1,n}$.

- **Tensor:**
  - In this lecture, the word tensor simply refers to a multi-dimensional array. A vector can be considered a 1st-order tensor, and a matrix a 2nd-order tensor. Tensors of 3rd order or higher are denoted by underlined bold italic uppercase letters, like $\underline{\boldsymbol{A}}$.
  - Students who have already learned about abstract tensors in mathematics or physics may feel uncomfortable calling a mere multi-dimensional array a tensor. However, if we always fix the basis to the standard basis, we can identify the mathematical tensor with its component representation (which becomes a multi-dimensional array), so the terminology is (somewhat) consistent.

3 Parametric Functions

We have seen that solving a task can be formulated as a function determination problem. Since we handle functions on a computer, the specification of that function must be uniquely determined by numerical data that a computer can store (specifically, a bit string, often a sequence of floating-point numbers).

Let's formulate this idea mathematically.

Definition 3.1 (Parametric Function)**.** If there exists a set Θ (the **parameter space**), and for each element $\boldsymbol{\theta} \in \Theta$, a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$ is defined, then the family of functions $(f_{\boldsymbol{\theta}})_{\boldsymbol{\theta} \in \Theta}$ is called a **parametric function**. Each $\boldsymbol{\theta}$ is called a **parameter**.

Given a parametric function, the problem of finding a function f is replaced by the problem of finding an appropriate parameter $\boldsymbol{\theta}$. Below are examples of parametric functions used in the field of AI. The immediate goal of this lecture is to understand neural networks

as parametric functions, but here we will first look at some examples of parametric functions from a broader perspective. There are several reasons for this.

- **Linear regression** and **logistic regression** are:
  - Simple and clear examples for understanding how a function changes when its parameters are changed.
  - They can be considered special cases of simple neural networks and serve as a foundation for later learning.
  - They are basic methods widely used in the field of statistics and are valuable to know as part of general knowledge.
- **Decision trees** and **regression trees** are:
  - Good examples of parametric functions consisting of functions that cannot be represented by (commonly used) neural networks with continuous activation functions, and are useful for understanding what neural networks are ”not”.
  - Good examples of parametric functions where the dimension of the parameters (the size and shape of the tree) is not fixed.
  - Very effective for explaining machine learning models to non-expert clients due to their intuitive nature of following a tree structure.
  - They form the basis of advanced methods like Gradient Boosting Trees, which demonstrate extremely high performance on tabular data such as customer information held by companies.
- **k-Nearest Neighbors (k-NN)** is:
  - Similar to regression and decision trees, a good example of a parametric function consisting of functions that cannot be represented by neural networks, useful for understanding what neural networks are ”not”.
  - A good example of a parametric function where the dimension of the parameters is not fixed.
  - An excellent first teaching material for understanding the steps of training and inference, as its training algorithm is extremely trivial.

3.1 Example 1: Linear Regression

Consider the case where the input is a d_{in}-dimensional vector $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ and the output is a 1-dimensional real number $y \in \mathbb{R}$.

- **Parameters:** $\boldsymbol{\theta} \coloneqq (\boldsymbol{w}, b)$, where $\boldsymbol{w} \in \mathbb{R}^{d_{\text{in}}}$, $b \in \mathbb{R}$.
- **Function:** $f_{\boldsymbol{w},b}(\boldsymbol{x}) \coloneqq \boldsymbol{w}^\top \boldsymbol{x} + b = \sum_{i=1}^{d_{\text{in}}} w_i x_i + b$.

Example 3.1. For $d_{\text{in}} = 2$, with parameters $\boldsymbol{w} = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}, b = 0.5$, the output for the input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$ is

$$\begin{aligned} y = f_{(\boldsymbol{w},b)}(\boldsymbol{x}) &= (1.0 \times 3.0) + (-2.0 \times 4.0) + 0.5 \\ &= 3.0 - 8.0 + 0.5 = -4.5 \end{aligned} \tag{1}$$

Exercise 3.1. For parameters $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$, calculate the output for the same input as in Example 1, $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$.

Answer.

$$\begin{aligned} y = f_{(\boldsymbol{w},b)}(\boldsymbol{x}) &= (-0.5 \times 3.0) + (3.0 \times 4.0) + (-1.0) \\ &= -1.5 + 12.0 - 1.0 = 9.5 \end{aligned} \tag{2}$$

(This shows that the output changes with different parameters for the same input.)

3.2 Example 2: Logistic Regression

The input is the same, $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, but the output is a probability value $y \in [0, 1]$.

- **Parameters:** Same as linear regression, $\boldsymbol{\theta} \coloneqq (\boldsymbol{w}, b)$.
- **Function:** $f_{\boldsymbol{w},b}(\boldsymbol{x}) \coloneqq \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$. Here, $\sigma(z) \coloneqq \frac{1}{1+e^{-z}}$ is the **sigmoid function**.

Example 3.2. Using the same parameters $\boldsymbol{w} = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}, b = 0.5$ and input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$ as in Example 1, first the output of the linear part is calculated as $z = \boldsymbol{w}^\top \boldsymbol{x} + b = -4.5$, and the final output is

$$\begin{aligned} y = \sigma(-4.5) &= \frac{1}{1 + e^{4.5}} \\ &\approx \frac{1}{1 + 90.017} \approx 0.011 \end{aligned} \tag{3}$$

Exercise 3.2. For the same input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$ as in Example 2, calculate the output using the parameters from Exercise 1, $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$.

Answer. First, the output of the linear part is calculated as $z = \boldsymbol{w}^\top \boldsymbol{x} + b = (-0.5 \times 3.0) + (3.0 \times$

$4.0) - 1.0 = -1.5 + 12.0 - 1.0 = 9.5$. The final output is

$$\begin{aligned} y = \sigma(9.5) &= \frac{1}{1 + e^{-9.5}} \\ &\approx \frac{1}{1 + 0.0000748} \approx 0.999925 \end{aligned} \tag{4}$$

3.3 Example 3: Decision Tree / Regression Tree

Decision trees and regression trees are examples where the parameters cannot be represented by a fixed-length vector. For an input $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, the output is determined by traversing a tree structure.

Mathematical Formulation: A decision/regression tree is represented by a binary tree structure. Nodes are assigned integer indices. Let the root node be 1, the left child of node $i \in \mathbb{Z}_{>0}$ be $\text{left}(i) \coloneqq 2i$, the right child be $\text{right}(i) \coloneqq 2i + 1$, and the parent node be $\text{parent}(i) \coloneqq \lfloor i/2 \rfloor$. The parameters $\boldsymbol{\theta}$ that define the tree structure and the behavior of each node are defined by the following set of elements. $\boldsymbol{\theta} \coloneqq (\mathcal{V}, ((j_i, t_i))_{i \in \mathcal{V}_{\text{int}}}, (c_l)_{l \in \mathcal{V}_{\text{leaf}}})$

- $\mathcal{V} \subset \mathbb{Z}_{>0}$: A finite set of indices of the nodes included in the tree. It must satisfy $1 \in \mathcal{V}$, and if $k \in \mathcal{V} \setminus \{1\}$, then $\text{parent}(k) \in \mathcal{V}$ (reachable from the root).
- The set of **internal nodes** $\mathcal{V}_{\text{int}} \coloneqq \{i \in \mathcal{V} \mid \{\text{left}(i), \text{right}(i)\} \subseteq \mathcal{V}\}$.
- The set of **leaf nodes** $\mathcal{V}_{\text{leaf}} \coloneqq \{i \in \mathcal{V} \mid \{\text{left}(i), \text{right}(i)\} \cap \mathcal{V} = \{\}\}$.
- The set of nodes must satisfy $\mathcal{V} = \mathcal{V}_{\text{int}} \cup \mathcal{V}_{\text{leaf}}$.
- $((j_i, t_i))_{i \in \mathcal{V}_{\text{int}}}$: A sequence of parameters for each internal node $i \in \mathcal{V}_{\text{int}}$. $j_i \in \{1, \ldots, d_{\text{in}}\}$ is the index of the input variable used for splitting, and $t_i \in \mathbb{R}$ is the threshold.
- $(c_l)_{l \in \mathcal{V}_{\text{leaf}}}$: A sequence of output values for each leaf node $l \in \mathcal{V}_{\text{leaf}}$. For a regression tree, $c_l \in \mathbb{R}$; for a classification tree, c_l is a class label.

The function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is defined as a recursive traversal starting from the root node 1.

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) \coloneqq \text{traverse}(\boldsymbol{x}, 1) \tag{5}$$

Here, the auxiliary function $\text{traverse}(\boldsymbol{x}, i)$ defines the process at node i as follows.

- If node i is a leaf node ($i \in \mathcal{V}_{\text{leaf}}$), then $\text{traverse}(\boldsymbol{x}, i) \coloneqq c_i$.
- If node i is an internal node ($i \in \mathcal{V}_{\text{int}}$), then

$$\text{traverse}(\boldsymbol{x}, i) \coloneqq \begin{cases} \text{traverse}(\boldsymbol{x}, \text{left}(i)) & \text{if } x_{j_i} < t_i \\ \text{traverse}(\boldsymbol{x}, \text{right}(i)) & \text{if } x_{j_i} \geq t_i \end{cases} \tag{6}$$

Example 3.3 (Regression Tree)**.** Let $d_{\text{in}} = 2$ and consider a regression tree defined by the following parameters $\boldsymbol{\theta}_1$ (see Figure 1).

- Node set $\mathcal{V} = \{1, 2, 3, 6, 7\}$
- (Derived from above) Internal node set $\mathcal{V}_{\text{int}} = \{1, 3\}$
- (Derived from above) Leaf node set $\mathcal{V}_{\text{leaf}} = \{2, 6, 7\}$
- Internal node parameters:
  - Node 1: $(j_1, t_1) = (1, 2.5)$
  - Node 3: $(j_3, t_3) = (2, 0.0)$
- Leaf node values:
  - Node 2: $c_2 = 10.0$
  - Node 6: $c_6 = -5.0$
  - Node 7: $c_7 = 8.0$

Calculate the output for the input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$.

1. Evaluate at node 1 (root): $x_{j_1} = x_1 = 3.0 \geq 2.5 = t_1$, so proceed to the right child node $\text{right}(1) = 3$.
2. Evaluate at node 3: $x_{j_3} = x_2 = -1.0 < 0.0 = t_3$, so proceed to the left child node $\text{left}(3) = 6$.
3. Node 6 is a leaf node ($\in \mathcal{V}_{\text{leaf}}$), and its value is $c_6 = -5.0$.

Therefore, the output is $y = f_{\boldsymbol{\theta}_1}(\boldsymbol{x}) = -5.0$.

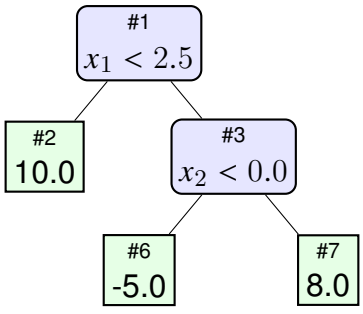

Figure 1: Parameters $\boldsymbol{\theta}_1$ for the regression tree used in Example 3

Exercise 3.3. Consider a regression tree defined by the following parameters $\boldsymbol{\theta}_2$ (see Figure 2).

- Node set $\mathcal{V} = \{1, 2, 3, 4, 5\}$
- (Derived from above) Internal node set $\mathcal{V}_{\text{int}} = \{1, 2\}$
- (Derived from above) Leaf node set $\mathcal{V}_{\text{leaf}} = \{3, 4, 5\}$
- Internal node parameters:
  - Node 1: $(j_1, t_1) = (2, 5.0)$
  - Node 2: $(j_2, t_2) = (1, 1.0)$
- Leaf node values:
  - Node 3: $c_3 = 20.0$
  - Node 4: $c_4 = 1.5$
  - Node 5: $c_5 = 3.0$

Calculate the output for the same input as in Example 3, $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$.

Answer. 1. Evaluate at node 1 (root): $x_{j_1} = x_2 = -1.0 < 5.0 = t_1$, so proceed to the left child node $\text{left}(1) = 2$.

2. Evaluate at node 2: $x_{j_2} = x_1 = 3.0 \geq 1.0 = t_2$, so proceed to the right child node $\text{right}(2) = 5$.

3. Node 5 is a leaf node ($\in \mathcal{V}_{\text{leaf}}$), and its value is $c_5 = 3.0$.

Therefore, the output is $y = f_{\boldsymbol{\theta}_2}(\boldsymbol{x}) = 3.0$.

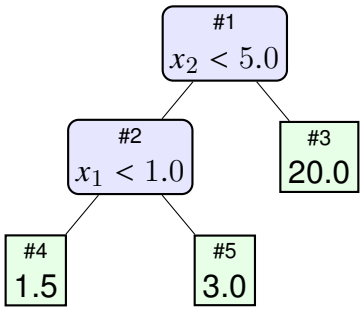

Figure 2: Parameters $\boldsymbol{\theta}_2$ for the regression tree used in Exercise 3

3.4 Example 4: k-Nearest Neighbors (k-NN)

k-NN is also an example where the parameters are of variable length.

Mathematical Formulation:

- **Parameters:** The sequence of past input-output data pairs itself. $\boldsymbol{\theta} \coloneqq \mathcal{D} = ((\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}$ The number of neighbors, k, can also be considered part of the parameters.

- **Function:** When a new input $\boldsymbol{x}_{new}$ is given,

  1. Calculate the distance $d(\boldsymbol{x}_{new}, \boldsymbol{x}_i)$ between $\boldsymbol{x}_{new}$ and all $\boldsymbol{x}_i$ in the dataset $\mathcal{D}$. The distance is usually the Euclidean distance $d(\boldsymbol{a}, \boldsymbol{b}) \coloneqq \sqrt{\sum_j (a_j - b_j)^2}$.
  2. Find the set of indices $\mathcal{N}_k(\boldsymbol{x}_{new}) \subseteq \{1, \ldots, N\}$ of the top k data points with the smallest distances.
  3. Aggregate their outputs $(y_i \mid i \in \mathcal{N}_k(\boldsymbol{x}_{new}))$ to obtain the final output.

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}_{new}) \coloneqq \text{aggregate}((y_i \mid i \in \mathcal{N}_k(\boldsymbol{x}_{new}))) \tag{7}$$

  The aggregation function aggregate is typically the mean for regression and the mode (majority vote) for classification.

Example 3.4 (Regression)**.** Let $k = 3$ and assume the parameters $\boldsymbol{\theta}_1$ consist of the following four data points. $\mathcal{D}_1 = ((\boldsymbol{x}_1, y_1) = (\begin{bmatrix}0\\0\end{bmatrix}, 5), (\boldsymbol{x}_2, y_2) = (\begin{bmatrix}1\\2\end{bmatrix}, 10), (\boldsymbol{x}_3, y_3) = (\begin{bmatrix}3\\0\end{bmatrix}, 20), (\boldsymbol{x}_4, y_4) = (\begin{bmatrix}4\\3\end{bmatrix}, 25))$ Calculate the output for a new input $\boldsymbol{x}_{new} = \begin{bmatrix}2\\1\end{bmatrix}$ (using the mean for regression).

1. Calculate the squared Euclidean distance to each data point (the square root can be omitted as it doesn't change the order).
   - $d(\boldsymbol{x}_{new}, \boldsymbol{x}_1)^2 = (2-0)^2 + (1-0)^2 = 4 + 1 = 5$
   - $d(\boldsymbol{x}_{new}, \boldsymbol{x}_2)^2 = (2-1)^2 + (1-2)^2 = 1 + 1 = 2$
   - $d(\boldsymbol{x}_{new}, \boldsymbol{x}_3)^2 = (2-3)^2 + (1-0)^2 = 1 + 1 = 2$
   - $d(\boldsymbol{x}_{new}, \boldsymbol{x}_4)^2 = (2-4)^2 + (1-3)^2 = 4 + 4 = 8$
2. Ordering by distance gives $\boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_1, \boldsymbol{x}_4$.
3. Since $k = 3$, we choose the top three: $(\boldsymbol{x}_2, y_2), (\boldsymbol{x}_3, y_3), (\boldsymbol{x}_1, y_1)$.
4. Their y values are $\{10, 20, 5\}$.
5. Calculate their mean: $y = \frac{10+20+5}{3} = \frac{35}{3} \approx 11.67$.

Thus, the output is $y \approx 11.67$.

Exercise 3.4. Let $k = 2$ and assume the parameters $\boldsymbol{\theta}_2$ consist of the following five data points. $\mathcal{D}_2 = ((\begin{bmatrix}1\\1\end{bmatrix}, 1), (\begin{bmatrix}2\\3\end{bmatrix}, 4), (\begin{bmatrix}-1\\2\end{bmatrix}, 6), (\begin{bmatrix}0\\-1\end{bmatrix}, 8), (\begin{bmatrix}4\\0\end{bmatrix}, 12))$ Calculate the output for the same new input as in Example 4, $\boldsymbol{x}_{new} = \begin{bmatrix}2\\1\end{bmatrix}$.

Answer. 1. Calculate the squared Euclidean distance to each data point.

- $d(\boldsymbol{x}_{new}, \begin{bmatrix}1\\1\end{bmatrix})^2 = (2-1)^2 + (1-1)^2 = 1 + 0 = 1$
- $d(\boldsymbol{x}_{new}, \begin{bmatrix}2\\3\end{bmatrix})^2 = (2-2)^2 + (1-3)^2 = 0 + 4 = 4$
- $d(\boldsymbol{x}_{new}, \begin{bmatrix}-1\\2\end{bmatrix})^2 = (2-(-1))^2 + (1-2)^2 = 9 + 1 = 10$
- $d(\boldsymbol{x}_{new}, \begin{bmatrix}0\\-1\end{bmatrix})^2 = (2-0)^2 + (1-(-1))^2 = 4 + 4 = 8$
- $d(\boldsymbol{x}_{new}, \begin{bmatrix}4\\0\end{bmatrix})^2 = (2-4)^2 + (1-0)^2 = 4 + 1 = 5$

2. The closest is $\begin{bmatrix}1\\1\end{bmatrix}$ (squared distance=1), and the next closest is $\begin{bmatrix}2\\3\end{bmatrix}$ (squared distance=4).
3. Since $k = 2$, we choose these two points: $(\begin{bmatrix}1\\1\end{bmatrix}, 1), (\begin{bmatrix}2\\3\end{bmatrix}, 4)$.
4. Their y values are $\{1, 4\}$.
5. Calculate their mean: $y = \frac{1+4}{2} = 2.5$.

Thus, the output is $y = 2.5$.

Remark 3.1. Models like k-NN, which store the training data itself or have a non-fixed number of parameters, are often called **non-parametric** models. This is historical terminology and slightly conflicts with this lecture's definition of a "parametric function." In this lecture, we consider non-parametric models as a type of "parametric function with a variable-length data structure as parameters."

4 Scheme of Training and Inference

By viewing machine learning models as parametric functions $(f_{\boldsymbol{\theta}})_{\boldsymbol{\theta}\in\Theta}$, the problem of solving a task is separated into two distinct phases. This distinction is legally important for the practical use of neural network models, as will be discussed later.

- **Training:** The process of finding the "optimal" parameters $\boldsymbol{\theta}^* \in \Theta$ that can best solve the task, using given **training data** $\mathcal{D}$ (e.g., a number of input-output pairs $((\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N))$). This is expressed as $\boldsymbol{\theta}^* = \mathfrak{A}((f_{\boldsymbol{\theta}}), \mathcal{D})$ using some **learning algorithm** $\mathfrak{A}$. In libraries like scikit-learn, this process is also called **fitting**.
- **Inference:** The process of fixing the parameters $\boldsymbol{\theta}^*$ obtained through training, constructing the specific function $f_{\boldsymbol{\theta}^*}$, and then calculating the output $\boldsymbol{y} = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$ for a new,

unknown input $\boldsymbol{x}$. This process is also called **prediction** (in fact, outside the context of neural networks, the term prediction is more common. Alternatively, the output $\boldsymbol{y}$ itself is sometimes called a prediction.).

This scheme of training and inference is a fundamental framework common to many machine learning models (Figure 3).

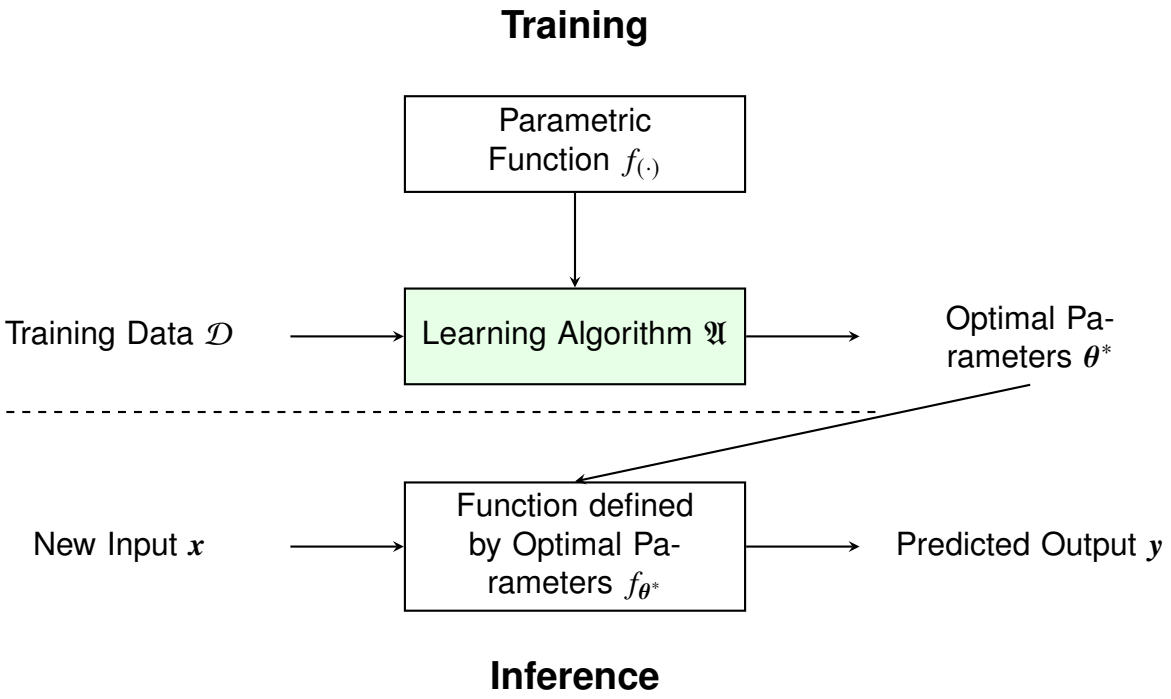

Figure 3: Scheme of training and inference using a parametric function. In the training phase, parameters that fit the data are determined. In the inference phase, predictions are made using the function embedded with those parameters.

Remark 4.1 (Why Emphasize the Difference Between Training and Inference?)**.** Students who have already learned the basics of machine learning will know the difference between training and inference (prediction) to some extent. Nevertheless, this course deliberately takes the time to explain it because the distinction between training and inference is essential for understanding issues that are unavoidable when we, as users, utilize neural network models, such as the publication and licensing of these models. For practical neural networks, the parameters for inference are often public, while the training algorithm is not. Also, the issue of licensing, which is unavoidable in practice, does not guarantee that the license is the same for the inference part and the training part. Therefore, without the distinction between training and inference, it is impossible to understand the essence of these problems. Handling neural network models without understanding their licenses carries legal risks, so it is dangerous to use them without a clear distinction between training and inference. The concepts of publication and licensing for neural network models will be covered in detail later in this course.

5 Specific Examples of Learning Algorithms

Inference is nothing more than the process of applying a new input $\boldsymbol{x}$ to a function $f_{\boldsymbol{\theta}^*}$ constructed using pre-trained parameters $\boldsymbol{\theta}^*$ to obtain an output $\boldsymbol{y}$, i.e., the computation $\boldsymbol{y} = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$. All the calculation examples we have seen so far correspond to this inference phase.

On the other hand, the training phase, the process of finding the "optimal" parameters $\boldsymbol{\theta}^*$ from given data $\mathcal{D}$, differs greatly depending on the type of parametric function. Here, we will look at specific examples of learning algorithms for some of the models mentioned so far. However, the goal here is not to memorize the learning algorithms, but to understand that training is indeed different from inference, that it is a process of determining parameters, and that the actual process generally differs depending on the parametric function.

5.1 Example 1: Training of k-Nearest Neighbors (k-NN)

The learning algorithm for k-NN is one of the simplest imaginable. The learning algorithm $\mathfrak{A}$ simply stores the given training data $\mathcal{D} = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N))$ as the parameters $\boldsymbol{\theta}$. In other words, training is completed just by "memorizing the data."

For example, the parameters $\boldsymbol{\theta}_1$ used in Example 4 of the previous section are precisely the training data sequence $\mathcal{D}_1 = ((\begin{bmatrix}0\\0\end{bmatrix}, 5), (\begin{bmatrix}1\\2\end{bmatrix}, 10), (\begin{bmatrix}3\\0\end{bmatrix}, 20), (\begin{bmatrix}4\\3\end{bmatrix}, 25))$ itself. Models like this, where most of the computation is deferred until inference time and the training process simply involves storing the data, are sometimes called **lazy learning** models.

5.2 Example 2: Training of Linear Regression

For models like linear regression, where parameters are represented by a fixed-length numerical vector, training is a more active process. In many cases, training is formulated as an **optimization problem** of "minimizing a function that measures the 'badness' of the model's predictions on the training data."

Definition 5.1 (Loss Function)**.** Given a parametric function f and a training data sequence $\mathcal{D}$, a function L that takes the parameters $\boldsymbol{\theta}$ of f as input and outputs a non-negative real value $L(\boldsymbol{\theta})$ representing how "bad" it is for the data sequence is called a **cost function**, **loss function**, or **objective function**. The goal of the learning algorithm is then to find the parameters $\boldsymbol{\theta}^*$ that minimize this cost function.

$$\boldsymbol{\theta}^* \coloneqq \arg\min_{\boldsymbol{\theta} \in \Theta} L(\boldsymbol{\theta}) \tag{8}$$

Remark 5.1 (On the terms Cost Function, Loss Function, and Objective Function)**.** The term objective function is general but can be used for cases where a smaller value is better as well as when a larger value is better, so caution is needed. The term loss function clearly

indicates that a smaller value is better, but it is usually used only when the function depends on f or $\boldsymbol{\theta}$ only through $f_{\boldsymbol{\theta}}$, and it refers to the part that excludes regularization terms. The term cost function is used for any function where a smaller value is better, but it is used somewhat less frequently.

5.2.1 Least Squares Method

In linear regression, the parameters are $\boldsymbol{w}$ and b. The **Sum of Squared Residuals (SSR)** is widely used as a cost function (loss function) for the parameters of parametric functions with real-valued outputs, including linear regression. Given training data $\mathcal{D} = ((\boldsymbol{x}_i, y_i))_{i=1}^N$, the SSR is defined as follows. A smaller value of this indicates that the model fits the training data well. The method of minimizing this cost (loss) is called the **least squares method**.

Definition 5.2 (Sum of Squared Residuals)**.** Given training data $\mathcal{D} = ((\boldsymbol{x}_i, y_i))_{i=1}^N$, the Sum of Squared Residuals (SSR) is defined as the sum of the squares of the differences (the **residuals**) between the actual outputs y_i and the model's predicted values $f(\boldsymbol{x}_i)$.

$$\begin{aligned} L_{\mathrm{SSR}}(\boldsymbol{w}, b) &\coloneqq \sum_{i=1}^{N} (y_i - f_{(\boldsymbol{w},b)}(\boldsymbol{x}_i))^2 \\ &= \sum_{i=1}^{N} (y_i - (\boldsymbol{w}^\top \boldsymbol{x}_i + b))^2 \end{aligned} \tag{9}$$

5.2.2 Least Squares Solution

The least squares solution can be found analytically using linear algebra. First, to include the bias term b in the weight vector, we consider a new vector $\tilde{\boldsymbol{x}}_i \coloneqq \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,d_{\mathrm{in}}} \\ 1 \end{bmatrix} \in \mathbb{R}^{d_{\mathrm{in}}+1}$ by appending a 1 to each input $\boldsymbol{x}_i \in \mathbb{R}^{d_{\mathrm{in}}}$. Similarly, the parameters become $\tilde{\boldsymbol{w}} \coloneqq \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{d_{\mathrm{in}}} \\ b \end{bmatrix} \in \mathbb{R}^{d_{\mathrm{in}}+1}$. Then, the model's prediction can be written concisely as $f(\boldsymbol{x}_i) = \tilde{\boldsymbol{w}}^\top \tilde{\boldsymbol{x}}_i$.

We represent the entire training data in matrix form. From the N data points $(\tilde{\boldsymbol{x}}_i, y_i)$, we

create a design matrix $\boldsymbol{X} \in \mathbb{R}^{N,d_{\text{in}}+1}$ and a target variable vector $\boldsymbol{y} \in \mathbb{R}^N$ as follows.

$$\boldsymbol{X} \coloneqq \begin{bmatrix} \tilde{\boldsymbol{x}}_1^\top \\ \tilde{\boldsymbol{x}}_2^\top \\ \vdots \\ \tilde{\boldsymbol{x}}_N^\top \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d_{\text{in}}} & 1 \\ x_{2,1} & x_{2,2} & \dots & x_{2,d_{\text{in}}} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,d_{\text{in}}} & 1 \end{bmatrix}, \quad \boldsymbol{y} \coloneqq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{10}$$

Then, the vector of predicted values $\hat{\boldsymbol{y}}$ is $\hat{\boldsymbol{y}} = \boldsymbol{X}\tilde{\boldsymbol{w}}$, and the loss function SSR can be written using the 2-norm $\|\cdot\|_2$ as follows.

$$\begin{aligned} L_{\text{SSR}}(\tilde{\boldsymbol{w}}) &= \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \|\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}}\|_2^2 \\ &= (\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}})^\top (\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}}) \end{aligned} \tag{11}$$

We want to find the minimizer of this cost function (loss function) SSR, which is given by the following normal equation.

Definition 5.3 (Normal Equation)**.** Given a design matrix $\boldsymbol{X} \in \mathbb{R}^{N,d_{\text{in}}+1}$ and a target variable vector $\boldsymbol{y} \in \mathbb{R}^N$, the following equation for $\tilde{\boldsymbol{w}} \in \mathbb{R}^{d_{\text{in}}+1}$ is called the **normal equation**.

$$(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}. \tag{12}$$

Theorem 5.1 (Least Squares Solution)**.** Given a design matrix $\boldsymbol{X} \in \mathbb{R}^{N,d_{\text{in}}+1}$ and a target variable vector $\boldsymbol{y} \in \mathbb{R}^N$, it is a necessary and sufficient condition for $\tilde{\boldsymbol{w}}$ to be a solution to the normal equation of Definition 5.3 that it minimizes the Sum of Squared Residuals (SSR) $L_{\text{SSR}}(\tilde{\boldsymbol{w}})$. In particular, if $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible (has an inverse), the normal equation has a unique solution given by:

$$\tilde{\boldsymbol{w}}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{13}$$

Therefore, $\tilde{\boldsymbol{w}}^*$ as expressed above is the unique vector that minimizes the SSR $L_{\text{SSR}}(\tilde{\boldsymbol{w}})$.

Proof. We find $\tilde{\boldsymbol{w}}$ that minimizes the loss $L(\tilde{\boldsymbol{w}})$. We expand $L(\tilde{\boldsymbol{w}})$ and set its gradient with respect to $\tilde{\boldsymbol{w}}$, $\nabla_{\tilde{\boldsymbol{w}}} L$, to 0. (Readers interested in the details of differentiation with respect to a vector may refer to the appendix at the end.)

$$L_{\text{SSR}}(\tilde{\boldsymbol{w}}) = \boldsymbol{y}^\top \boldsymbol{y} - 2\tilde{\boldsymbol{w}}^\top \boldsymbol{X}^\top \boldsymbol{y} + \tilde{\boldsymbol{w}}^\top \boldsymbol{X}^\top \boldsymbol{X}\tilde{\boldsymbol{w}}, \tag{14}$$

$$\nabla_{\tilde{\boldsymbol{w}}} L_{\text{SSR}} = -2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\tilde{\boldsymbol{w}}. \tag{15}$$

Rearranging the equation $\nabla_{\tilde{\boldsymbol{w}}} L_{\text{SSR}} = \boldsymbol{0}$, we see that it is equivalent to the normal equation below.

$$(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}. \tag{16}$$

We confirm that the solution to this equation minimizes the loss. The Hessian matrix (second derivative) of L_{SSR} is $\nabla_{\tilde{\boldsymbol{w}}}^2 L = 2\boldsymbol{X}^\top \boldsymbol{X}$. For any vector $\boldsymbol{z}$, $\boldsymbol{z}^\top (\boldsymbol{X}^\top \boldsymbol{X})\boldsymbol{z} = (\boldsymbol{X}\boldsymbol{z})^\top (\boldsymbol{X}\boldsymbol{z}) = \|\boldsymbol{X}\boldsymbol{z}\|_2^2 \geq 0$, so $\boldsymbol{X}^\top \boldsymbol{X}$ is a positive semi-definite matrix. This means that $L_{\text{SSR}}(\tilde{\boldsymbol{w}})$ is a convex function, and

therefore $\nabla_{\tilde{\boldsymbol{w}}} = 0$ is a necessary and sufficient condition for minimizing the Sum of Squared Residuals (SSR) $L_{\text{SSR}}(\tilde{\boldsymbol{w}})$.

If $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible, the equivalence between $(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}$ and $\tilde{\boldsymbol{w}}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$ is clear.

□

Geometric Interpretation: The least squares method can be seen as the operation of **orthogonal projection** of the target variable vector $\boldsymbol{y}$ onto the subspace spanned by the column vectors of the design matrix $\boldsymbol{X}$ (the column space $C(\boldsymbol{X})$). In other words, it is the problem of finding the vector $\hat{\boldsymbol{y}} = \boldsymbol{X}\tilde{\boldsymbol{w}}$ in the column space that minimizes the Euclidean distance $\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|$ to $\boldsymbol{y}$. In this case, the error vector $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ must be orthogonal to the column space $C(\boldsymbol{X})$. This means that the inner product of $\boldsymbol{e}$ with every column vector of $\boldsymbol{X}$ is 0, i.e., $\boldsymbol{X}^\top \boldsymbol{e} = \boldsymbol{0}$.

$$\boldsymbol{X}^\top(\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}}) = \boldsymbol{0} \implies \boldsymbol{X}^\top \boldsymbol{X}\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y} \tag{17}$$

This is precisely the normal equation, showing that the least squares method is geometrically calculating an orthogonal projection.

Example 5.1 (Least Squares Solution for Linear Regression)**.** Suppose we are given the following training data $\mathcal{D}_A$ consisting of three data points.

$$\mathcal{D}_A = ((\boldsymbol{x}_1, y_1) = (\begin{bmatrix}3\\4\end{bmatrix}, -4.5), (\boldsymbol{x}_2, y_2) = (\begin{bmatrix}1\\0\end{bmatrix}, 1.5), (\boldsymbol{x}_3, y_3) = (\begin{bmatrix}0\\1\end{bmatrix}, -1.5)) \tag{18}$$

First, we construct the design matrix $\boldsymbol{X}$ and the target variable vector $\boldsymbol{y}$.

$$\boldsymbol{X} = \begin{bmatrix}3 & 4 & 1\\ 1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix}-4.5\\ 1.5\\ -1.5\end{bmatrix} \tag{19}$$

Next, we calculate $\boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{X}^\top \boldsymbol{y}$.

$$\begin{aligned}\boldsymbol{X}^\top \boldsymbol{X} &= \begin{bmatrix}3 & 1 & 0\\ 4 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}3 & 4 & 1\\ 1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix}\\ &= \begin{bmatrix}10 & 12 & 4\\ 12 & 17 & 5\\ 4 & 5 & 3\end{bmatrix}\end{aligned} \tag{20}$$

$$\begin{aligned}\boldsymbol{X}^\top \boldsymbol{y} &= \begin{bmatrix} 3 & 1 & 0 \\ 4 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -4.5 \\ 1.5 \\ -1.5 \end{bmatrix} \\ &= \begin{bmatrix} -13.5 + 1.5 \\ -18.0 - 1.5 \\ -4.5 + 1.5 - 1.5 \end{bmatrix} = \begin{bmatrix} -12.0 \\ -19.5 \\ -4.5 \end{bmatrix}\end{aligned} \tag{21}$$

We solve the normal equation $(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}$.

$$\begin{bmatrix} 10 & 12 & 4 \\ 12 & 17 & 5 \\ 4 & 5 & 3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} -12.0 \\ -19.5 \\ -4.5 \end{bmatrix} \tag{22}$$

Solving this system of linear equations (e.g., by calculating $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ and multiplying from the left), the solution is

$$\tilde{\boldsymbol{w}}^* = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} 1.0 \\ -2.0 \\ 0.5 \end{bmatrix} \tag{23}$$

Thus, training is the process of determining parameters to fit the data.

Note that this result matches the parameters $(\boldsymbol{w} = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}, b = 0.5)$ used in the example in the previous section. Please review that example to reconfirm how inference and prediction can be done using the obtained function.

Exercise 5.1 (Least Squares Exercise)**.** Suppose we are given the following training data $\mathcal{D}_B$ consisting of four data points.

$$\mathcal{D}_B = ((\begin{bmatrix} 3 \\ 4 \end{bmatrix}, 9.5), (\begin{bmatrix} 2 \\ 0 \end{bmatrix}, -2.0), (\begin{bmatrix} 0 \\ 2 \end{bmatrix}, 5.0), (\begin{bmatrix} 1 \\ 1 \end{bmatrix}, 1.5)) \tag{24}$$

Using the least squares method, find the parameters $(\boldsymbol{w}, b)$ of the linear regression model that best fits this data.

Answer. First, construct the design matrix $\boldsymbol{X}$ and the target variable vector $\boldsymbol{y}$.

$$\boldsymbol{X} = \begin{bmatrix} 3 & 4 & 1 \\ 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} 9.5 \\ -2.0 \\ 5.0 \\ 1.5 \end{bmatrix} \tag{25}$$

Next, calculate $\boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{X}^\top \boldsymbol{y}$.

$$\boldsymbol{X}^\top \boldsymbol{X} = \begin{bmatrix} 3 & 2 & 0 & 1 \\ 4 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 3 & 4 & 1 \\ 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 14 & 13 & 6 \\ 13 & 21 & 7 \\ 6 & 7 & 4 \end{bmatrix}$$

$$\boldsymbol{X}^\top \boldsymbol{y} = \begin{bmatrix} 3 & 2 & 0 & 1 \\ 4 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 9.5 \\ -2.0 \\ 5.0 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 28.5 - 4.0 + 0 + 1.5 \\ 38.0 + 0 + 10.0 + 1.5 \\ 9.5 - 2.0 + 5.0 + 1.5 \end{bmatrix} = \begin{bmatrix} 26.0 \\ 49.5 \\ 14.0 \end{bmatrix}$$

Solve the normal equation $(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}$.

$$\begin{bmatrix} 14 & 13 & 6 \\ 13 & 21 & 7 \\ 6 & 7 & 4 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} 26.0 \\ 49.5 \\ 14.0 \end{bmatrix} \tag{26}$$

Solving this system of linear equations, the solution is

$$\tilde{\boldsymbol{w}}^* = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} -0.5 \\ 3.0 \\ -1.0 \end{bmatrix} \tag{27}$$

This matches the parameters $(\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0)$ used in Exercise 1 of the previous section.

5.3 Diversity of Learning Algorithms and Neural Networks

As we have seen, the learning algorithm for k-NN is just to store the data, while for linear regression it is to minimize a loss function (specifically, by solving the normal equation). Thus, different parametric functions require completely different learning algorithms. The learning for decision trees (not detailed here, but generally a greedy algorithm that finds the feature and threshold that best splits the information) also takes a different approach. This need to consider a dedicated learning algorithm for each model is one of the cumbersome aspects of learning machine learning.

In contrast, **neural networks**, which we will learn about from the next lecture onwards, can represent a wide variety of parametric functions, from simple functions similar to linear regression to extremely complex functions that handle images and audio, by changing their structure (architecture). And one of its greatest strengths is that, despite being able to represent such a wide variety of functions, its learning algorithm is **unified**. Specifically, most neural networks can be trained under a common framework: calculating the gradient of the

loss function using **backpropagation** and minimizing the loss using **stochastic gradient descent (SGD)** or its variants.

This ”high expressive power” and ”uniformity of learning method” are among the reasons why neural networks form the core of modern AI technology.

6 Summary and Future Outlook

6.1 Today's Summary

- If a **parametric function**, where a specific function is determined by specifying the values of its parameters, is given, many of the tasks that humans want to solve can be reduced to a parameter determination problem.
- Various AI models such as linear regression, logistic regression, decision trees, and k-NN can be understood uniformly as parametric functions.
- Machine learning is separated into a **training** phase, which finds the optimal parameters of a parametric function from past data, and an **inference** phase, which calculates the output for an unknown input using those parameters.

6.2 Next Time

This time, we viewed various machine learning models within the unified framework of parametric functions. This framework is an important foundation for understanding neural networks, which we will learn about from the next lecture onwards.

Next time, we will strictly define what a **neural network** is—a particularly expressive type of parametric function that forms the core of modern AI technology—using the mathematical tool of a **Directed Acyclic Graph (DAG)**. Then, we will learn about the distinction between ”architecture” and ”checkpoints,” concepts that are extremely important in the practical application of modern AI.

A Appendix: Differentiation by a Vector

In the derivation of the least squares solution in the main text, we used the calculation of differentiating a vector by a vector (gradient calculation). Here, we supplement the notation and formulas used in that calculation. The notation and formulas in this appendix conform to The Matrix Cookbook [1].

A.1 Definition of Gradient

The derivative (gradient) of a scalar-valued function $f(\boldsymbol{x})$ with respect to a vector $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ is defined as a vector of partial derivatives with respect to each element (Denominator layout).

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} \coloneqq \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{28}$$

A.2 Formulas Used in the Text

The following two formulas were used in the gradient calculation of the loss function in the main text.

A.2.1 Formula 1: Derivative of a Linear Term

The gradient of the inner product (a scalar) of a constant vector $\boldsymbol{a}$ and a variable vector $\boldsymbol{x}$ with respect to $\boldsymbol{x}$ is $\boldsymbol{a}$ itself.

$$\frac{\partial(\boldsymbol{a}^\top \boldsymbol{x})}{\partial \boldsymbol{x}} = \boldsymbol{a} \tag{29}$$

This can be confirmed by noting that $\boldsymbol{a}^\top \boldsymbol{x} = \sum_i a_i x_i$, and taking the partial derivative with respect to each x_j gives $\frac{\partial}{\partial x_j} \sum_i a_i x_i = a_j$. Summarizing this in vector form gives $\boldsymbol{a}$.

A.2.2 Formula 2: Derivative of a Quadratic Form

The gradient of a quadratic form (a scalar) created from a constant matrix $\boldsymbol{A}$ and a variable vector $\boldsymbol{x}$ with respect to $\boldsymbol{x}$ is as follows.

$$\frac{\partial(\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x})}{\partial \boldsymbol{x}} = (\boldsymbol{A} + \boldsymbol{A}^\top)\boldsymbol{x} \tag{30}$$

In particular, if the matrix $\boldsymbol{A}$ is symmetric ($\boldsymbol{A} = \boldsymbol{A}^\top$), this formula simplifies.

$$\frac{\partial(\boldsymbol{x}^\top \boldsymbol{A} \boldsymbol{x})}{\partial \boldsymbol{x}} = 2\boldsymbol{A}\boldsymbol{x} \quad (\text{if } \boldsymbol{A} \text{ is symmetric}) \tag{31}$$

A.3 Derivation of the Loss Function Gradient

Using the formulas above, we calculate the gradient of the loss function $L(\tilde{\boldsymbol{w}})$ from the main text.

$$L(\tilde{\boldsymbol{w}}) = \boldsymbol{y}^\top \boldsymbol{y} - 2\tilde{\boldsymbol{w}}^\top \boldsymbol{X}^\top \boldsymbol{y} + \tilde{\boldsymbol{w}}^\top (\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} \tag{32}$$

We differentiate each term of this equation with respect to $\tilde{\boldsymbol{w}}$.

1. **1st term $\boldsymbol{y}^\top \boldsymbol{y}$:** This is a constant that does not depend on $\tilde{\boldsymbol{w}}$, so its derivative is $\boldsymbol{0}$.

2. **2nd term $-2\tilde{\boldsymbol{w}}^\top \boldsymbol{X}^\top \boldsymbol{y}$:** This has the form $-2\boldsymbol{a}^\top \tilde{\boldsymbol{w}}$ (where $\boldsymbol{a} = \boldsymbol{X}^\top \boldsymbol{y}$). Using Formula 1, the gradient of this term is:
$$\frac{\partial(-2\tilde{\boldsymbol{w}}^\top (\boldsymbol{X}^\top \boldsymbol{y}))}{\partial \tilde{\boldsymbol{w}}} = -2(\boldsymbol{X}^\top \boldsymbol{y}) \tag{33}$$

3. **3rd term $\tilde{\boldsymbol{w}}^\top (\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}}$:** This has the form $\tilde{\boldsymbol{w}}^\top \boldsymbol{A}\tilde{\boldsymbol{w}}$ (where $\boldsymbol{A} = \boldsymbol{X}^\top \boldsymbol{X}$). Here, since $\boldsymbol{A}^\top = (\boldsymbol{X}^\top \boldsymbol{X})^\top = \boldsymbol{X}^\top (\boldsymbol{X}^\top)^\top = \boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{A}$, the matrix $\boldsymbol{A}$ is symmetric. Therefore, using Formula 2 (for the symmetric case), the gradient of this term is:
$$\frac{\partial(\tilde{\boldsymbol{w}}^\top (\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}})}{\partial \tilde{\boldsymbol{w}}} = 2(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} \tag{34}$$

Summing these up, we obtain the gradient equation shown in the main text.

$$\begin{aligned} \nabla_{\tilde{\boldsymbol{w}}} L &= \boldsymbol{0} - 2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\tilde{\boldsymbol{w}} \\ &= -2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\tilde{\boldsymbol{w}} \end{aligned} \tag{35}$$

References

[1] Kaare Brandt Petersen and Michael Syskind Pedersen. The Matrix Cookbook. Technical report, Technical University of Denmark, 2012. Version: November 15, 2012.