AI Applications Lecture 2

Machine Learning Models and Parametric Functions

SUZUKI, Atsushi
Jing WANG

Outline

Introduction

Preparation: Mathematical Notations

Parametric Functions

Scheme of Training and Inference

Specific Examples of Learning Algorithms

Summary and Future Outlook

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**.

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**.
- Real-world entities such as text, images, and audio can be converted into numerical data (vectors or tensors).

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**.
- Real-world entities such as text, images, and audio can be converted into numerical data (vectors or tensors).
- Thus, solving a task with AI boils down to a **function determination problem**: finding an appropriate **function** from an input space to an 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**.

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.

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 determining parameters using data, and explain the difference between **training** and **inference**.

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.

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.

1.3 Policy of This Lecture

As a first step, we will learn about "parametric functions," a mathematical framework common to many machine learning models, including neural networks.

1.3 Policy of This Lecture

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.
- Clearly distinguish between the two major phases of AI development: **training** and **inference**.

1.3 Policy of This Lecture

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.
- Clearly distinguish between the two major phases of AI development: **training** and **inference**.

This distinction is an essential foundation for understanding neural network applications and practical issues such as licensing.

Preparation: Mathematical Notations

2. Preparation: Mathematical Notations

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

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: The left-hand side is defined by the right-hand side.

2. Preparation: Mathematical Notations

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

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: The left-hand side is defined by the right-hand side.
- **Set:** Denoted by uppercase calligraphic letters (e.g., $\mathcal{A}$).
  - $x \in \mathcal{A}$: element x belongs to set $\mathcal{A}$.
  - $\{a, b, c\}$: The set consisting of elements a, b, c.
  - $\{x \in \mathcal{A} | P(x)\}$: The set of elements in $\mathcal{A}$ for which $P(x)$ is true.
  - $\mathbb{R}$ (reals), $\mathbb{Z}$ (integers), with subscripts like $_{>0}$ (positive) or $_{\geq 0}$ (non-negative).

2. Preparation: Mathematical Notations

- **Function:**
  - $f : X \to Y$: f is a map from set X to set Y.
  - $y = f(x)$: The output of f for input $x \in X$ is $y \in Y$.

2. Preparation: Mathematical Notations

- **Function:**
  - $f : X \to Y$: f is a map from set X to set Y.
  - $y = f(x)$: The output of f for input $x \in X$ is $y \in Y$.
- **Vector:** Denoted by bold italic lowercase letters (e.g., $\boldsymbol{v}$).
  - A vector is a column of numbers.
  - $\boldsymbol{v} \in \mathbb{R}^n$: an n-dimensional real vector.
  - The i-th element is v_i. $\boldsymbol{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}^\top$.

2. Preparation: Mathematical Notations

- **Matrix:** Denoted by bold italic uppercase letters (e.g., $\boldsymbol{A}$).
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$: an $m \times n$ real matrix.
  - The element in the i-th row, j-th column is $a_{i,j}$.
  - Transpose is denoted as $\boldsymbol{A}^\top$, where $(\boldsymbol{A}^\top)_{ij} = a_{ji}$.

2. Preparation: Mathematical Notations

- **Matrix:** Denoted by bold italic uppercase letters (e.g., $\boldsymbol{A}$).
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$: an $m \times n$ real matrix.
  - The element in the i-th row, j-th column is $a_{i,j}$.
  - Transpose is denoted as $\boldsymbol{A}^\top$, where $(\boldsymbol{A}^\top)_{ij} = a_{ji}$.
- **Tensor:**
  - In this lecture, a tensor is simply a multi-dimensional array.
  - Vector = 1st-order tensor, Matrix = 2nd-order tensor.
  - Higher-order tensors are denoted by $\underline{\boldsymbol{A}}$.

Parametric Functions

3. Parametric Functions

We have seen that solving a task can be formulated as a function determination problem.

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 function must be specified by numerical data (e.g., a sequence of floating-point numbers).

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 function must be specified by numerical data (e.g., a sequence of floating-point numbers).

Let's formalize this idea.

Definition (Parametric Function)

If there exists a set Θ (the **parameter space**), and for each element $\boldsymbol{\theta} \in \Theta$, a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ 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**.

3. Parametric Functions

Given a parametric function, finding a function f is replaced by finding an appropriate parameter $\boldsymbol{\theta}$.

3. Parametric Functions

Given a parametric function, finding a function f is replaced by finding an appropriate parameter $\boldsymbol{\theta}$.

Why look at models other than neural networks?

- **Linear/Logistic Regression**: Simple, foundational, and a special case of neural networks.
- **Decision Trees**: A different kind of function not representable by typical NNs. Parameters are not a fixed-size vector. Intuitive and powerful.
- **k-Nearest Neighbors (k-NN)**: Another non-NN example. Also has variable-size parameters. Extremely simple training algorithm, which is great for illustrating the concepts of training and inference.

3.1 Example 1: Linear Regression

Considers mapping a d_{in}-dimensional vector $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ to a 1-dimensional real number $y \in \mathbb{R}$.

3.1 Example 1: Linear Regression

Considers mapping a d_{in}-dimensional vector $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ to a 1-dimensional real number $y \in \mathbb{R}$.

- **Parameters:** $\boldsymbol{\theta} \coloneqq (\boldsymbol{w}, b)$, where $\boldsymbol{w} \in \mathbb{R}^{d_{\text{in}}}$ is a weight vector and $b \in \mathbb{R}$ is a bias.

3.1 Example 1: Linear Regression

Considers mapping a d_{in}-dimensional vector $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ to a 1-dimensional real number $y \in \mathbb{R}$.

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

3.1 Example 1: Linear Regression

Considers mapping a d_{in}-dimensional vector $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$ to a 1-dimensional real number $y \in \mathbb{R}$.

- **Parameters:** $\boldsymbol{\theta} \coloneqq (\boldsymbol{w}, b)$, where $\boldsymbol{w} \in \mathbb{R}^{d_{\text{in}}}$ is a weight vector and $b \in \mathbb{R}$ is a bias.
- **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

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

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

Exercise

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

Exercise

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

Answer:

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

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

3.2 Example 2: Logistic Regression

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

3.2 Example 2: Logistic Regression

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

- **Parameters:** Same as linear regression, $\boldsymbol{\theta} \coloneqq (\boldsymbol{w}, b)$.

3.2 Example 2: Logistic Regression

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

- **Parameters:** Same as linear regression, $\boldsymbol{\theta} \coloneqq (\boldsymbol{w}, b)$.
- **Function:** $f_{\boldsymbol{w},b}(\boldsymbol{x}) \coloneqq \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$.

3.2 Example 2: Logistic Regression

The input is $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, 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**.

3.2 Example 2: Logistic Regression

Example

Using the same parameters and input as the first linear regression example ($\boldsymbol{w} = \begin{bmatrix} 1.0 \\ -2.0 \end{bmatrix}, b = 0.5, \boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$), we first calculate the linear part $z = \boldsymbol{w}^\top \boldsymbol{x} + b = -4.5$.

The final output is:

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

3.2 Example 2: Logistic Regression

Exercise

For the same input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$, calculate the output using the parameters from the linear regression exercise, $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$.

Exercise

For the same input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$, calculate the output using the parameters from the linear regression exercise, $\boldsymbol{w} = \begin{bmatrix} -0.5 \\ 3.0 \end{bmatrix}, b = -1.0$.

Answer: First, the linear part is $z = \boldsymbol{w}^\top \boldsymbol{x} + b = 9.5$. The final output is:

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

3.3 Example 3: Decision Tree / Regression Tree

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

3.3 Example 3: Decision Tree / Regression Tree

Decision trees are examples where parameters are not 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 binary tree structure with nodes indexed by integers.
- **Parameters θ define:**
  - The set of nodes in the tree ($\mathcal{V}$).
  - For each **internal node**: which input feature to check (j_i) and a threshold (t_i).
  - For each **leaf node**: the output value (c_l).
- **Function $f_{\boldsymbol{\theta}}(\boldsymbol{x})$:**
  - Start at the root node.
  - At each internal node, compare x_{j_i} with t_i to decide whether to go left or right.
  - When a leaf node is reached, return its value c_l.

3.3 Example 3: Regression Tree

Example (Regression Tree)

Let $d_{\text{in}} = 2$. Consider a tree defined by parameters $\boldsymbol{\theta}_1$:

- Internal nodes:
  - Node 1: split on $x_1 < 2.5$
  - Node 3: split on $x_2 < 0.0$
- Leaf nodes:
  - Node 2: value = 10.0
  - Node 6: value = -5.0
  - Node 7: value = 8.0

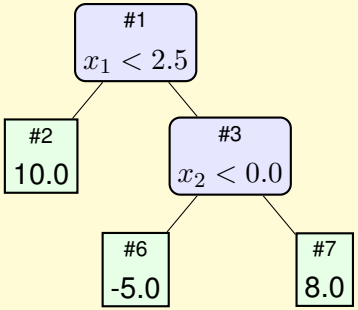


Figure 1: Parameters $\boldsymbol{\theta}_1$

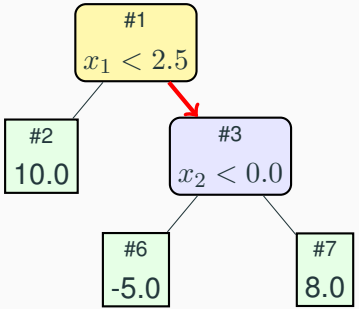
Calculate the output for input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$.

3.3 Example 3: Regression Tree

Input: $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$

Step 1: Evaluate at root node 1.

- The rule is: $x_1 < 2.5$.
- Our input has $x_1 = 3.0$.
- Since $3.0 \geq 2.5$, the condition is false.
- We proceed to the **right** child, node 3.

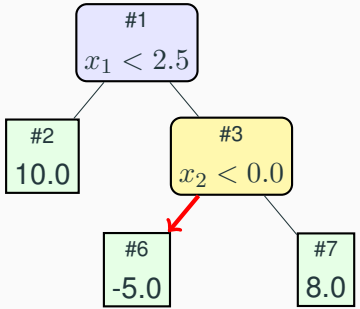


3.3 Example 3: Regression Tree

Input: $x = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$

Step 2: Evaluate at node 3.

- The rule is: $x_2 < 0.0$.
- Our input has $x_2 = -1.0$.
- Since $-1.0 < 0.0$, the condition is true.
- We proceed to the **left** child, node 6.



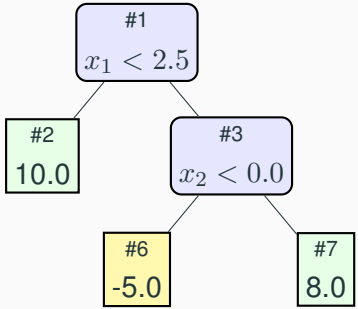
3.3 Example 3: Regression Tree

Input: $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$

Step 3: Arrive at leaf node 6.

- Node 6 is a leaf node.
- The value associated with this leaf is $c_6 = -5.0$.
- This is our final output.

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



Exercise

Consider a different tree defined by parameters $\boldsymbol{\theta}_2$. Calculate the output for the same input $\boldsymbol{x} = \begin{bmatrix} 3.0 \\ -1.0 \end{bmatrix}$.

- Internal nodes:
  - Node 1: $x_2 < 5.0$
  - Node 2: $x_1 < 1.0$
- Leaf nodes:
  - Node 3: 20.0
  - Node 4: 1.5
  - Node 5: 3.0

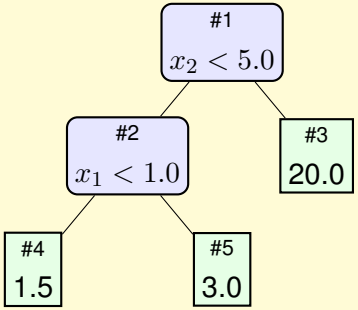


Figure 2: Parameters $\boldsymbol{\theta}_2$

3.3 Example 3: Regression Tree

Answer:

1. At node 1 (root): $x_2 = -1.0 < 5.0$ (true) $\rightarrow$ go left to node 2.
2. At node 2: $x_1 = 3.0 \geq 1.0$ (false) $\rightarrow$ go right to node 5.
3. Node 5 is a leaf node. Its value is $c_5 = 3.0$.

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

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

k-NN is another model where the parameters are of variable length.

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

k-NN is another model where the parameters are of variable length.

Mathematical Formulation:

- **Parameters:** The training dataset itself, $\boldsymbol{\theta} \coloneqq \mathcal{D} = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N))$, and the number of neighbors, k.

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

k-NN is another model where the parameters are of variable length.

Mathematical Formulation:

- **Parameters:** The training dataset itself, $\boldsymbol{\theta} \coloneqq \mathcal{D} = ((\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N))$, and the number of neighbors, k.
- **Function:** For a new input $\boldsymbol{x}_{new}$:
  1. Calculate the distance from $\boldsymbol{x}_{new}$ to every $\boldsymbol{x}_i$ in the dataset.
  2. Find the k data points $(\boldsymbol{x}_i, y_i)$ with the smallest distances.
  3. Aggregate their outputs y_i to get the final result. (e.g., mean for regression, majority vote for classification).

Example (Regression)

Let $k = 3$. The parameters $\boldsymbol{\theta}_1$ are the dataset $\mathcal{D}_1$:

$$\mathcal{D}_1 = \left(\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, 5 \right), \left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}, 10 \right), \left(\begin{bmatrix} 3 \\ 0 \end{bmatrix}, 20 \right), \left(\begin{bmatrix} 4 \\ 3 \end{bmatrix}, 25 \right) \right)$$

Calculate the output for a new input $\boldsymbol{x}_{new} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$.

3.4 Example 4: k-NN Regression

Input: $\boldsymbol{x}_{new} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$. **Data:** $\mathcal{D}_1$.

Step 1: Calculate squared Euclidean distance to each data point.

- $d(\boldsymbol{x}_{new}, \boldsymbol{x}_1)^2 = (2-0)^2 + (1-0)^2 = 4 + 1 = 5$
- $d(\boldsymbol{x}_{new}, \boldsymbol{x}_2)^2 = (2-1)^2 + (1-2)^2 = 1 + 1 = 2$
- $d(\boldsymbol{x}_{new}, \boldsymbol{x}_3)^2 = (2-3)^2 + (1-0)^2 = 1 + 1 = 2$
- $d(\boldsymbol{x}_{new}, \boldsymbol{x}_4)^2 = (2-4)^2 + (1-3)^2 = 4 + 4 = 8$

Step 2: Find the top $k = 3$ nearest neighbors.

- The distances are: 5, 2, 2, 8.
- The smallest three distances correspond to data points $\boldsymbol{x}_2$ (dist 2), $\boldsymbol{x}_3$ (dist 2), and $\boldsymbol{x}_1$ (dist 5).
- Their corresponding outputs are $y_2 = 10, y_3 = 20, y_1 = 5$.

Step 2: Find the top $k = 3$ nearest neighbors.

- The distances are: 5, 2, 2, 8.
- The smallest three distances correspond to data points $\boldsymbol{x}_2$ (dist 2), $\boldsymbol{x}_3$ (dist 2), and $\boldsymbol{x}_1$ (dist 5).
- Their corresponding outputs are $y_2 = 10, y_3 = 20, y_1 = 5$.

Step 3: Aggregate the outputs (mean for regression).

- We calculate the average of the neighbors' output values.

$$y = \frac{10 + 20 + 5}{3} = \frac{35}{3} \approx 11.67$$

The final output is $y \approx 11.67$.

Exercise

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 the previous example, $\boldsymbol{x}_{new} = \begin{bmatrix}2\\1\end{bmatrix}$.

Answer:

- Squared distances are: 1, 4, 10, 8, 5.
- The two nearest neighbors are $(\begin{bmatrix}1\\1\end{bmatrix}, 1)$ and $(\begin{bmatrix}2\\3\end{bmatrix}, 4)$.
- Their outputs are $\{1, 4\}$.
- The mean is $y = \frac{1+4}{2} = 2.5$.

Remark

Models like k-NN, which store the training data itself or have a non-fixed number of parameters, are often called **non-parametric** models.

Remark

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."

Remark

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."

Scheme of Training and Inference

4. Scheme of Training and Inference

Viewing models as parametric functions, $(f_{\boldsymbol{\theta}})_{\boldsymbol{\theta}\in\Theta}$, separates the problem-solving process into two distinct phases.

- **Training:** The process of finding the "optimal" parameters $\boldsymbol{\theta}^* \in \Theta$ that can best solve the task, using given **training data** $\mathcal{D}$. This is often called **fitting**.

$$\boldsymbol{\theta}^* = \mathfrak{A}((f_{\boldsymbol{\theta}}), \mathcal{D})$$

4. Scheme of Training and Inference

Viewing models as parametric functions, $(f_{\boldsymbol{\theta}})_{\boldsymbol{\theta}\in\Theta}$, separates the problem-solving process into two distinct phases.

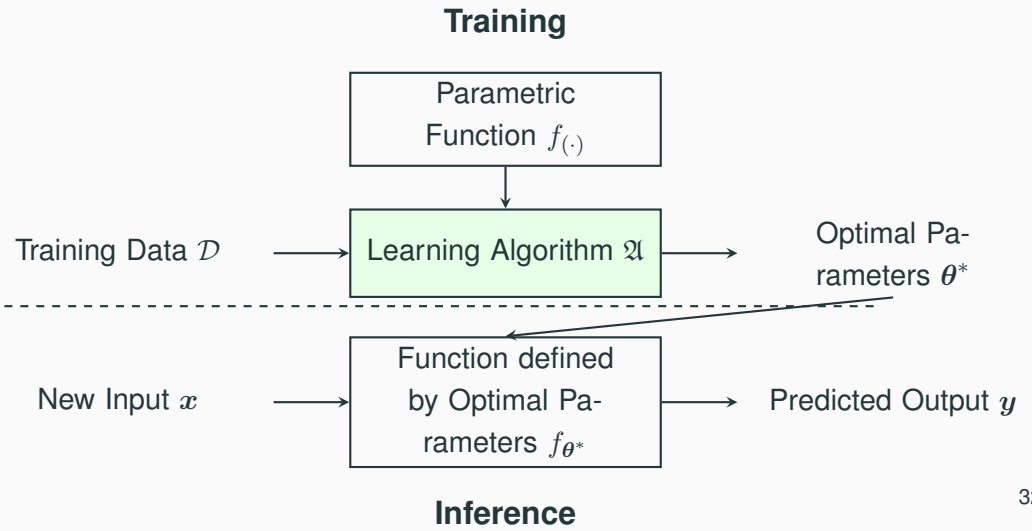
- **Training:** The process of finding the "optimal" parameters $\boldsymbol{\theta}^* \in \Theta$ that can best solve the task, using given **training data** $\mathcal{D}$. This is often called **fitting**.

$$\boldsymbol{\theta}^* = \mathfrak{A}((f_{\boldsymbol{\theta}}), \mathcal{D})$$

- **Inference:** The process of fixing the parameters $\boldsymbol{\theta}^*$ and using the specific function $f_{\boldsymbol{\theta}^*}$ to calculate the output $\boldsymbol{y}$ for a new, unknown input $\boldsymbol{x}$. This is also called **prediction**.

$$\boldsymbol{y} = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$$

4. Scheme of Training and Inference



Remark (Why Emphasize the Difference Between Training and Inference?)

The distinction is essential for understanding practical issues when using neural network models, such as publication and licensing.

Remark (Why Emphasize the Difference Between Training and Inference?)

The distinction is essential for understanding practical issues when using neural network models, such as publication and licensing.

- For practical neural networks, the parameters for inference ($\boldsymbol{\theta}^*$) are often public, while the training algorithm ($\mathfrak{A}$) and data ($\mathcal{D}$) are not.

Remark (Why Emphasize the Difference Between Training and Inference?)

The distinction is essential for understanding practical issues when using neural network models, such as publication and licensing.

- For practical neural networks, the parameters for inference ($\boldsymbol{\theta}^*$) are often public, while the training algorithm ($\mathfrak{A}$) and data ($\mathcal{D}$) are not.
- The license for the inference part and the training part may not be the same.

Remark (Why Emphasize the Difference Between Training and Inference?)

The distinction is essential for understanding practical issues when using neural network models, such as publication and licensing.

- For practical neural networks, the parameters for inference ($\boldsymbol{\theta}^*$) are often public, while the training algorithm ($\mathfrak{A}$) and data ($\mathcal{D}$) are not.
- The license for the inference part and the training part may not be the same.
- Handling neural network models without understanding their licenses carries legal risks.

Specific Examples of Learning Algorithms

Inference is simply the computation $\boldsymbol{y} = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$. All the calculation examples we have seen so far correspond to this inference phase.

Inference is simply the computation $\boldsymbol{y} = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$. All the calculation examples we have seen so far correspond to this inference phase.

The training phase, finding the "optimal" parameters $\boldsymbol{\theta}^*$ from data $\mathcal{D}$, differs greatly depending on the type of parametric function.

Inference is simply the computation $\boldsymbol{y} = f_{\boldsymbol{\theta}^*}(\boldsymbol{x})$. All the calculation examples we have seen so far correspond to this inference phase.

The training phase, finding the "optimal" parameters $\boldsymbol{\theta}^*$ from data $\mathcal{D}$, differs greatly depending on the type of parametric function.

The goal here is not to memorize algorithms, but to understand that training is the process of determining parameters, and it's different from inference.

5.1 Example 1: Training of k-NN

The learning algorithm for k-NN is one of the simplest imaginable.

5.1 Example 1: Training of 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 $\boldsymbol{\theta}$.

5.1 Example 1: Training of 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 $\boldsymbol{\theta}$.

Training is completed just by "memorizing the data."

5.1 Example 1: Training of 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 $\boldsymbol{\theta}$.

Training is completed just by "memorizing the data."

Models like this, where most of the computation is deferred until inference time, are sometimes called **lazy learning** models.

5.2 Example 2: Training of Linear Regression

For models with fixed-length numerical parameters like linear regression, training is a more active process.

5.2 Example 2: Training of Linear Regression

For models with fixed-length numerical parameters like linear regression, training is a more active process.

Training is often formulated as an **optimization problem**: "minimizing a function that measures the 'badness' of the model's predictions on the training data."

5.2 Example 2: Training of Linear Regression

For models with fixed-length numerical parameters like linear regression, training is a more active process.

Training is often formulated as an **optimization problem**: "minimizing a function that measures the 'badness' of the model's predictions on the training data."

Definition (Loss Function)

A function $L(\boldsymbol{\theta})$ that takes parameters $\boldsymbol{\theta}$ and outputs a non-negative real value representing how "bad" the model is for the data is called a **cost function**, **loss function**, or **objective function**. The goal is to find:

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

5.2.1 Least Squares Method

For linear regression, a widely used loss function is the **Sum of Squared Residuals (SSR)**.

5.2.1 Least Squares Method

For linear regression, a widely used loss function is the **Sum of Squared Residuals (SSR)**.

The method of minimizing this loss is called the **least squares method**.

5.2.1 Least Squares Method

For linear regression, a widely used loss function is the **Sum of Squared Residuals (SSR)**.

The method of minimizing this loss is called the **least squares method**.

Definition (Sum of Squared Residuals)

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

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

5.2.2 Least Squares Solution

The least squares solution can be found analytically using linear algebra.

5.2.2 Least Squares Solution

The least squares solution can be found analytically using linear algebra.

Step 1: Augment vectors and matrices.

- Append a 1 to each input vector: $\tilde{\boldsymbol{x}}_i \coloneqq \begin{bmatrix} \boldsymbol{x}_i \\ 1 \end{bmatrix}$.
- Combine weights and bias: $\tilde{\boldsymbol{w}} \coloneqq \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}$.
- Now, the prediction is a simple dot product: $f(\boldsymbol{x}_i) = \tilde{\boldsymbol{w}}^\top \tilde{\boldsymbol{x}}_i$.

Step 2: Represent data in matrix form.

- Design Matrix $\boldsymbol{X}$: Each row is an augmented input vector $\tilde{\boldsymbol{x}}_i^\top$.
- Target Vector $\boldsymbol{y}$: A column vector of all true outputs y_i.

$$\boldsymbol{X} \coloneqq \begin{bmatrix} \tilde{\boldsymbol{x}}_1^\top \\ \vdots \\ \tilde{\boldsymbol{x}}_N^\top \end{bmatrix}, \quad \boldsymbol{y} \coloneqq \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

5.2.2 Least Squares Solution

With the matrix formulation, the vector of all predictions $\hat{\boldsymbol{y}}$ is simply $\hat{\boldsymbol{y}} = \boldsymbol{X}\tilde{\boldsymbol{w}}$.

5.2.2 Least Squares Solution

With the matrix formulation, the vector of all predictions $\hat{\boldsymbol{y}}$ is simply $\hat{\boldsymbol{y}} = \boldsymbol{X}\tilde{\boldsymbol{w}}$.

The loss function SSR becomes the squared L2-norm of the difference vector:

$$L_{\mathrm{SSR}}(\tilde{\boldsymbol{w}}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \|\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}}\|_2^2$$

5.2.2 Least Squares Solution

With the matrix formulation, the vector of all predictions $\hat{\boldsymbol{y}}$ is simply $\hat{\boldsymbol{y}} = \boldsymbol{X}\tilde{\boldsymbol{w}}$.

The loss function SSR becomes the squared L2-norm of the difference vector:

$$L_{\text{SSR}}(\tilde{\boldsymbol{w}}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \|\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}}\|_2^2$$

The minimizer of this loss function is the solution to the following equation:

Definition (Normal Equation)

The following equation for $\tilde{\boldsymbol{w}}$ is called the **normal equation**.

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

5.2.2 Least Squares Solution

Theorem (Least Squares Solution)

A vector $\tilde{\boldsymbol{w}}$ minimizes the Sum of Squared Residuals (SSR) if and only if it is a solution to the normal equation.

5.2.2 Least Squares Solution

Theorem (Least Squares Solution)

A vector $\tilde{\boldsymbol{w}}$ minimizes the Sum of Squared Residuals (SSR) if and only if it is a solution to the normal equation.

In particular, if the matrix $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible, the normal equation has a unique solution given by:

$$\tilde{\boldsymbol{w}}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{2}$$

5.2.2 Least Squares Solution

Theorem (Least Squares Solution)

A vector $\tilde{\boldsymbol{w}}$ minimizes the Sum of Squared Residuals (SSR) if and only if it is a solution to the normal equation.

In particular, if the matrix $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible, the normal equation has a unique solution given by:

$$\tilde{\boldsymbol{w}}^* = (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}. \tag{2}$$

Geometric Interpretation: The least squares method finds the **orthogonal projection** of the target vector $\boldsymbol{y}$ onto the subspace spanned by the columns of the design matrix $\boldsymbol{X}$. The normal equation expresses the condition that the error vector $(\boldsymbol{y} - \boldsymbol{X}\tilde{\boldsymbol{w}})$ must be orthogonal to that subspace.

Example

Let's find the optimal parameters for the following training data $\mathcal{D}_A$:

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

5.2.2 Least Squares Solution

Step 1: Construct the design matrix X and target vector y.

- The input vectors are $\begin{bmatrix} 3 \\ 4 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.
- The augmented input vectors (add a 1) are $\begin{bmatrix} 3 \\ 4 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$.
- The outputs are $-4.5, 1.5, -1.5$.

5.2.2 Least Squares Solution

Step 1: Construct the design matrix X and target vector y.

- The input vectors are $\begin{bmatrix}3\\4\end{bmatrix}, \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}$.
- The augmented input vectors (add a 1) are $\begin{bmatrix}3\\4\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}$.
- The outputs are $-4.5, 1.5, -1.5$.

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

Step 2: Calculate $\boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{X}^\top \boldsymbol{y}$.

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

Step 2: Calculate $\boldsymbol{X}^\top \boldsymbol{X}$ and $\boldsymbol{X}^\top \boldsymbol{y}$. (Continued)

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

5.2.2 Least Squares Solution

Step 3: Solve the normal equation $(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}$.

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

5.2.2 Least Squares Solution

Step 3: Solve the normal equation $(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}} = \boldsymbol{X}^\top \boldsymbol{y}$.

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

Solving this system of linear equations (e.g., by finding the inverse of the matrix on the left), we get the solution:

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

These are exactly the parameters we used in the first inference example!

5.3 Diversity of Learning Algorithms and Neural Networks

As we have seen, learning algorithms can be completely different:

- **k-NN:** Just store the data ("lazy learning").
- **Linear Regression:** Solve a system of linear equations (the normal equation).
- **Decision Trees:** Use a greedy algorithm to find the best splits (not detailed here).

5.3 Diversity of Learning Algorithms and Neural Networks

As we have seen, learning algorithms can be completely different:

- **k-NN:** Just store the data ("lazy learning").
- **Linear Regression:** Solve a system of linear equations (the normal equation).
- **Decision Trees:** Use a greedy algorithm to find the best splits (not detailed here).

In contrast, one of the greatest strengths of **neural networks** is the **uniformity of their learning algorithm**.

5.3 Diversity of Learning Algorithms and Neural Networks

As we have seen, learning algorithms can be completely different:

- **k-NN:** Just store the data ("lazy learning").
- **Linear Regression:** Solve a system of linear equations (the normal equation).
- **Decision Trees:** Use a greedy algorithm to find the best splits (not detailed here).

In contrast, one of the greatest strengths of **neural networks** is the **uniformity of their learning algorithm**.

Most neural networks can be trained under a common framework:

1. Calculate the gradient of the loss function using **backpropagation**.
2. Minimize the loss using **stochastic gradient descent (SGD)** or its variants.

Summary and Future Outlook

6.1 Today's Summary

- A task can be reduced to a parameter determination problem for a **parametric function**.

6.1 Today's Summary

- A task can be reduced to a parameter determination problem for a **parametric function**.
- Various AI models like linear regression, decision trees, and k-NN can be understood uniformly as parametric functions.

6.1 Today's Summary

- A task can be reduced to a parameter determination problem for a **parametric function**.
- Various AI models like linear regression, decision trees, and k-NN can be understood uniformly as parametric functions.
- Machine learning is separated into a **training** phase (finding optimal parameters from data) and an **inference** phase (using those parameters to make predictions).

6.2 Next Time

This time, we viewed various machine learning models within the unified framework of parametric functions. This is an important foundation for understanding neural networks.

6.2 Next Time

This time, we viewed various machine learning models within the unified framework of parametric functions. This is an important foundation for understanding neural networks.

Next time, we will strictly define what a **neural network** is—a particularly expressive type of parametric function. We will use the mathematical tool of a **Directed Acyclic Graph (DAG)** to do this.

6.2 Next Time

This time, we viewed various machine learning models within the unified framework of parametric functions. This is an important foundation for understanding neural networks.

Next time, we will strictly define what a **neural network** is—a particularly expressive type of parametric function. We will use the mathematical tool of a **Directed Acyclic Graph (DAG)** to do this.

Then, we will learn about the distinction between "architecture" and "checkpoints," concepts that are extremely important in the practical application of modern AI.

Appendix: Differentiation by a Vector i

The gradient of a scalar function $f(\boldsymbol{x})$ with respect to a vector $\boldsymbol{x}$ is defined as a vector of partial derivatives.

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} := \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}^\top$$

1. **Derivative of a Linear Term:**

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

2. **Derivative of a Quadratic Form:** For a symmetric matrix $\boldsymbol{A}$:

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

Appendix: Differentiation by a Vector ii

Recall the loss function:

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

Differentiating each term w.r.t. $\tilde{\boldsymbol{w}}$:

- 1st term ($\boldsymbol{y}^\top \boldsymbol{y}$): Constant, derivative is $\mathbf{0}$.
- 2nd term ($-2\tilde{\boldsymbol{w}}^\top (\boldsymbol{X}^\top \boldsymbol{y})$): Linear term, derivative is $-2\boldsymbol{X}^\top \boldsymbol{y}$.
- 3rd term ($\tilde{\boldsymbol{w}}^\top (\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}}$): Quadratic form with a symmetric matrix ($\boldsymbol{X}^\top \boldsymbol{X}$), derivative is $2(\boldsymbol{X}^\top \boldsymbol{X})\tilde{\boldsymbol{w}}$.

Summing them up gives the gradient:

$$\nabla_{\tilde{\boldsymbol{w}}} L = -2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\tilde{\boldsymbol{w}}$$