IT Talent Training Course

Aug. 2025.

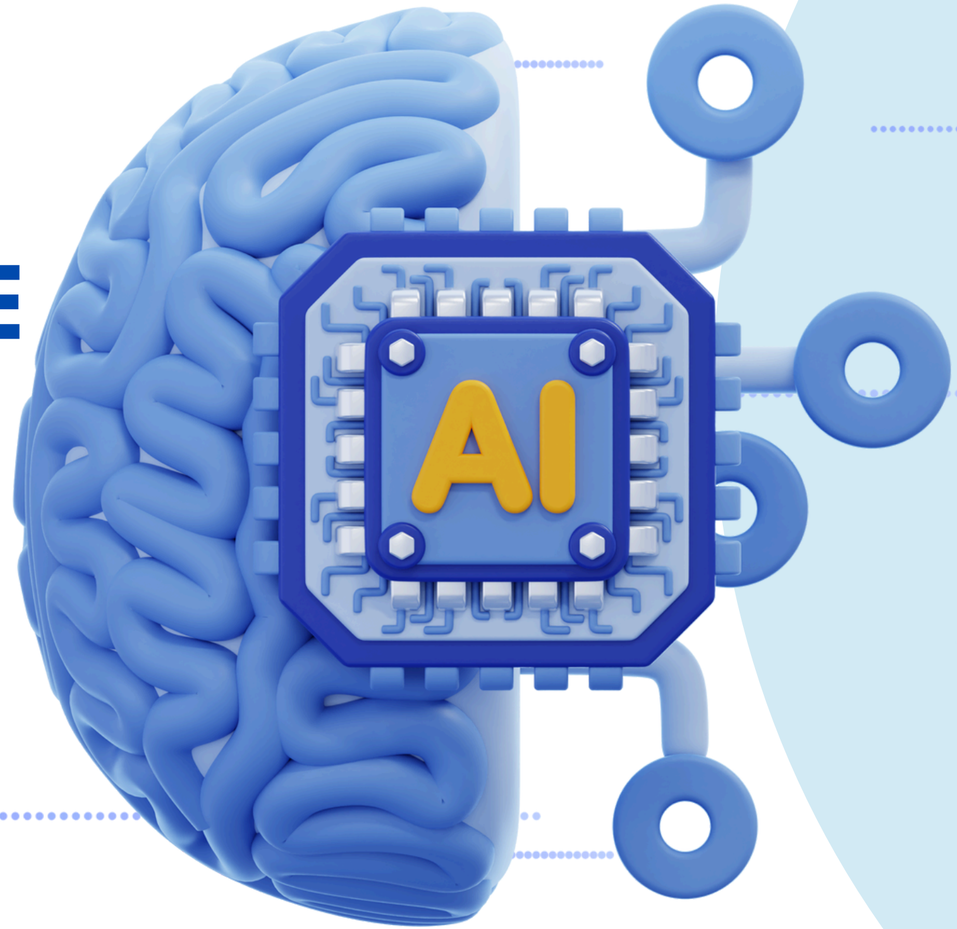
A.I. PROGRAMMING WITH PYTORCH

Instructor :
Daesung Kim

1st Day - Part 01

INDEX

01
ARTIFICIAL INTELLIGENCE OVERVIEW



Keywords

Artificial Intelligence, Machine Learning, Deep Learning, Data, Education

CONCEPTS AND INTERRELATIONS OF AI, ML, DL - 1

Keywords

Artificial Intelligence, Machine Learning, Deep Learning, Data, Education

1. Artificial Intelligence

- **Definition:** Artificial Intelligence (AI) encompasses a broad concept that refers to the technologies and fields enabling computers to perform cognitive functions similar to those executed by human intelligence, including learning, problem-solving, pattern recognition, decision-making, and language comprehension.

- **Objective:** The ultimate aim is to create a system that thinks, learns, and acts like a human.

- **History:** Since its inception in the 1950s, AI has experienced several periods of stagnation, often referred to as 'AI winters.' However, due to advancements in big data, computing power (such as GPUs), and algorithms, it has achieved remarkable progress in recent years.

- **Examples:** autonomous vehicles, voice assistants (Siri, Google Assistant), recommendation systems (Netflix, YouTube), medical diagnostic support systems, etc.

CONCEPTS AND INTERRELATIONS OF AI, ML, DL - 2

2. Machine Learning

- **Definition:** Machine learning is a branch of artificial intelligence that focuses on developing computer algorithms capable of improving performance through learning from data without explicit programming.

- **Core Idea:** The objective is to enable computers to analyze data, identify patterns, learn from them, and subsequently predict future outcomes or make decisions. Unlike traditional programming methods (such as If-Else statements), this approach allows data to 'inform' the rules rather than requiring developers to explicitly write every rule.

- **Operation Method:**
  - Data Collection: We gather the data necessary for problem-solving.
  - Model Training: Utilizing the collected data, a specific algorithm (model) is trained. The model identifies relationships and patterns within the data.
  - Prediction/Decision: The trained model is applied to new data to make predictions or decisions.

- **Examples:** spam email filtering, credit card fraud detection, housing price prediction, customer churn prediction, etc.

CONCEPTS AND INTERRELATIONS OF AI, ML, DL - 3

3. Deep Learning

- **Definition:** Deep learning is a subfield of machine learning that uses 'artificial neural networks' that mimic the neural network structure of the human brain to learn complex patterns from large amounts of data. 'Deep' refers to a structure that stacks multiple layers of neural networks (layers).

- **Characteristics:**
  - **Multilayer neural networks:** Automatically extract hierarchical features from data using multiple hidden layers. Early layers learn low-level features (e.g. lines and outlines in images), and later layers learn high-level features (e.g. faces and objects in images).
  - **Big data and GPUs:** It requires a large amount of data, and when high-performance computing resources such as GPUs (graphics processing units) are utilized to process complex calculations in parallel, it shows significant performance improvements.

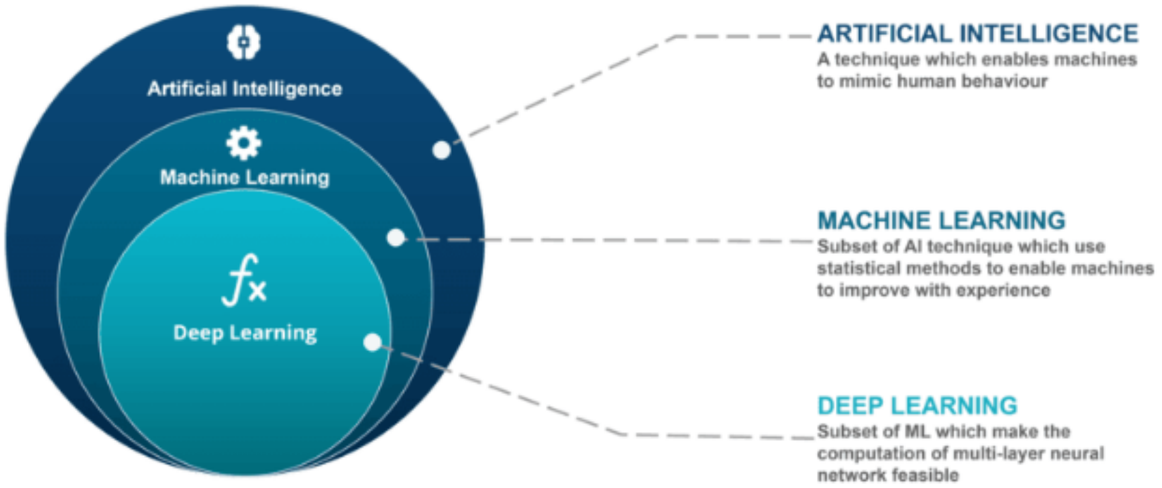
- **Innovation:** Deep learning has brought about innovative advances that surpass the performance of existing machine learning techniques, especially in the fields of image recognition, speech recognition, and natural language processing.

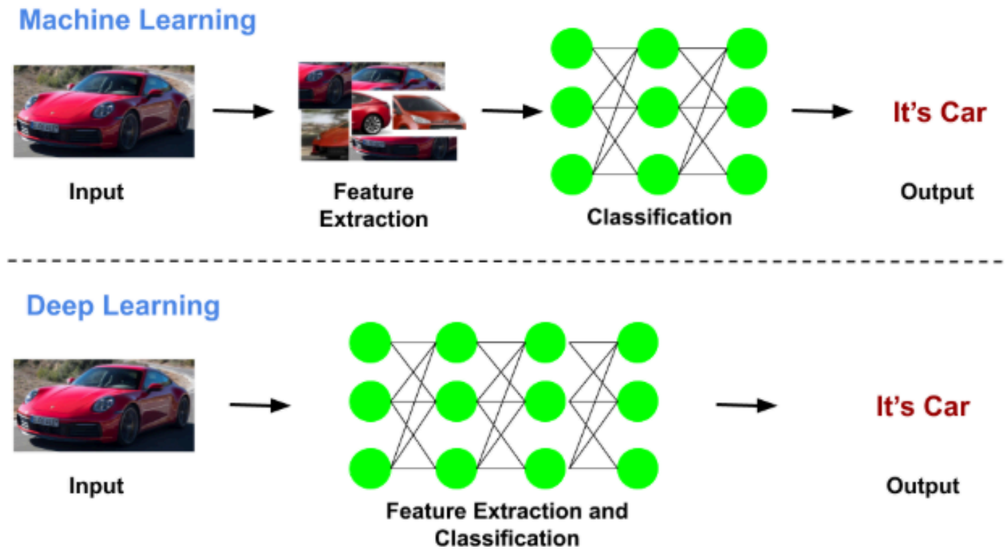
- **Examples:** face recognition, speech recognition (speech-to-text), machine translation, Go-playing AI (AlphaGo), large-scale language models (LLMs) (Gemini, etc.).

CONCEPTS AND INTERRELATIONS OF AI, ML, DL - 4

4. Comparison of AI, ML, and DL

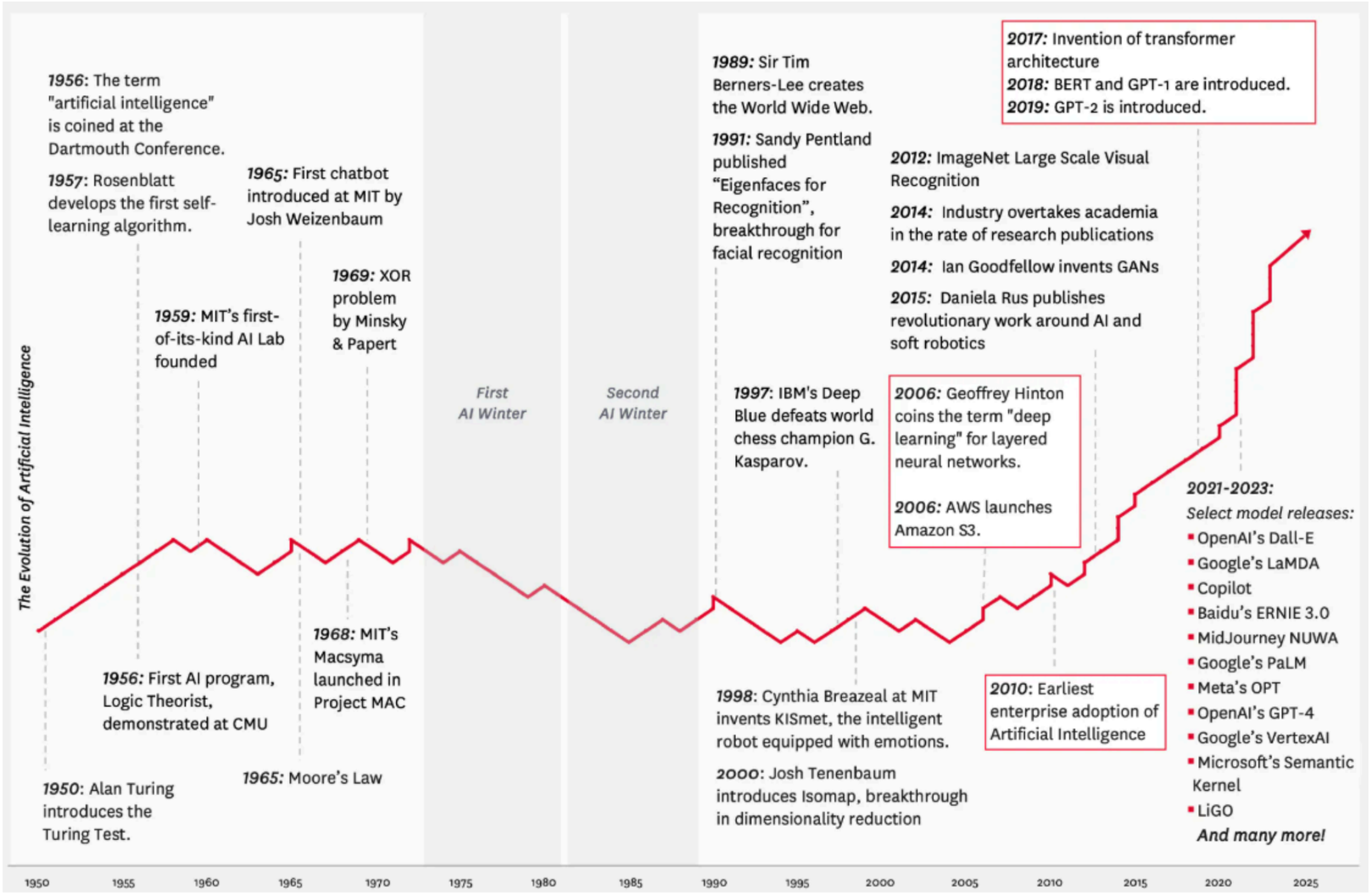
- **AI** is the largest concept, encompassing all technologies that mimic human intelligence.
- **ML** refers to methods of learning through data within AI.
- **DL** refers to learning methods within ML, specifically artificial neural networks, and among them, 'deep' neural network structures.





HISTORY OF ARTIFICIAL INTELLIGENCE

Time Line



THE ENGINE OF THE AI REVOLUTION: PROCESSOR ADVANCES -1

1. CPU(Central Processing Unit)

- The 'brain' of the computer, designed to process complex and sequential commands quickly.
- All early AI research was based on CPUs, but CPUs with a small number of cores had limitations in processing complex parallel operations of AI models.

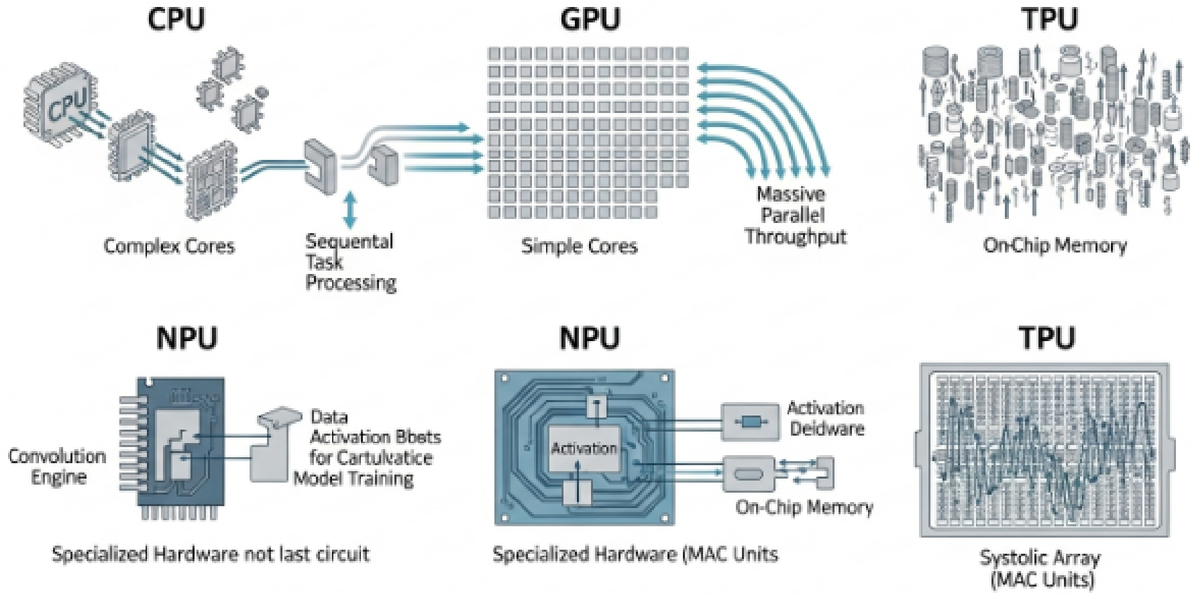
2. GPGPU(General Purpose Graphics Processing Unit)

- Originally born for 3D graphics processing, it specializes in processing simple calculations simultaneously and in parallel using thousands of small cores.
- The core of deep learning, matrix multiplication, is exactly this kind of parallel calculation, and the advent of GPUs caused a "big bang" in deep learning. As GPUs can complete learning that would take weeks with CPUs in just a few days, researchers can experiment with larger and more complex models.
- (Important) In particular, thanks to the GPU support provided for free by Google Colab, which is used in this course, learners can efficiently experience the complex calculations of deep learning models without expensive hardware.

THE ENGINE OF THE AI REVOLUTION: PROCESSOR ADVANCES -2

3. AI Processor

- It is a processor more specialized for deep learning operations.
- **NPU (Neural Processing Unit):** It is designed to accelerate AI operations by imitating human neural networks, and is especially efficient in low-power environments such as mobile devices.
- **TPU (Tensor Processing Unit):** It is an AI accelerator developed by Google, and shows excellent performance and power efficiency in learning and inference of large-scale AI models.

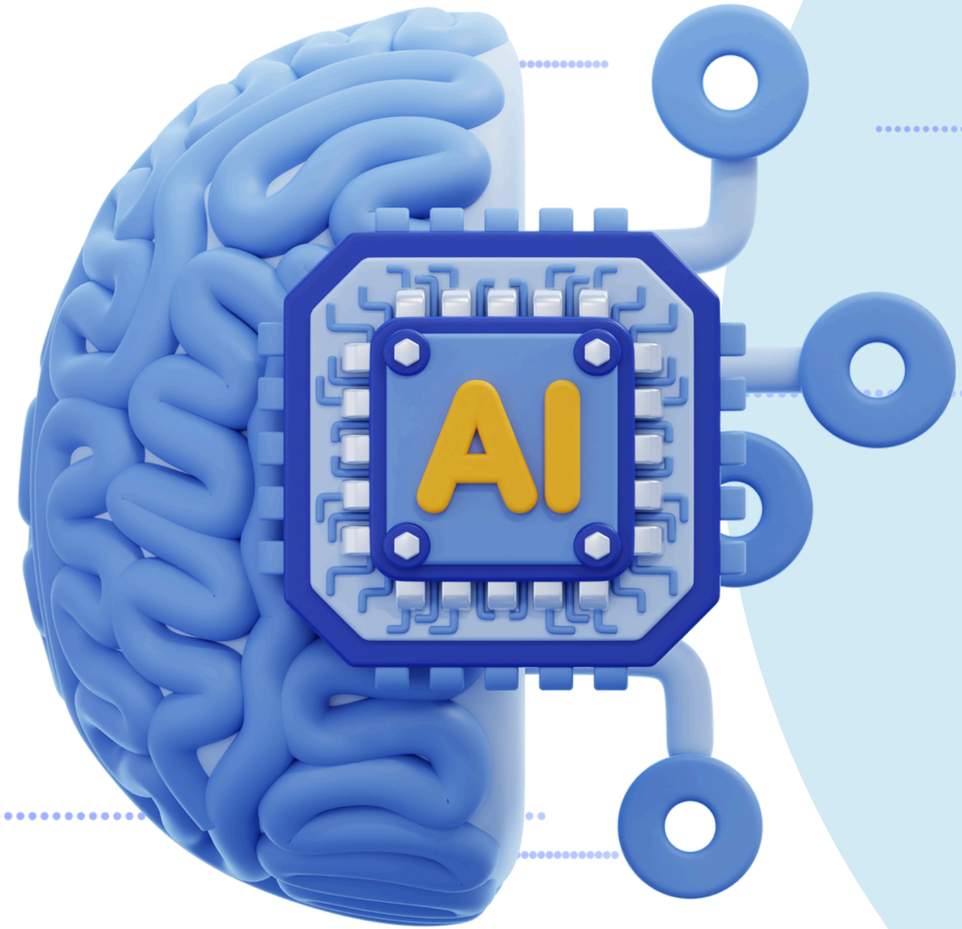


SUMMARY

Summary

AI is the biggest dream of imitating human intelligence, and within it, there is machine learning that learns through data, and deep learning that uses artificial neural networks within machine learning (AI ⊃ ML ⊃ DL). The history of AI has developed through repeated expectations and disappointments (AI winter). In particular, the explosive growth of deep learning was possible thanks to the parallel processing capabilities of GPUs, the increase in big data, and the improvement of algorithms, and currently, more specialized AI processors such as NPUs and TPUs are leading the development.

02
HOW DOES ML LEARN?



Keywords

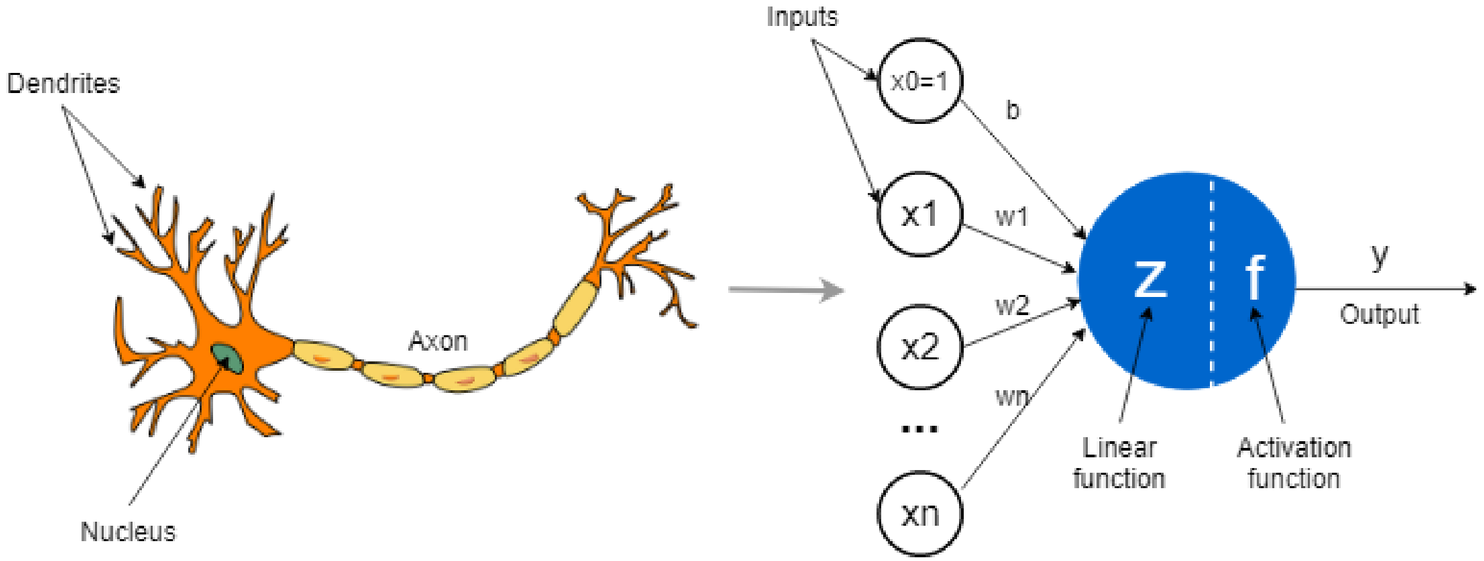
ANN, Supervised learning, Unsupervised learning, Reinforcement learning, Linear regression, Logistic regression, Label (correct answer), Reward, Sigmoid function, Binary cross entropy

ANN(ARTIFICIAL NEURAL NETWORK) - 1

ANN is a computational model inspired by the neural network structure of our brain, and can learn patterns in complex data and perform tasks such as prediction or classification. It is an important concept that forms the basis of advanced deep learning models such as CNN and RNN that we will learn later.

1. Biological neurons to artificial neurons

Biological Neuron: Neurons in the brain receive electrochemical signals from other neurons through **dendrites**. When these signals gather in the **cell body** and exceed a certain threshold, they transmit signals to other neurons through the **axon**.



ANN(ARTIFICIAL NEURAL NETWORK) - 2

Artificial Neuron (Perceptron):
A mathematically very simple model of this structure is the artificial neuron, or **perceptron**.

- **Input:**
  - This corresponds to the signal received from other neurons. Each **input value(x_1, x_2, ...)** comes in with its own importance, or **weight (w_1,w_2, ...)**. A larger weight means that the corresponding input signal has a greater influence on the result.

- **Computation of Neural Networks:**
  - Similar to the computational process that occurs in the cell body. Add up all the input values and their corresponding weights ($w_1x_1+w_2x_2+....$). **Add a value called Bias (b)** to calculate the final sum. The bias acts as a kind of 'reference point' that determines how easily the neuron will activate.

- **Activation Function:**
  - This acts as a 'threshold' that determines whether the neuron will transmit a signal or not. The calculated **final sum passes through the activation function** to produce the final output value (y).

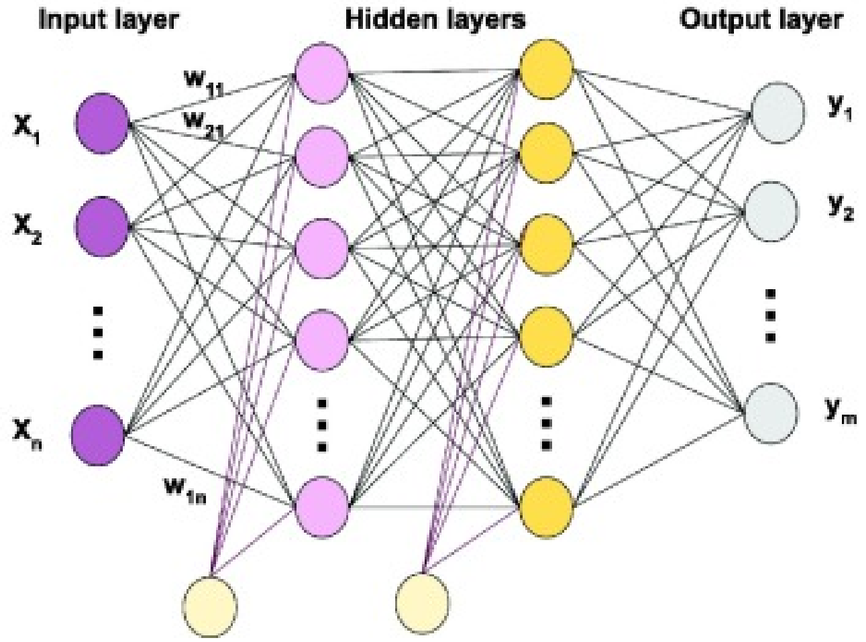
$$z = \sum_{i=1}^{n} (w_i x_i) + b = (w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) + b$$

$$y = f(z)$$

ANN(ARTIFICIAL NEURAL NETWORK) - 3

An artificial neural network usually consists of **three types of layers**.

- **Input Layer:** This is the layer where data first enters. The number of neurons in this layer is equal to the number of features in the data.
  - For example, if you input a handwritten digit image consisting of 784 pixels, the number of neurons in the input layer will be 784. The input layer simply receives data and passes it to the next layer, but does not perform any special calculations.



ANN(ARTIFICIAL NEURAL NETWORK) - 4

- **Hidden Layer:** This refers to all layers between the input layer and the output layer. The hidden layer extracts and combines features contained in the input data to create new features that help the output layer make the final decision. The number of hidden layers and the number of neurons in each layer are important hyperparameters (values that the user must set) that greatly affect the complexity and performance of the model. A neural network with two or more hidden layers is usually called a **deep neural network (DNN), which is the origin of deep learning**.

- **Output Layer:** This is the layer that outputs the final calculation result of the neural network. The number of neurons and the activation function of the output layer are determined depending on the type of problem to be solved.
  - **Regression:** When predicting continuous values, such as predicting house prices, the output layer neuron is usually 1 and the activation function is often not used.
  - **Binary Classification:** When matching one of two things, such as 'dog' or 'cat', the output layer neuron is 1 and the sigmoid is often used as the activation function.
  - **Multi-class Classification:** When matching one of several classes, such as handwritten numbers from 0 to 9, the number of neurons in the output layer is equal to the number of classes (in this case, 10) and the activation function is often used as the softmax.

HOW DO NEURAL NETWORKS LEARN? - 1

The learning process of a neural network largely consists of two stages: **forward propagation** and **backpropagation**. This is similar to the process of a student solving a problem (forward propagation) and comparing it to the correct answer and correcting the wrong parts (backpropagation).

1. Feedforwad propagation

Forward propagation is the process of calculating the predicted value by propagating input data in one direction, starting from the input layer, passing through the hidden layer, and reaching the output layer.

- **Input:** Data to be learned (e.g. handwritten image) is input to the input layer.
- **Computation:** The neurons in each layer multiply their weights and add a bias to the output received from the previous layer, and then pass it through the activation function to the next layer.
- **Final prediction:** When this process is repeated up to the output layer, the final predicted value of the neural network is produced.

Since the weights and biases are initially set to random values, the first predicted value through forward propagation will be very different from the actual correct answer.

HOW DO NEURAL NETWORKS LEARN? - 2

2. Loss Function

We need a metric to quantitatively measure how 'wrong' the model is. This is called a loss function or cost function. The loss function calculates the difference between the **prediction value (haty)** of the neural network and the actual answer (y).

- **Mean Squared Error (MSE):** Mainly used in regression problems.
- **Cross-Entropy Error:** Mainly used in classification problems.

The goal of neural network training is to find the weights (w) and bias (b) that minimize this loss.

HOW DO NEURAL NETWORKS LEARN? - 3

3. Backpropagation and Gradient Descent

How do we update the weights and biases in a way that minimizes the loss? This is where **Gradient Descent and Backpropagation algorithms** come into play.

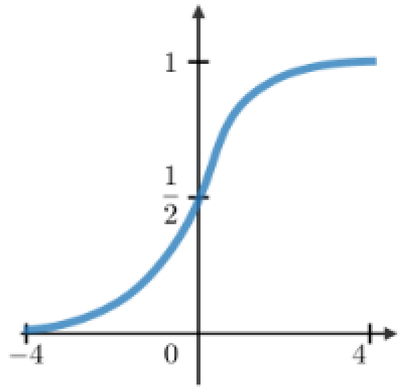
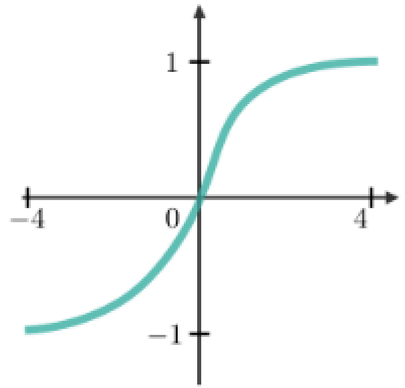
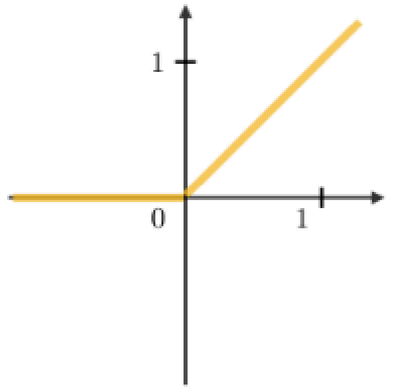
- **Gradient Descent:** Calculate the derivative of the loss function with respect to each weight, or **gradient**. This gradient represents the direction in which the loss increases the steepest. Therefore, we can reduce the loss by moving the weights in the opposite direction of this gradient. This is similar to when you go down a foggy mountain to find the lowest point, you take one step at a time down the steepest slope from your current position.

- **Backpropagation:** An efficient way to calculate the gradient for each of the numerous weights in a complex neural network. As the name suggests, the loss (error) calculated in the output layer is propagated backwards, that is, from the output layer to the hidden layer and toward the input layer, and the weights of each layer are calculated using the chain rule to determine how much they contributed to the final loss. Deep learning frameworks such as PyTorch automatically calculate this complex differentiation process with a single line of code called loss.backward(), so we do not need to implement all the formulas ourselves.

By repeating this **[forward propagation → loss calculation → backpropagation (gradient calculation) → weight update]** process for a large number of data (epochs), the neural network gradually adjusts the weights and biases in the direction of lower loss, and eventually learns to become a model that matches the correct answer well.

ACTIVATION FUNCTION - 1

The activation function plays a very important role in providing **non-linearity** to the neural network. If there is no activation function (or it is linear, like the identity function), no matter how many layers you stack, the entire network will eventually become just a linear transformation. This makes it impossible to properly represent complex real-world data.

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$	$g(z) = \max(0, z)$
		

ACTIVATION FUNCTION - 2

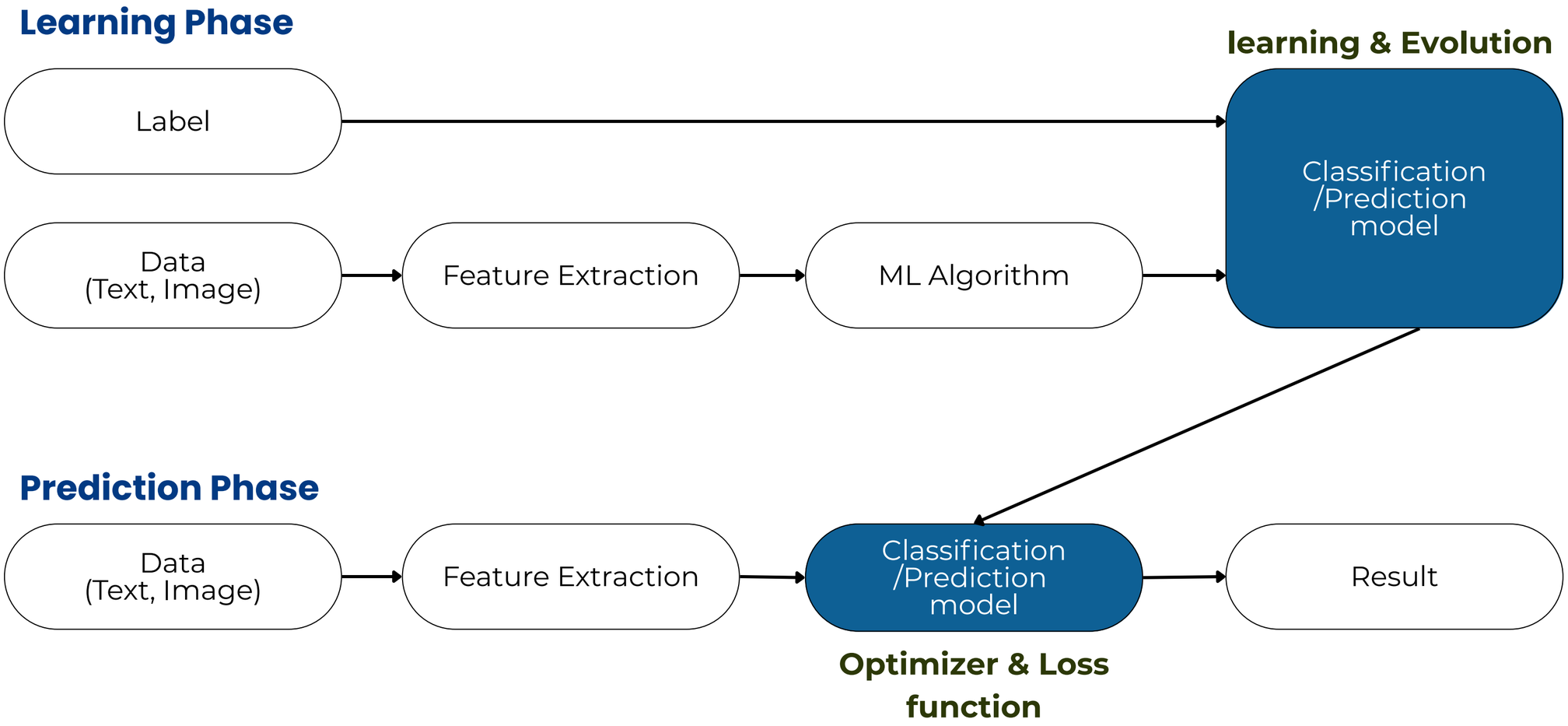
The activation function takes the weighted sum (z) of the neurons as input and transforms it in a specific way to produce the output (y).

- **Sigmoid Function:**
  - It has the characteristic that the output value is always between 0 and 1, so it was used a lot in the past to express probability. However, it has a vanishing gradient problem where the gradient approaches 0 as the input value gets farther from 0, so it is not often used in hidden layers. It is mainly used in the output layer of binary classification problems.

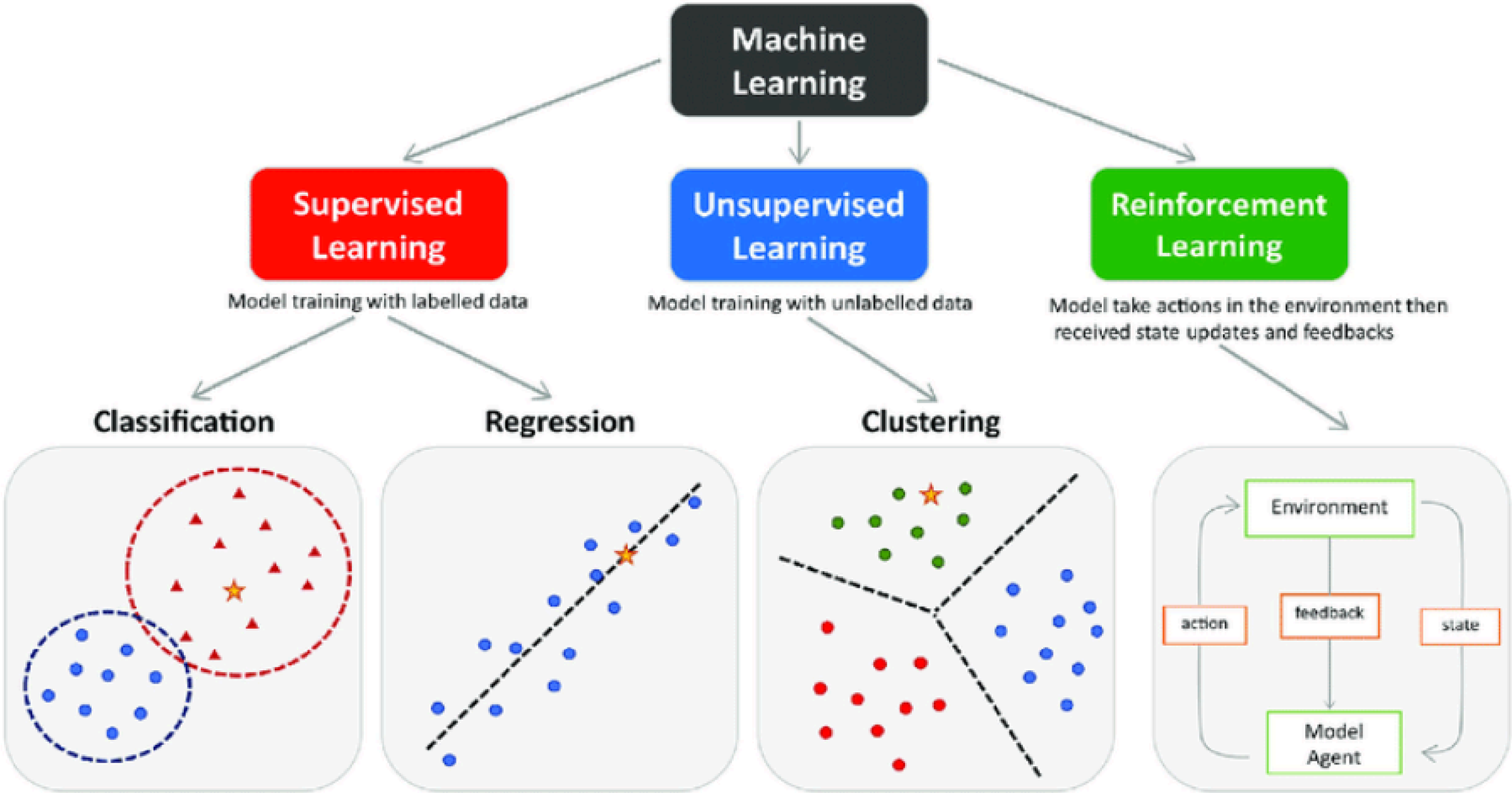
- **Hyperbolic Tangent Function (Tanh):**
  - It is known to have better learning efficiency than sigmoid because the output value is between -1 and 1, and the center of the function is 0. However, the vanishing gradient problem still exists.

- **ReLU (Rectified Linear Unit) function:**
  - This is the most widely used activation function. If the input is greater than 0, it outputs as is, and if it is less than 0, it outputs 0. Its structure is very simple, so the calculation speed is fast, and it has greatly contributed to improving deep learning performance by solving the gradient vanishing problem of sigmoid or Tanh.

MACHINE LEARNING PROCESS



MACHINE LEARNING CLASSIFICATION

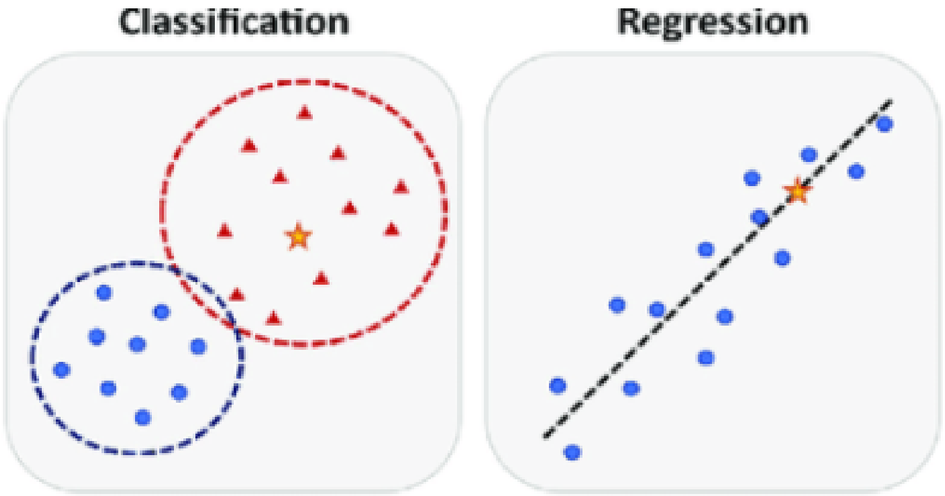


SUPERVISED LEARNING - 1

This is a method of learning with data that has a 'correct answer (label).' By providing input data (problem) and the corresponding correct answer (label) together, the model learns the relationship between the problem and the correct answer.

- **Goal:** When a new problem is given, accurately predict the correct answer

- **Representative examples:**
  - **Regression:** A problem of predicting continuous values. (Example: Predicting 'house price' with house size and location information)
  - **Classification:** A problem of guessing which group the given data belongs to. (Example: Classifying whether an email is 'spam' or 'normal')



SUPERVISED LEARNING - 2

1. Linear Regression

Linear regression is an algorithm that finds the straight line that best represents the relationship between multiple data. It models the linear relationship when one variable (x) affects another variable (y), such as study time and test scores.

- **Hypothesis & Goal:**
  - Linear regression aims to find the equation of the straight line that best represents the data.
    - **H(x)=Wx+b,**
      - where H(x) is the model's prediction.
      - x: Input data (e.g., study time)
      - W: Weight, the slope of the line. It indicates how much influence x has on the outcome.
      - b: Bias, the y-intercept of the line. It indicates the starting point.
  - **Learning:** A machine learning model looks at a lot of data (x, y pairs) and finds the optimal W and b that minimizes the average error between the actual answer (y) and the model's prediction (H(x)).

SUPERVISED LEARNING - 3

- **Model Optimization:** Finding the Optimal Straight Line with Gradient Descent
  - When a model learns, it is a process of **minimizing the loss or cost.** In other words, it creates an indicator that indicates how wrong the model's prediction is, and finds W and b that minimize this value.
  - **Loss Function - Mean Squared Error (MSE):**
    - It measures "how wrong the model is." The most representative loss function is **Mean Squared Error (MSE)**.
    - This is the difference between the actual answer (y_i) and the model's prediction value ($H(x_i)$) for each data, squared, and then added up and averaged.

$$Cost(W,b) = \frac{1}{n}\sum_{i=1}^{n}(H(x_i)-y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(Wx_i+b-y_i)^2$$

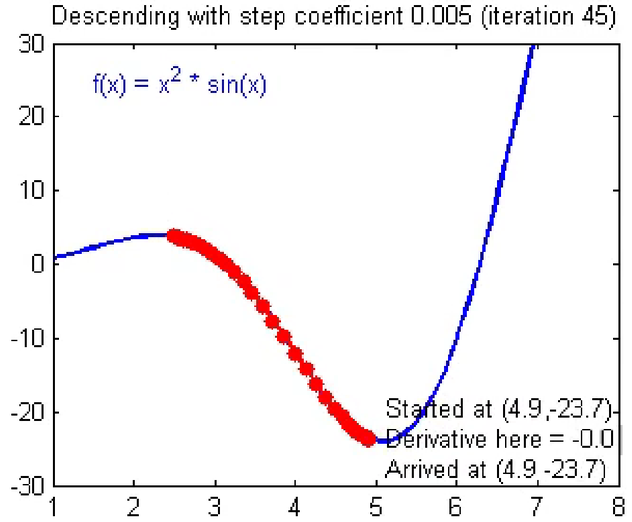
SUPERVISED LEARNING - 4

- **Gradient Descent:** A method to "move in the direction of reducing loss".
  - It is an iterative optimization method that finds the direction (the opposite direction of the gradient) that can reduce the loss function as quickly and as low as possible, and moves W and b in that direction little by little by the step size called the learning rate.
  - Update Rule:
    - Weight (W) update rules:

$$W \leftarrow W - \alpha \times \frac{\partial}{\partial W} Cost(W, b)$$

    - Bias (b) Update Rule:

$$b \leftarrow b - \alpha \times \frac{\partial}{\partial b} Cost(W, b)$$



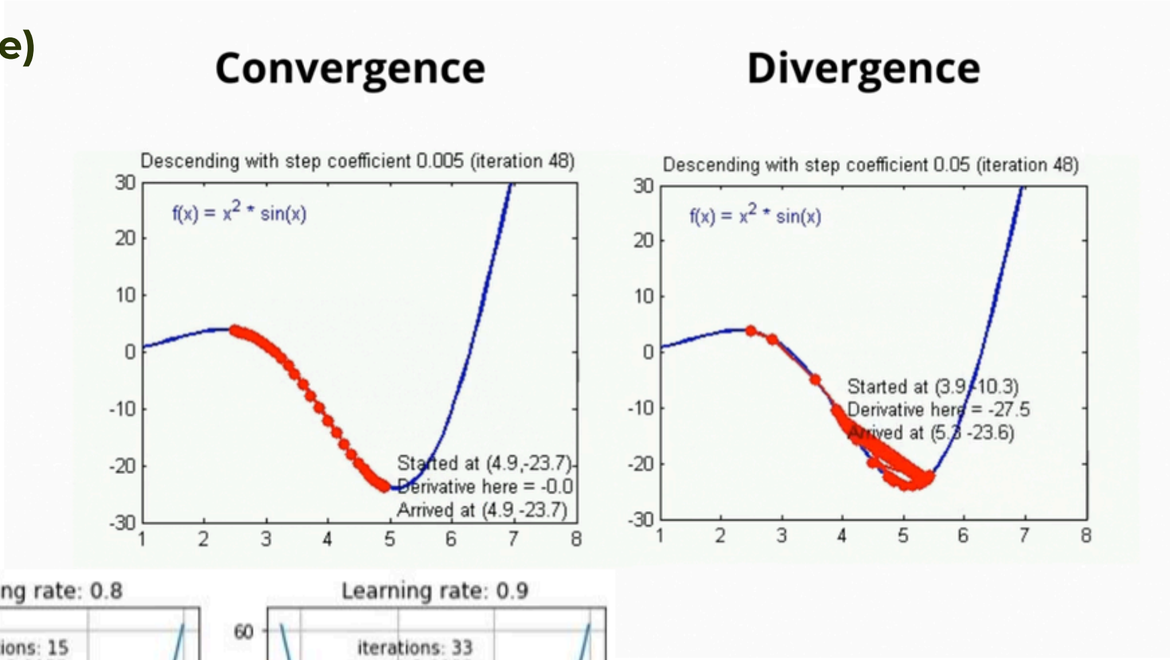
$$\alpha \times \frac{\partial}{\partial W} Cost(W, b)$$

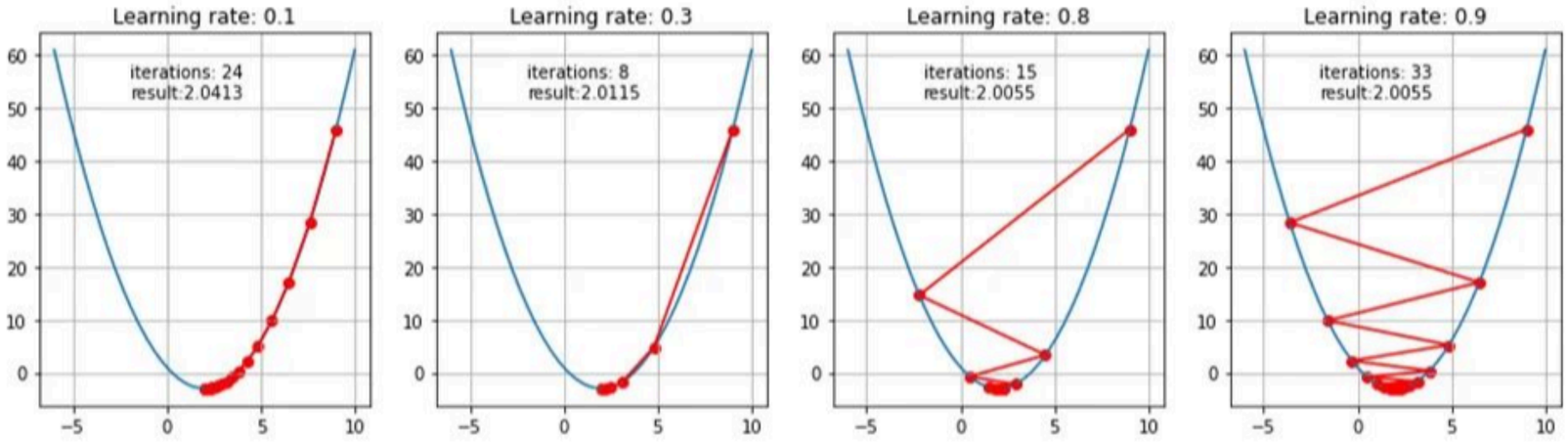
Learning rate

The value (slope) of the partial differentiation of the loss function with respect to the weight W

SUPERVISED LEARNING - 5

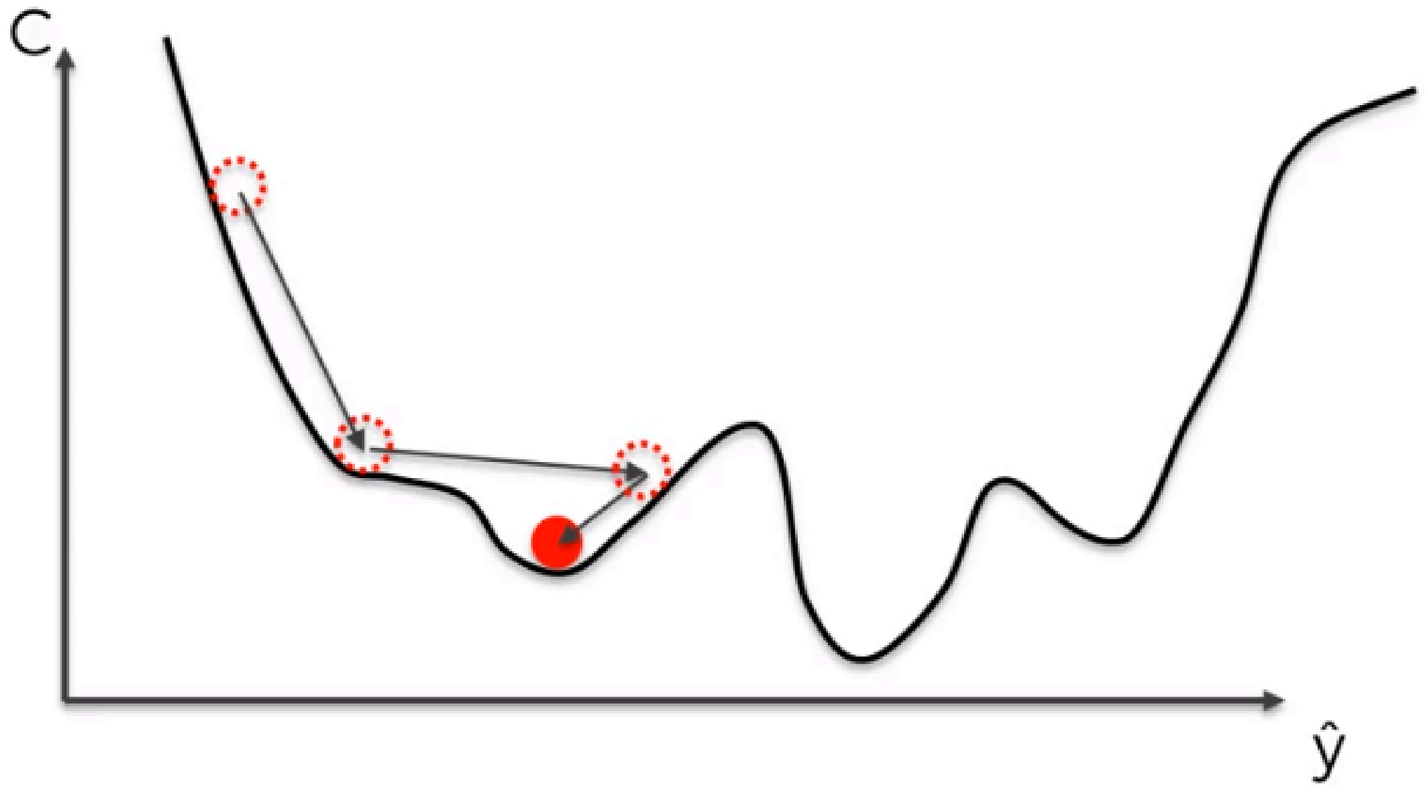
- **Gradient Descent Problem-1 (Learning Rate)**





SUPERVISED LEARNING - 6

- **Gradient Descent Problem-2 (Local minima)**



SUPERVISED LEARNING - 7

2. Linear Regression Coding Practice

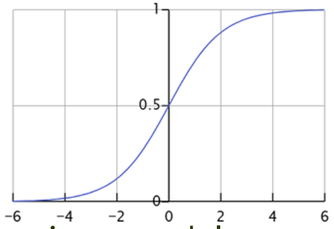
[Code] Day01 - 01_Linear_Regression.py github

SUPERVISED LEARNING - 8

3. Logistic Regression

Logistic regression is easy to confuse because of its name, 'regression', but it is actually an algorithm for classification. It is mainly used for problems where you have to decide between two options (e.g. 'pass' or 'fail', 'spam' or 'normal').

- **Core idea: Sigmoid function** $H(x) = \text{sigmoid}(z) = \frac{1}{1+e^{-z}} \quad (\text{where } z = Wx + b)$



  - The result of linear regression (Wx+b) can have any real value, but the classification problem must be expressed as a probability value between 0 and 1.

  - The sigmoid function takes any real input (z) and always outputs a value in the form of an S-shaped curve between 0 and 1, converting the result of linear regression into a probability.

  - The predicted value H(x) that passes through this function means "given the input x, the probability that the correct answer is 1." Generally, if this probability is greater than 0.5, it is classified as 1 (pass), and if it is less than 0.5, it is classified as 0 (fail).

SUPERVISED LEARNING - 9

- **Loss Function: Binary Cross-Entropy (BCE)**
  - For classification problems, the Binary Cross-Entropy (BCE) loss function is more appropriate than MSE. BCE effectively measures the difference between the probability predicted by the model and the actual correct answer (either 0 or 1).
  - **How it works:** If the model gets the correct answer right (e.g., the predicted probability is close to 1 when the correct answer is 1), the loss approaches 0, and if the model predicts the wrong answer with high confidence (e.g., the predicted probability is close to 0 when the correct answer is 1), the loss increases to infinity, imposing a reasonable penalty.

$$Cost(W, b) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(H(x_i)) + (1 - y_i)\log(1 - H(x_i))]$$

SUPERVISED LEARNING - 10

4. Logistic Regression Coding Practice

[Code] Day01 - 02_Logistic_Regression.py github

UNSUPERVISED LEARNING - SUMMARY

This is a learning method using data without a 'correct answer (label)'. There is no answer sheet, but it allows you to find the structure, pattern, and features hidden in the data itself.

- **Input:** Only the problem (Input) exists
- **Goal:** Discover the hidden structure or pattern of the data
- Representative example:
  - **Clustering:** Grouping data with similar characteristics. (Example: Analyzing the purchasing patterns of customers and dividing them into 'customer groups' with similar tendencies)
  - **Dimensionality Reduction:** Reducing the amount (dimension) of data while maintaining the important features of the data.

The detailed algorithm will be covered when we proceed with in-depth learning on related topics.

REINFORCEMENT LEARNING - SUMMARY

This is a learning method through 'reward'. Unlike supervised learning, the correct answer is not set, but when a certain action is taken, it receives feedback such as 'good job (reward)' or 'bad job (penalty)' and learns.

- **Input:** Current state (State) and reward (Reward)
- **Goal:** Find the optimal action policy (Policy) that maximizes the accumulated reward
- Representative example:
  - **Game AI:** Learns winning strategies by playing numerous games on its own in games such as Go and chess (e.g. AlphaGo)
  - **Robot control:** A robot that learns how to walk without falling over

The detailed algorithm will be covered when we proceed with in-depth learning on related topics.

SUMMARY

Summary

Machine learning methods are largely divided into supervised learning, which studies with correct answers, unsupervised learning, which finds patterns by looking at only data, and reinforcement learning, which learns behavior through rewards and punishments.

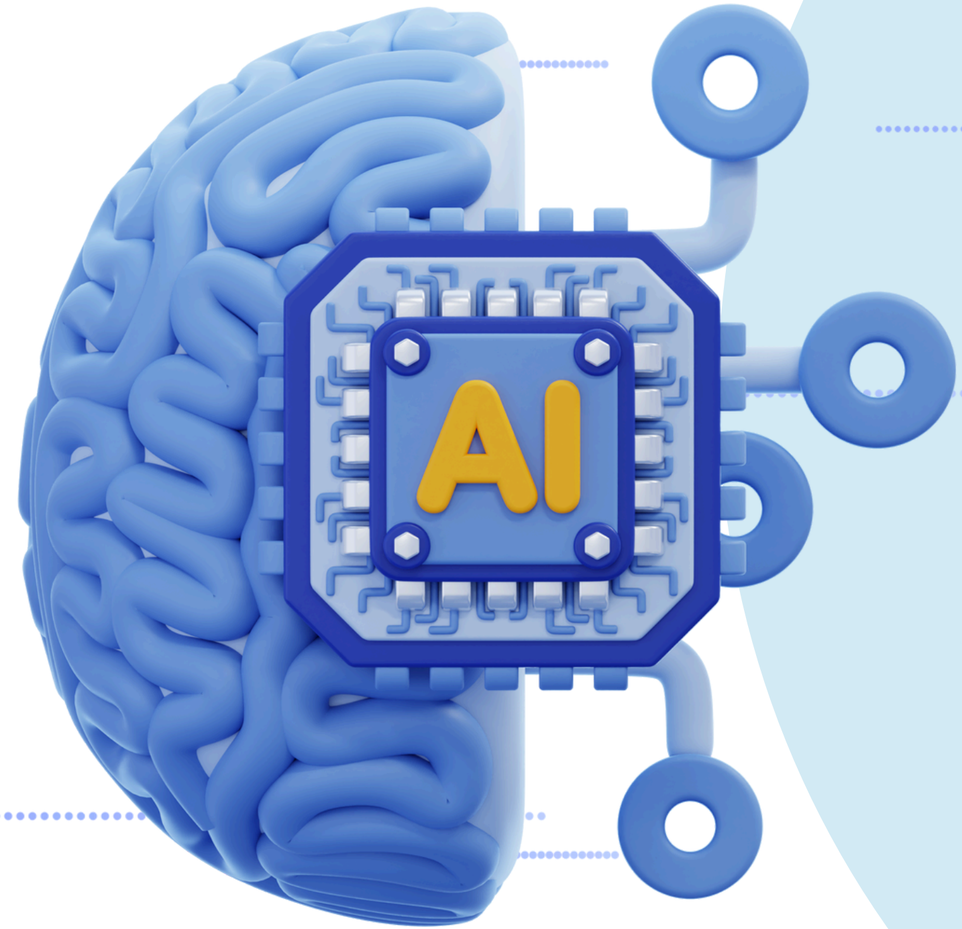
Linear regression, a representative example of supervised learning, predicts continuous values by finding a 'straight line' that best explains the data, and logistic regression solves classification problems by calculating the probability between 0 and 1 with an 'S-curve (sigmoid function)'. Both models define a loss function (MSE, BCE) and learn parameters (W, b) that minimize the loss using gradient descent.

Reinforcement learning, like supervised learning, learns behavior in the direction of maximizing rewards without a set correct answer. It's like a game character learning the optimal game method by repeatedly experiencing gaining (reward) or losing (punishment) points when performing a specific action. The agent, which is the subject of learning, recognizes the current state in the given environment and takes some action. Through the rewards received as a result of that action, it learns on its own the optimal decision-making rule, or Policy, to obtain greater rewards in the future.

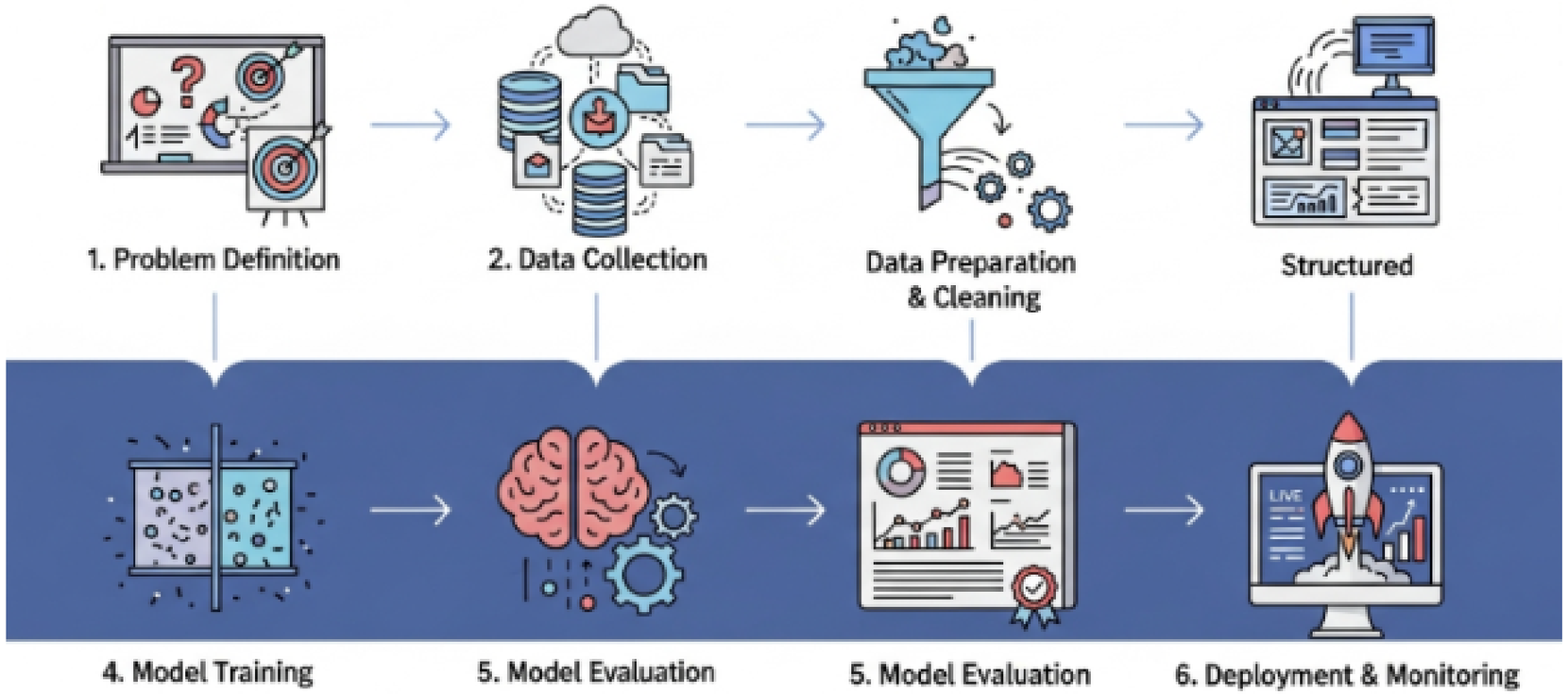
03
MACHINE LEARNING LIFECYCLE

Keywords

Problem definition, Data Collection & Processing, Modeling, Training, Evaluation, Deployment & Monitoring



MACHINE LEARNING WORKFLOW



MACHINE LEARNING WORKFLOW

3. Logistic Regression

- **(Step-1) Problem Definition**
  - This is the most important first step. We clearly define what we want to solve with AI. (For example, what are our business goals and how will we measure success?)

- **(Step-2) Data Collection & Preprocessing**
  - This is the process of collecting the data needed to solve the defined problem and cleaning it into a form that the model can learn from.
  - This includes tasks such as filling in missing values (empty values) or unifying the format of the data. This is often the most time-consuming step in a project.

- **(Step-3) Modeling**
  - This is the step of selecting the most appropriate machine learning algorithm (model) for the problem.
  - For image problems, select a model that fits the characteristics of the data and the problem, such as CNN for image problems and RNN for sequential text problems.

MACHINE LEARNING WORKFLOW

3. Logistic Regression

- **(Step-4) Training**
  - This is the process of inputting prepared data into the model so that the model learns the patterns in the data.
  - In this process, the parameters inside the model are adjusted to the optimal values.

- **(Step-5) Evaluation**
  - This is an objective indicator of how well the model that has completed learning performs.
  - Use new data (test data) that was not used for learning to check the generalization performance of the model. (e.g. accuracy, precision, etc.)

- **(Step-6) Deployment & Monitoring**
  - This is the final step of deploying a model whose performance has been verified to a service (web, app, etc.) that can be used by actual users.
  - Continuously monitor whether the model's performance is maintained even after deployment, and update the model with new data if necessary.

IT Talent Training Course

Aug. 2025.

A.I. PROGRAMMING WITH PYTORCH

Instructor :
Daesung Kim

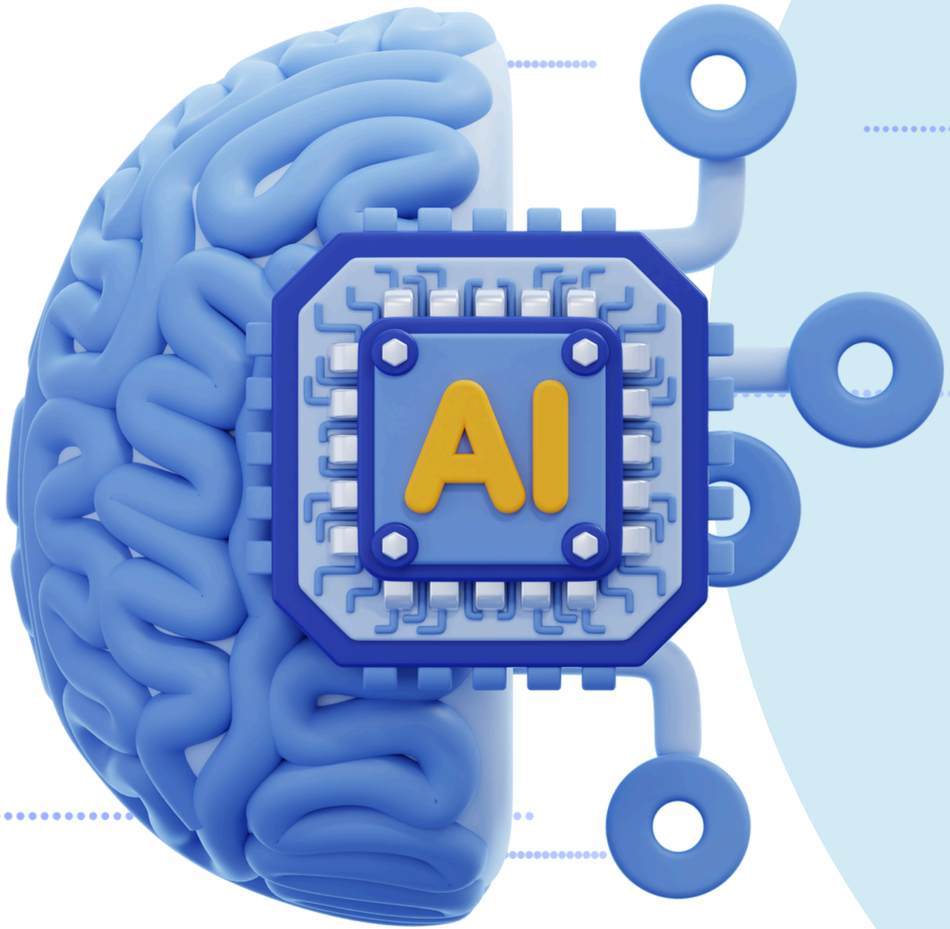
1st Day - Part 02

INDEX

01
NumPy (Numerical Python)

Keywords

Python basic data types



WHY NUMPY INSTEAD OF PYTHON LISTS? - 1

1. Limitations of Python lists

Python has a list that can store multiple data. It is convenient, but it has a fatal disadvantage in that it is very slow for large-scale operations such as adding or multiplying millions of numbers at once.

- **Store various data types:**
  - Python lists can store all kinds of data, such as integers, real numbers, and strings. Because of this flexibility, the computer has to check what kind of data each item in the list is, which slows down the operation speed.

- **Relying on loops:**
  - If you want to add 5 to all elements of a list, you need to use a for loop. If there are 1 million data, 1 million iteration commands should be executed.

```
# Operations using Python lists (example)
my_list = list(range(1000000))
result = []

# Repeat 1 million times to multiply each element by 2
for i in my_list:
result.append(i * 2)

# You can check how fast it takes.
```

WHY NUMPY INSTEAD OF PYTHON LISTS? - 2

2. The power of NumPy and vectorized operations

NumPy solves this problem with the concept of **vectorization**. Vectorization is a method of applying operations to all elements of an array at once without loops.

- **Fast:**
  - NumPy arrays (ndarray) are internally implemented in the C language. C is a low-level language that is easy for computers to understand, so it is very fast to execute. It is like processing loops with optimized C code instead of Python.

- **Concise code:**
  - A million loops are replaced with a single line of code. The code becomes shorter and easier to read.

```
# Vectorization operations using NumPy (example)
import numpy as np

my_array = np.arange(1000000)

# Multiply all elements by 2 in one line without loops
result = my_array * 2

# If you compare the speed with the Python list example, you can feel the overwhelming difference.
```

WHY NUMPY INSTEAD OF PYTHON LISTS? - 3

3. The core of NumPy, ndarray

- Everything in NumPy starts with a data structure called ndarray (N-dimensional array).
  - An ndarray can only contain data of the same type (e.g. all integers, all real numbers).
  - This allows data to be stored contiguously in memory and processed very efficiently.

- Key properties:
  - ndarray.ndim: The number of dimensions of the array (e.g. 1D, 2D)
  - ndarray.shape: The size of each dimension is represented as a tuple
    - (e.g. (3, 4) is a 2D array with 3 rows and 4 columns)
  - ndarray.dtype: The data type of the elements in the array (e.g. int64, float64)

```
# Create ndarray and check properties
arr = np.array([[1, 2, 3], [4, 5, 6]]) # Create a 2-dimensional array of size 2x3

print(f"Array contents: \n{arr}")
print(f"Array dimensions: {arr.ndim}") # Output: 2
print(f"Array shape: {arr.shape}") # Output: (2, 3)
print(f"Array data type: {arr.dtype}") # Output: int64 (may be int32 depending on the system)
```

CREATING AND MANIPULATING NDARRAYS - 1

1. Creating arrays in various ways

There are several convenient functions for creating arrays depending on the situation.

- **np.array():**
  - Creates an array from a Python list or tuple. This is the most basic method.
- **np.zeros():**
  - Creates an array with all elements being 0. This is useful when initializing the weights of a model.
- **np.ones():**
  - Creates an array with all elements being 1.
- **np.arange():**
  - Similar to Python's range() function, but it allows specifying not only integers but also real intervals, and returns an ndarray as a result.

CREATING AND MANIPULATING NDARRAYS - 2

```
# Practice code: Various array creation functions

# 1. Create from a list
list1 = [1, 2, 3, 4, 5]
arr1 = np.array(list1)
print(f"np.array() result:\n{arr1}\n")

# 2. Create an array filled with 0s
# Create a float type array of shape (2, 3) by filling it with 0s
arr2 = np.zeros((2, 3))
print(f"np.zeros() result:\n{arr2}\n")

# 3. Create an array filled with 1s
# Create an int type array of shape (2, 3, 4) by filling it with 1s
arr3 = np.ones((2, 3, 4), dtype=np.int16)
print(f"np.ones() result:\n{arr3}\n")

# 4. Consecutive Create an array by value
# An array that increases by 1 from 0 to 10
arr4 = np.arange(10)
print(f"np.arange(10) result:\n{arr4}\n")

# Create an array from 0 to 1 with an interval of 0.1
arr5 = np.arange(0, 1, 0.1)
print(f"np.arange(0, 1, 0.1) result:\n{arr5}\n")
```

CREATING AND MANIPULATING NDARRAYS - 3

2. Reshape an array

The reshape() function is used to change the dimensions and shape of an array while keeping the data intact. It is used very frequently in deep learning when the shape of data needs to be adjusted to input data into the input layer of a model.

- **NOTE!** reshape can only be used when the number of elements in the original shape exactly matches the number of elements in the shape you are trying to reshape (e.g. you can't reshape 12 elements into a 3x5=15 shape).

```
# Practice code: Changing the shape of an array with reshape

# Create a 1D array with values from 0 to 11
arr = np.arange(12)
print(f"Original array (shape: {arr.shape}):\n{arr}\n")

# Change a 1D array (12 elements) -> 2D 3x4 array
reshaped_arr1 = arr.reshape(3, 4)
print(f"Array reshaped to 3x4 (shape: {reshaped_arr1.shape}):\n{reshaped_arr1}\n")

# Change a 1D array (12 elements) -> 2D 2x6 array
reshaped_arr2 = arr.reshape(2, 6)
print(f"Array reshaped to 2x6 (shape: {reshaped_arr2.shape}):\n{reshaped_arr2}\n")

# Using -1: Automatically calculate from other dimension values
# When you want to make 12 elements into 4 columns, the rows will automatically be 3 (12 / 4 = 3)
reshaped_arr3 = arr.reshape(-1, 4)
print(f"Reshape using -1 (shape: {reshaped_arr3.shape}):\n{reshaped_arr3}\n")
```

FREE ACCESS TO DATA: INDEXING AND SLICING

1. 1D Array Indexing/Slicing

Once you have created an array, you need to get the data you want out of it. Indexing is a method of getting one element at a specific location, and slicing is a method of getting a specific range of elements.

```
# Practice code: 1D array indexing and slicing

arr = np.arange(10)
print(f"Original array: {arr}")

# Indexing (starting from 0)
print(f"arr[0]: {arr[0]}") # First element
print(f"arr[5]: {arr[5]}") # Sixth element
print(f"arr[-1]: {arr[-1]}") # Last element

# Slicing (start:end -> From start index to end-1 index)
print(f"arr[2:5]: {arr[2:5]}") # From index 2 to index 4
print(f"arr[:5]: {arr[:5]}") # From index 4 to index 5
print(f"arr[5:]: {arr[5:]}") # From index 5 to the end
```

2D ARRAY INDEXING/SLICING

2. 2D Array Indexing/Slicing

NumPy's true value starts with two-dimensional arrays (matrices). We access each dimension (axis) using commas (,). The format is arr[row, column].

```
# Practice code: 2D array indexing and slicing

arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(f"Original 2D array:\n{arr2d}\n")

# Indexing a specific element: arr[row_index, column_index]
print(f"arr2d[0, 2]: {arr2d[0, 2]}") # Element in row 0, column 2 -> 3
print(f"arr2d[1, 1]: {arr2d[1, 1]}") # Element in row 1, column 1 -> 5

# Slicing a specific row/column
# The : symbol means 'all elements in the corresponding dimension'. print(f"1st row (arr2d[1, :]):\n{arr2d[1, :]}\n")
print(f"2nd column (arr2d[:, 2]):\n{arr2d[:, 2]}\n")

# Partial region slicing
# Part corresponding to rows 0~1 and columns 1~2
sub_arr = arr2d[:2, 1:]
print(f"0~1 rows and 1~2 column slicing result:\n{sub_arr}")
```

VECTORIZED OPERATIONS AND BROADCASTING - 1

1. Operations between arrays (vectorization)

When performing arithmetic operations on NumPy arrays, the operations are performed on elements at the same location (index). This is called **element-wise operation**. Also, no loops are needed.

```
# Practice code: Vectorization operations between arrays

arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])

# Addition
print(f"arr1 + arr2:\n{arr1 + arr2}\n")

# Multiplication
print(f"arr1 * arr2:\n{arr1 * arr2}\n")

# Scalar (single-valued) operations: Apply the same operation to all elements
print(f"arr1 * 10:\n{arr1 * 10}\n")
```

VECTORIZED OPERATIONS AND BROADCASTING - 2

Note: Matrix Multiplication (Dot Product)

- Matrix multiplication, as we learn in mathematics, is different from regular multiplication (*). In NumPy, we use the np.dot() function or the @ operator.
  - (A, B) @ (B, C) -> (A, C) matrix is generated.

```
# Practice code: Matrix multiplication

arr1 = np.array([[1, 2], [3, 4]]) # (2, 2)
arr2 = np.array([[5, 6], [7, 8]]) # (2, 2)

# Matrix multiplication
dot_product = np.dot(arr1, arr2)
# dot_product = arr1 @ arr2 Same as
print(f"Matrix multiplication result (np.dot):\n{dot_product}")
```

VECTORIZED OPERATIONS AND BROADCASTING - 3

2. Broadcasting: Operations on arrays of different sizes

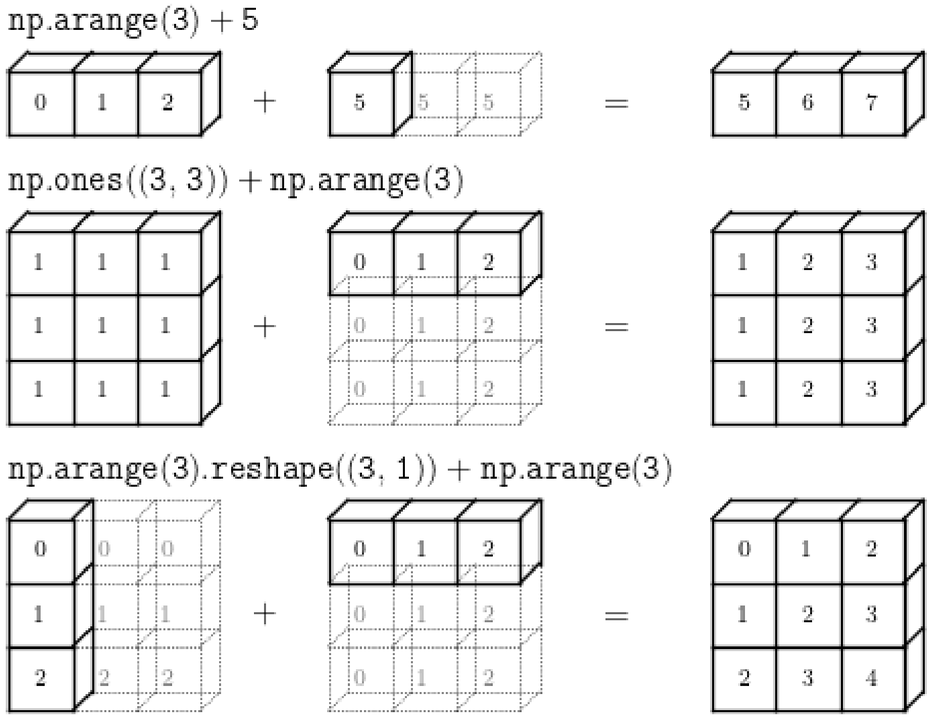
This is one of the most powerful and convenient features of NumPy. **It is a function that virtually expands arrays so that operations can be performed on arrays with different shapes** if they satisfy certain rules.

- **Rule:** When comparing the dimensions of two arrays from the back, if (1) the sizes of the dimensions are the same, or (2) one of the two dimensions is 1, broadcasting is possible.

- **Example-1)** (3, 3) array + (3,) array (1 dimension)
  a. Compare the dimensions of (3, 3) and (3,).
  b. Consider the (3,) array as the (1, 3) array.
  c. Compare from the back dimension: 3 == 3 (equal -> OK)
  d. Compare the front dimension: 3 vs 1 (one side is 1 -> OK)
  e. Conclusion: Broadcasting is possible! The (1, 3) array is copied 3 times to become (3, 3) and then operated on.

VECTORIZED OPERATIONS AND BROADCASTING - 4

Example-2)



Example-3)

```
# Practice Code: Broadcasting

# 2D array of shape (3, 3)
arr1 = np.array([[0, 0, 0], [10, 10, 10], [20, 20, 20]])

# 1D array of shape (3,)
arr2 = np.array([0, 1, 2])

# arr2 is copied 3 times and added to each row of arr1
result = arr1 + arr2
print(f"Broadcasting result:\n{result}")

# Predict the result:
# [[ 0, 1, 2],
# [10, 11, 12],
# [20, 21, 22]]
```

BOOLEAN MASKING - 1

There are many times when you want to extract only the data that satisfies a certain condition from an array. For example, 'find all values greater than 0'. In this case, using **Boolean Masking** is very efficient.

1. Creating a Boolean Mask

When you apply a comparison operator to an array, an array (a Boolean mask) is created with True and False values depending on whether each element satisfies the condition.

```
# Practice code: Create a boolean mask

arr = np.array([[1, -2, 3], [-4, 5, -6]])
print(f"Original array:\n{arr}\n")

# Create a mask that only returns True for values greater than 0
mask = arr > 0
print(f"Boolean mask (arr > 0):\n{mask}")
```

BOOLEAN MASKING - 2

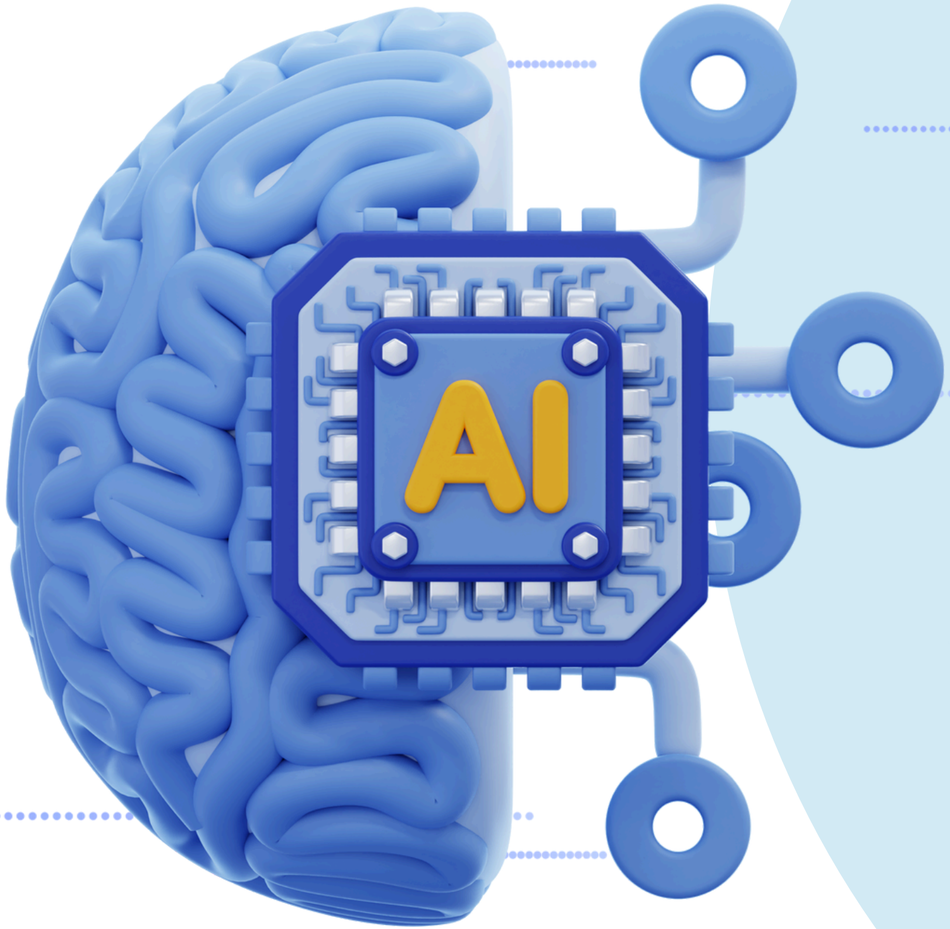
1. Data extraction using masks

By using the generated boolean mask as an index into an array, you can extract only the values at positions corresponding to True.

```
# Practice code: Extract and change data using masks

arr = np.array([[1, -2, 3], [-4, 5, -6]])
mask = arr > 0
print(f"Boolean mask: \n{mask}\n")

# Extract only values greater than 0 using masks
positive_values = arr[mask]
print(f"Values greater than 0: {positive_values}\n")

# Application: Change all values less than 0 to 0
arr[arr < 0] = 0
print(f"Array of values less than 0 changed to 0: \n{arr}")
```

This technique is very useful in data preprocessing to remove outliers or change values within a certain range.

02
INTEGRATED PRACTICE EXAMPLE



LET'S ANALYZE SOME FICTITIOUS DATA USING ALL THE FEATURES OF NUMPY WE LEARNED TODAY.

Problem:
We have test scores for 3 subjects (Math, English, Science) for 4 students.

- Student A: 80, 85, 90
- Student B: 75, 92, 88
- Student C: 95, 80, 78
- Student D: 88, 88, 92

Requirements:

1. Make this data into a 4x3 NumPy array scores.
2. Calculate the mean score for each subject. (Hint: np.mean() function, axis argument)
3. Calculate the mean score for each student, and print the names of the students who scored 90 or higher.
4. Count how many scores are below 80 out of the total scores.
5. Print all subject scores for students who scored 90 or higher in Math.