



1.

## INTRODUCTION TO AI

- · Artificial Intelligence is an exciting scientific discipline that studies how we can make computers exhibit intelligent behavior, e.g. do those things that human
- Originally, computers were invented by Charles Babbage to operate on numbers following a well-defined procedure an algorithm. Modern computers, even thou original model proposed in the 19th century, still follow the same idea of controlled computations. Thus it is possible to program a computer to do something if we need to do in order to achieve the goal.
- DIFFERENCE BETWEEN WEAK AI AND STRONG AI
- Weak AI (Narrow AI):
  - **Definition**: AI that is designed and trained for a specific task.

Capabilities: It can simulate human intelligence but doesn't truly understand or possess consciousness.

### Examples:

Voice assistants like Siri. Alexa Recommendation systems (e.g., Netflix, Amazon) Chatbots and image recognition tools

- **☑** Good at performing one task extremely well
- X Cannot generalize to other tasks
- Strong AI (Artificial General Intelligence, AGI):
- **Definition**: AI with **human-like cognitive abilities**—able to understand, learn, and apply knowledge across a wide range of tasks.
- Capabilities: It would have consciousness, self-awareness, and true understanding—not just simulating intelligence.
- **Examples**: Purely theoretical at this point—no current system is truly Strong AI.
  - **M** Can think, reason, and adapt like a human
  - X Still hypothetical and under research

## **TURING TEST**

- When speaking about AGI we need to have some way to tell if we have created a truly intelligent system. Alan Turing proposed a way called a Turing Test, which also a compares a given system to something inherently intelligent - a real human being, and because any automatic comparison can be bypassed by a computer program, w being is unable to distinguish between a real person and a computer system in text-based dialogue - the system is considered intelligent.
- <u>Different Approaches to Al</u>

There are two possible approaches to this problem:

Top-down Approach (Symbolic Reasoning)	Bottom-up Approach (Neura
A top-down approach models the way a person reasons to solve a problem. It involves extracting knowledge from a human being, and representing it in a computer-readable form. We also need to develop a way to model reasoning inside a computer.	A bottom-up approach models the structure of a human brain, consist neurons. Each neuron acts like a weighted average of its inputs, and problems by providing training data.

-There are also some other possible approaches to intelligence:

- An Emergent, Synergetic or multi-agent approach are based on the fact that complex intelligent behaviour can be obtained by an interaction of a large number of simp cybernetics, intelligence can emerge from more simple, reactive behaviour in the process of metasystem transition.
- An Evolutionary approach, or genetic algorithm is an optimization process based on the principles of evolution.
- In a top-down approach, we try to model our reasoning. Because we can follow our thoughts when we reason, we can try to formalize this process and program it insic <u>reasoning.</u>

People tend to have some rules in their head that guide their decision making processes. For example, when a doctor is diagnosing a patient, he or she may realize that some inflammation going on inside the body. By applying a large set of rules to a specific problem a doctor may be able to come up with the final diagnosis. This approach relies heavily on knowledge representation and reasoning. Extracting knowledge from a human expert might be the most difficult part, because a doctor he or she is coming up with a particular diagnosis. Sometimes the solution just comes up in his or her head without explicit thinking. Some tasks, such as determining

Bottom-Up Approach

be at all reduced to manipulating knowledge.

Alternately, we can try to model the simplest elements inside our brain - a neuron. We can construct a so-called artificial neural network inside a computer, and then examples. This process is similar to how a newborn child learns about his or her surroundings by making observations

- Machine Learning (ML)
- How it works: Learns from data and patterns, not from hard-coded rules.

## Types:

Supervised learning – Learns from labeled examples (e.g., spam vs. not spam) Unsupervised learning – Finds patterns in unlabeled data Reinforcement learning – Learns through trial and error (like training a dog)

Very powerful and flexible

X Needs lots of data

# A Brief History of AI

- How did AI start?
- In the 1950s, scientists wanted computers to think like people.
- They used rules and logic to build programs.
- *One big success was expert systems* programs that gave advice like a doctor or engineer.
- **BUT** it was hard:
- Writing all the rules was slow.
- Keeping the computer's knowledge updated was too much work.
- So, people lost interest. This was called the AI Winter in the 1970s.
- ₩ What changed?
- Computers became cheaper and faster.
- We got lots of data (thanks to the internet).
- Scientists started using **neural networks** a way for computers to **learn from examples**.
- Neural networks became really good at:
- Recognizing images (computer vision).
- Understanding speech.
- **∠** Chess A Cool Example
- *Old method:* Computers guessed lots of moves and picked the best one using logic (search and rules).
- Better method: Computers learned from past human games (case-based reasoning).
- Today: AI learns by playing with itself and improving over time using neural networks + reinforcement learning (like how humans practice).
- That's how programs like AlphaZero can beat world champions!
- **W** Other games AI learned to play:
- Go (AlphaGo beat a world champ!)
- Poker
- StarCraft II
- Dota 2

## AI FOR BEGINNER PART-2 / SYMBOLIC AI

## Knowledge Representation and Expert Systems----

## What is Knowledge?

- Knowledge is what we know and understand about the world.
- It's not just information we see or hear it's what we learn and connect in our minds.
- For example, you read a book (data), understand the meaning (information), and then remember and use it in real life that's knowledge.

OneNote

## DIKW Pyramid (From Data to Wisdom)

- 1. Data Just raw facts.
  - Example: The word "computer" printed in a book.
  - ➤ It's just text doesn't mean anything until someone reads it.
- 2. Information When we understand what the data means.
  - Example: You read "computer" and know it' s a machine.
  - ➤ Now the word has meaning.
- 3. Knowledge When we connect information to what we already know.
  - Sexample: You learn how a computer works, what it's used for, and where to buy one.
  - ➤ It becomes part of your personal understanding of the world.
- 4. Wisdom Knowing how and when to use your knowledge.
  - 🤔 Example: You decide when using a computer is helpful or why someone might not need one.
  - ➤ It's smart decision-making based on knowledge.





## Classifying Computer Knowledge Representations-----

## Network Representations (Semantic Networks)

- Think of a mind map or a web of ideas.
- In our brain, we connect ideas like: "Python  $\rightarrow$  is a  $\rightarrow$  programming language"
- A semantic network does the same thing on a computer it shows concepts (nodes) and relationships (arrows or edges) between them.

## *Section 2 Example:*

[Python] — is —> [Untyped Language] [Python] — invented by —> [Guido van Rossum] [Python] — block syntax —> [Indentation]

### **※** Object-Attribute-Value Triplets

Another way to store this network in a computer is to break each connection into 3 parts: Object – Attribute – Value

## Example:

Object	Attribute	Value
Python	is	Untyped-Language
Python	invented-by	Guido van Rossum
Python	block-syntax	indentation
Untyped-Language	doesn't have	type definitions

This is easy for a computer to store, search, and connect.

## Hierarchical Representations - Like a Family Tree

- We humans think in hierarchies big categories with smaller ones inside.
- Example:
  - Canary is a Bird
  - **>** Bird is an Animal
- From this, we know:
- All birds (including canaries) have wings.
- So if something is a bird, it inherits bird properties.
  - Frame Representation Like a Form or Template
- A frame is like a profile or a form that describes an object.
- It has slots, like fields in a form.
- Each slot holds values, default values, or even rules.
  - Example: Frame for Python (programming language)

Slot	Value	Default Value	Range or Notes
Name	Python		
Is-A	Untyped-Language		(category it belongs to)
Variable Case		CamelCase	(default case style)
Program Length			5–5000 lines
Block Syntax	Indent		(uses indentation)

- This is like saying:
- Python is an untyped language.
- If we don't know the case style, we assume it's CamelCase by default.
- Most Python programs are between 5 to 5000 lines.
- It uses indentation for blocks.

## Procedural Representations – "Knowledge as Actions"

- In this type, knowledge is stored as a set of actions or steps to take when something happens.
- It's like if-this-happens  $\rightarrow$  then-do-this.

### 1. Production Rules – IF-THEN Statements

- These are simple rules that help us make decisions.
- Example (Doctor's Rule):
  - ← IF a patient has high fever OR high C-reactive protein
- Once we know one part is true, we can use it to conclude something else.

### 2. Algorithms – Step-by-Step Procedures

- Algorithms are a set of fixed steps to solve a problem.
- Like a recipe in cooking:
- Step 1: Boil water
- Step 2: Add pasta
- Step 3: Cook for 10 minutes
  - Programming aren't used much as direct knowledge they're more for programming, not for understanding "facts."

## 3. Logic - Representing Universal Knowledge

- Logic is a formal way to represent facts and reasoning.
- It started with Aristotle, who tried to describe how humans think using rules.

## **Types of Logic Used in AI:**

- ➤ Predicate Logic
- Used to describe facts like:
- It's very powerful, but too complex for computers to handle fully.
  - V So we use simpler parts, like Horn clauses (used in Prolog).
  - ➤ Description Logic
- Used to describe hierarchies and relationships between objects.
- Commonly used in the Semantic Web to define and connect knowledge online.

## What is an Expert System?

An Expert System is a computer program that mimics the decision-making ability of a human expert in a specific field, like medical diagnosis, engineering, or finan

### Knowledge Base (like a brain full of expert facts)

- It stores **expert knowledge**—facts, rules, and information about a specific topic.
- This knowledge is taken from real human experts and entered into the system.
- It doesn't change during each use—it's like the system's memory.
- It helps the system move from one problem to a solution, so it's also called dynamic system

## **# Inference Engine** (like the thinking part of the brain)

- This is the part that thinks and makes decisions.
- It looks at the problem, searches through the knowledge base, and decides what rule to apply.
- If it needs more info, it **asks questions** to the user (like a doctor asking about symptoms).
- Its job is to **find a path from the problem to the solution** by using the rules.

## Introduction to Neural Networks-

## Machine Learning

Neural Networks are a part of a larger discipline called Machine Learning, whose goal is to use data to train computer models that are able to solve problems. Machine Intelligence

we will consider the two most common machine learning problems:

#### 1. Classification



In **classification**, the goal is to **put things into categories** (or classes).

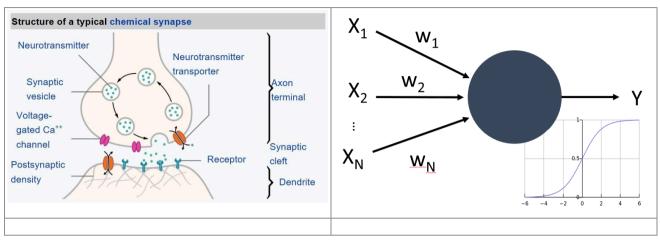
## 2. Regression

**What it is:** 

In regression, the goal is to predict a number.

#### A Model of a Neuron

From biology we know that our brain consists of neural cells, each of them having multiple "inputs" (axons), and an output (dendrite). Axons and dendrites can conduct electrical signals can exhibit different degrees of conductivity (controlled by neuromediators).



Thus, the simplest mathematical model of a neuron contains several inputs X1, ..., XN and an output Y, and a series of weights W1, ..., WN. An output is calculated as:

$$Y = f\left(\sum_{i=1}^{N} X_i W_i\right)$$

Introduction to Neural Networks: Perceptron

### · Perceptron Kya Hota Hai?

Perceptron ek simple sa machine learning model hai jo 2 cheezon mein farq karna seekhta hai — jaise:

- · Email spam hai ya nahi
- Image mein cat hai ya dog (Only 2 options — binary classification)
- 📥 Input Aur 📤 Output
- Hum kuch inputs/features dete hain, jaise [x1, x2, x3...]
- Har input ka ek weight hota hai [w1, w2, w3...]
- Ye inputs aur weights multiply hoke total score dete hain

Formula

$$V(X) = f(W \cdot X)$$

- $W \cdot X = (W1 \times X1 + W2 \times X2 + ... + Wn \times Xn)$
- f() = step function: Agar result > 0 ho to output +1, warna -1
  - @ Perceptron Ka Goal

Training ka goal hota hai:

🗾 Aise weights dhoondhna jo zyada se zyada inputs ko sahi classify karein.

X Jab Galti Hoti Hai...

Agar perceptron kisi input ko galat predict karta hai to hum weights ko update karte hain taaki agli baar sahi ho.

Error ko define karte hain:

mathematica

$$E(W) = -\sum (W \cdot X_i \times t_i)$$

- $Yahaan x_i = input vector$
- $t_i$  = true label (either +1 ya -1)
- Sirf unhi inputs ko include karte hain jo galat classify huye
  - Gradient Descent (Update Ka Tareeka)

Weights ko update karne ka rule:

ini

$$W = W + \eta \times X_i \times t_i$$

- $\eta$  = learning rate (kitna update karna hai)
- Agar galti hui hai to weights mein thoda sa change karo
- Ye process bar-bar repeat hota hai (iterations mein)
  - **\*** Introduction to Neural Networks & Multi-Layer Perceptron (MLP)

What's a Neural Network?

Think of a **neural network** like a smart calculator that learns to make predictions by looking at examples. It takes some **input**, does some math, and gives an **output** li. (e.g., house price).

## One-Layer Perceptron:

This is the simplest model. It draws a straight line (or plane) to separate two groups of points (like red dots and blue dots) based on their features. But... it only works if the data can be separated by a straight line (linearly separable).

## Multi-Laver Perceptron (MLP):

To solve more complex problems, we add more layers in between input and output. These are called hidden layers.

Each layer transforms the data step by step until we get a useful output.

Now, the network can:

- Handle more than 2 classes
- Predict numbers (regression)
- Separate curved/complex data shapes

## Machine Learning as Math

We want to learn a function f(x) that gives the correct output (y) for each input (x).

To know how good or bad our guess is, we use a loss function  $\mathcal{L}$  — a score for how wrong we are.

- For numbers  $\rightarrow$  Use squared error
- For classes  $\rightarrow$  Use accuracy or logistic loss

The better the model, the lower the loss.

### Parameters: w and b

- w (weights): Think of them as knobs to control how much each input matters
- b (bias): A fixed shift to adjust the result

We call them  $\theta = \langle w, b \rangle$ , and we want to tune them to reduce the loss.

## Gradient Descent: Learning Step by Step

Like hiking down a mountain

- Start with random values of w and b
- Check which direction reduces the loss

- Take small steps (controlled by learning rate  $\eta$ )
- Repeat until we reach the bottom (lowest loss)

## MLP Example Math

With more layers, the function becomes:

```
\rightarrow Hidden layer z2 = w2 \cdot activation(z1) + b2 <math>\rightarrow Output layer f = softmax(z2)
                                                                                                                                        \rightarrow Final prediction
z1 = w1 \cdot x + b1
```

- activation() = adds non-linearity so we can handle curves (like ReLU or sigmoid)
- softmax() = turns outputs into probabilities for classification

## Backpropagation: Learning Backwards

To update weights correctly, we use backpropagation — a method that applies the chain rule of calculus:

- Start from the loss
- Work backwards through the layers
- Adjust each weight using the gradient

It's like following the steps of a recipe in reverse, figuring out how each ingredient (weight) affected the final taste (loss).

Neural Network Frameworks:

#### What Do We Need to Train Neural Networks?

To train neural networks well, we need two things:

- *Do math with tensors (big arrays of numbers)* like addition, multiplication, applying functions like **sigmoid or softmax**.
- *Compute gradients* this helps the model learn by improving step-by-step (called gradient descent).

#### Manual vs. Automatic Gradients

- With NumPy, we can do math, but we have to manually code how the model learns (called backpropagation).
- Good frameworks (like PyTorch and TensorFlow) automatically calculate gradients much easier!
- o CPU vs. GPU
- Neural networks are heavy! So, we use GPUs or even TPUs to make training faster by parallelizing (doing many things at once).

## **\*** Two Popular Frameworks:

Туре	TensorFlow	PyTorch
Low-Level API	TensorFlow	PyTorch
High-Level API	Keras	PyTorch Lightning

- Low-Level API = more control, but more work (good for research)
- High-Level API = much easier, just build layers and train (good for projects)

You can mix both! Like:

- · Build a custom layer using low-level API
- Train with high-level API

### Which to Learn?

Choose either PyTorch or TensorFlow — both are good. Many people prefer PyTorch because it's easier to debug and learn.

## What is Overfitting?

Overfitting = When your model is too smart on training data, but bad on new data.

## Example:

*Imagine you have only 5 dots to draw a line through:* 

- *Simple model* (*left*): fits the trend  $\rightarrow$  good prediction.
- **Too complex model** (right): passes through every dot perfectly  $\rightarrow$  bad prediction on new data.

Model	Training Error	Validation Error
Simple	5.3	5.1 🗸

Too complex 0 🎉 20 **X** 

- Why Overfitting Happens:
- Too little training data
- Model is too powerful (too many parameters)
- Noisy data
- How to Spot Overfitting:
- Training error is low 🖊
- Validation error is high 🗶

## **1** How to Prevent Overfitting:

- 1. Add more training data
- 2. Use a simpler model
- 3. Use **regularization** (like **Dropout**) a trick to stop overfitting

### Bias-Variance Tradeoff (Simple View)

Term	Meaning
Bias	$Model\ is\ too\ simple  ightarrow misses\ patterns\ (underfitting)$
Variance	Model is too complex $\rightarrow$ learns noise (overfitting)

Goal: Find a balance so your model learns the real pattern, not noise.

#### Conclusion:

- Use frameworks like PyTorch or TensorFlow to make your life easier.
- Use high-level APIs to quickly build models.
- Understand overfitting to make your model work well on real-world data.
- Introduction to Computer Vision---

### What is Computer Vision?

Computer Vision is a way to help computers "see" and understand images, just like humans do.

For example, with computer vision, a computer can:

- Find where an object is in a photo (like a cat or a car)
- Understand what's happening in a picture (like someone dancing)
- Write a description for a photo (like "A dog playing with a ball")
- Rebuild a 3D scene from a flat image
- Recognize faces, guess someone's age or emotion from a photo

### How is Computer Vision Done?

Computer Vision is a part of Artificial Intelligence (AI).

Today, it's usually done using neural networks — these are models that learn by looking at lots of data (images in this case).

The most common neural network used in vision is called a:

← Convolutional Neural Network (CNN) — you'll learn more about this later!

### Before Using a Neural Network...

Sometimes, we need to clean or adjust the image first. This is called image processing — and it helps the model understand the image better.

### Popular Python Tools for Image Processing

Here are some tools (Python libraries) to help with image tasks:

- 4. imageio
- Can read and save images.
- Can turn videos into image frames.
- 5. Pillow (PIL)
- Can change images (like colors, shapes).
- Good for simple edits.
- 6. OpenCV
- Super powerful!
- Can do almost everything related to images and videos.
- Very popular in the industry.
- 7. **dlib**
- Used for more advanced stuff.
- Good at finding faces and facial features (like eyes, nose, mouth).

· Uses machine learning inside.

## What is OpenCV?

OpenCV is a toolbox for working with images and videos.

It's written in C++ (super fast!) but you can use it in Python too.

Even though we won't learn all of OpenCV in this course, we'll use it to do useful things with images, like resizing, filtering, or converting colors.

## ■ How Do You Load an Image in Python?

- In Python, images are treated like grids of numbers using a library called NumPy.
- A grayscale image is like a table of numbers: height x width
- A color image is like: height x width x 3 (because it has 3 colors: Red, Green, Blue) Here's how to load and show an image:

### Here's how to load and show an image:

```
import cv2
import matplotlib.pyplot as plt
im = cv2.imread('image.jpeg')
                               # Load the image
plt.imshow(im)
                               # Show the image
BUT — there's a catch!
OpenCV uses a BGR color order (Blue-Green-Red), but most tools like matplotlib expect RGB.
So, we need to convert the image:
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
```

You can also use this to:

- Convert to grayscale
- Convert to **HSV** (Hue-Saturation-Value)

## Can OpenCV Handle Videos?

Yes! OpenCV can read videos frame-by-frame, just like reading one image at a time. This is super helpful for analyzing or editing videos.

### **%** What Can You Do with OpenCV? (Image Processing)

Before giving images to a neural network, we often **clean or prepare** them. OpenCV helps you do this!

Here are some common things you can do:

## 1-Resize the image

Make it smaller or bigger to fit your model:

im = cv2.resize(im, (320, 200))

### 2-Blur the image

Useful to remove noise

im = cv2.medianBlur(im, 3)

im = cv2. Gaussian Blur(im, (3,3), 0)

## 3-Change brightness or contrast

You can tweak image brightness using simple math with NumPy.

## 4-Thresholding

Turn an image into black & white, often used in scanning or document reading:

```
_, im = cv2.threshold(im, 127, 255, cv2.THRESH_BINARY)
```

## 5-Transform the image shape

- a) Affine transformation Rotate, move, stretch image
- b) Perspective transformation Fix the angle of photos (like making a tilted paper look straight)

#### 6-Understand motion in videos

You can use optical flow to see what's moving from one frame to another.

## imp

- The MNIST dataset (Modified National Institute of Standards and Technology dataset) is a large, well-known collection of handwritten digit images. It's widely used fo learning and computer vision, especially for image classification tasks.
- Why is MNIST Important?
- It is simple and small, making it ideal for:
- Learning and practicing ML concepts
- Testing algorithms quickly
- It's like the "Hello World" of deep learning and computer vision.
- It's often used to benchmark models like logistic regression, SVM, k-NN, CNNs, etc.

### Convolutional Neural Networks

### What's the Problem?

In real life, we want computers to recognize things in pictures, like cats, dogs, cars, etc.

But the same object (like a cat) can appear anywhere in the image — top-left, center, bottom-right...

So we need a smart way to find patterns, not just exact positions.

## What is an Image for a Computer?

A picture is just a grid of numbers to a computer.

- A black-and-white image is a 2D matrix (just rows and columns of numbers).
- A color image is a 3D matrix (red, green, blue channels).

Each number says how bright a pixel is.

### What is a Filter (or Kernel)?

A **filter** is a **tiny grid** (like  $3\times3$  or  $5\times5$ ) of numbers.

Example of a  $3\times3$  filter:

```
[[0, 1, 0],
[1, -4, 1],
[0, 1, 0]]
```

This filter is like a **tool** or **lens** that looks for **certain patterns** in the image — like edges, corners, lines.

## What is Convolution (Filter Application)?

Imagine a small square (the filter) sliding over the whole image, one spot at a time.

At each spot:

- It **looks at 3 \times 3 pixels** from the image (if it's a  $3 \times 3$  filter).
- It multiplies each pixel value with the corresponding filter value.
- It adds them all up to get one new number.
- That number goes into a **new image** (called a feature map).

This process is called **convolution**.

## How Does This Help Find a Cat?

Let's say you want to find a cat in a photo.

- First, the filter may look for horizontal lines (whiskers!).
- Another filter may look for circles (eyes!).
- Then we combine these patterns to say: "Hmm, these features look like a cat!"

OneNote