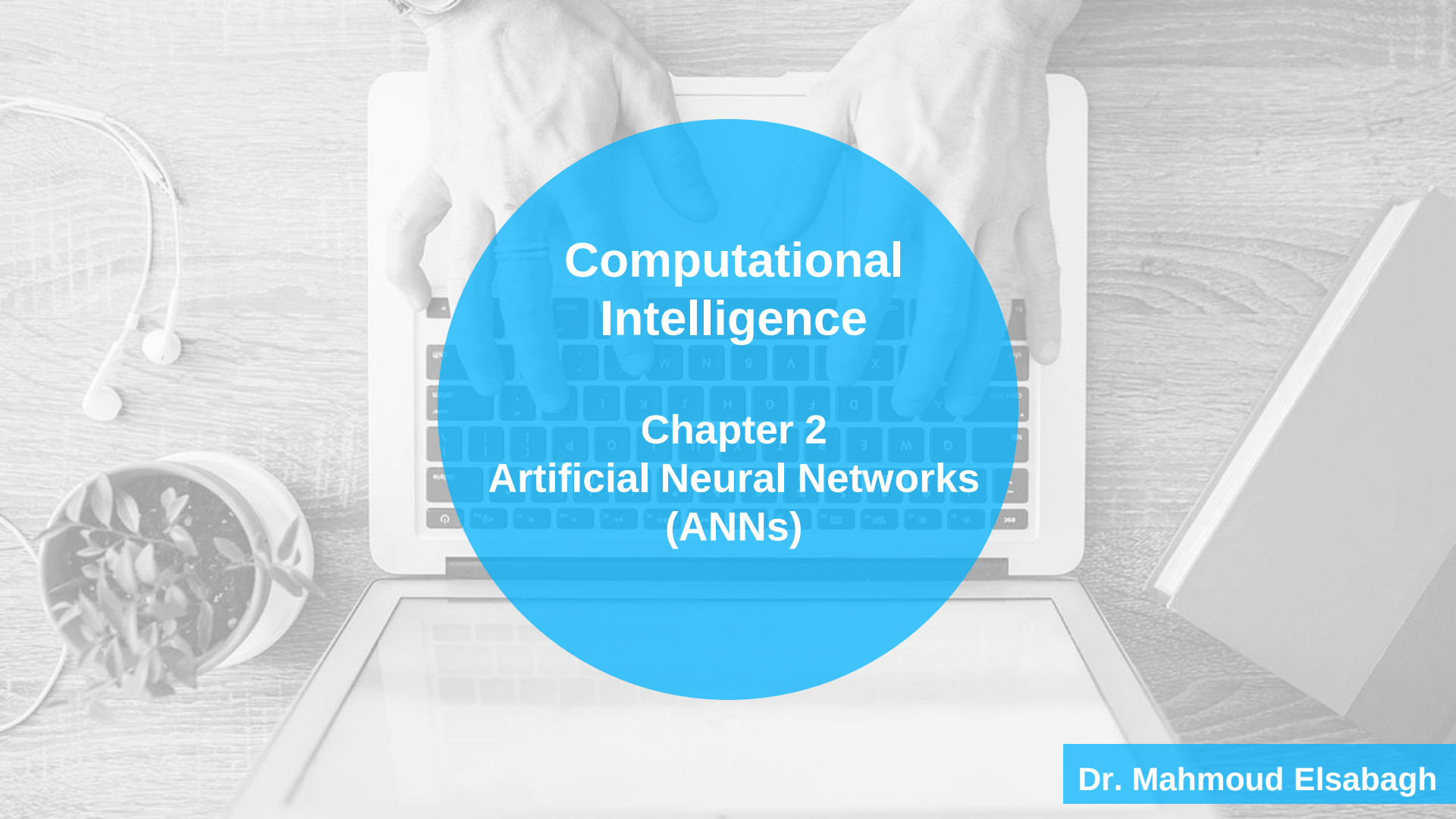
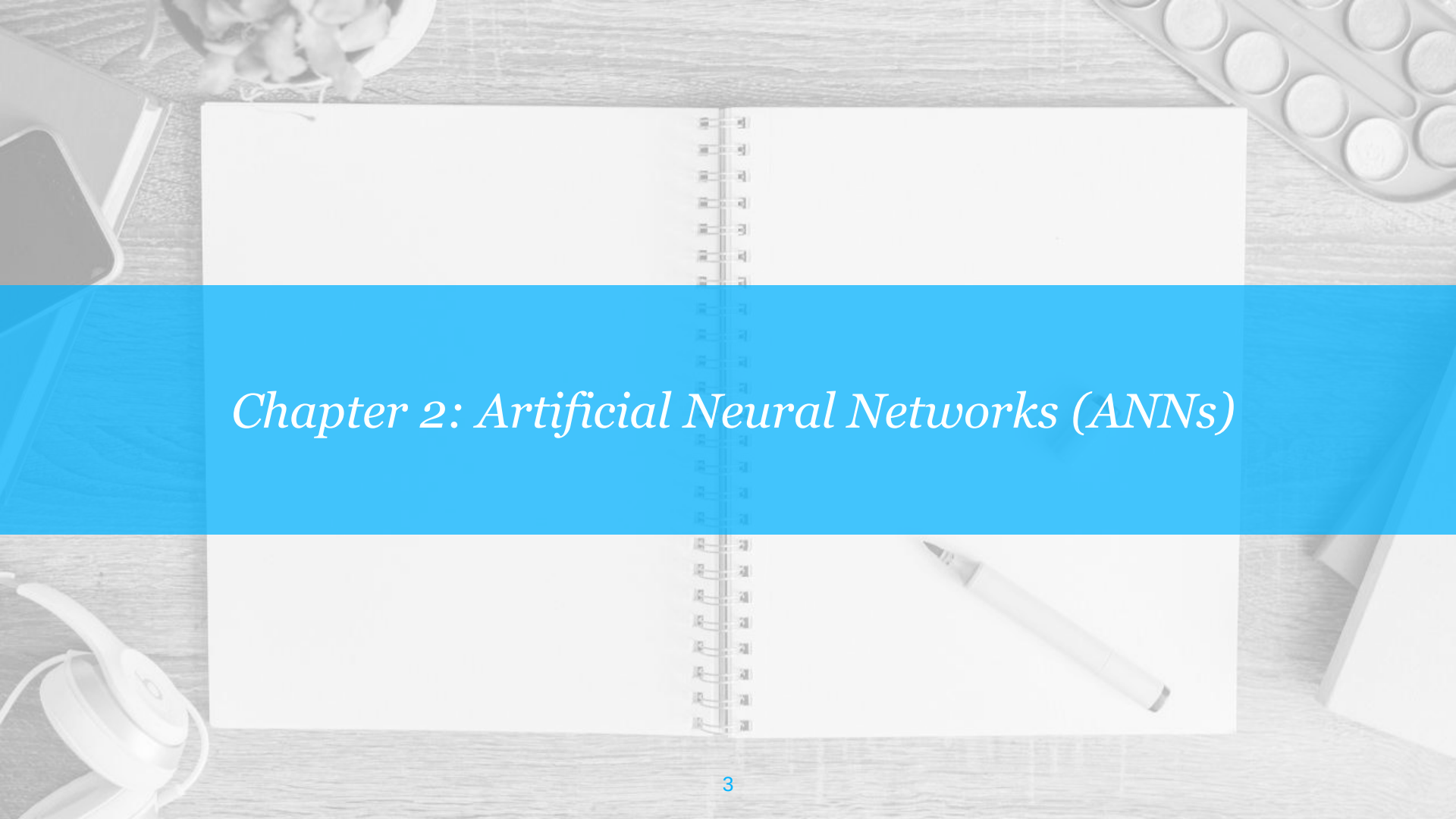


Computational Intelligence

Chapter 2
Artificial Neural Networks (ANNs)

Dr. Mahmoud Elsabagh

Contents



Chapter 2: Artificial Neural Networks (ANNs)

Contents

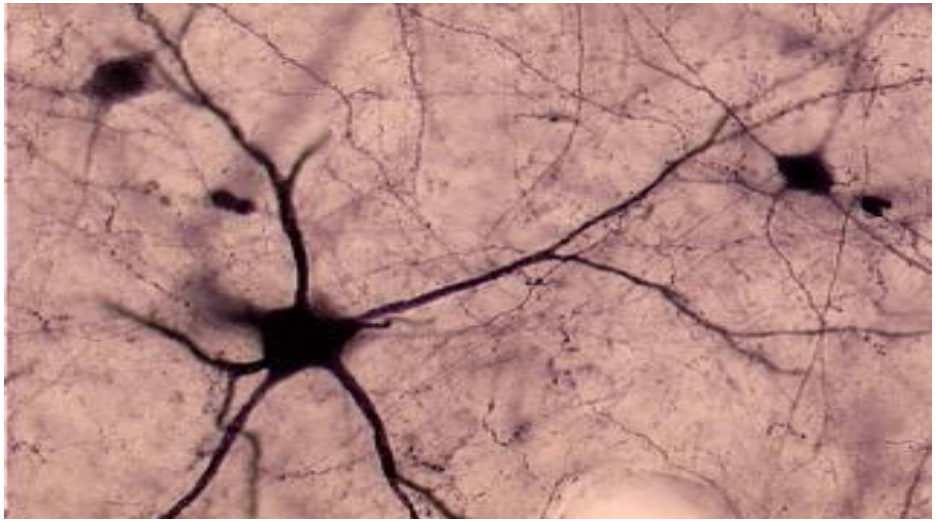
1. Introduction to Artificial Neural Networks

- **Artificial Neural Networks (ANNs)** are among the most fundamental tools in computational intelligence, providing solutions to complex problems such as image recognition, natural language processing, and autonomous control. This lecture will cover the basic building blocks of ANNs, the mathematics behind them, different types of ANNs, and their real-world applications. Along with textual explanations, the key concepts will be illustrated through diagrams and figures.
- An Artificial Neural Network (ANN) is a computational model that mimics the structure of biological neural networks. ANNs are used for tasks such as pattern recognition, classification, regression, and more, by learning from data. The core of an ANN is the artificial neuron (also known as a perceptron), which processes inputs and generates outputs.

Biological Background

➜ **Neuron consists of:**

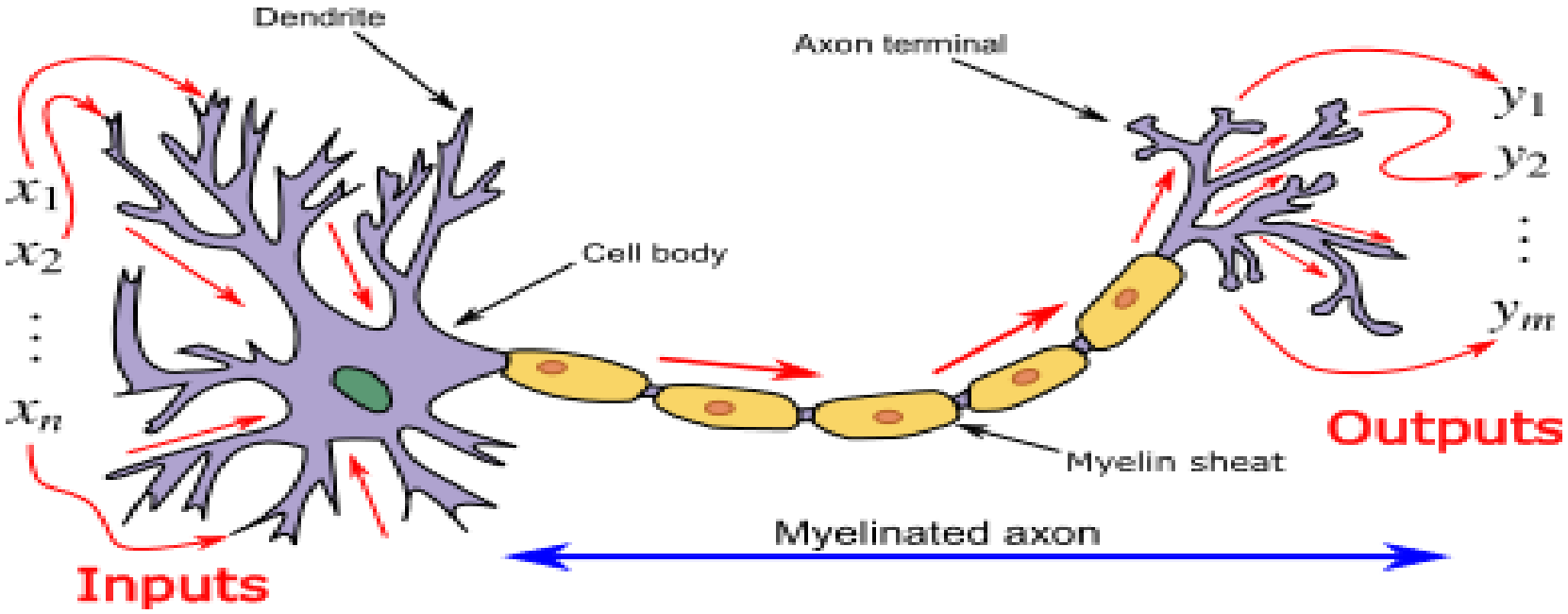
- Cell body
- Dendrites
- Axon
- Synapses



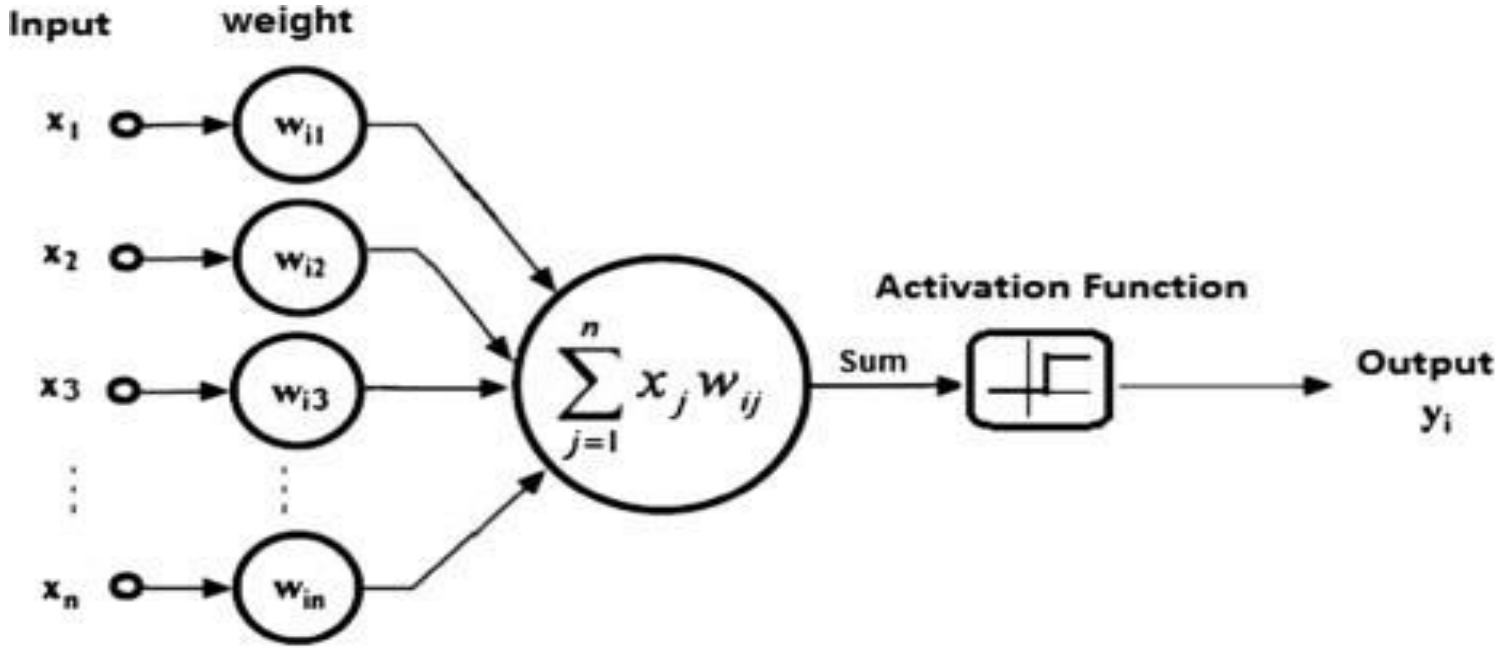
➜ **Neural activation :**

- Throught dendrites/axon
- Synapses have different strengths

ANN ingredients



Simulation on ANN



1. Introduction to Artificial Neural Networks

➜ **Key Components:**

- **Inputs (X)**: Features or data points provided to the model.
- **Weights (W)**: Parameters that control the influence of inputs on the neuron's output.
- **Activation Function (f)**: Non-linear function that determines the neuron's output.
- **Output (Y)**: The final result after processing inputs through layers of neurons.

2. Structure of an Artificial Neuron (Perceptron)

➜ An **artificial neuron** (often referred to as a **perceptron**) processes multiple inputs by applying weights and an activation function to produce an output.

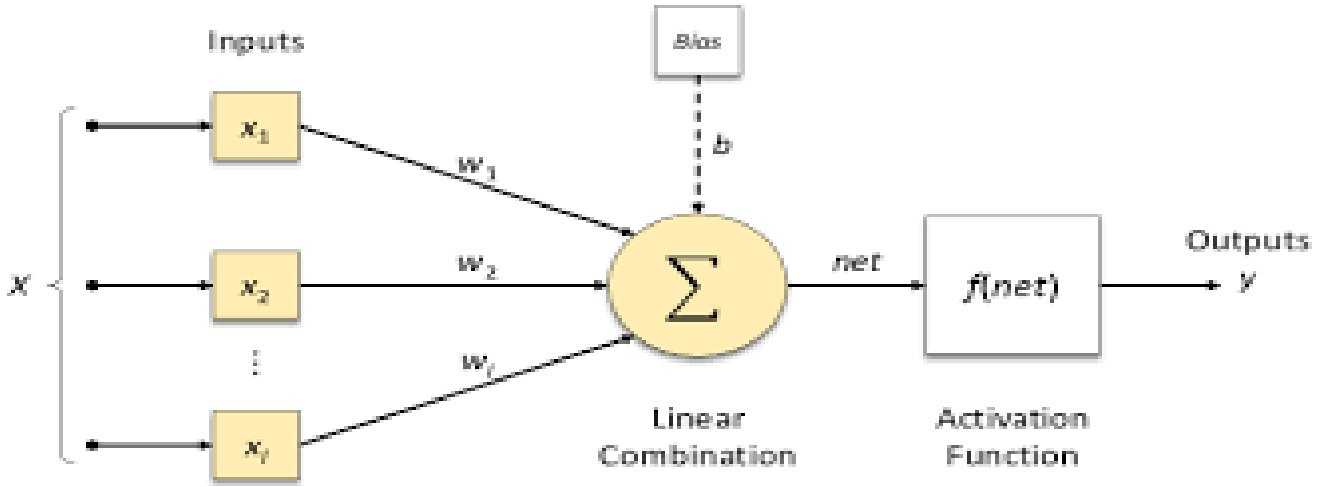
Mathematical representation of a neuron:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Where:

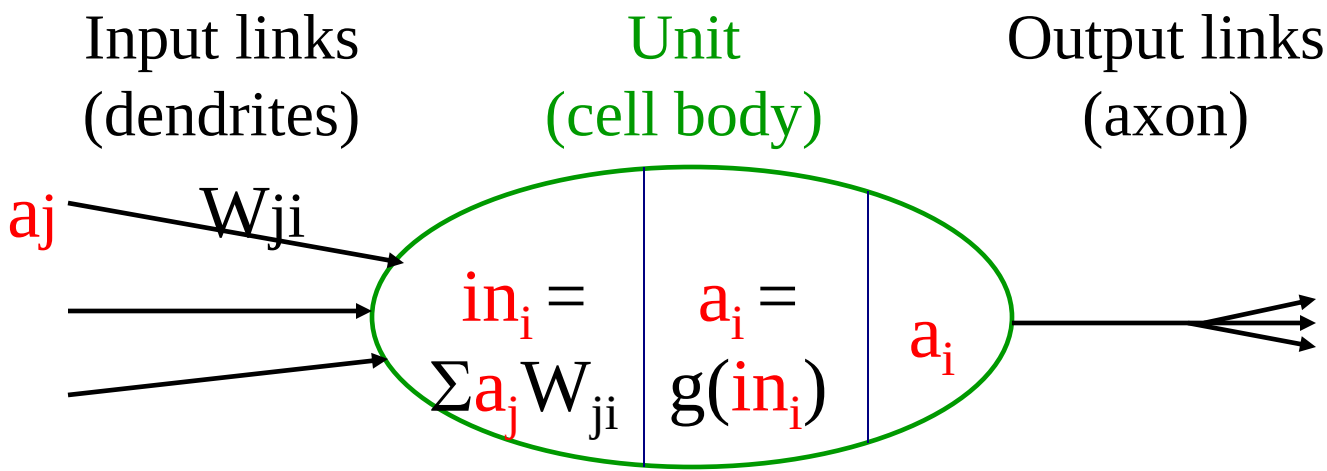
- x_i: Inputs
- w_i: Weights
- b: Bias term
- f: Activation function (e.g., sigmoid, ReLU)
- y: Output

2. Structure of an Artificial Neuron (Perceptron)

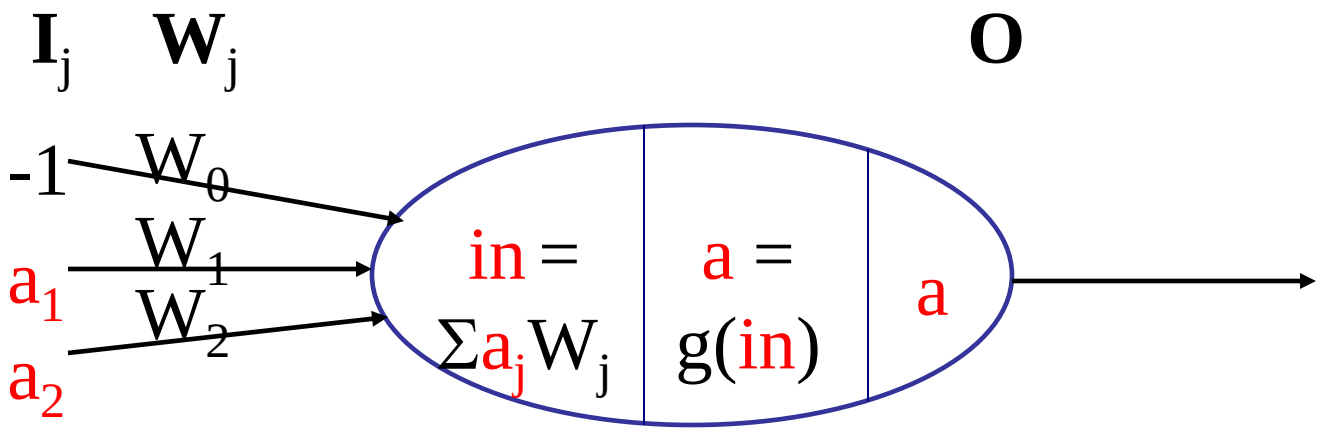


Perceptron Structure

Artificial Neuron Example



Perceptron



$$a = g(-W_0 + W_1 a_1 + W_2 a_2) \qquad g(in) = \begin{cases} 0, & in<0 \\ 1, & in>0 \end{cases}$$

3. ANN Architecture: Input, Hidden, and Output Layers

➜ An ANN typically consists of:

- **Input Layer**: Receives raw data.

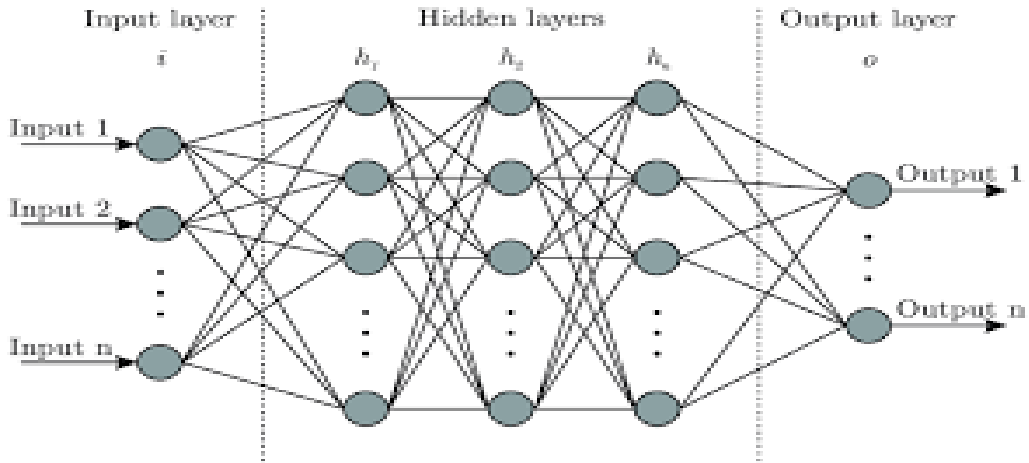
- **Hidden Layers**: Perform intermediate computations; multiple layers allow the network to learn complex features.

- **Output Layer**: Provides the final prediction or classification.

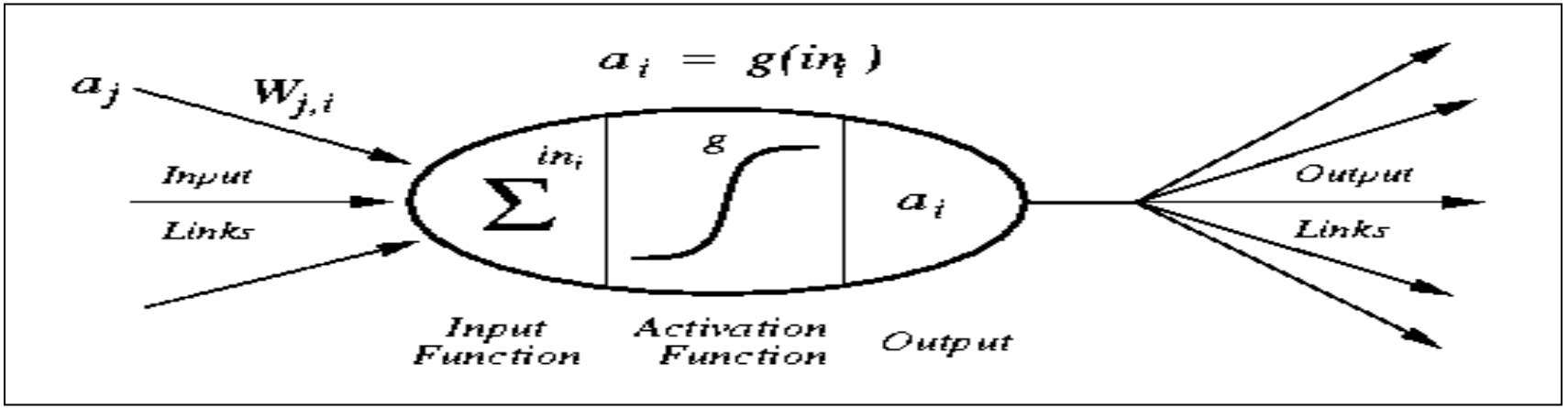
3. ANN Architecture: Input, Hidden, and Output Layers

➜ **Architecture of an Artificial Neural Network**

➜ Each layer consists of neurons, and each neuron in one layer is connected to all neurons in the next layer.



Modelling a Neuron



$$in_i = \sum_j W_{j,i} a_j$$

a_j :Activation value of unit j
$w_{j,I}$:Weight on the link from unit j to unit i
in_I :Weighted sum of inputs to unit i
a_I :Activation value of unit i
g :Activation function

4. Activation Functions: Bringing Non-Linearity

To allow the network to solve non-linear problems, **activation functions** are used after the weighted sum of inputs.

Common Activation Functions:

1. **Sigmoid:**

$$f(x) = \frac{1}{1 + e^{-x}}$$

Range: (0, 1) **Usage:** Often used in binary classification problems.

2. **Tanh:**

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Range: (-1, 1) **Usage:** Helps with problems requiring outputs with both positive and negative values.

3. **ReLU (Rectified Linear Unit):**

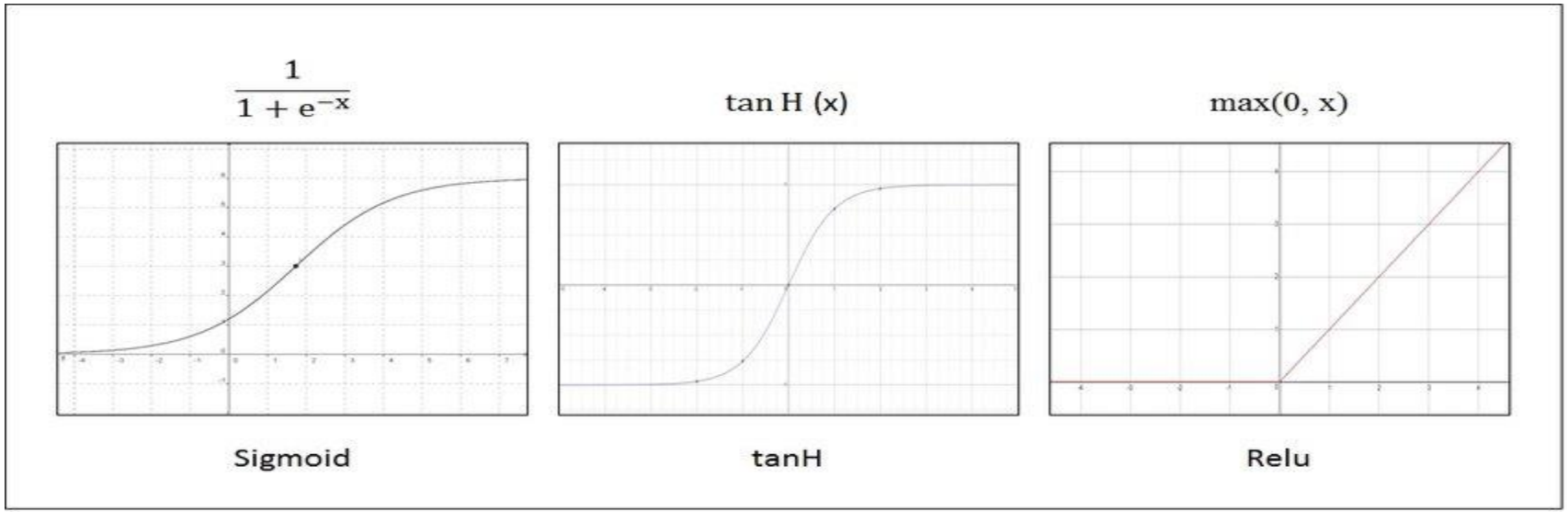
$$f(x) = \max(0, x)$$

Usage: ReLU is the most widely used activation function in deep learning.

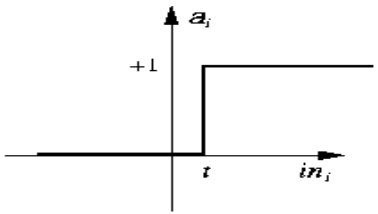
4. Activation Functions: Bringing Non-Linearity

➜ **Key Points**:

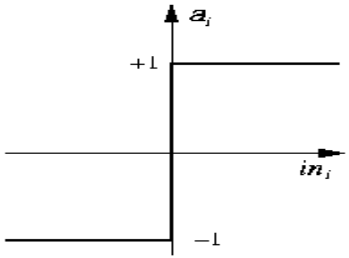
- Activation functions introduce non-linearity, allowing the network to model more complex patterns.
- ReLU is especially popular because of its simplicity and efficiency in training deep networks.



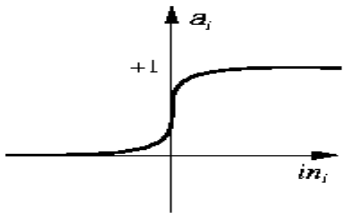
Activation Functions



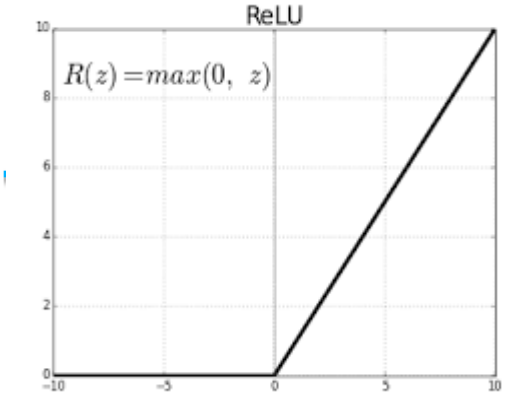
(a) Step function



(b) Sign function



(c) Sigmoid function



$Step_t(x)$ = 1 if x >= t, else 0

Sign(x) = +1 if x >= 0, else –1

Sigmoid(x) = $1/(1+e^{-x})$

Identity Function

Relu(x) = max(o,X)

5. Backpropagation and Gradient Descent

➜ **Backpropagation** is the learning algorithm used in ANNs, where the error (loss) is propagated backward through the network to update weights. The goal is to minimize the error using an optimization technique called **gradient descent**.

➜ **Key Steps in Backpropagation:**

- **Forward Pass**: Compute the network output by propagating the input forward through the layers.

- **Calculate Loss**: Measure the error between predicted output and actual output using a loss function (e.g., Mean Squared Error, Cross-Entropy).

- **Backward Pass**: Update the weights by calculating the gradients of the loss with respect to each weight using the chain rule.

- **Weight Update**: Update weights using the following rule:

$$w_{new} = w_{old} - \eta \cdot \frac{\partial L}{\partial w}$$

where η is the learning rate, and L is the loss.

6. Types of Artificial Neural Networks

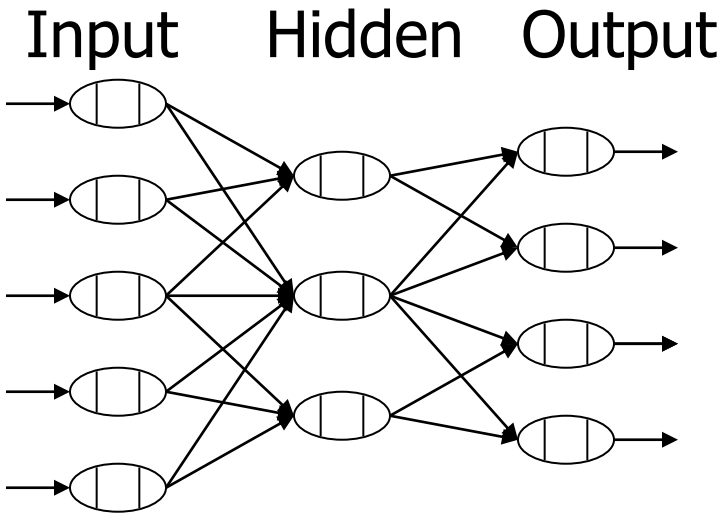
➜ **Feed-forward Neural Networks (FNNs)**:

- The simplest type of neural network, where data flows in one direction—from input to output.

- Used in tasks like classification and regression.

- **Multiple layers:**
  - hidden layer(s)

Feed-forward:
Output links only connected to input links in the next layer



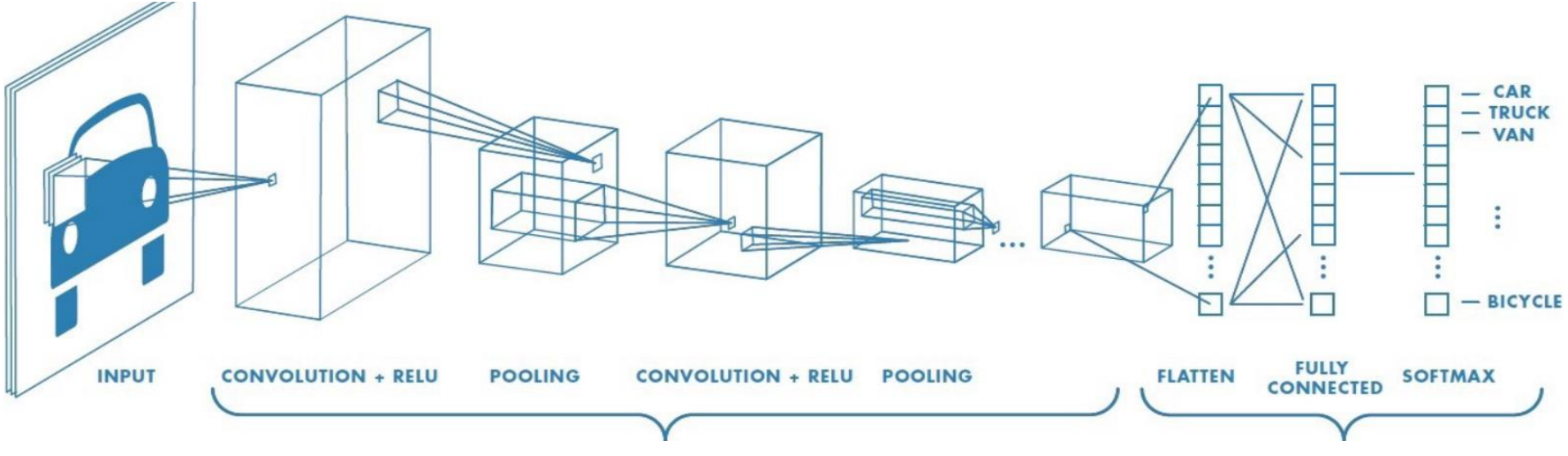
- **Complex non-linear functions can be represented**

6. Types of Artificial Neural Networks

➜ **Convolutional Neural Networks (CNNs)**:

- Primarily used for image-related tasks. They consist of convolutional layers that capture spatial hierarchies in the data.
- **Applications**: Image classification, object detection, face recognition.

➜ **CNN Architecture**

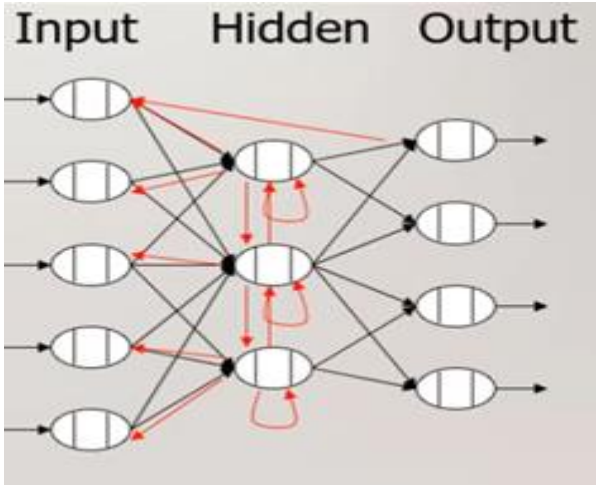


6. Types of Artificial Neural Networks

➜ **Recurrent Neural Networks (RNNs)**:

- Used for sequential data, such as time series or text. RNNs maintain a **hidden state** that captures information from previous time steps.
- **Applications**: Natural language processing, speech recognition, time-series forecasting.

- No restrictions on connections
- Behaviour more difficult to predict/ understand
- Apps: Voice to text



7. Training Process of ANNs

➜ **Data Preprocessing**:

- **Normalization**: Input data is normalized to ensure that all features have similar scales.

- **Train-Test Split**: The dataset is split into training, validation, and testing sets to evaluate the model.

➜ **Hyperparameters**:

- **Learning Rate (η)**: Controls how big the step is when updating weights.

- **Epochs**: The number of times the entire dataset is passed through the network during training.

- **Batch Size**: Number of samples processed before the model is updated.

8. Real-World Applications of ANNs

➜ **Image Recognition**:

CNNs are widely used in systems like facial recognition (e.g., on smartphones) and medical image analysis (e.g., tumor detection in MRIs).

➜ **Natural Language Processing (NLP)**:

RNNs and transformers power machine translation, chatbots, and text generation.

➜ **Autonomous Vehicles**:

Neural networks are used for sensor fusion, object detection, and decision-making in self-driving cars (e.g., Tesla's Autopilot).

➜ **Healthcare**:

ANNs assist in diagnosis (e.g., identifying diseases from medical scans) and personalized medicine (predicting patient responses to treatments).

9. Challenges and Limitations of ANNs

➜ **Data Requirements**:

ANNs need a large amount of labeled data to learn effectively.

➜ **Computational Power**:

Training deep networks is computationally intensive, requiring GPUs or cloud computing resources.

➜ **Black Box Nature**:

Understanding how a trained ANN makes its decisions is often difficult, which raises concerns in critical applications (e.g., healthcare, legal systems).

➜ **Overfitting**:

ANNs may perform well on training data but poorly on unseen data if overfitting occurs. Regularization techniques such as dropout are used to mitigate this.

Thanks!

Any questions?