**1. Structure of an Artificial Neuron & Similarity to Biological Neuron**

An **artificial neuron** is a mathematical model inspired by biological neurons.

**Components of an Artificial Neuron:**

1. **Inputs (xix\_i)** – Features or data points.
2. **Weights (wiw\_i)** – Adjustable parameters that determine input importance.
3. **Bias (bb)** – Shifts the activation threshold.
4. **Summation Function (∑wixi+b\sum w\_i x\_i + b)** – Computes weighted sum.
5. **Activation Function** – Determines the neuron’s output (e.g., step, sigmoid).

**Similarity to Biological Neuron:**

* **Dendrites (Inputs in ANN)**: Receive signals.
* **Cell Body (Summation in ANN)**: Processes inputs.
* **Axon (Output in ANN)**: Passes information to other neurons.

**2. Types of Activation Functions**

1. **Step Function**
   * f(x)=1f(x) = 1 if x≥0x \geq 0, else 00.
   * Used in early perceptrons (binary classification).
2. **Sigmoid Function**
   * f(x)=11+e−xf(x) = \frac{1}{1 + e^{-x}}
   * Outputs values between (0,1).
   * Problem: **Vanishing gradient** in deep networks.
3. **ReLU (Rectified Linear Unit)**
   * f(x)=max⁡(0,x)f(x) = \max(0, x)
   * Avoids vanishing gradient.
   * Popular in deep learning.
4. **Tanh (Hyperbolic Tangent)**
   * f(x)=ex−e−xex+e−xf(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}
   * Outputs between (-1,1).
   * Stronger gradient than sigmoid.
5. **Softmax**
   * Converts logits into probabilities.
   * Used in **multi-class classification**.

**3. Rosenblatt’s Perceptron Model**

A **single-layer perceptron** is a binary classifier:

1. Compute **weighted sum**: S=w0+w1x1+w2x2S = w\_0 + w\_1 x\_1 + w\_2 x\_2.
2. Apply **step function**: Output 1 if S≥0S \geq 0, otherwise 0.
3. **Adjust weights** using perceptron learning rule: wi=wi+η(y−y^)xiw\_i = w\_i + \eta (y - \hat{y}) x\_i where η\eta is the learning rate.

**3.1 Classifying Data Using a Perceptron**

Given weights: w0=−1w\_0 = -1, w1=2w\_1 = 2, w2=1w\_2 = 1,  
Decision function: f(x)=−1+2x1+x2f(x) = -1 + 2x\_1 + x\_2.  
For each point (x1,x2)(x\_1, x\_2):

* (3,4): −1+2(3)+4=9-1 + 2(3) + 4 = 9 → Class 1
* (5,2): −1+2(5)+2=11-1 + 2(5) + 2 = 11 → Class 1
* (1,-3): −1+2(1)+(−3)=−2-1 + 2(1) + (-3) = -2 → Class 0
* (-8,-3): −1+2(−8)+(−3)=−20-1 + 2(-8) + (-3) = -20 → Class 0
* (-3,0): −1+2(−3)+0=−7-1 + 2(-3) + 0 = -7 → Class 0

**4. Multi-Layer Perceptron (MLP) & XOR Problem**

A **single perceptron fails** for XOR since it’s not linearly separable.

**Solution – MLP (2 hidden neurons + 1 output neuron)**

1. **Hidden layer transforms inputs**:
   * Two neurons learn **non-linear decision boundaries**.
2. **Output layer combines them**:
   * Uses a weighted sum + activation function.
3. **Backpropagation updates weights** iteratively.

**5. Artificial Neural Network (ANN) & Architectures**

An **ANN** is a collection of interconnected neurons.

**Architectures:**

1. **Feedforward ANN (FNN)** – No cycles, used in classification.
2. **Recurrent ANN (RNN)** – Has memory, used in time-series/NLP.
3. **Convolutional Neural Networks (CNNs)** – Used for image processing.
4. **Deep Neural Networks (DNNs)** – Multiple layers for complex tasks.

**6. Learning Process of ANN & Weight Assignment Challenges**

1. **Initialization** – Assign random weights.
2. **Forward Pass** – Compute outputs.
3. **Compute Error** – Compare output with expected.
4. **Backpropagation** – Adjust weights using gradient descent.

**Challenges:**

* If weights are **too small**, learning is slow.
* If weights are **too large**, convergence is unstable.
* **Solution**: Use adaptive methods (e.g., Adam, Xavier initialization).

**7. Backpropagation Algorithm & Limitations**

**Steps:**

1. Compute forward pass.
2. Compute loss.
3. Compute gradient (chain rule).
4. Update weights (gradient descent).

**Limitations:**

* Slow for deep networks (**vanishing gradient** issue).
* Requires large labeled datasets.

**8. Adjusting Interconnection Weights**

Weight update:

wi=wi−η∂L∂wiw\_i = w\_i - \eta \frac{\partial L}{\partial w\_i}

where LL is loss, η\eta is learning rate.

**9. Multi-Layer Network & Backpropagation Steps**

1. **Initialize weights** randomly.
2. **Compute forward pass**.
3. **Compute error**.
4. **Propagate gradients backward**.
5. **Update weights**.
6. **Repeat until convergence**.

**Why Multi-Layer Network?**

* A **single-layer perceptron cannot learn complex patterns**.
* **MLPs with hidden layers can model non-linear relationships**.

**10. Short Notes**

**1. Artificial Neuron**

* A mathematical model of a **biological neuron**.
* Consists of **inputs, weights, activation function, and output**.

**2. Multi-Layer Perceptron (MLP)**

* Consists of an **input layer, hidden layer(s), and an output layer**.
* Uses **backpropagation for training**.

**3. Deep Learning**

* **ANN with multiple hidden layers**.
* Used in **image recognition, NLP, reinforcement learning**.

**4. Learning Rate (η\eta)**

* Controls the **speed of weight updates** in training.
* Too high → **Diverges**, Too low → **Slow convergence**.

**11. Differences**

**1. Activation Function vs Threshold Function**

| **Feature** | **Activation Function** | **Threshold Function** |
| --- | --- | --- |
| Definition | Defines how neuron output is computed | A special case of activation function (Step function) |
| Range | Continuous or discrete | Binary (0 or 1) |
| Examples | Sigmoid, ReLU, Tanh | Step Function |

**2. Step Function vs Sigmoid Function**

| **Feature** | **Step Function** | **Sigmoid Function** |
| --- | --- | --- |
| Output | Binary (0/1) | Continuous (0 to 1) |
| Differentiable? | No | Yes |
| Used in | Perceptron | Neural Networks |

**3. Single Layer vs Multi-Layer Perceptron**

| **Feature** | **Single-Layer Perceptron** | **Multi-Layer Perceptron** |
| --- | --- | --- |
| Layers | 1 (Input → Output) | Multiple (Hidden layers) |
| Learnability | Only linearly separable data | Can learn non-linear functions |
| Solves XOR? | No | Yes |