**Module – 2 (Supervised Learning Network)**

* Perceptron Networks– Learning rule, Training and testing algorithm.
* Adaptive Linear Neuron– Architecture, Training and testing algorithm.
* Back propagation Network – Architecture, Training and testing algorithm.

3 State the testing algorithm used in Perceptron Network. (3)



4 List the stages involved in Backpropagation Network. 3)



3 Describe the role of bias in a perceptron model and demonstrate with an example how changing the bias value alters the final output. (3)



#### ✅ **What is Bias?**

* Bias acts like a **threshold shift**.
* It allows the activation function to be **moved left or right**, improving the flexibility of the model.
* Bias helps a neuron fire even when input is zero.

**Bias** controls the threshold of neuron activation and can dramatically change the output.

4 Draw the architecture of the Adaline Network. (3)



3 Explain the training algorithm of Perceptron Network. (3)



4 Explain the architecture and the delta rule used for weight updation in Adaline network (3)



3 Draw the architecture of Adaline Network. What is the training rule for Adaline network? (3)

4 Mention the applications of perceptron networks. (3)



3. Explain the Widrow-Hoff learning rule for supervised learning in neural networks with help of an example. Why is it sometimes called the LMS learning rule?

4. Implement one epoch of Adaline algorithm for AND logic function with binary inputs and bipolar outputs. Initial weights are w1=0.2, w2=0.1 and learning rate parameter η=0.2.



13. (a) Discuss the training algorithm and explain the weight updates in backpropagation networks. (10)

(b) Implement one epoch of Perceptron training algorithm for OR logic function with binary input and bipolar output. (4)



14. (a) Explain how synaptic weights are adapted iteration by iteration using error correction rule in Perceptron convergence algorithm for an OR gate with bipolar inputs and outputs. Initial weights are all zero and learning rate parameter η=0.1. (10)



(b) Explain Perceptron convergence theorem and discuss Perceptron algorithm based on XOR logic function. (4)



13 a) Draw the architecture and explain training algorithm of Back Propagation network. Write its testing algorithm. (10)



b) State the testing algorithm used in perceptron networks. (4)

14 a) Implement AND logical function using Perceptrons. (5)



b) Use Adaline to train ANDNOT function with bipolar inputs and targets. Perform 2 epochs of training. (9)



13 a) State the significance of error portions δk , δj and how they are calculated in BPN algorithm. Explain the architecture of BPN with proper labelling (8)



**Significance and Calculation of Error Portions (δk and δj)**

In the **Backpropagation Algorithm**, the **δ values (delta)** represent the **error signal** at each unit and are used to update weights during training.

**✅ δk – Error at Output Units**

* Represents how far the **actual output** yk​ is from the **target output** tk​.
* Used to adjust weights between **hidden and output layers**.

**Formula:**

δk=(tk−yk)⋅f′(netk)

Where:

* tk​ = target output
* yk​ = actual output
* f′(netk) = derivative of the activation function (e.g., sigmoid)

**✅ δj – Error at Hidden Units**

* Used to propagate error from output layer **back to hidden layer**.
* Helps in adjusting weights between **input and hidden layers**.

**Formula:**

δj=f′(netj)⋅∑δk⋅wjk ​

Where:

* f′(netj) = derivative of activation function at hidden unit
* wjk= weight connecting hidden unit jjj to output unit kkk

b) Implement the logic function OR with binary inputs and bipolar targets using Perceptron network up to two epochs. (6)



14 a) Explain the Adailne training algorithm for single output class (6)



b) Implement one epoch of Adaline algorithm for AND logic function with binary inputs and bipolar outputs. Initial weights are w1=0.2, w2=0.1 and learning rate parameter η=0.2. (8)



13 a) Design and implement OR function with bipolar inputs and targets using Adaline network? Find the total mean square error of 3 epochs. (8)



b) Find the weights required to perform the following classifications using a perceptron network: The vectors (2, 2, -2, -2) and (2, -2, 2, -2) belong to a class with a target value 1. The vectors (-2, -2, -2, 2) and (-2, -2, 2, 2) belong to a class with a target value -1. Assume a learning rate of 1 and initial weights of [0,0,0,0][0, 0, 0, 0][0,0,0,0]. (6)

14 a) Explain the architecture and training algorithm of the Back Propagation Network. Describe the various terminologies used in the algorithm (9)



b) Explain the training and testing algorithm for an Adaptive Linear Neuron. (5)



13 a) Implement AND function with binary inputs and bipolar targets using perceptron training algorithm. 8



b) Draw the architecture of Back propagation Network and explain the training algorithm. 6



14 a) What is Adaline? Draw the model of an Adaline Network. 4



b) Use Adaline to train OR function with bipolar inputs and targets. Perform 2 epochs of training. 10

1. Find the weights required to perform classification of patterns shown below using perceptron network. The patterns (1,1,-1) and ( 1, -1,-1) are belonging to the target class -1. The patterns (-1,1,1) and (-1,-1,1) are belonging to the target class +1. Assume suitable learning rate and initial weights.

2. Explain the architecture and training algorithm of Adaline network . Use Adaline nerwork to train NOR logic function with bipolar inputs and targets. Perform 2 epochs of training.



If ty; then, wij (new) = wij (old) + atjxi by(new) = b;(old) + at, wij (new) = wij (old)in bj(new) = by(old) else, we have Step 6: Train the network until there is no weight change. Otherwise, start again from Step 2.

**🧠 Perceptron Learning Rule – Summary**

**🔹 What is it?**

A supervised learning algorithm used for binary classification tasks where weights are updated based on the difference between **target output** and **actual output**.

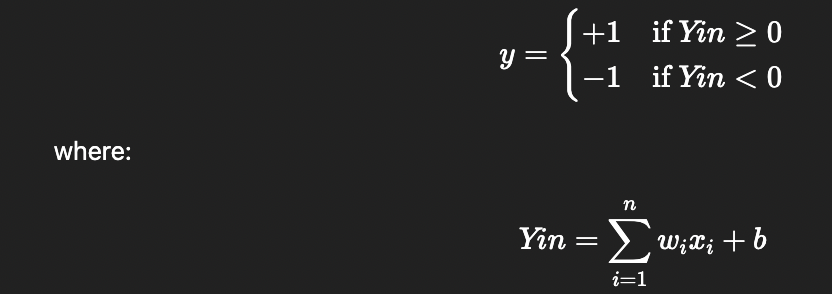
**🧾 Key Concepts:**

* **Learning Rate (α):**
  + Controls how much the weights are adjusted.
  + **Too small:** Converges slowly.
  + **Too large:** May overshoot, never converge.
  + Value typically lies in the range: 0<α<10 < \alpha < 10<α<1
* **Error Calculation:**

Error=t−y

where:

* + t is the target output
  + y is the actual output
* **Activation Function (Sign Function):**



**🔁 Perceptron Training Algorithm (Single Output Class)**

**Step-by-Step:**

| **Step** | **Action** |
| --- | --- |
| 0 | Initialize weights wiw\_iwi​, bias bbb, and learning rate α\alphaα. |
| 1 | Repeat Steps 2–6 until stopping condition (e.g., no weight updates). |
| 2 | For each training pair (x, t): |
| 3 | Input layer applies identity activation function. |
| 4 | Compute net input: Yin=∑wixi+b and apply activation function to get output y. |
| 5 | If y≠t , update:   else no update. |
| 6 | Continue until weights stop changing. |

**🧮 Perceptron Training Algorithm (Multiple Output Classes)**

Very similar to the single-output version, but:

* We maintain multiple weight vectors and biases, one for each class.
* The algorithm updates only those weights and biases for classes that had errors.

| **Step** | **Action** |
| --- | --- |
| 0 | Initialize weights wij​ and biases bj​ for all outputs. |
| 1–3 | As in single-output. |
| 4 | Compute: Yinj​=∑wij​xi​+bj​ for each output class j. |
| 5 | If yj≠tj ​, update:  ​ |
| 6 | Repeat until convergence. |

**🔍 Role of Critic Information**

* **Critic information** is the **feedback** (error signal or reward) used to guide weight updates.
* In **perceptron**, it's the **difference between desired output and actual output**:

Error=t−y

* This error determines **whether to update the weights** and by how much (scaled by α\alphaα).

**🧪 Perceptron Testing Algorithm (After Training)**

**🔹 Step-by-Step Process:**

| **Step** | **Action** |
| --- | --- |
| **Step 0** | Use the **final weights** obtained from training. |
| **Step 1** | For each new input vector **X**, do Steps 2–3. |
| **Step 2** | Set the activations of input units. Use the input vector X=[x1,x2,...,xn] |
| **Step 3** | Compute **net input** and **output** of the neuron:    Then apply the **activation function**: |

**⚙️ Adaline Training Algorithm (Steps)**

**Step 0:** Initialize weights and bias randomly (not zero).  
**Set**: learning rate α (e.g., 0.1 to 1.0)

**Step 1:** Repeat until stopping condition is met:

* **Step 2:** For each bipolar training pair s:t :
  + **Step 3:** Assign inputs xi=si
  + **Step 4:** Compute net input:

net=∑(wi⋅xi)+b

* + **Step 5:** Update weights & bias:

wi←wi+α⋅(t−net)⋅xi ​

b←b+α⋅(t−net)

**Step 6:** Calculate Es = ∑(t-net)2 , If maximum weight change (error Es) <= tolerance (Et)→ **STOP**, else continue.

**🧪 Adaline Testing Algorithm**

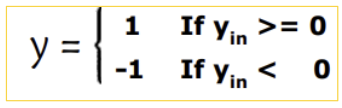
**Step 0:** Use trained weights and bias.

**Step 1:** For each input vector x:

* **Step 2:** Set input activations xi
* **Step 3:** Compute net input:

net=∑(wi⋅xi)+b

* **Step 4:** Apply activation function (identity or sign)



**🧠 Backpropagation – Training Process**

The **training** of a neural network using the backpropagation algorithm includes **three main stages**:

**1️⃣ Feedforward Phase:**

* Each **input unit** xi​ sends input signals to all **hidden units** Zj.
* Each **hidden unit** computes its activation and passes the signal to **output units** Yk​.
* Each **output unit** computes its output activation yk​, which is the network's response.

**2️⃣ Backpropagation of Error:**

* For each output unit Yk, calculate the error:

δk=(tk−yk)⋅f′(netk)

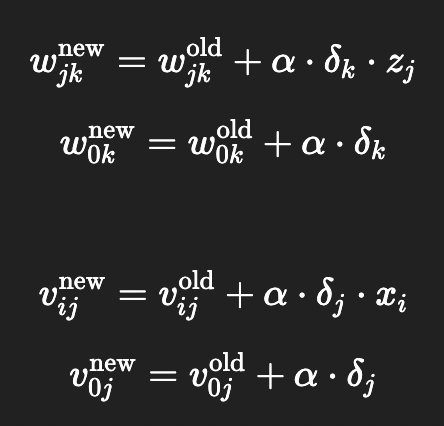
where f′ is the derivative of the activation function.

* Propagate δk ​ backward to compute δj for each hidden unit:

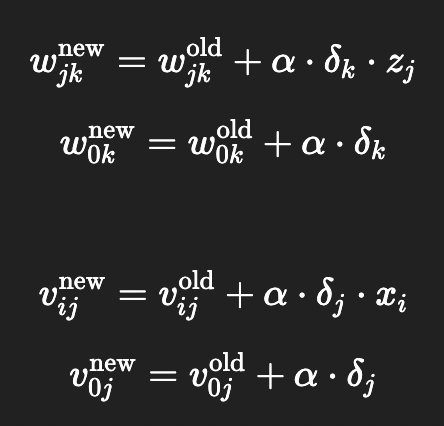
δj=f′(netj)⋅∑δk⋅wjk

**3️⃣ Weight & Bias Adjustment:**

* **Output layer weights update**:

****

* **Hidden layer weights update**:

****

### 🧪 ****Backpropagation Neural Network – Testing Algorithm (Summary)****

1. **Load trained weights and biases**  
   (from input to hidden & hidden to output layers)
2. **For each input vector** x:
   * **Compute hidden layer activations**:

netj=∑xi⋅vij+v0j, zj=f(netj)

* + **Compute output layer activations**:

netk=∑zj⋅wjk+w0k, yk=f(netk)

1. **Return predicted output** y=[y1,y2,...,ym] \