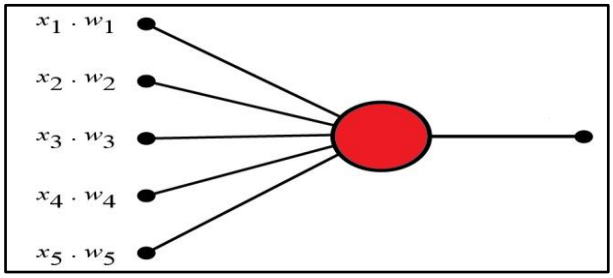
Unit III Supervised Learning Network

01. WHAT IS PERCEPTRON? DRAW AND EXPLAIN. (1 TIME)

1. What is a Perceptron?



It is an ***artificial neuron*** that forms the building blocks of machine learning.

2. Who Invented the Perceptron?

The perceptron was invented by ***Frank Rosenblatt (1928 – 1971),*** an ***American psychologist*** notable in the field of artificial intelligence. In ***1957,*** he developed a perceptron program on an ***IBM 704 computer*** at the ***Cornell Aeronautical Laboratory.***

3. How Does a Perceptron Work?

Scientists discovered that neurons in the ***brain receive input from our senses via electrical signals.*** These ***neurons use electrical signals to store information*** and ***make decisions based on previous input.*** Rosenblatt had the idea that perceptron's could simulate these brain principles, ***with the ability to learn and make decisions.***

The original perceptron was designed to take a binary input and produce one binary output (0 or 1). The idea was to ***use different weights to represent the importance of each input,*** and that the sum of the values should be greater than a threshold value before making a decision like ***yes or no (true or false) (0 or 1).***

4. Perceptron Example

Imagine a perceptron in your brain trying to decide if you should go to a concert. It considers various factors such as whether the **artist is good, the weather is good, a friend will come, food is served, and alcohol is served.** Each of these factors is assigned a weight to represent its importance.

Here's a table representing this:

Criteria	**Input (x)**	**Weight (w)**
Artists is Good	0 or 1	0.7
Weather is Good	0 or 1	0.6
Friend will Come	0 or 1	0.5

Criteria	Input (x)	Weight (w)
Food is Served	0 or 1	0.3
Alcohol is Served	0 or 1	0.4

5. The Perceptron Algorithm

Frank Rosenblatt suggested the following algorithm for a perceptron:

1. Set a threshold value (e.g., ***1.5***).
2. Multiply all inputs with their weights (e.g., x1 * w1 = 1 * 0.7 = 0.7).
3. Sum all the results (e.g., 0.7 + 0 + 0.5 + 0 + 0.4 = 1.6).
4. **Activate the output:** Return true if the sum > 1.5 (***"Yes, I will go to the Concert"***).

02. IMPLEMENT LOGICAL OR USING PERCEPTRON NETWORK. (1 TIME)

Perceptron Network:

A perceptron network, also known as a ***single-layer feed-forward network,*** is a type of artificial neural network with a simple architecture. It consists of three main units:

1. **Sensory Unit (Input Unit):** Receives input signals from the environment or other sources. Each ***input unit represents a feature or attribute*** of the input data.
2. **Associator Unit (Hidden Unit):** Processes the input signals and ***computes intermediate representations.*** In traditional perceptron's, the ***associator unit is fixed, and its connections to the sensory unit have predetermined weights.***
3. **Response Unit (Output Unit):** ***Produces the final output of the network*** based on the processed input signals. The ***result of output unit is determined by applying an activation function to the net input.***

Key Points about Perceptron Network:

- Sensory units are connected to associator units with ***fixed weights.***
- Response unit produces binary output based on a ***fixed threshold.***
- The output is determined by applying an activation function to the net input.
- ***Perceptron learning rule*** is used to ***adjust weights*** between the ***associator and response units*** based on training data.

Unit III Supervised Learning Network

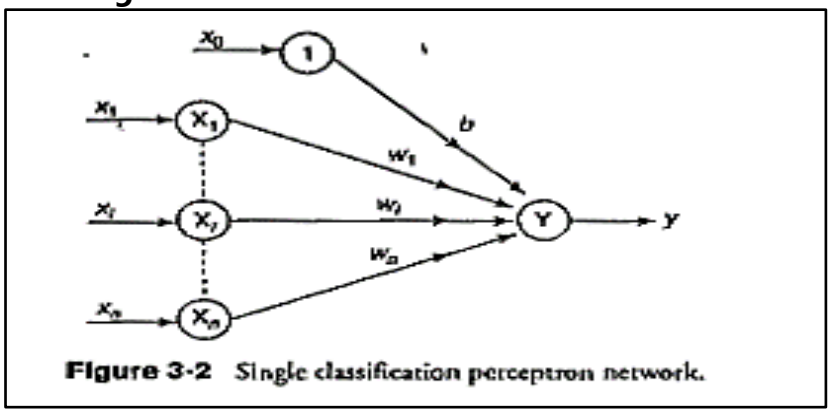


Figure 3-2 Single classification perceptron network.

Implementing Logical OR using Perceptron Network:

Logical OR is a binary function that returns true (1) if at least one of its inputs is true (1). Here's how to implement Logical OR using a perceptron network:

1. **Define the Problem:** We want the perceptron to output 1 if either input 1x1 or input 2x2 (or both) is 1, and output 0 otherwise.
2. **Initialize Weights:** Set the weights connecting the sensory unit to the associator unit as follows:
   - $w1=w2=1$ (representing positive influence from both inputs)
   - Set the bias $b=-0.5$ (adjusts the threshold for activation)
3. **Activation Function:** Use a binary step function as the activation function:
   - If the net input $\sum_{i=1}^{n} w_i x_i$ is greater than or equal to the threshold (0 in this case), output 1.
   - Otherwise, output 0.
4. **Training the Network:** Train the perceptron using training data that represents the input-output mapping of the Logical OR function.
5. **Testing:** Test the trained perceptron with different input combinations to verify if it correctly implements the Logical OR function.

By adjusting weights and biases through training, the perceptron can learn to correctly classify input patterns and produce the desired output, effectively implementing the Logical OR function.

03. IMPLEMENT LOGICAL 'AND' USING ADALINE TRAINING PROCESS. (1 TIME)

Adaptive Linear Neuron (Adaline):

1. **Theory:**
   - An Adaptive Linear Neuron (Adaline) is a type of ***single-linear-unit*** neural network with a ***linear activation function.***

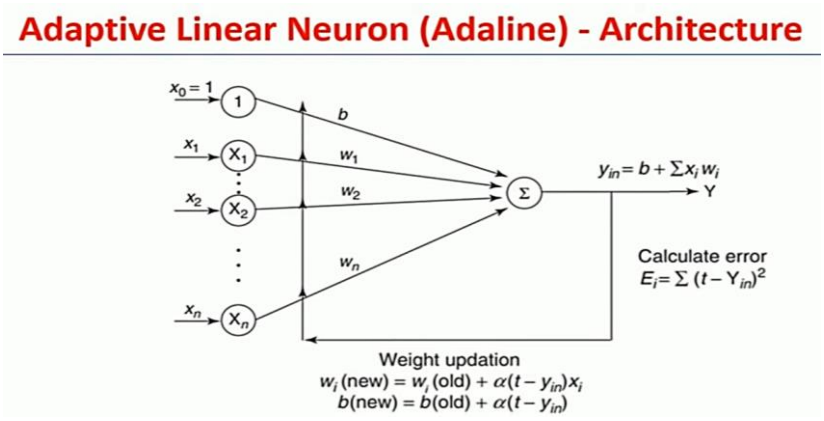
Unit III Supervised Learning Network

- In Adaline, the ***input-output relationship is linear,*** and it utilizes ***bipolar activation*** for both input signals and target outputs.
- The ***network consists of adjustable weights*** between the input and output units, where the bias acts as an adjustable weight connected to a unit with constant activation.
- Adaline aims to ***minimize the mean-squared error*** between the network's output and the target value, typically trained using the ***delta rule*** or the ***least mean square (LMS) rule.***

2. Delta Rule for Single Output Unit:

- The ***delta rule, also known as the Widrow-Hoff rule or least mean square (LMS) rule,*** updates the weights to minimize the difference between the network's output and the target value.
- It adjusts the weights based on the difference between the actual output and the target output multiplied by the input activations and the learning rate.
- The formula: ***wi(new)=wi(old) + α (t−yin)xi,*** where ***wi(new)*** is the weight change, **α** is the learning rate, ***t*** is the target output, ***yin*** is the actual output, and ***xi*** is input unit.

3. Architecture:



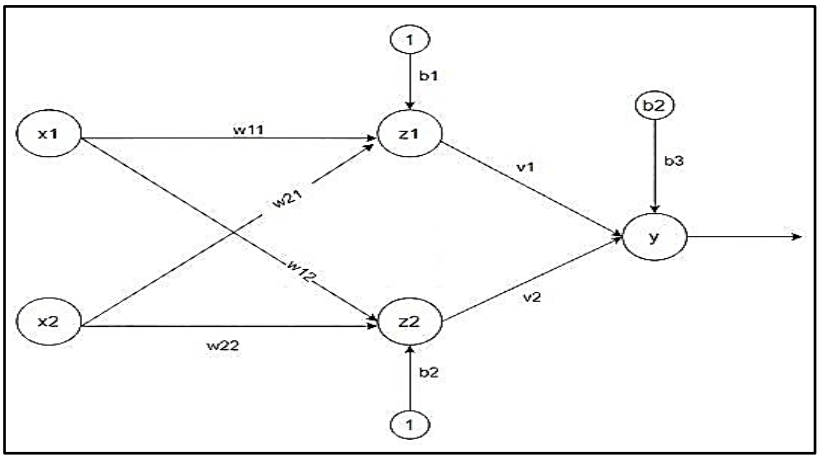
- Adaline consists of a ***single-linear-unit*** neuron that receives inputs from multiple units, including a bias unit.
- The model includes trainable weights, where inputs are either +1 or -1, and weights are initially assigned randomly.
- The net input calculated is applied to a ***quantizer transfer function*** that produces an output of either +1 or -1.

Implementing Logical AND using Adaline:

1. **Define the input patterns for the Logical AND function:** (1,1), (1,-1), (-1,1), and (-1,-1).
2. **Set the target outputs for the AND function accordingly:** 1 for (1,1) and -1 for the other input patterns.

3. Train the Adaline network using the training algorithm described above, adjusting the weights to minimize the error between the actual output and the target output.
4. Once trained, test the Adaline network by applying the testing algorithm to classify the input patterns and observe the output to verify that it behaves like the Logical AND function.

04. EXPLAIN MADALINE TRAINING PROCESS. (3 TIMES)



Architecture of Madaline NN

Madaline, short for ***Multiple Adaptive Linear Neurons,*** is a neural network model ***comprising multiple Adaline units with a single output unit.*** Here's a breakdown of the Madaline training process:

1. **Initialization:**
   - Weights are initialized for the connections between the ***input layer and the hidden Adaline layer.***
   - Initial small random values are set for Adaline weights, and a learning rate ***α*** is chosen.
2. **Training Algorithm:**
   - The training process iterates until a stopping condition is met.
   - For each training pair ***s : t,*** where ***s*** is the input pattern and ***t*** is the target output:
     - Input layer units are activated by setting the input values ***xi.***
     - Net input to each hidden Adaline unit ***Zin*** is calculated using the weights ***Wij*** and bias ***bj.***
     - The output ***Zj*** of each hidden unit is obtained by ***applying the activation function f(Zin).***
     - The net input to the output unit ***Yin*** is calculated using the ***fixed weights Vi*** and ***bias b0.***
     - The output ***Y*** of the net is obtained by applying the activation function ***f(Yin).***

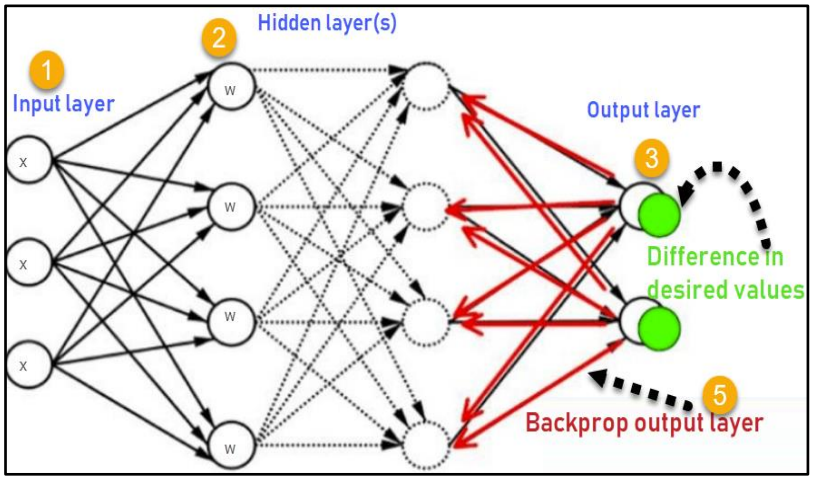
- The error between the ***actual output Y*** and the ***target output t*** is calculated.
- Weights are updated based on the error:
  - If $t=Y$, no weight updation is required.
  - If $t \neq Y$ and $t=+1$, weights on the hidden units ***Zj*** with net input closest to zero are updated.
  - If $t \neq Y$ and $t=-1$, weights on units ***Zk*** with positive net input are updated.

3. **Stopping Condition:**
   - The training process continues until a stopping condition is met, which could be:
     - No weight changes occur.
     - Weights reach a satisfactory level.
     - A specified maximum number of iterations of weight updates have been performed.
4. **Functionality:**
   - Madalines can be configured to perform logic functions using the weights on the output unit.
   - For example, if only two hidden units are present, the **"majority vote rule"** function may be used.

The Madaline training process involves adjusting the weights between the input layer and the hidden Adaline layer to minimize errors in the output unit. It is an iterative process that continues until the network achieves satisfactory performance or meets a stopping condition.

05. ILLUSTRATE BACK PROPAGATION NETWORK TRAINING WITH EQUATIONS. (4 TIMES)

Backpropagation is a ***key algorithm used to train neural networks.*** It's based on the ***chain rule of calculus*** and allows us to adjust the weights of the network to minimize the error between the predicted output and the actual output. Below, I'll illustrate the backpropagation algorithm with equations.



Unit III Supervised Learning Network

Let's consider a simple feedforward neural network with one hidden layer. We'll denote:

- *xi* as the input to the network,
- *yi* as the output of the network,
- *wij* as the weight connecting the *i*-th neuron in the input layer to the *j*-th neuron in the hidden layer,
- *vjk* as the weight connecting the *j*-th neuron in the hidden layer to the *k*-th neuron in the output layer,
- *zj* as the output of the hidden layer neuron *j*,
- *f* as the activation function of the hidden layer, and
- *g* as the activation function of the output layer.

Here is a detailed explanation of the steps involved in the backpropagation algorithm, which is a fundamental technique used in training neural networks. The steps are as follows:

1. **Forward Pass:**
    a. Compute the outputs of the hidden layer neurons: zj = f(∑i wijxi)
    b. Compute the outputs of the output layer neurons: yk = g(∑j vjkzj)
2. **Compute the Error:**
    a. Calculate the error between the predicted output yk and the actual output dk: E = 1/2 ∑k(dk - yk)^2
3. **Backward Pass:**
    a. Compute the gradient of the error with respect to the output layer weights: ∂E/∂vjk = -(dk - yk) · g'(inputk) · zj
    b. Compute the gradient of the error with respect to the hidden layer weights: ∂E/∂wij = -∑k(dk - yk) · g'(inputk) · vjk · f'(inputj) · xi
4. **Update Weights:**
    a. Update the weights using the gradients and a learning rate α: vjk ← vjk - α · ∂E/∂vjk wij ← wij - α · ∂E/∂wij
    b. Repeat steps 1-4 for multiple iterations until the error converges to a minimum.

It also explains that ***g'(inputk)*** and ***f'(inputj***) denote the derivatives of the activation functions of the output and hidden layers, respectively, evaluated at their respective inputs. This process is iterated through many examples in a dataset, updating the weights each time, until the network learns to approximate the desired function.

Unit III Supervised Learning Network

06. EXPLAIN TRAINING PROCESS OF RADIAL BASIS FUNCTION NETWORK. (3 TIMES)

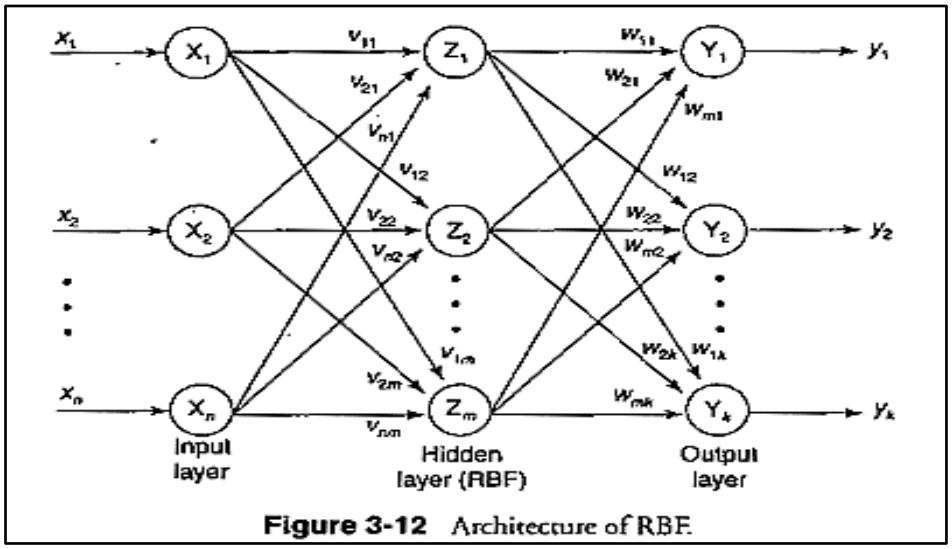


Figure 3-12 Architecture of RBF

1. **Introduction to RBFN:**
   - The Radial Basis Function Network (RBFN) is a neural network architecture developed by ***M.J.D. Powell.***
   - It utilizes nonlinearities such as ***sigmoidal and Gaussian kernel functions.***
   - The network consists of two layers: an input layer and a hidden layer (RBF), followed by an output layer.
2. **Architecture:**
   - The RBFN architecture comprises an input layer, a hidden layer consisting of RBF nodes, and an output layer.
   - The output nodes in both layers form a linear combination of kernel functions computed by the RBF nodes.
3. **Training Algorithm:**
   - **Step 0:** Initialize the weights to small random values.
   - **Step 1:** Perform subsequent steps until the stopping condition is met.
   - **Step 2:** For each input:
     - **Step 3:** Transmit input signals from each input unit to the next hidden layer unit.
     - **Step 4:** Calculate the radial basis function.
     - **Step 5:** Select the centres for the radial basis functions from the input vectors.
   - **Step 6:** Calculate the output from the hidden layer unit using the Gaussian function:

$$v_i(x) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right)$$

- *xi*: centre of the RBF unit for input variables
- *σi*: width of the RBF unit
- *xj*: the j-th variable of the input pattern

- **Step 7:** Calculate the output of the neural network using a linear combination of RBF unit outputs:
  - $$Y_n = \sum_{i=1}^{k} W_{im} v_i(x_n) + w_o$$
  - *k*: number of hidden layer nodes (RBF functions)
  - *Wim*: weight between i-th RBF unit and m-th output node
  - *wo*: biasing term at the output node
- **Step 8:** Calculate the error and test for the stopping condition, which may be based on the number of epochs or a specified threshold for weight changes.

The training process for RBFN involves selecting centres for radial basis functions, calculating their outputs based on input patterns, and adjusting weights iteratively until a stopping condition is met. This process combines unsupervised learning in the hidden layer with supervised learning in the output layer to optimize the network's performance.