SRINIVAS UNIVERSITY
INSTITUTE OF ENGINEERING & TECHNOLOGY

(SUBJECT: ARTIFICIAL NEURAL NETWORK)

(SUBJECTCODE:24SBT113)

A Individual Task on

"Build a Perceptron model solve binary classification and apply perceptron learning law"

Submitted in the partial fulfillment of the requirements for the fourth semester

BACHELOR OF TECHNOLOGY IN AIML

Submitted By,

SUDESH(01SU24AI104)

UNDER THE GUIDANCE OF

Prof. Mahesh Kumar V B

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

SRINIVAS UNIVERSITY INSTITUTE OF ENGINEERING & TECHNOLOGY
MUKKA, MANGALURU-574146

2025-26

SRINIVAS UNIVERSITY

INSTITUTEOF ENGINEERING & TECHNOLOGY

Department of ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

CERTIFICATE

This is to certify that **SUDESH(01SU24AI104)** has satisfactorily completed the assessment (Individual-Task – Module 1) in **"ARTIFICIAL NEURAL NETWORK "** prescribed by the Srinivas University for the 4st semester B. Tech course during the year **2025-26**.

MARKS AWARDED

<u>Staff In charge</u>

Name: Prof. Mahesh Kumar V B

Assistant Professor, Department Of AIML

TABLE OF CONTENTS

1. Introduction:-

One of the earliest and most influential models developed to solve such problems is the **Perceptron**, introduced by Frank Rosen blatt in 1958. The Perceptron represents a computational abstraction of a biological neuron, designed to mimic how the human brain processes information. It takes multiple inputs, applies weights to them, sums the results, and passes the sum through an activation function to produce an output. This output is then compared with the desired target, and the model adjusts its weights accordingly. Through repeated iterations, the Perceptron gradually learns to classify inputs correctly.

The model will be implemented in a step-by-step manner, either on paper or in a spreadsheet, to make the learning process transparent and easy to follow. We will apply the **Perceptron learning law**, which governs how weights and biases are updated based on errors, and demonstrate how the model converges to a solution. By working through a concrete example, such as the logical AND function, we will illustrate the mechanics of the algorithm and show how the decision boundary evolves during training.

2. Biological Inspiration:-

 Inputs**:- (x1,x2,…..,xn)** correspond to signals received by dendrites.

- **Weights (w1,w2,….,wn)** represent the strength of synaptic connections, determining the influence of each input.
- The **summation unit** parallel the soma, aggregating all weighted inputs.
- The **activation function** mimics the firing threshold of a neuron, producing an output only if the aggregated signal is strong enough.
- The **output** corresponds to the axon’s signal, which can be passed to other neurons or used as a final decision

3. Perceptron Architecture:-

3.1 Components of the Perceptron:-

1. **Input Layer:-**
   - The Perceptron receives multiple inputs, denoted as x1,x2,….,xn.
   - Each input represents a feature of the dataset. For example, in a binary classification problem distinguishing between two types of objects, inputs could represent measurable attributes such as size, weight, or colour intensity.

2. **Weights:-**
   - Each input is associated with a weight (w1,w2,…. ,wn).
   - Weights determine the importance of each input in the decision-making process. A higher weight means the corresponding input has a stronger influence on the output.

Initially, weights are assigned randomly or set to zero, and they are adjusted during training using the Perceptron learning law.

3. **Bias:-**

- The bias term (b) allows the decision boundary to shift away from the origin.
- Without bias, the Perceptron could only classify data with boundaries passing through the origin, which is often too restrictive.

4. **Summation Function:-**

- The Perceptron computes a weighted sum of inputs plus the bias:
- This summation represents the total input signal received by the model.

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \ , \ y = f(z)$$

5. **Activation Function:-**

- The activation function determines the output of the Perceptron based on the summation.

The simplest activation function is the **step function**:

$$y = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

6. **Output:-**

- The final output (y) is either 0 or 1, representing the predicted class.
- This output can be used directly for classification or passed to other layers in more complex networks.

3.2 Decision Boundary:-

The Perceptron defines a **linear decision boundary** in the input space. For two inputs (x1,x2), the boundary is given by:

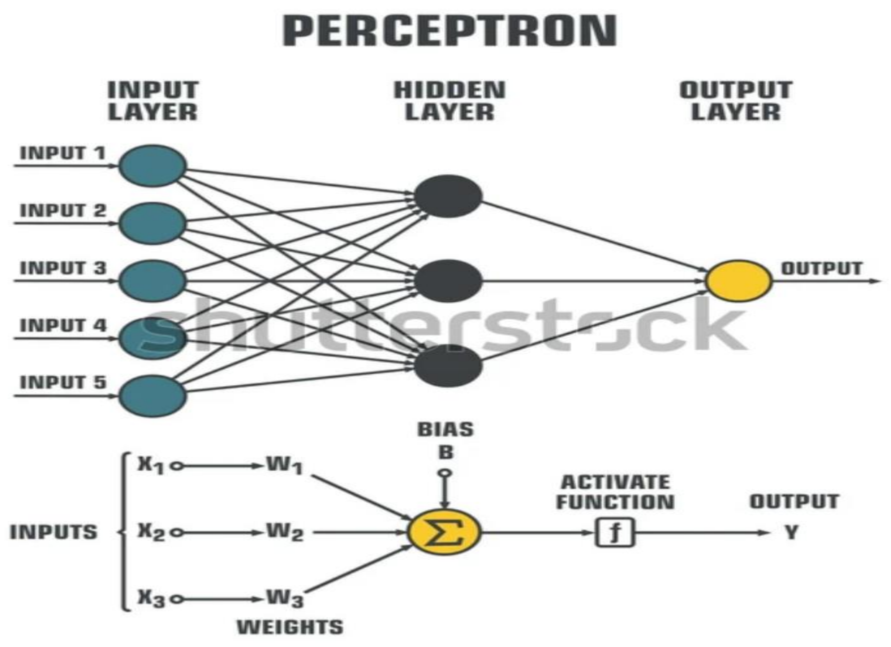
w_1x_1+w_2x_2+b=0

This equation represents a straight line that separates the input space into two regions:

- One region where the Perceptron outputs 1 (Class A).
- Another region where it outputs 0 (Class B).

The orientation and position of this line depend on the values of the weights and bias. During training, the Perceptron adjusts these parameters to correctly classify the training data.

3.3 Diagrammatic Representation:



4. Perceptron Learning Law:

4.1: Mathematical Formulation

The update rules are defined as follows:

- **Weight update:**
- $w_i^{new} = w_i^{old} + \eta(t\text{-}y)x_i$

- **Bias update:**

$$b^{new} = b^{old} + \eta(t\text{-}y)$$

Where:

- wi = weight associated with input xi
- b = bias term
- eta = learning rate (a small positive constant controlling the step size of updates)
- t = target output (true label, 0 or 1)
- y = predicted output (0 or 1)

Learning Process

The Perceptron learning process can be summarized in the following steps:

1. **Initialization:-**
   - Assign initial values to weights and bias (often zeros or small random numbers).
2. **Prediction:-**
   - Compute the weighted sum of inputs plus bias.
   - Apply the activation function to produce output y.
3. **Error Calculation:-**
   - Compare predicted output y with target t.
   - Compute error as (t-y).
4. **Update:-**
   - Adjust weights and bias using the learning law.
5. **Iteration:-**

- Continue for multiple epochs until convergence or until a stopping criterion is met.
- 4.5 Example Application
- Consider a Perceptron with two inputs trained to learn the logical AND function:

Training example: x_1=1,x_2=1,t=1

Initial weights: w_1=0,w_2=0,b=0

- Prediction: y=0 (since weighted sum = 0)
- Error: t-y=1-0=1

Update:

w_1=0+1.1 .1=1

w_2=0+1.1. 1=1

b=0+1.1=1

5. Dataset:

The truth table for the AND function is shown below:

5.1.Training Process:

The Perceptron learning law updates weights and bias iteratively. We will go through the data-set multiple times (epochs) until the model converges.

1. **Input (0,0), Target = 0**
   - Weighted sum: z=0.0+0. 0+0=0
   - Prediction: y=1 (since z=0)
   - Error: t-y=0-1=-1
   - Update:

w_1=0+(-1).0=0

w_2=0+(-1). 0=0

b=0+(-1)=-1

 2.**Input (0,1), Target = 0**

- Weighted sum: z=0.0+0. 1-1=-1
- Prediction: y=0 → Correct → No update.

3.Input (1,0), Target = 0

- Weighted sum: z=0.1+0. 0-1=-1
- Prediction: y=0 → Correct → No update.

4.Input (1,1), Target = 1

- Weighted sum: z=0. 1+0 . 1-1=-1
- Prediction: y=0 → Error = 1
- Update:

w_1=0+1. 1=1

w_2=0+1. 1=1

b=-1+1=0

5.2. Convergence

After several epochs, the Perceptron converges to a solution that correctly classifies all inputs:

- Final weights: w_1=1,w_2=1
- Final bias: b=-1

Decision boundary:

x_1+x_2-1=0

This line separates the input space into two regions:

- Region where output = 1 (only when both inputs are 1).
- Region where output = 0 (all other cases).

6.Spreadsheet Implementation:-

6.1 Structure of the Spreadsheet

The spreadsheet can be organized into columns representing each stage of the Perceptron computation:

Column	**Description**
Input 1 (x_1)	**First feature of the dataset**
Input 2 (x_2)	**The second feature of the dataset**
Target (t)	**Desired output (0 or 1)**
Weighted Sum (z)	**Computed as $w_1 x_1 + w_2 x_2 + b$**
Prediction ($\hat{y}$)	**Output after applying the step function**
Error ($t - \hat{y}$)	**Difference between target and prediction**
Weight Update ($\Delta w_1, \Delta w_2$)	**Adjustment based on the learning law**
Bias Update (Δb)	**Adjustment to bias**

6.2 Example set up and functions:-

Input (x_1, x_2)	**Target (t)**
(0, 0)	**0**
(0, 1)	**0**

Input (x_1, x_2)	**Target (t)**
(1, 0)	**0**
(1, 1)	**1**

Initial values:

- w_1=0,w_2=0,b=0
- Learning rate \eta =1

6.3 Spreadsheet Formulas:-

1. **Weighted Sum (z):**

   z= (w1 * Input1) + (w2 * Input2) + b

2. **Prediction (y):**

   y =IF(z>=0,1,0)

3. **Error (t – y):**

   t-y = Target – Prediction

4. **Weight Updates:**

   Δw1 = η * Error * Input1

   Δw2 = η * Error * Input2

5. **Bias Update:**

   Δb = η * Error

6. **New Weights and Bias:**

   w1(new) = w1(old) + Δw1

   w2(new) = w2(old) + Δw2

   b(new) = b(old) + Δb

6.4 Advantages of Spreadsheet Implementation:-

- Transparency: Each step of the learning process is visible and traceable.

- Interactivity: Users can manually adjust weights, bias, or learning rate to observe effects.
- Visualization: Charts and plots make abstract concepts concrete.
- Accessibility: No programming knowledge is required; anyone familiar with spreadsheets can implement the Perceptron**.**

7. Visualization:-

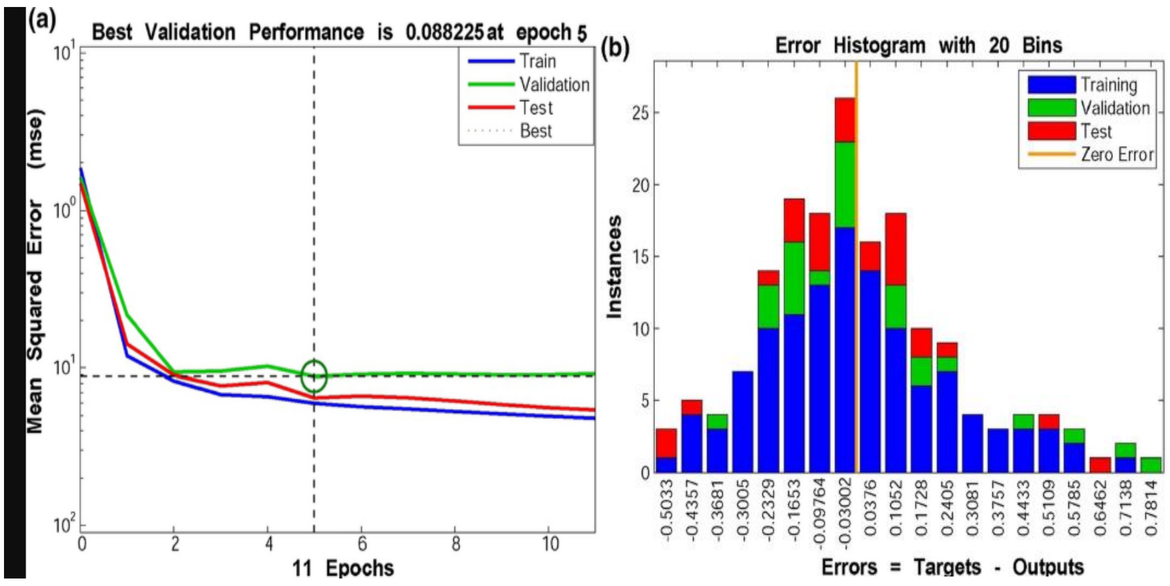
Visualization plays a crucial role in understanding how the Perceptron model learns and adapts during training. By graphically representing the data points, decision boundaries, and error trends, we can observe the dynamics of the learning process in a way that is far more intuitive than numerical tables alone. This section highlights the key visualizations that can be used to illustrate the Perceptron operation.

7.1 Advantages of Visualization

- **Clarity:** Visuals make abstract mathematical concepts easier to grasp.
- **Transparency:** Graphs show how the decision boundary evolves during training.
- **Insight:** Error plots reveal convergence behaviour and highlight potential issues.
- **Accessibility:** Visualizations make the Perceptron understandable even to non-technical audiences.

7.2 Error vs. Epoch Graph

- Another useful visualisation is a line chart showing the number of misclassifications per epoch. At the beginning of training, the Perceptron may misclassify several points. As weights and bias are updated, the number of errors decreases. Eventually, the error count reaches zero, indicating convergence.
- This graph demonstrates the effectiveness of the Perceptron learning law and provides insight into the speed of convergence, which depends on factors such as the learning rate and initial weights.



8. Applications of Perceptron:-

8.1 Pattern Recognition

The Perceptron was originally designed for image recognition tasks. Early experiments by Rosenblatt demonstrated that the model could distinguish between simple visual patterns, such as shapes or letters. By mapping pixel values as inputs, the Perceptron could classify images into categories, laying the groundwork for modern computer vision.

8.2 Document and Email Classification

Binary classification tasks such as **spam detection** can be modeled using a Perceptron. Inputs may represent features like the presence of certain keywords, frequency of links, or sender information. The Perceptron learns to separate spam from legitimate emails, providing one of the earliest approaches to automated filtering.

8.3 Medical Diagnosis

In healthcare, Perceptron have been used to classify patients into categories such as “healthy” vs. “diseased” based on diagnostic features. For example, inputs could include blood pressure, cholesterol levels, or test results. While modern systems use more advanced models, the Perceptron demonstrated the feasibility of machine learning in medical decision support.

8.4 Industrial Quality Control

Manufacturing industries often require classification of products into “defective” vs. “non-defective.” A Perceptron can be trained using measurable attributes such as weight, dimensions, or sensor readings. This application highlights the model’s utility in automating inspection processes.

8.5 Early Speech and Signal Processing

Perceptron have also been applied to basic signal classification tasks, such as distinguishing between different tones or speech patterns. Inputs derived from frequency or amplitude features allow the model to separate signals into categories, forming the basis for later advances in speech recognition.

8.6 Educational Use

Today, the Perceptron is widely used as a teaching tool. Its simplicity makes it ideal for introducing students to neural networks, machine learning, and the concept of iterative weight adjustment. Implementations on paper or spreadsheets provide hands-on experience with the mechanics of learning algorithms.

9. Strengths and Limitations:

9.1 Strengths:

1. **Simplicity and Transparency:-**
   - The Perceptron is easy to understand and implement. Its architecture and learning law are straightforward, making it an ideal entry point for students and researchers exploring neural networks.
2. **Efficiency for Linearly Separable Problems:-**
   - When data can be separated by a straight line (or hyperplane in higher dimensions), the Perceptron converges reliably to a solution. This makes it effective for tasks such as basic logical functions (AND, OR) and simple classification problems.
3. **Historical Importance:-**
   - As the first learning algorithm capable of adjusting weights automatically, the Perceptron laid the foundation for modern machine learning. Concepts such as weights, bias, activation functions, and iterative training originated here and remain central to advanced models.

4. **Educational Value:-**

- The Perceptron is widely used as a teaching tool. Its step-by-step learning process can be demonstrated on paper or in spreadsheets, providing hands-on experience with the mechanics of supervised learning.

9.2 Limitations

1. **Restricted to Linear Boundaries:-**
   - The Perceptron can only solve problems where classes are linearly separable. It fails on nonlinear problems such as the XOR function, where no straight line can separate the classes.
2. **Binary Output Only:-**
   - The classic Perceptron uses a step activation function, producing only binary outputs (0 or 1). This restricts its ability to handle more complex tasks requiring continuous or probabilistic outputs.
3. **Sensitivity to Learning Rate and Initialization;-**
   - The speed of convergence depends on the choice of learning rate and initial weights. Poor choices may lead to slow learning or oscillations in weight updates.

10. Extensions:-

10.1 Multi-Layer Perceptron (MLP)

The most significant extension of the Perceptron is the **Multi-Layer Perceptron (MLP)**. Unlike the single-layer Perceptron, which can only form linear decision boundaries, the MLP introduces one or more **hidden layers** between the input and output. Each hidden layer consists of multiple neurons, each applying a nonlinear activation function.

Key Features:-

- Hidden layers allow the network to learn complex, nonlinear relationships.
- Nonlinear activation functions (e.g., sigmoid, tan h, RELU) enable the model to approximate any continuous function.
- MLPs can solve problems like the XOR function, which a single Perceptron cannot handle.

10.2 Advanced Activation Functions:-

The original Perceptron used a step function, producing binary outputs. Modern extensions employ smoother, differentiable activation functions:

- **Sigmoid Function:** Maps inputs to values between 0 and 1, useful for probabilistic interpretation.
- **Hyperbolic Tangent (tan h):** Outputs between -1 and 1, often used for centred data.
- **Rectified Linear Unit (RELU):** Outputs zero for negative inputs and linear values for positive inputs, enabling faster training and reducing vanishing gradient problems.

10.3 Learning Algorithms Beyond the Perceptron Rule:-

The Perceptron learning law is effective for linearly separable problems but limited otherwise. Extensions introduced more general optimisation techniques:

- **Gradient Descent:** Minimizes error by adjusting weights in the direction of steepest descent of the loss function.
- **Back-propagation:** A systematic method for computing gradients in multi-layer networks, enabling efficient training of deep architectures.

11. Conclusion:-

a Perceptron model was constructed and trained to solve a binary classification problem using the Perceptron learning rule. The experiment demonstrated how a single-layer neural network can learn from labeled data by adjusting its weights and bias based on classification errors. During the training process, the model iteratively updated its parameters whenever the predicted output differed from the actual target value. This error-correction mechanism allowed the model to gradually improve its accuracy over multiple iterations.

The results show that the Perceptron successfully learns to classify linearly separable data such as logical functions like AND and OR. As training progresses, the number of classification errors decreases until the model converges to a stable solution. This confirms the effectiveness of the Perceptron learning law in finding a decision boundary (a straight line or hyperplane) that separates two classes.

12. References:-

- Rosenblatt, F. (1958). *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*.
- Hay kin, S. (1999). *Neural Networks: A Comprehensive Foundation*.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*