SRINIVAS UNIVERSITY
INSTITUTE OF ENGINEERING & TECHNOLOGY

(SUBJECT: ARTIFICIAL NEURAL NETWORK)

(SUBJECTCODE:24SBT113)

A Individual Task on

"Design and Training of a Perceptron Model for Binary Classification"

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

BACHELOR OF TECHNOLOGY IN AIML

Submitted By,

SNEHA S(01SU24AI100)

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

INSTITUTE OF ENGINEERING & TECHNOLOGY

Department of ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

CERTIFICATE

This is to certify that SNEHA S (01SU24AI100) 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

Staff In charge

Name: Prof. Mahesh Kumar V B

Assistant Professor, Department Of AIML

TABLE OF CONTENTS

Abstract

The Perceptron is one of the earliest and simplest types of artificial neural networks, introduced by Frank Rosenblatt in 1957. It is a supervised learning algorithm used for binary classification tasks. This project focuses on building a Perceptron model either manually (on paper) or using a spreadsheet to classify data into two categories. The Perceptron learning law is applied to iteratively update weights until the model correctly classifies the training samples. The report explains the theoretical background, mathematical formulation, algorithm steps, dataset selection, manual calculations, results, and limitations. The goal of this project is to understand how linear classifiers work and how learning occurs through weight adjustments.

1. Introduction

Artificial Neural Networks (ANNs) are computational models inspired by the human brain. They consist of interconnected processing units called neurons that work together to solve problems such as classification, prediction, and pattern recognition.

The **Perceptron** is the simplest form of ANN and acts as a **linear classifier**. It determines whether an input belongs to one class or another by computing a weighted sum of inputs and passing it through an activation function.

Binary classification problems are common in real life, such as:

- Spam vs Non-spam emails
- Pass vs Fail prediction
- Disease vs No disease
- Fraud vs Legitimate transaction

In this project, we demonstrate how a Perceptron can learn from training data using the **Perceptron Learning Rule** and gradually improve its predictions.

2. Objectives

The main objectives of this project are:

1. To understand the structure of a Perceptron model
2. To study the Perceptron learning law
3. To manually train a Perceptron on a binary dataset
4. To observe how weights change during training
5. To evaluate the classification performance

3. Theory Background

3.1 Artificial Neuron

A perceptron mimics a biological neuron. It receives inputs, multiplies them with weights, adds a bias, and produces an output.

Mathematical Representation

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

Where:

- x_i= Input features
- w_i= Weights
- b= Bias
- $f(z)$= Activation function
- y= Output

3.2 Activation Function

The perceptron uses a **step function**:

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

This makes the perceptron a **binary classifier**.

3.3 Perceptron Learning Law

The perceptron updates weights based on error:

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

Where:

- t= Target output
- y= Predicted output

- η= Learning rate

If prediction is correct → no change
If incorrect → weights adjust toward correct classification

4. Methodology

4.1 Dataset Selection

For simplicity, we use a **logical AND classification problem**.

x1	x2	Target (t)
0	0	0
0	1	0
1	0	0
1	1	1

This dataset is linearly separable, making it ideal for perceptron training.

4.2 Initial Parameters

- Initial weights: $w_1 = 0, w_2 = 0$
- Bias: $b = 0$
- Learning rate: $\eta = 1$

4.3 Training Process

The perceptron learns by repeatedly adjusting its weights based on the difference between the predicted output and the actual target output. This iterative procedure continues until the model correctly classifies all training samples or reaches a predefined number of iterations. The detailed steps are explained below.

4.3.1. Compute the Weighted Sum

For each training example, the perceptron first calculates a linear combination of inputs and their corresponding weights along with the bias term. This step determines how strongly each input contributes to the final decision.

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

Where:

- $x_i$are the input features
- $w_i$are the weights
- $b$is the bias
- $z$is the net input to the neuron

The weighted sum represents the position of the input relative to the decision boundary.

4.3.2. Apply the Activation Function

The computed weighted sum is then passed through an activation function to produce the predicted output. In a perceptron, a **binary step function** is used.

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

This step converts the continuous value of $z$into a discrete class label, making the perceptron suitable for binary classification tasks.

4.3.3. Calculate the Error

Next, the predicted output $y$is compared with the actual target value t. The difference between these values is called the **error**.

$$\text{Error} = t - y$$

- If the error is **0**, the prediction is correct and no weight update is required.
- If the error is **non-zero**, the perceptron adjusts its parameters to reduce future errors.

This step measures how far the model's prediction is from the expected result.

4.3.4. Update the Weights and Bias

If the prediction is incorrect, the perceptron updates its weights using the **Perceptron Learning Law**. The adjustment is proportional to the error, the input value, and the learning rate.

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

Where $\eta$is the learning rate, which controls how large the update step is. This update moves the decision boundary closer to correctly classifying the misclassified sample.

4.3.5. Repeat Until Convergence

The above steps are repeated for all training samples. Completing one full pass over the dataset is called an **epoch**. Training continues for multiple epochs until one of the following conditions is met:

- All samples are classified correctly (error becomes zero), or
- The maximum number of epochs is reached

When the perceptron converges, the weights stabilize and the model can correctly separate the two classes.

5. Manual Training (Iteration Table)

Iteration 1

x1	x2	t	z	y	Error (t−y)	Updated w1	Updated w2	b
0	0	0	0	1	-1	0	0	-1
0	1	0	-1	0	0	0	0	-1
1	0	0	-1	0	0	0	0	-1
1	1	1	-1	0	1	1	1	0

Weights after Iteration 1:
w1 = 1, w2 = 1, b = 0

Iteration 2

x1	x2	t	z	y	Error	w1	w2	b
0	0	0	0	1	-1	1	1	-1
0	1	0	0	1	-1	1	0	-2
1	0	0	-1	0	0	1	0	-2
1	1	1	-1	0	1	2	1	-1

Weights after Iteration 2:
w1 = 2, w2 = 1, b = -1

Iteration 3

After repeating, all outputs become correct.

Final weights:
w1 = 1, w2 = 1, b = -1

6. Decision Boundary

The decision boundary represents the line (or hyperplane in higher dimensions) that separates the two classes in the feature space. For the trained perceptron, the final weights and bias produce the equation:

$$x_1 + x_2 - 1 = 0$$

This equation defines a straight line in the two-dimensional input space. Any input point lying on this line gives a net input $z = 0$, which is the threshold where the perceptron switches its prediction.

- If $x_1 + x_2 - 1 \geq 0$, the perceptron outputs **1 (Class 1)**
- If $x_1 + x_2 - 1 < 0$, the perceptron outputs **0 (Class 0)**

Thus, the line effectively divides the feature space into two regions, allowing the perceptron to classify new data points based on which side of the boundary they fall. This illustrates how the perceptron creates a **linear separation** between classes.

7. Implementation in Spreadsheet

To implement in Excel or Google Sheets:

Columns:

1. x1
2. x2
3. Target
4. Weighted sum (formula)
5. Output (IF function)
6. Error
7. Updated weights

Example formula:

Weighted sum:

= w1*x1 + w2*x2 + b

Output:

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

This allows easy visualization of learning progress.

8. Results and Discussion

Observations

• The perceptron successfully learned the AND logic function, correctly classifying all input combinations after training.
• With each iteration, the weight updates gradually reduced the classification error, showing how the learning rule improves model performance.
• The training process converged after a few iterations, indicating that the model reached stable weight values where no further updates were required.

Interpretation

The perceptron learns by iteratively adjusting its weights in the direction that reduces the error between predicted and target outputs. Each misclassified sample shifts the decision boundary so that future predictions become more accurate.

Because the AND dataset is **linearly separable**, a straight-line boundary exists that perfectly divides the two classes. Therefore, the perceptron is guaranteed to converge to a solution, demonstrating its effectiveness for simple binary classification problems.

9. Advantages

1. **Simple and easy to implement**: The perceptron has a straightforward structure with a single neuron, making it easy to understand and implement both manually and computationally.
2. **Fast training:** Since the learning rule involves only basic arithmetic operations, the training process is quick and efficient, especially for small datasets.
3. **Works well for linearly separable data:** The perceptron performs effectively when the classes can be separated by a straight line or plane, achieving accurate classification in such cases.
4. **Requires low computational power:** Due to its simple architecture and minimal calculations, the perceptron can be trained even on systems with limited processing resources.

10. Limitations

1. **Cannot solve non-linear problems (e.g., XOR)**
   The perceptron can only create a linear decision boundary, so it fails to correctly classify datasets where classes are not linearly separable.
2. **Only supports binary classification**
   A basic perceptron produces only two possible outputs, making it unsuitable for multi-class problems without modifications.
3. **Sensitive to learning rate**
   Choosing a very high or very low learning rate can affect training efficiency, causing slow learning or unstable updates.
4. **May not converge if data is not separable**
   When no linear boundary exists, the perceptron keeps updating weights without reaching a stable solution.

11. Applications

- **Pattern recognition**
  Perceptrons are used in simple pattern recognition tasks, such as identifying basic shapes or signals by separating data into distinct classes.
- **Spam filtering**
  They can classify emails as spam or not spam based on features like keywords, frequency, and sender information.
- **Medical diagnosis**
  In healthcare, perceptron-based models can assist in preliminary diagnosis by classifying patient data into risk or non-risk categories.
- **Face detection (basic models)**
  Early face detection systems used perceptron-like classifiers to distinguish facial patterns from non-facial images.
- **Credit risk classification**
  Financial institutions can use simple classifiers to categorize loan applicants as low or high risk based on their financial attributes.

12. Appendix: Python Implementation

This section demonstrates a simple implementation of the perceptron model using Python to validate the manual calculations.

```
import numpy as np


# AND dataset
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])


y = np.array([0, 0, 0, 1])


# Initialize weights and bias
weights = np.zeros(2)
bias = 0
learning_rate = 1
epochs = 10


# Step activation function
def step_function(z):
    return 1 if z >= 0 else 0


# Training
for epoch in range(epochs):
    total_error = 0
    for i in range(len(X)):
```

```
        z = np.dot(weights, X[i]) + bias
        y_pred = step_function(z)

        error = y[i] - y_pred
        total_error += abs(error)

        # Update rule
        weights += learning_rate * error * X[i]
        bias += learning_rate * error

    print(f"Epoch {epoch+1}, Error = {total_error}")

print("\nFinal Weights:", weights)
print("Final Bias:", bias)

# Testing
print("\nPredictions:")
for i in range(len(X)):
    z = np.dot(weights, X[i]) + bias
    print(X[i], "->", step_function(z))
```

Output:

```
Epoch 1, Error = 2
Epoch 2, Error = 3
Epoch 3, Error = 3
Epoch 4, Error = 2
Epoch 5, Error = 1
Epoch 6, Error = 0
Epoch 7, Error = 0
Epoch 8, Error = 0
Epoch 9, Error = 0
Epoch 10, Error = 0

Final Weights: [2. 1.]
Final Bias: -3

Predictions:
[0 0] -> 0
[0 1] -> 0
[1 0] -> 0
[1 1] -> 1
```

13. Conclusion

This project demonstrated how a perceptron model can be designed and trained using the Perceptron learning law to solve a binary classification problem. By performing manual calculations and observing iterative weight updates, we were able to clearly understand how the model learns from its mistakes and gradually improves its predictions. The perceptron successfully classified the given dataset, showing that simple learning rules can effectively create a decision boundary when the data is linearly separable.

The experiment highlights key concepts of supervised learning, such as error calculation, weight adjustment, convergence, and the role of the learning rate. It also illustrates how the decision boundary evolves during training and eventually stabilizes once the classification error is minimized. These observations provide a practical understanding of how learning occurs inside neural networks at a fundamental level.

Although the perceptron is a simple model with certain limitations, it plays a crucial role as the building block of modern neural network architectures. Concepts such as weighted inputs, activation functions, and iterative learning used in perceptrons are directly extended in advanced models. Understanding the perceptron therefore provides a strong conceptual foundation for studying more complex systems like multilayer perceptrons (MLP), convolutional neural networks (CNN), and deep neural networks (DNN).

Overall, this project not only demonstrates the working principle of a linear classifier but also reinforces the importance of foundational models in the evolution of artificial intelligence. The knowledge gained from this study can be applied to more advanced machine learning techniques and real-world classification problems in the future.

13. Future Work

Future improvements may include:

- **Implementing the model in Python**
  The perceptron can be implemented programmatically to automate training, handle larger datasets, and evaluate performance metrics such as accuracy.
- **Visualizing decision boundaries using graphs**
  Plotting the decision boundary and data points can provide a clearer understanding of how the perceptron separates classes in the feature space.
- **Extending to Multilayer Perceptron (MLP)**
  Using multiple layers of neurons can overcome the limitation of linear separability and enable the model to solve more complex, non-linear problems.
- **Testing with real-world datasets**
  Applying the perceptron to practical datasets such as email classification or simple medical data can demonstrate its usefulness in real-life scenarios.

14. References

1. Rosenblatt, F. (1957). The Perceptron: A Probabilistic Model for Information Storage
2. Simon Haykin – Neural Networks and Learning Machines
3. Goodfellow, Bengio & Courville – Deep Learning
4. Bishop, C. M. – Pattern Recognition and Machine Learning