SRINIVAS UNIVERSITY
INSTITUTE OF ENGINEERING & TECHNOLOGY

(SUBJECT: ARTIFICIAL NEURAL NETWORKS)

(SUBJECT CODE: 24SBT113)

AN INDIVIDUAL TASK REPORT ON

"Build a Perceptron model"

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

BACHELOR OF TECHNOLOGY
IN
ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Submitted By,

SANUROOP CK-01SU24AI094

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, **SANUROOP CK (01SU24AI094),** has satisfactorily completed the assessment (Group Task) in **ARTIFICIAL NEURAL NETWORKS (24SBT113)** prescribed by the Srinivas University for the 4th semester B. Tech course during the year **2025-26**.

MARKS AWARDED

Staff In charge

Name: Prof. Mahesh Kumar V B

Date: 23.02.2026

TABLE OF CONTENTS

INTRODUCTION

In the field of Artificial Intelligence and Machine Learning, learning algorithms play a crucial role in enabling machines to make decisions based on data. One of the earliest and most influential learning models is the Perceptron, which laid the foundation for modern neural networks. The perceptron is a simple yet powerful computational model inspired by the functioning of biological neurons in the human brain. It is primarily used for binary classification problems, where the goal is to assign inputs into one of two possible classes, such as true/false, yes/no, or 0/1. Understanding how a perceptron learns is essential for grasping more advanced neural network models.

A key mechanism behind perceptron training is Error-Correction Learning. This learning approach is based on the principle that a system should adjust itself whenever it makes a mistake. In other words, learning occurs by reducing the difference between the desired output (target) and the actual output produced by the model. This difference is called the error, and it directly influences how the perceptron updates its internal parameters, namely the weights and bias. Error-correction learning is a form of supervised learning, as the model is trained using labeled input-output pairs.

The perceptron model consists of input signals, associated weights, a bias term, a summation unit, and an activation function. During training, each input is multiplied by its corresponding weight, summed together with the bias, and passed through an activation function—commonly a unit step function. The output is then compared with the target value. If the output matches the target, no learning is required. However, if the output is incorrect, the perceptron learning rule modifies the weights and bias in a direction that reduces future errors. This iterative process continues over multiple training cycles, known as epochs, until the error becomes zero or minimal.

To demonstrate error-correction learning effectively, logical functions such as AND and OR are commonly used. These tasks are ideal because they are linearly separable, meaning a straight line can divide the input space into correct classes. When a perceptron is trained on AND or OR datasets, the total classification error decreases gradually with each epoch, clearly illustrating the learning process. Plotting the error against the number of epochs provides a visual confirmation of convergence and learning efficiency.

An important factor influencing error-correction learning is the learning rate. The learning rate determines how much the weights change in response to an error. A small learning rate leads to slow but stable convergence, while a large learning rate speeds up learning but may cause oscillations or instability. Choosing an appropriate learning rate is therefore essential for achieving efficient and reliable convergence.

In summary, error-correction learning in a perceptron offers a simple yet insightful demonstration of how machines learn from mistakes. By training a perceptron on basic binary classification tasks like AND and OR, one can clearly observe error reduction, convergence behavior, and the effect of learning rate. This foundational concept not only helps in understanding neural learning principles but also serves as a stepping stone toward more complex models in machine learning and deep learning.

OBJECTIVE

The primary objective of this experiment is to study and demonstrate the concept of Error-Correction Learning using a single-layer Perceptron model for binary classification tasks. This experiment aims to provide a clear understanding of how a perceptron learns from training data by adjusting its weights and bias based on the classification error, thereby improving its prediction accuracy over successive learning iterations known as epochs.

One of the major objectives is to implement and analyze the perceptron learning process for simple logical problems such as AND and OR gates. These problems are chosen because they are linearly separable, making them suitable for demonstrating the effectiveness of the perceptron learning algorithm. By training the perceptron on these datasets, the experiment seeks to show how the model identifies a decision boundary that correctly separates input patterns into two distinct classes.

Another important objective is to understand and apply the Error-Correction Learning Rule, which forms the foundation of supervised learning in perceptrons. The experiment focuses on observing how weight updates are directly proportional to the error between the target output and the predicted output. Through repeated exposure to training patterns, the perceptron modifies its parameters in a direction that reduces the overall classification error. This objective helps in understanding how learning from mistakes enables neural networks to improve performance.

The experiment also aims to analyze the behavior of the perceptron during the training process by monitoring the error reduction across epochs. By calculating the total error at the end of each epoch, the experiment demonstrates how learning progresses over time. Plotting or tabulating the error values allows visualization of the learning curve, making it easier to interpret convergence behavior. This objective emphasizes the importance of iterative learning in neural networks.

A further objective is to study the effect of learning rate on the convergence of the perceptron model. The learning rate controls the magnitude of weight updates during training. By experimenting with different learning rate values—small, moderate, and large—the experiment aims to understand how learning speed and stability are influenced. This objective highlights that while a small learning rate results in slow but stable learning, an excessively large learning rate may cause oscillations or delayed convergence.

In addition, the experiment seeks to enhance conceptual understanding of linearly separable and non-linearly separable problems. By successfully training the perceptron on AND and OR tasks, learners can clearly identify the types of problems that a single-layer perceptron can solve. This objective also indirectly introduces the limitation of the perceptron model, particularly its inability to solve problems such as XOR, thereby motivating the need for multi-layer neural networks.

Another objective of this experiment is to develop practical skills in implementing neural network algorithms either manually, using spreadsheets, or through programming tools. By performing step-by-step calculations of net input, output, error, and weight updates, learners gain hands-on experience with the internal working of neural networks. This practical exposure strengthens theoretical concepts and builds confidence in applying machine learning techniques to real-world problems.

Finally, the overall objective of this experiment is to bridge the gap between theoretical knowledge and practical implementation of neural networks. By demonstrating how a simple perceptron learns through error correction, the experiment lays a strong foundation for understanding more advanced neural network models and learning algorithms used in modern machine learning and artificial intelligence applications.

STRUCTURE OF THE PERCEPTRON MODEL

The perceptron model is the simplest form of an artificial neural network designed for binary classification problems. It is inspired by the biological neuron and serves as the foundational building block for more complex neural network architectures. Understanding the structure of a perceptron is essential for comprehending how learning and decision-making occur in artificial neural systems.

At its core, the perceptron consists of five main components: input layer, weights, bias, summation unit, and activation function. Each component plays a specific role in transforming input data into a meaningful output.

The input layer represents the features or attributes of the data being processed. These inputs are typically numerical values such as binary values, integers, or real numbers. For example, in logic gate problems like AND or OR, the inputs are binary values (0 or 1). The perceptron can handle multiple inputs, and each input corresponds to one feature of the data. The number of input nodes depends on the dimensionality of the problem.

Each input is associated with a weight, which determines the importance or influence of that input on the final output. Weights are real-valued parameters that are adjusted during the learning process. If a weight has a large positive value, it means the corresponding input strongly contributes to activating the neuron. Conversely, a negative weight reduces the influence of the input. Initially, weights are usually assigned small random values or zeros and are modified during training using a learning rule.

The bias is an additional parameter that allows the perceptron to shift its decision boundary. It acts as an offset added to the weighted sum of inputs. Without bias, the decision boundary would always pass through the origin, which limits the model's flexibility. The bias enables the perceptron to correctly classify data that cannot be separated by a boundary passing through the origin. Conceptually, the bias can be thought of as a weight connected to an input that is always equal to one.

The summation unit computes the weighted sum of the inputs and the bias. Mathematically, this operation is expressed as:

This net input represents the total activation received by the perceptron before applying the activation function. The summation unit integrates all incoming signals and prepares them for decision-making.

The output of the summation unit is then passed to the activation function, which determines the final output of the perceptron. In a basic perceptron, a unit step (threshold) function is commonly used. This function produces a binary output: it outputs 1 if the net input is greater than or equal to a threshold value, and 0 otherwise. The activation function introduces non-linearity and enables the perceptron to perform classification tasks.

The final component is the output layer, which consists of a single neuron that produces the classification result. Since the perceptron is a single-layer network, there are no hidden layers. The output represents the predicted class label, such as true or false, yes or no, or 1 or 0.

In summary, the structure of a perceptron model is simple yet powerful. By combining inputs with adjustable weights, adding a bias term, computing a weighted sum, and applying an activation function, the perceptron is able to learn and classify patterns. Although it has limitations in solving non-linearly separable problems, the perceptron remains a fundamental model for understanding the principles of neural networks and supervised learning.

MATHEMATICAL MODEL OF THE PERCEPTRON

The perceptron is one of the earliest and most fundamental models in artificial neural networks, widely used for solving binary classification problems. Its mathematical model provides a formal representation of how input data is transformed into an output decision through weighted summation and an activation function. Understanding this mathematical formulation is essential for analyzing the learning behavior and limitations of the perceptron.

Consider a perceptron with n input features, represented as a vector

$$\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$$

Associated with each input is a weight, represented as

$$\mathbf{w} = (w_1, w_2, w_3, \ldots, w_n)$$

In addition to the weighted inputs, the perceptron includes a bias term, denoted by . The bias allows the decision boundary to be shifted away from the origin, increasing the flexibility of the model. Without the bias term, the perceptron would be constrained to decision boundaries that pass through the origin, which is insufficient for many classification tasks.

The first step in the perceptron's mathematical operation is the computation of the net input, also known as the activation potential. It is calculated as the weighted sum of the inputs plus the bias:

$$net = \sum_{i=1}^{n} w_i x_i + b$$

This equation represents a linear combination of the input features. In vector notation, it can be written more compactly as:

$$net = \mathbf{w} \cdot \mathbf{x} + b$$

The net input value determines how strongly the perceptron is activated before applying the activation function.

The next component of the mathematical model is the activation function, which converts the continuous net input into a discrete output. In the classical perceptron, a unit step (threshold) function is used:

$$y = \begin{cases} 1, & \text{if } net \ge 0 \\ 0, & \text{if } net < 0 \end{cases}$$

Here, represents the predicted output of the perceptron. The activation function introduces a decision-

making mechanism that enables binary classification. If the net input exceeds the threshold (usually set to zero), the perceptron outputs one class; otherwise, it outputs the other class.

The perceptron's output is then compared with the target output, denoted by . The difference between the target and predicted output is known as the error:

$$e = t - y$$

This error term plays a crucial role in learning, as it indicates whether the perceptron has classified the input correctly or incorrectly.

To reduce this error, the perceptron uses an error-correction learning rule to update its weights and bias. The update equations are given by:

$$w_i^{new} = w_i^{old} + \eta e x_i$$

$$b^{new} = b^{old} + \eta e$$

where is the learning rate, a positive constant that controls the size of weight updates. A suitable learning rate ensures stable and efficient convergence.

Geometrically, the mathematical model of the perceptron represents a linear decision boundary in the input space. The equation

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

In conclusion, the mathematical model of the perceptron combines linear algebra and threshold-based decision-making to perform binary classification. Despite its simplicity and limitations, it forms the theoretical foundation for more advanced neural network models and learning algorithms used in modern machine learning.

ACTIVATION FUNCTION

An activation function is a crucial component of an artificial neural network, including the perceptron model. It determines whether a neuron should be activated or not by converting the net input received by the neuron into an output signal. In simple terms, the activation function decides the final output of a neuron based on the weighted sum of inputs and the bias. Without an activation function, a neural network would behave like a simple linear system and would not be capable of performing classification tasks.

In the perceptron model, the net input to the neuron is calculated as the weighted sum of inputs plus the bias:

net = \sum_{i=1}^{n} w_i x_i + b

The most commonly used activation function in the classical perceptron is the unit step function, also known as the threshold function. This function produces a binary output, making it suitable for binary classification problems. The unit step activation function is defined as:

f(net) =
\begin{cases}
1, & \text{if } net \ge \theta \\
0, & \text{if } net < \theta
\end{cases}

One of the primary advantages of the unit step activation function is its simplicity. It is easy to implement and computationally efficient, making it ideal for demonstrating the fundamental principles of neural networks. In logic gate problems such as AND and OR, the unit step function enables the perceptron to correctly map input combinations to their corresponding binary outputs.

However, the unit step function also has certain limitations. It is a non-differentiable function, meaning its derivative is zero almost everywhere and undefined at the threshold. As a result, gradient-based learning methods cannot be directly applied. This limitation restricts the perceptron to simple learning rules such as error-correction learning and prevents it from being extended to more complex, multi-layer networks using gradient descent.

Apart from the unit step function, other activation functions have been developed to overcome these limitations. For example, the sigmoid activation function produces a smooth, continuous output between 0 and 1. It is defined as:

f(net) = \frac{1}{1 + e^{-net}}

Another commonly used activation function is the hyperbolic tangent (tanh) function, which outputs values between –1 and +1. It is similar to the sigmoid function but is zero-centered, which often leads to faster convergence during training. Additionally, the ReLU (Rectified Linear Unit) activation function has become popular in deep learning due to its computational efficiency and ability to reduce the vanishing gradient problem.

ERROR-CORRECTION LEARNING RULE

The Error-Correction Learning Rule is a fundamental supervised learning mechanism used in artificial neural networks, particularly in the perceptron model. The core idea behind this learning rule is that a neural network should adjust its parameters in such a way that the error between the desired output and the actual output is minimized. Learning occurs only when the network makes a mistake, and corrections are applied to reduce future errors.

In supervised learning, each input pattern is associated with a known target output. During training, the perceptron processes an input pattern and produces an output based on its current weights and bias. This output is then compared with the target value. If the predicted output matches the target, no learning occurs. If there is a mismatch, the perceptron updates its weights and bias to correct the error. This approach closely resembles the way humans learn from mistakes.

Mathematically, the perceptron computes the net input as:

$$net = \sum_{i=1}^{n} w_i x_i + b$$

The error is defined as the difference between the target output and the actual output :

$$e = t - y$$

The Error-Correction Learning Rule updates the weights using the following equation:

$$w_i^{new} = w_i^{old} + \eta e x_i$$

$$b^{new} = b^{old} + \eta e$$

Conceptually, the error-correction rule works by shifting the decision boundary of the perceptron. When a data point is misclassified, the weights are adjusted so that the decision boundary moves closer to correctly classifying that point. Over repeated training iterations, known as epochs, the decision boundary gradually aligns itself in a way that separates the input patterns into their respective classes.

One of the most important properties of the Error-Correction Learning Rule is its convergence guarantee for linearly separable data. If the training data can be separated by a linear decision boundary, the perceptron learning algorithm is guaranteed to converge to a solution in a finite number of steps, provided an appropriate learning rate is chosen. This makes the learning rule particularly effective for simple binary classification problems such as AND and OR logic gates.

However, the error-correction learning rule also has limitations. It cannot solve non-linearly separable problems such as the XOR problem. In such cases, the perceptron fails to converge because no single linear decision boundary can separate the classes. This limitation led to the development of multi-layer neural networks and more advanced learning algorithms.

TRAINING THE AND GATE DATASET

Training the AND gate dataset using a perceptron is a fundamental exercise for understanding supervised learning and error-correction mechanisms in neural networks. The AND gate is a simple logical function that outputs a value of 1 only when both inputs are 1; otherwise, it outputs 0. Because the AND gate is a linearly separable problem, it is well suited for demonstrating the learning capability and convergence behavior of a single-layer perceptron.

The AND gate dataset consists of two binary input variables and one binary target output. The dataset can be represented as four training patterns. When both inputs are zero or when only one input is one, the target output is zero. Only when both inputs are one does the target output become one. This clear separation makes it possible for the perceptron to learn an appropriate decision boundary.

At the beginning of the training process, the perceptron parameters such as weights and bias are initialized, usually to zero or small random values. A learning rate is selected to control the magnitude of weight updates during learning. The perceptron then processes each training pattern one by one. For each input pattern, the net input is calculated as the weighted sum of the inputs plus the bias. This value represents the total activation received by the perceptron before classification.

Once the net input is computed, it is passed through a unit step activation function to produce the output. If the net input is greater than or equal to zero, the output is set to one; otherwise, it is set to zero. This predicted output is then compared with the target output of the AND gate. If the predicted output matches the target output, the classification is correct and no weight update is performed. If the predicted output differs from the target, an error is generated.

The error is calculated as the difference between the target output and the predicted output. This error value determines how the perceptron parameters are adjusted. Using the error-correction learning rule, each weight is updated in proportion to the error, the corresponding input value, and the learning rate. The bias is also updated using the same error term. This adjustment shifts the decision boundary in a direction that reduces the likelihood of making the same error again.

Training proceeds through multiple iterations called epochs. In each epoch, the perceptron is exposed to all four training patterns of the AND gate dataset. During the early epochs, errors are common because the weights and bias are not yet optimized. However, as training continues, the perceptron gradually learns the correct mapping between inputs and outputs. The number of misclassifications decreases with each epoch, indicating improvement in learning.

A key aspect of training the AND gate dataset is observing the reduction in total error across epochs. The total error for an epoch is usually calculated as the sum of absolute errors for all training patterns. As the perceptron learns, this total error decreases and eventually reaches zero. Zero error signifies that the perceptron has successfully learned the AND gate function and correctly classifies all input patterns.

TRAINING PROCEDURE

The training procedure of a perceptron refers to the systematic process by which the model learns to classify input patterns correctly by adjusting its weights and bias. This process is based on supervised learning, where each input pattern is associated with a known target output. The goal of training is to minimize classification errors and achieve convergence, where the perceptron correctly classifies all training patterns.

The first step in the training procedure is the initialization of parameters. The weights associated with each input and the bias term are initialized to small random values or zeros. Proper initialization ensures that the perceptron begins learning without bias toward any particular input feature. Along with initialization, a learning rate is selected. The learning rate is a positive constant that controls the size of weight updates during learning. Choosing an appropriate learning rate is important to ensure stable and efficient convergence.

The next step involves presenting the training dataset to the perceptron. Each training pattern consists of input values and a corresponding target output. The perceptron processes one pattern at a time. For a given input pattern, the net input is calculated as the weighted sum of the inputs plus the bias. This value represents the total stimulation received by the perceptron neuron.

After computing the net input, the perceptron applies an activation function to generate the predicted output. In the classical perceptron model, a unit step function is used to produce a binary output. The predicted output is then compared with the target output to determine whether the classification is correct or incorrect.

If the predicted output matches the target output, no change is made to the weights and bias. This indicates that the perceptron has correctly classified the input pattern. However, if there is a mismatch between the predicted output and the target output, an error occurs. This error is computed as the difference between the target output and the predicted output.

Once the error is calculated, the perceptron updates its weights and bias using the error-correction learning rule. Each weight is adjusted in proportion to the learning rate, the error value, and the corresponding input. The bias is updated using the same error term. These updates modify the decision boundary of the perceptron, moving it closer to correctly classifying the misclassified pattern.

The process of presenting all training patterns once and updating weights accordingly is called an epoch. After completing one epoch, the perceptron may still have classification errors, especially during the initial stages of training. Therefore, the training procedure repeats for multiple epochs. In each successive epoch, the perceptron's performance typically improves as the weights and bias become better tuned.

An important aspect of the training procedure is monitoring convergence. Convergence occurs when the perceptron correctly classifies all training patterns and the total error becomes zero. For linearly separable datasets, such as AND and OR logic gates, convergence is guaranteed within a finite number of epochs. The training procedure can be terminated when convergence is achieved or when a predefined maximum number of epochs is reached.

LIMITATIONS OF THE PERCEPTRON

The perceptron is one of the earliest and simplest models of artificial neural networks and has played a crucial role in the development of machine learning and neural computation. While it is effective for solving basic binary classification problems, the perceptron has several important limitations that restrict its applicability to real-world and complex tasks. Understanding these limitations is essential for appreciating the need for more advanced neural network architectures.

One of the most significant limitations of the perceptron is its inability to solve non-linearly separable problems. A perceptron can only learn problems where the data can be separated using a single straight line (in two dimensions) or a hyperplane (in higher dimensions). For example, while it can successfully learn AND and OR logic gates, it fails to learn the XOR problem because XOR data cannot be separated by a single linear boundary. This fundamental limitation restricts the perceptron to a narrow class of problems.

Another major limitation is that the perceptron uses a hard threshold activation function, such as the unit step function. This function produces abrupt changes in output and is non-differentiable. As a result, gradient-based optimization methods like gradient descent cannot be applied. This prevents the perceptron from being extended to multi-layer architectures using traditional backpropagation learning, thereby limiting its learning capacity.

The perceptron is also restricted to binary output classification. It can produce only two output classes, typically represented as 0 and 1 or –1 and +1. Although extensions exist to handle multi-class problems, the basic perceptron model does not naturally support them. This makes it unsuitable for applications requiring classification into multiple categories without significant modifications.

Another limitation is related to its sensitivity to input scaling. The performance of a perceptron can be affected by the magnitude of input values. If inputs are not properly normalized or scaled, certain features may dominate the learning process, leading to biased or suboptimal decision boundaries. This sensitivity increases the need for careful data preprocessing.

The perceptron learning algorithm may also exhibit slow convergence or oscillatory behavior depending on the choice of learning rate. A small learning rate results in very slow learning, requiring many epochs to converge. On the other hand, a large learning rate may cause the weights to overshoot optimal values, leading to instability or failure to converge. Selecting an appropriate learning rate is therefore critical but not always straightforward.

Another important limitation is that the perceptron does not provide a measure of confidence in its predictions. The output is binary, offering no probabilistic interpretation or indication of how certain the model is about its decision. This lack of confidence estimation makes it less useful in applications where uncertainty information is important, such as medical diagnosis or risk assessment.

CONCLUSION

This study has successfully presented a comprehensive analysis of the perceptron model and its learning behavior using the error-correction learning approach. The perceptron, as one of the earliest artificial neural network models, serves as a fundamental building block for understanding more advanced neural architectures. Through systematic experimentation and theoretical explanation, the essential concepts governing the perceptron's operation have been clearly demonstrated.

The implementation of the perceptron on simple binary classification tasks, particularly the AND and OR logic gate datasets, has highlighted the effectiveness of supervised learning based on error correction. By iteratively adjusting weights and bias in response to classification errors, the perceptron was able to improve its performance over successive epochs. The gradual reduction of error during training illustrates how learning occurs through continuous feedback, reinforcing the idea that neural networks adapt by minimizing discrepancies between predicted and desired outputs.

A key outcome of this work is the demonstration of the convergence property of the perceptron learning algorithm. For linearly separable datasets, the perceptron consistently achieved zero classification error after a finite number of training iterations. This behavior confirms the theoretical guarantee of convergence and emphasizes the suitability of the perceptron for solving simple decision-making problems. The successful training of AND and OR gate datasets clearly shows how a linear decision boundary can be learned and adjusted through weight updates.

The study also emphasized the role of the activation function in the perceptron model. The use of a unit step activation function enabled binary decision-making, making the perceptron appropriate for two-class classification tasks. Although simple and computationally efficient, this activation function imposes limitations on the model, particularly in terms of differentiability and learning flexibility. Recognizing these constraints is important for understanding why more advanced activation functions are required in multi-layer neural networks.

Another significant aspect examined in this study is the effect of the learning rate on the training process. The learning rate was shown to directly influence the speed and stability of convergence. A small learning rate led to slow but steady learning, whereas an excessively large learning rate caused unstable behavior and delayed convergence. This observation highlights the importance of selecting appropriate training parameters to ensure effective learning.

Despite its simplicity and successful performance on basic tasks, the perceptron exhibits several inherent limitations. Most notably, it cannot solve non-linearly separable problems such as the XOR function. This limitation arises from the absence of hidden layers and the reliance on a linear decision boundary. Additionally, the perceptron's inability to handle noisy or overlapping data restricts its practical applicability in complex real-world scenarios. These limitations underline the necessity of extending the perceptron into multi-layer architectures with more sophisticated learning algorithms.

Overall, this study provides valuable insight into the fundamental principles of neural network learning. By examining the structure, mathematical model, activation function, learning rule, training procedure, and limitations of the perceptron, a solid foundation has been established for understanding more complex neural network models.