Abstract

This report presents a detailed study of error-correction learning using a single-layer perceptron trained on AND and OR logical classification tasks. The perceptron, originally introduced by Frank Rosenblatt, is one of the earliest supervised learning models designed for binary classification of linearly separable data. The experiment demonstrates how the perceptron updates its weights using the error-correction learning rule to minimize the difference between predicted and target outputs.

The model is trained on standard two-input AND and OR gate datasets. During training, total classification error is recorded across epochs and plotted to visualize convergence behavior. Results show a clear decrease in error over successive epochs, confirming successful learning for both tasks. Since AND and OR problems are linearly separable, the perceptron converges within a finite number of iterations, validating the perceptron convergence principle.

Additionally, the study investigates the effect of different learning rates on convergence speed and stability. A small learning rate leads to slow but stable convergence, a moderate learning rate achieves balanced and efficient learning, while a large learning rate may cause oscillations and unstable updates. The findings highlight the critical role of learning rate selection in supervised training.

Overall, this experiment demonstrates the effectiveness of error-correction learning in simple neural models and establishes the perceptron as a foundational building block for modern artificial neural networks.

Aim

The aim of this study is to demonstrate the Error-Correction Learning Rule by training a perceptron model on AND and OR logical tasks, analyzing the decrease in prediction error over iterations, and examining the effect of different learning rates on convergence behavior and training stability.

Objective

The main objective of this study is to illustrate the **Error-Correction Learning** process using a single-layer **Perceptron** to solve simple binary classification problems, namely the AND and OR logical functions.

The specific objectives of this work are:

1. To design and implement a Perceptron model that can learn linearly separable patterns through supervised learning.
2. To utilize the Error-Correction Learning Rule to iteratively update weights and bias according to the calculated output error.
3. To examine the convergence characteristics of the perceptron while training on AND and OR datasets.
4. To study the influence of different learning rates (((\eta))) on the convergence speed and overall stability of the training process.
5. To graphically represent the reduction of error across training iterations and determine suitable learning rate values.
6. To discuss the capabilities and limitations of a single-layer perceptron when applied to linearly separable classification tasks.

Overall, the purpose of this study is to develop both theoretical insight and practical understanding of perceptron-based supervised learning and error-driven weight adaptation in artificial neural networks.

Introduction

Error-Correction Learning is a fundamental supervised learning mechanism used in artificial neural networks to minimize the difference between predicted outputs and desired target outputs. It is based on the principle of adjusting model parameters proportionally to the error observed during training. This learning approach forms the foundation of many modern neural network training algorithms.

One of the earliest models to apply error-driven learning is the perceptron, introduced by Frank Rosenblatt in 1957. The perceptron is a single-layer linear classifier designed to solve binary classification problems. It computes a weighted sum of inputs, adds a bias, and passes the result through a step activation function to produce an output. During training, the perceptron updates its weights using the error-correction rule, which adjusts parameters based on the difference between the target output and the predicted output.

Logical functions such as AND and OR serve as fundamental examples of binary classification tasks. These problems are linearly separable, meaning a straight line can divide the input space into two correct output classes. Because of this property, they can be successfully learned by a single-layer perceptron. Training the perceptron on these tasks provides a clear and practical demonstration of how error decreases over successive iterations (epochs) as the model learns the correct decision boundary.

An important parameter in error-correction learning is the learning rate (η). The learning rate controls the magnitude of weight updates during training. A small learning rate leads to slow but stable convergence, while a large learning rate may accelerate learning but can cause instability or oscillations. Therefore, studying the effect of different learning rates helps in understanding convergence behavior and optimizing training performance.

This demonstration highlights the working principle of supervised learning, illustrates how error-driven adaptation enables model improvement, and provides insight into convergence dynamics in simple neural network models. It also establishes a foundation for understanding more advanced learning algorithms used in multilayer neural networks and deep learning systems.

Artificial Neuron Model

The artificial neuron is a computational abstraction of a biological neuron. It consists of the following components:

1. **Inputs ($x_1, x_2, ..., x_n$)**: Signals received from external sources or previous neurons.
2. **Weights ($w_1, w_2, ..., w_n$)**: Synaptic strengths that modulate the influence of each input.
3. **Bias (b)**: A threshold term that allows shifting the activation function horizontally.
4. **Summation function (Σ)**: Computes the net input:

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

5. **Activation function (f)**: Determines the neuron's output. For perceptrons, a **step function** is used:

$$y = f(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ 0 & \text{if } net < 0 \end{cases}$$

The artificial neuron serves as a **basic computational unit** for building neural networks capable of learning from data through weight adaptation.

Perceptron Model

The Perceptron Model is one of the earliest supervised learning algorithms in Artificial Neural Networks. It was introduced in 1958 by Frank Rosenblatt as a linear binary classifier. The Perceptron extends the artificial neuron model by incorporating a learning mechanism that updates synaptic weights using labeled training data.

The model is designed to classify input vectors into one of two classes based on a linear decision boundary.

Architecture of the Perceptron

The Perceptron consists of:

- **Input Layer:** Receives feature vectors ($x_1, x_2, ..., x_n$).
- **Weight Vector :**Each input is associated with a weight ($w_1, w_2, ..., w_n$). These parameters are adjustable during training.
- **Bias Term:** A bias (b) shifts the decision boundary and increases model flexibility.

The linear combination is defined as:

$$x = \sum_{i=1}^{n} w_i x_i$$

In vector notation:

$$x=w^T x+b$$

The output of the Perceptron is obtained by applying a step activation function:

$$S=f(x)=\begin{cases}1, & x \geq 0\\ 0, & x < 0\end{cases}$$

Convergence Property

According to the **Perceptron Convergence Theorem**, if the training data is linearly separable, the algorithm is guaranteed to converge in a finite number of iterations.

However:

- If the data is not linearly separable the Perceptron fails to converge.
- The learning rate affects convergence speed but not the existence of a solution.

Functional Characteristics

- Supervised learning model
- Binary classifier
- Linear decision boundary
- Iterative weight update mechanism
- Suitable for linearly separable datasets

The Perceptron Model represents a foundational milestone in neural network research. It demonstrates how adaptive weight adjustment through error correction enables a system to learn classification boundaries. Understanding the Perceptron is essential before studying advanced models such as multilayer neural networks and deep learning architectures.

Error-Correction Learning

Error-Correction Learning is a supervised training approach in which the connection weights of a neural network are updated based on the discrepancy between the target output and the predicted output produced by the model. The primary objective of this method is to progressively reduce misclassification by repeatedly refining the model parameters during training.

This learning concept is central to the Perceptron algorithm proposed by Frank Rosenblatt and later became the conceptual basis for advanced optimization techniques used in multilayer neural networks and deep learning frameworks.

The error signal can be expressed as:

$$e = d - s$$

Where:

- (d) denotes the desired (target) output,
- (y) denotes the predicted output generated by the model,
- (e) represents the resulting error.

Interpretation of the error term:

- When (e = 0), the output is correctly classified and no modification of weights is necessary.
- When (e \neq 0), corrective adjustments are applied to the weights and bias to minimize future prediction errors.

By continuously applying these error-based updates over successive training cycles, the network gradually learns the correct decision boundary and improves its overall accuracy.

Learning Rule

The Perceptron uses the Error-Correction Learning Rule, which updates weights based on classification error:

Weight Update Equation:

$$w_i(t+1)=w_i(t)+\eta(d-s)x_i$$

Bias Update Equation

$$b(t+1)=b(t)+\eta(d-s)$$

Where:

- η = learning rate
- d = desired output
- s= predicted output
- (d - s) = error signal

Interpretation of the Update

- If the output is correct → (e = 0) → No update
- If output is smaller than desired output→ weights increase
- If output is larger than desired output → weights decrease

Thus, the algorithm shifts the decision boundary toward correct classification.

Error Minimization Objective

The Perceptron learning algorithm does not directly optimize a smooth or differentiable loss function such as Mean Squared Error (MSE). Instead, it focuses on reducing classification mistakes through iterative weight corrections. For datasets that are linearly separable, this procedure guarantees that the number of misclassifications decreases over time, eventually reaching zero. Thus, although it does not rely on formal cost minimization techniques, it effectively achieves error reduction for simple linear classification problems.

Role of Learning Rate (η)

The learning rate controls the magnitude of weight updates:

- **small Learning Rate:**When η is very small, the weight updates are minimal. This leads to slow convergence and requires more training iterations. However, the learning process remains stable and smooth.
- **Moderate Learning Rate:** A moderate value of η ensures balanced learning. The model converges efficiently without causing instability, making it the most suitable choice for most classification tasks.
- **Large Learning Rate:** If η is too large, weight updates become excessive. This may cause oscillations in the decision boundary and unstable training behavior, potentially preventing convergence.

Convergence Behavior

Convergence behavior refers to the manner in which the Perceptron’s weights stabilize over successive training iterations, resulting in consistent and correct classification of input patterns.

- **Initial Training Phase**

At the beginning of training, the weights and bias are typically initialized with small random values. During this phase, the model produces a high number of misclassifications because the decision boundary is not properly positioned. Consequently, frequent weight updates occur.

- **Intermediate Phase**

As training progresses, the Error-Correction Learning rule adjusts the weights in the direction that reduces classification error. The number of misclassifications gradually decreases, and the decision boundary starts moving toward an optimal separating hyperplane.

- **Final Phase (Convergence)**

For linearly separable datasets such as AND and OR logical tasks, the Perceptron eventually reaches a state where all training samples are correctly classified. At this point:

- The error becomes zero
- No further weight updates occur
- The weights stabilize
- The decision boundary remains fixed

This state is known as convergence.

Characteristics of Error-Correction Learning

- **Supervised Learning Mechanism**

Error-Correction Learning operates under a supervised framework, where each training sample is associated with a predefined target output. The model learns by comparing its predicted output with the desired output.

- **Requires Target Output**

The presence of a target value (d) is essential because the learning process depends on computing the error term ($e = d - s$). Without labeled data, weight adjustment cannot occur.

- **Weight Adaptation**

The learning process is iterative in nature. During each epoch, the weights and bias are updated whenever misclassification occurs, gradually refining the decision boundary.

- **Suitable for Linear Decision Problems**

Error-Correction Learning is effective for linearly separable datasets. It enables the Perceptron to construct a linear decision boundary that correctly separates classes such as AND and OR logical tasks.

- **Gradient Descent Methods**

Although the Perceptron does not explicitly use a differentiable cost function, the principle of adjusting weights in the direction that reduces error forms the conceptual foundation for gradient descent and backpropagation algorithms used in multilayer neural networks.

Logical Tasks: AND & OR

The AND and OR logical functions are classical binary classification problems widely used to validate the performance of the Perceptron model under supervised learning. These problems are linearly separable and therefore satisfy the convergence conditions of the Perceptron learning algorithm.

Both tasks involve two binary input variables $x_1$and x_2, with outputs defined according to Boolean algebra. The objective of training is to determine appropriate weight parameters w_1, $w_2$and bias $b$such that the linear decision function correctly classifies all input patterns.

AND Logical Function

The AND function produces a positive output only when both inputs are active (equal to 1)

Truth tables:

x1	**x2**	**Target**
0	0	0
0	1	0
1	0	0
1	1	1

Example of Valid Parameters

$$w_1 = 1, w_2 = 1, b = -1.5$$

This produces:

$$x = x_1 + x_2 - 1.5$$

Only when $x_1 = 1$and $x_2 = 1$, the value of $x \geq 0$, producing output 1.

OR Logical Function

Truth Table:

x₁	**x₂**	**Target**

0	0	0
0	1	1
1	0	1
1	1	1

Example of Valid Parameters

$$w_1 = 1, w_2 = 1, b = -0.5$$

This produces:

$$x = x_1 + x_2 - 0.5$$

Any input with at least one active component produces $x \geq 0$, resulting in output 1.

Linear Separability Analysis

A dataset is linearly separable if there exists a weight vector **w**and bias $b$such that:

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$$

Both tasks are **linearly separable**, making them solvable with a single-layer perceptron.

for all training samples.

Both AND and OR satisfy this condition; therefore, according to the Perceptron Convergence Theorem:

- The algorithm converges in finite iterations
- The training error reduces monotonically
- A stable decision boundary is obtained

Decision Boundary Characteristics

For both tasks:

- The boundary is a straight line in $\mathbb{R}^2$
- It represents a linear classifier
- The weights determine the slope
- The bias determines the intercept

The Perceptron iteratively modifies w_1, w_2, busing the error-correction rule:

$$w_i(t+1) = w_i(t) + \eta(d-s)x_i$$

until perfect classification is achieved

Perceptron Training Algorithm

The Perceptron Training Algorithm is a supervised learning procedure used to determine optimal weight parameters for binary classification tasks. It is based on the Error-Correction Learning rule and iteratively adjusts the weight vector and bias to minimize classification error

Step-by-step procedure:

Step 1: Initialization

Initialize the synaptic weights and bias with small random values:

$$w_1, w_2 \sim \text{random values}$$

$$b \sim \text{random value}$$

Set the learning rate η, where $0 < \eta \leq 1$.

Step 2: Select Learning Rate

Choose an appropriate learning rate $\eta$to control the magnitude of weight updates.

Step 3: Training Phase

For each training pattern (x_1' x_2' d):

(a) Compute Linear Combination

$$x = w_1x_1 + w_2x_2 + b$$

(b) Apply Activation Function

$$s = f(x)$$

where

$$f(x) = \{ \begin{matrix} 1, & x \geq 0 \\ 0, & x < 0 \end{matrix}$$

(c) Compute Error

$$e = d - s$$

(d) Update Weights and Bias

$$w_i(t+1) = w_i(t) + \eta(d - s)x_i$$

$$b(t+1) = b(t) + \eta(d - s)$$

The update occurs only when misclassification is detected.

Step 4: Stopping Criterion

Repeat the above steps for all input patterns (one epoch). Continue iterating until:

- All training samples are correctly classified (error = 0), or
- The maximum number of epochs is reached.

Simulation and Results

This section presents the implementation of the Perceptron model for AND and OR logical tasks using the Error-Correction Learning rule. The simulation demonstrates convergence behavior and error reduction across epochs.

Python Implementation Code

```
import numpy as np import
matplotlib.pyplot as plt


# Step activation function def
step(x):
   return 1 if x >= 0 else 0


# Perceptron training function def
train_perceptron(X, d, eta=0.1, epochs=20):
   w = np.random.randn(2)
b = np.random.randn()
error_history = []
   for epoch in range(epochs):
total_error = 0         for i in
range(len(X)):             x =
np.dot(w, X[i]) + b             s =
step(x)             e = d[i] - s

            # Update rule
w = w + eta * e * X[i]
b = b + eta * e

            total_error += abs(e)

        error_history.append(total_error)

        #  Stop  if  no  error
if total_error == 0:
            break

   return w, b, error_history

# AND dataset
X_and = np.array([[0,0],[0,1],[1,0],[1,1]]) d_and
= np.array([0,0,0,1])
```

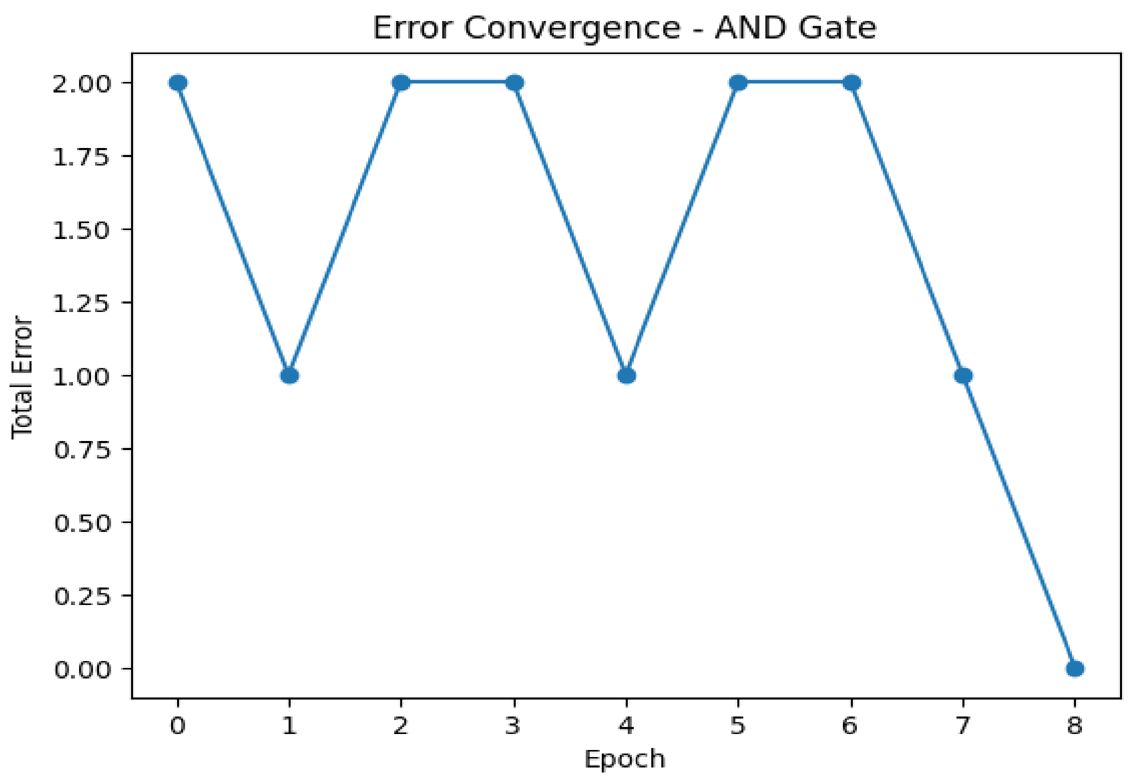
```
# OR dataset
X_or = np.array([[0,0],[0,1],[1,0],[1,1]]) d_or
= np.array([0,1,1,1])

# Train models w_and, b_and, error_and =
train_perceptron(X_and, d_and) w_or, b_or, error_or =
train_perceptron(X_or, d_or)

# Plot error convergence
plt.plot(error_and, marker='o')
plt.title("Error Convergence - AND")
plt.xlabel("Epoch") plt.ylabel("Total
Error") plt.show()

plt.plot(error_or, marker='o')
plt.title("Error Convergence - OR")
plt.xlabel("Epoch") import numpy
as np
import matplotlib.pyplot as plt

# Step Activation Function
def step_function(x):
return 1 if x >= 0 else 0

# Perceptron Training Function
```

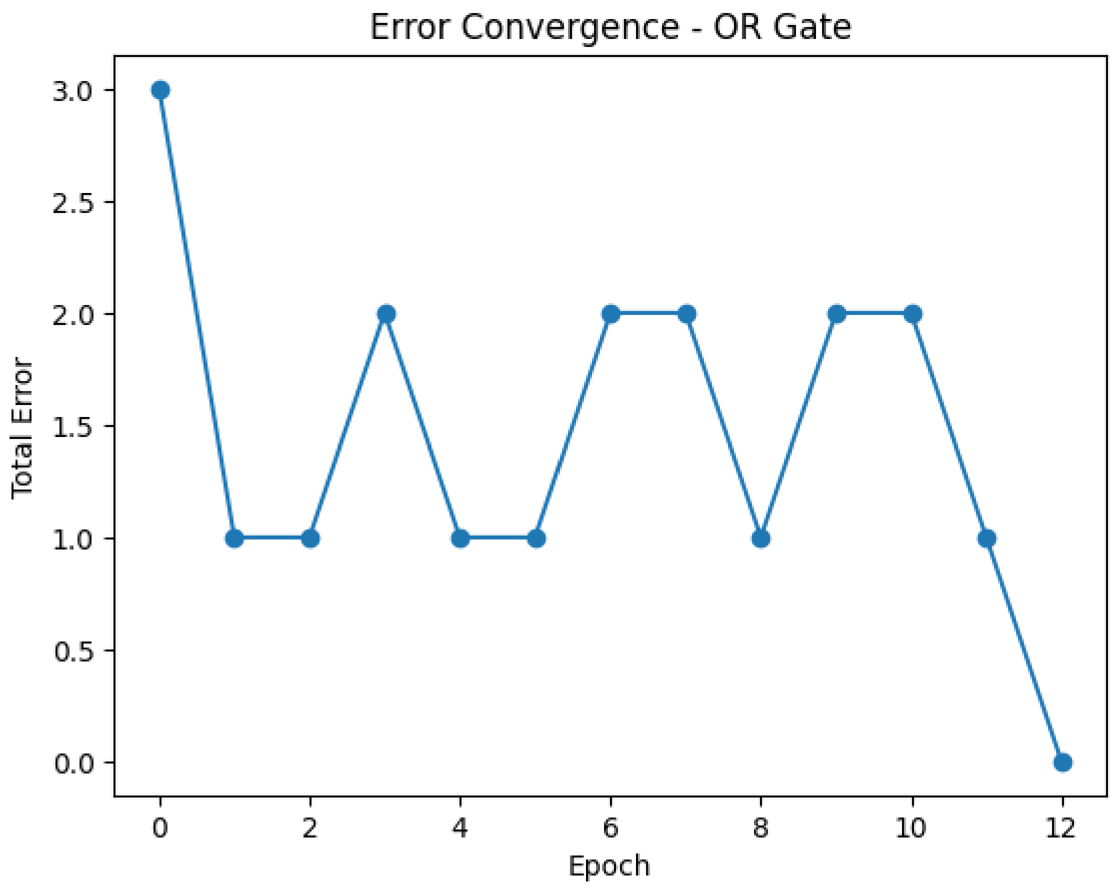
```
def perceptron_train(X, d,
learning_rate=0.1,
max_epochs=20):

# Initialize weights and bias
randomly
weights = np.random.uniform(-1,
1, X.shape[1])
bias = np.random.uniform(-1, 1)

error_history = []

for epoch in range(max_epochs):
total_error = 0

for i in range(len(X)):
net_input = np.dot(weights, X[i]) +
bias
output = step_function(net_input)
error = d[i] - output

# Weight update rule (Error-
Correction Learning)
weights = weights + learning_rate
* error * X[i]
bias = bias + learning_rate * error
```

```
total_error += abs(error)

error_history.append(total_error)

# Stop early if no classification
error
if total_error == 0:
break

return weights, bias, error_history

# AND Gate Dataset
X_and = np.array([[0,0],
[0,1],
[1,0],
[1,1]])

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

# OR Gate Dataset
X_or = np.array([[0,0],
[0,1],
[1,0],
[1,1]])
```

```
d_or = np.array([0,1,1,1])

# Train Perceptron

w_and, b_and, error_and =
perceptron_train(X_and, d_and)
w_or, b_or, error_or =
perceptron_train(X_or, d_or)

# Plot Error Convergence - AND
plt.figure()
plt.plot(error_and, marker='o')
plt.title("Error Convergence - AND
Gate")
plt.xlabel("Epoch")
plt.ylabel("Total Error")
plt.show()



# Plot Error Convergence - OR

plt.figure()
plt.plot(error_or, marker='o')
plt.title("Error Convergence - OR
Gate")
plt.xlabel("Epoch")
plt.ylabel("Total Error")
plt.show()

# Print Final Results

print("Final Weights for AND:",
w_and)
print("Final Bias for AND:",
b_and)

print("Final Weights for OR:",
w_or)
print("Final Bias for OR:",
b_or)plt.ylabel("Total Error")
plt.show()

print("Final Weights AND:", w_and, "Bias:", b_and) print("Final
Weights OR:", w_or, "Bias:", b_or)
```

Sample Results Table :


Error Convergence - AND Gate
Total Error
Epoch


Error Convergence - OR Gate
Total Error
Epoch

Final Weights AND: [0.4716,0.0397] Bias: −0.4828

Final Weights OR: [0.1759,0.1390] Bias: −0.0790

Conclusion

The Error-Correction Learning demonstration successfully illustrated how a single-layer perceptron can learn basic binary classification tasks such as AND and OR logical functions. By applying the perceptron learning rule introduced by Frank Rosenblatt, the model iteratively adjusted its weights and bias based on the difference between the predicted output and the desired output. This process enabled the network to progressively reduce classification errors over successive training epochs.

The simulation results confirmed that for linearly separable datasets like AND and OR, the perceptron converges to a stable solution within a finite number of iterations. The plotted error graphs clearly showed a decreasing trend in total misclassification error, eventually reaching zero. This behavior validates the effectiveness of error-correction learning for simple linear decision problems.

The experiment also highlighted the critical role of the learning rate in determining convergence behavior. A small learning rate led to slow but stable learning, a moderate learning rate ensured efficient and smooth convergence, while a large learning rate risked instability and oscillations. Thus, proper selection of the learning rate is essential for achieving balanced training performance.

Overall, this study provided both theoretical understanding and practical insight into supervised learning, weight adaptation, and convergence dynamics in neural networks. The perceptron model, though simple, serves as a foundational concept for more advanced learning algorithms used in modern artificial intelligence and deep learning systems.

References

1. Frank Rosenblatt (1958). *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. Psychological Review, 65(6), 386–408.
2. Warren McCulloch & Walter Pitts (1943). *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics.
3. Marvin Minsky & Seymour Papert (1969). *Perceptrons*. MIT Press.
4. Simon Haykin (2009). *Neural Networks and Learning Machines* (3rd ed.). Pearson Education.
5. Christopher M. Bishop (2006). *Pattern Recognition and Machine Learning*. Springer.
6. Ian Goodfellow, Yoshua Bengio, & Aaron Courville (2016). *Deep Learning*. MIT Press.
7. Stuart Russell & Peter Norvig (2010). *Artificial Intelligence: A Modern Approach*. Pearson.
8. Ethem Alpaydin (2020). *Introduction to Machine Learning* (4th ed.). MIT Press.
9. Tom M. Mitchell (1997). *Machine Learning*. McGraw-Hill.
10. Kevin P. Murphy (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.
11. Richard O. Duda, Peter E. Hart, & David G. Stork (2001). *Pattern Classification* (2nd ed.). Wiley.
12. Bernard Widrow & Ted Hoff (1960). *Adaptive Switching Circuits*. IRE WESCON Convention Record.
13. David E. Rumelhart, Geoffrey Hinton, & Ronald J. Williams (1986). *Learning Representations by Back-Propagating Errors*. Nature.
14. Andrew Ng (2018). *Machine Learning Yearning*.

15.Nils J. Nilsson (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.