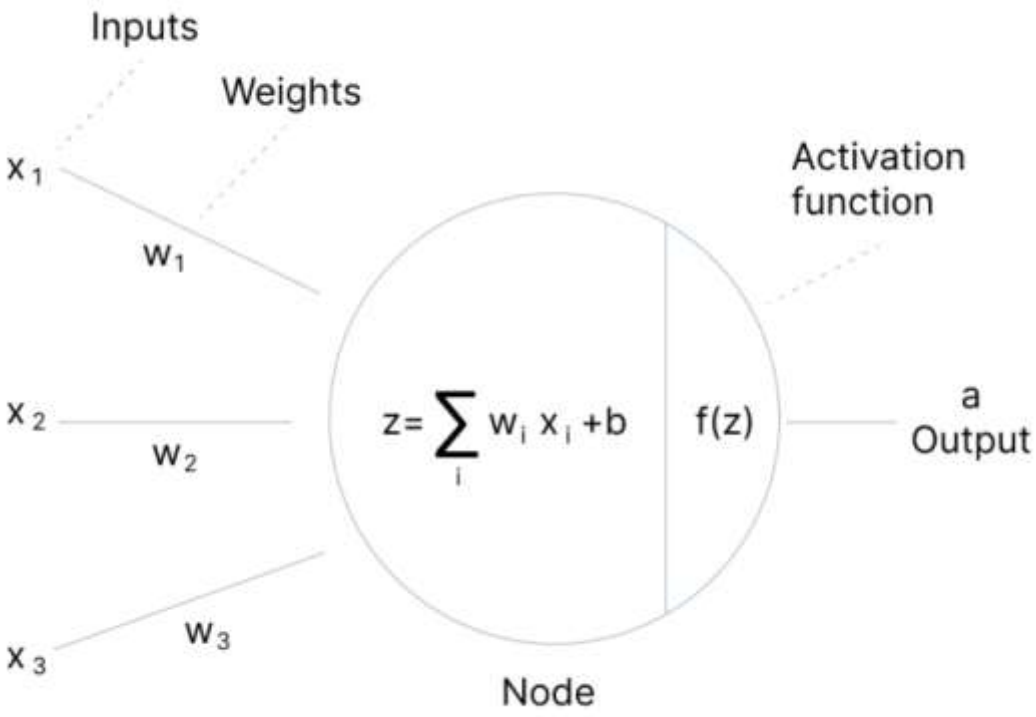
MODULE 4

➢ Introduction neural network:

- Neural Networks, abbreviated as NN or ANN (Artificial Neural Network), represent a crucial paradigm in the field of artificial intelligence and machine learning.
- It's a network which can solve Artificial intelligence problems.
- ANN is a supervised learning system built of a large number of simple elements, called neurons or perceptrons.
- Each neuron can make simple decisions, and feeds those decisions to other neurons, organized in interconnected layers.

- **Basic Terms**



1. **Neuron (Node):**
   - The fundamental unit of a neural network.
   - Neurons receive inputs, apply transformations, and produce outputs.

2. **Input Layer:**
   - First layer receiving external inputs.
   - Passes inputs to subsequent layers without applying operations.
   - Its size depends on the shape of the training data.

3. **Output Layer:**
    - Final layer producing network predictions.
    - Outputs are generated based on inputs processed by hidden layers.
    - Number of neurons in the output layer depends on the task.
4. **Hidden Layer:**
    - Intermediate layers between the input and output layers.
    - Hidden layers process inputs through complex transformations.
    - They are essential for capturing intricate patterns in data.
5. **Connections:**
    - Links between neurons that transmit information.
    - Each connection possesses a weight, influencing the signal's strength.
6. **Bias:**
    - Additional input to neurons, usually set to 1.
    - Bias helps neurons activate even when inputs are absent.
    - Balancing bias and variance is crucial for model performance.
7. **Weights:**
    - Numeric values associated with connections.
    - Weights determine the influence of inputs on neuron outputs.
    - They are adjusted during training to minimize errors.
8. **Activation Function:**
    - Function applied to neuron outputs, determining their activation.
    - Common functions include sigmoid, TanH, and ReLU.
    - Activation functions introduce non-linearity, crucial for learning complex patterns.
9. **Error Function:**
    - Measures the disparity between actual and predicted outputs.
    - Loss functions like Mean Squared Error (MSE) quantify prediction errors.
    - Minimizing error functions guides the network towards accurate predictions.
10. **Variance:**

- Reflects how well the model adapts to new, unseen data.
- High variance may lead to overfitting, while low variance may cause underfitting.

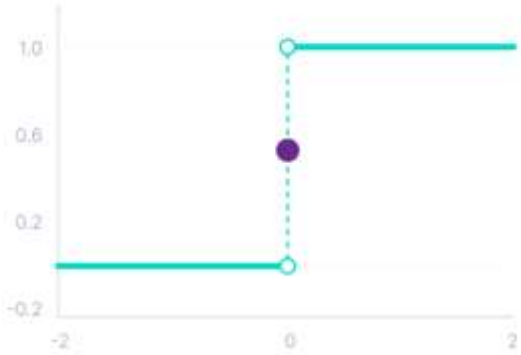
11. **Hyperparameters:**
    - Settings influencing the network's architecture and behavior.
    - Examples include learning rate, number of hidden layers, and neurons per layer.
    - Tuning hyperparameters optimizes network performance for specific tasks.

➢ Activation functions

- Function applied to neuron outputs, determining their activation.
- Common functions include sigmoid, TanH, and ReLU.
- Activation functions introduce non-linearity, crucial for learning complex patterns

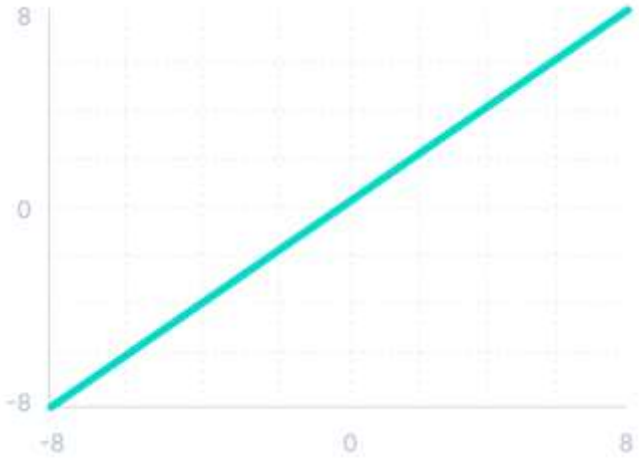
Types of activation function

1. Step Function



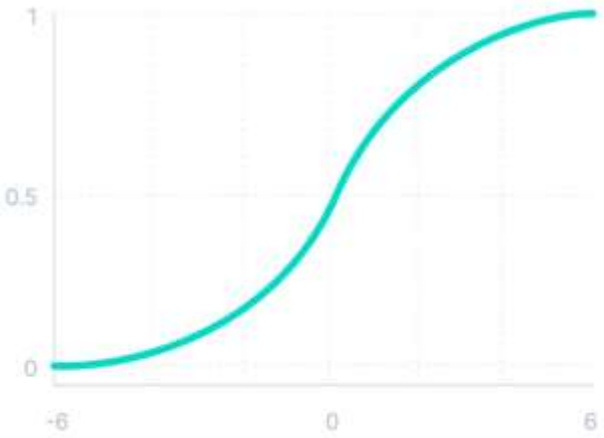
- **Equation:**
  A = 1 if Y > threshold, 0 otherwise.
- It's a binary activation function where a neuron is activated if the output exceeds a threshold.
- **Drawback:** Limited applicability in multi-class classification due to its binary nature. Multiple activated neurons may complicate decision-making.

2. Linear Function



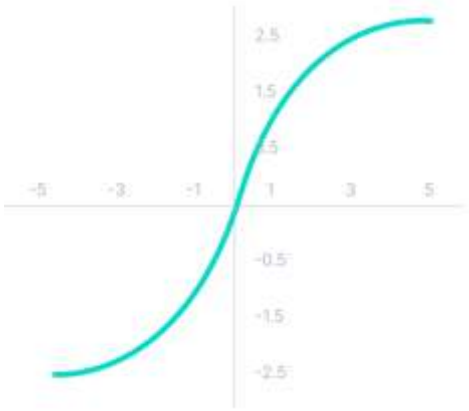
- **Equation:** A = cx.
- Activation is directly proportional to input, offering a range of activations.
- **Problem:** Gradient remains constant regardless of input changes, hindering adaptive learning during backpropagation.

3. Sigmoid Function



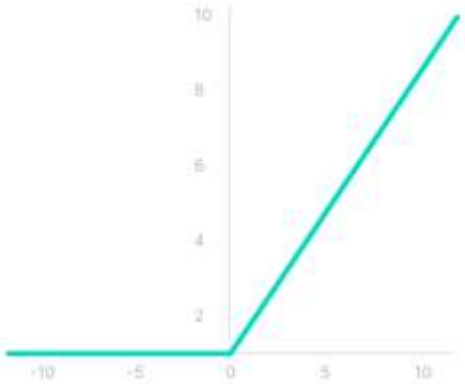
- **Equation:** A = $\frac{1}{1+e^{-x}}$
- Nonlinear function suitable for binary classification tasks.
- Smooth gradient facilitates stable training.
- **Advantages:** Provides analog activations, bound within (0,1) range, and allows stacking layers.
- **Limitation:** Suffers from vanishing gradient problem at extreme ends, slowing down or halting learning.

4. Tanh Function



- **Equation:** A =(2/1+e^-2x)-1
- Similar to sigmoid but scaled to (-1,1) range, suitable for time series data.
- Stronger gradient than sigmoid, aiding in faster learning.
- **Limitation:** Still susceptible to vanishing gradient problem.

5. ReLU (Rectified Linear Unit)

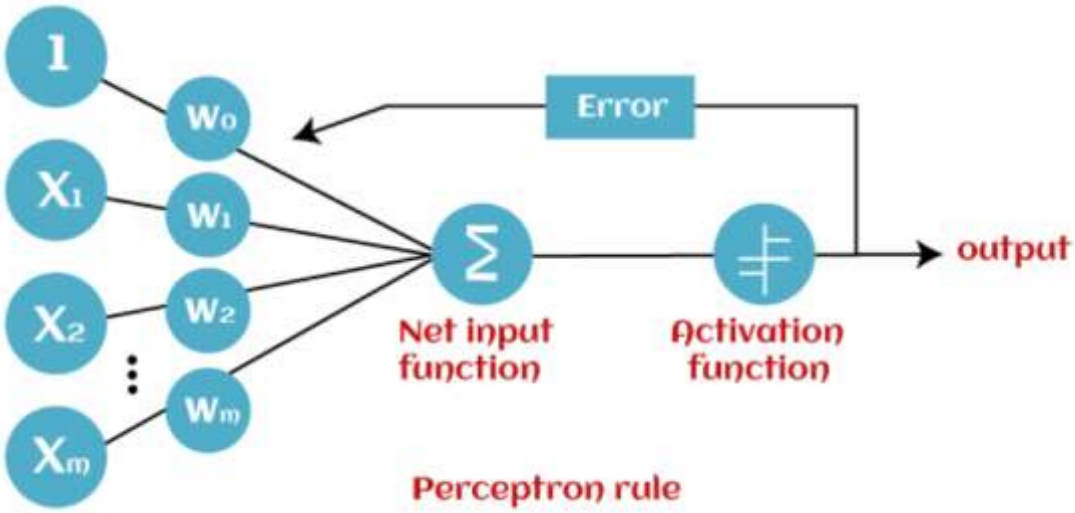


- **Equation:** A(x) = max(0,x).
- Nonlinear function with efficient computation.
- Offers sparse activations, reducing computational load.
- **Drawback:** Suffers from "dying ReLU" problem where neurons become inactive, hindering learning in affected regions.
- Variations like **Leaky ReLU** mitigate this issue by maintaining a non-zero gradient for negative inputs.

➢ Perceptron

Introduction to Perceptron:

- A perceptron is a basic unit in an artificial neural network, mimicking the functionality of biological neurons.
- It's capable of learning and solving simple binary classification problems.
- Inputs are weighted, summed, and passed through an activation function to produce an output.



Types of Perceptrons:

1. **Single Layer Perceptron:**
    - Learns linearly separable patterns.
    - Limited to single-layered models.
2. **Multilayer Perceptron (Feedforward Neural Networks):**
    - Comprises multiple layers of perceptrons, allowing for the accurate resolution of complex problems.
    - Typically includes an input layer, one or more hidden layers, and an output layer.

Perceptron Learning:

- Automatically learns optimal weight coefficients through training.
- Inputs are multiplied by weights, and if the sum exceeds a threshold, the neuron fires.
- Each iteration through the training set is an epoch, and training continues until the error ceases to improve.

Examples:

1) Realization of AND function using perceptron learning rule?

AND Gate

x_0	x_1	x_2	Target
1	1	1	1
1	1	-1	-1
1	-1	1	-1
1	-1	-1	-1

Here x_0 = bias
x_1 = ? } i/p
x_2 = }

Initial weights $w_0 = 0$ } bias weight of x_0
$w_1 = 0$ } weights of x_1 & x_2
$w_2 = 0$ }

Solution:

Input vector = $I = \langle 1\ 1\ 1 \rangle$ weights = $\langle 0\ 0\ 0 \rangle$
output = 1 (Target o/p)

produced o/p = $w_0 x_0 + w_1 x_1 + w_2 x_2$
= $1 \times 0 + 1 \times 0 + 1 \times 0$
= 0

produced o/p ≠ Target o/p

∴ update weights = $\langle 1\ 1\ 1 \rangle + \langle 0\ 0\ 0 \rangle$
= $\langle 1\ 1\ 1 \rangle$

[New updated weights = $\langle 1\ 1\ 1 \rangle$]

Similary, we get

	Inputs x_0 x_1 x_2	Target o/p	Initial weights w_0 w_1 w_2	Produced o/p	updated weights w_0 w_1 w_2
Epoch1	1 1 1	1	0 0 0	0 (0)	1 1 1
	1 1 -1	-1	1 1 1	1 (1)	0 0 2
	1 -1 1	-1	0 0 2	2 (1)	-1 1 1
	1 -1 -1	-1	-1 1 1	-3 (-1)	-1 1 1
Epoch2	1 1 1	1	-1 1 1	1	-1 1 1
	1 1 -1	-1	-1 1 1	-1	-1 1 1
	1 -1 +1	-1	-1 1 1	-1	-1 1 1
	1 -1 -1	-1	-1 1 1	-1	-1 1 1

So iteration stop as every target o/p matched with produced o/p.

∴ Final weigh vector <-1 1 1>

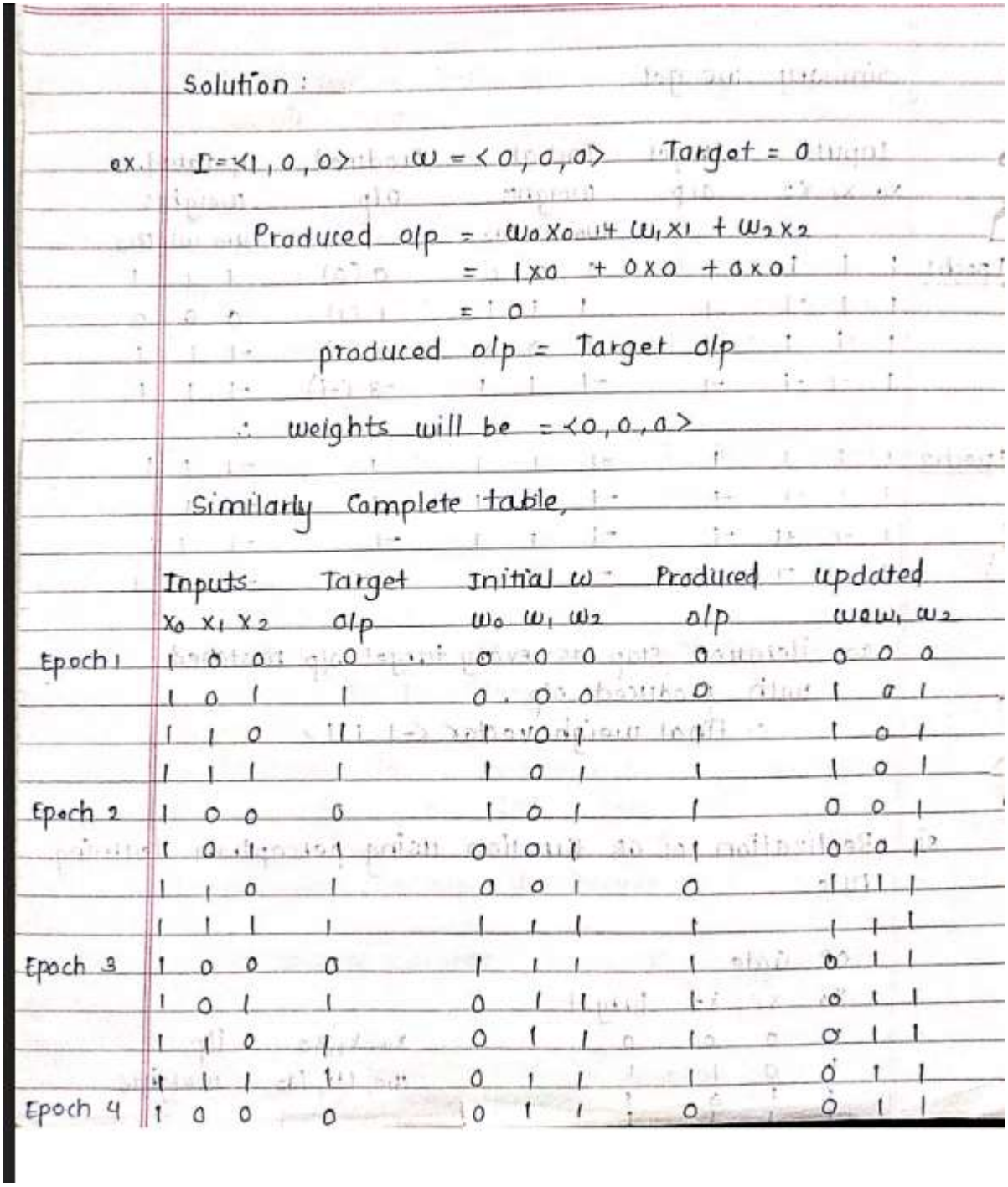
9] Realization of OR function using perceptron learning rule.

OR Gate

x_0	x_1	x_2	Target
1	0	0	0
1	0	1	1
1	1	0	1

x_0, x_1, x_2 = i/p

w_0, w_1, w_2 = weights



Solution :

ex. $I = \langle 1, 0, 0 \rangle$ $w = \langle 0, 0, 0 \rangle$ Target = 0

Produced o/p = $w_0x_0 + w_1x_1 + w_2x_2$

$= 1 \times 0 + 0 \times 0 + 0 \times 0$

$= 0$

produced o/p = Target o/p

∴ weights will be $= \langle 0, 0, 0 \rangle$

Similarly Complete table,

	Inputs			Target	Initial w			Produced	updated		
	x_0	x_1	x_2	o/p	w_0	w_1	w_2	o/p	w_0	w_1	w_2
Epoch 1	1	0	0	0	0	0	0	0	0	0	0
	1	0	1	1	0	0	0	0	1	0	1
	1	1	0	1	1	0	1	1	1	0	1
	1	1	1	1	1	0	1	1	1	0	1
Epoch 2	1	0	0	0	1	0	1	1	0	0	1
	1	0	1	1	0	0	1	1	0	0	1
	1	1	0	1	0	0	1	0	1	1	1
	1	1	1	1	1	1	1	1	1	1	1
Epoch 3	1	0	0	0	1	1	1	1	0	1	1
	1	0	1	1	0	1	1	1	0	1	1
	1	1	0	1	0	1	1	1	0	1	1
	1	1	1	1	0	1	1	1	0	1	1
Epoch 4	1	0	0	0	0	1	1	0	0	1	1

➢ Applications of ANN :

1. Image Recognition and Computer Vision
2. Natural Language Processing (NLP)
3. Speech Recognition and Synthesis
4. Healthcare (Medical Image Analysis, Patient Outcome Prediction)

5. Finance and Trading (Financial Forecasting, Algorithmic Trading)
6. Autonomous Vehicles (Object Detection, Path Planning)
7. Recommendation Systems
8. Manufacturing and Industry 4.0 (Predictive Maintenance, Quality Control)
9. Cybersecurity (Intrusion Detection, Malware Detection)
10. Environmental Monitoring (Weather Forecasting, Pollution Prediction)

➢ Gradient Descent :

- Gradient Descent is a widely used optimization algorithm for training machine learning models, including neural networks.
- It iteratively adjusts model parameters to minimize the error between actual and predicted outputs. Let's delve into its workings and variations:

Working Principle

- **Slope Measurement:** Measures how the output of a function changes with respect to small changes in inputs.
- **Local Minimum/Maximum:** Moving towards negative gradients leads *to local minimum*, while positive gradients lead *to local maximum*, helping in convergence towards optimal solutions.

Formula

$$b = a - \alpha \cdot \nabla F(a)$$

- b: New value
- a: Current value
- α: Learning rate
- $\nabla F(a)$: Gradient (direction of steepest descent)

Minimization Algorithm

- Start with initializing weights (W) and bias (b).
- Calculate the slope (gradient) using the derivative.
- Adjust parameters using the tangent line to minimize the cost function.
- Update weights and bias iteratively until convergence, where the cost function is minimized

➢ Types of Gradient Descent

1. **Batch Gradient Descent (BGD):**
    - Calculates error for all training points before updating parameters.
    - Updates the model after evaluating all training examples.
    - Suitable for small datasets but computationally expensive for large datasets due to evaluating all examples at once.
2. **Stochastic Gradient Descent (SGD):**
    - Processes each training example individually within a dataset.
    - Updates parameters one example at a time.
    - Faster computation but noisy updates, leading to erratic convergence.

3. **Mini-Batch Gradient Descent:**
    - Hybrid approach combining BGD and SGD.
    - Divides training datasets into small batches and updates parameters on these batches separately.
    - Strikes a balance between computational efficiency and convergence stability.

➢ Learning neural network is minimization problem:

Learning neural networks is a minimization problem.

Justification:

1. **Objective Function:** In neural network training, the primary goal is to minimize the error or loss function. The error function quantifies the disparity between the predicted outputs of the neural network and the actual outputs. The aim is to minimize this error, ensuring that the network's predictions are as close as possible to the ground truth.
2. **Gradient Descent:** The optimization algorithm commonly used to train neural networks, such as gradient descent, operates by iteratively adjusting the model parameters (weights and biases) in the direction that minimizes the error function. This process involves

computing the gradient of the error function with respect to the model parameters and updating them to move towards the direction of decreasing error.

3. **Local Minimum:** The objective of the minimization problem is to find the local minimum of the error function. At the local minimum, the error function reaches its lowest value, indicating that further adjustments to the model parameters would not significantly reduce the error. Therefore, training a neural network involves iteratively updating the parameters until convergence to a local minimum of the error function is achieved