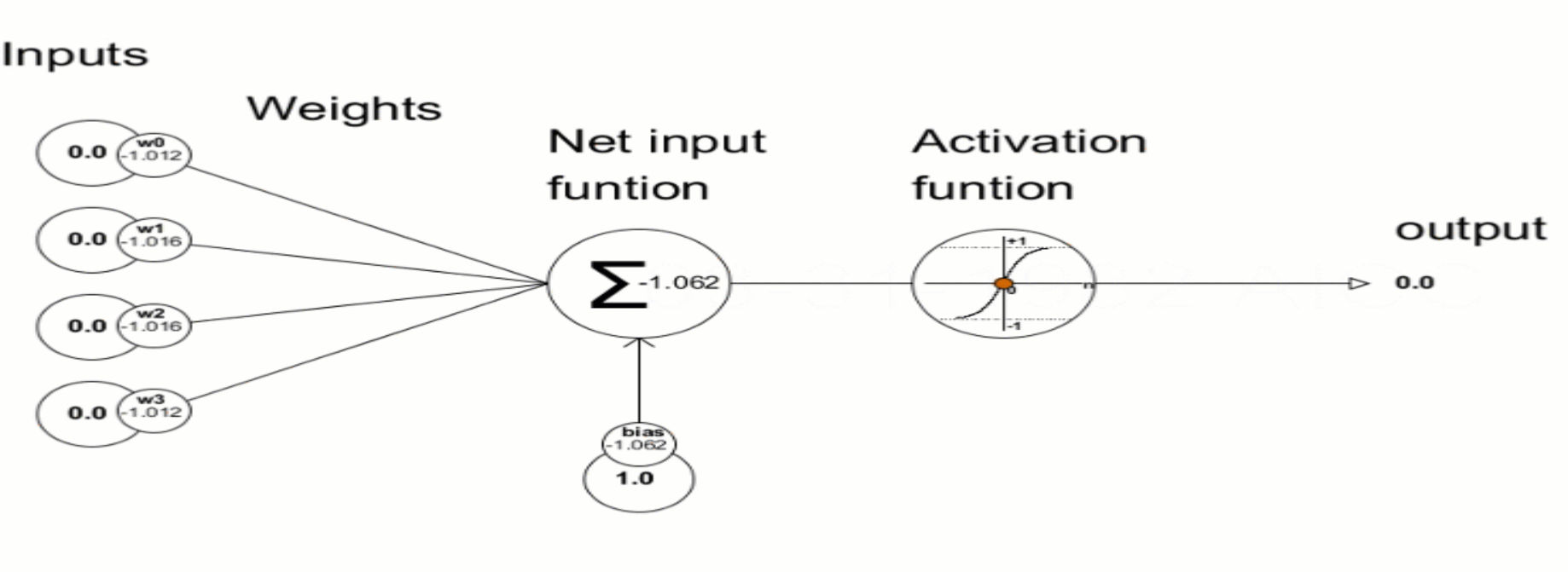
Bias:



In Artificial Neural Networks (ANNs), a bias term is an additional parameter that is added to the weighted sum of the inputs before passing through the activation function. While weights determine the strength of the connections between neurons and inputs, the bias term allows the activation function to be shifted, influencing the output of the neuron regardless of the input.

Including a bias term enables the neural network to capture patterns in the data that might not necessarily pass through the origin (0,0) on a graph. It essentially allows the activation function to be flexible in terms of its positioning, which can be crucial for learning complex relationships in the data.

Mathematically, the weighted sum of inputs plus bias can be represented as:

$$\text{weighted sum} = \sum_{i=1}^{n} \text{input}_i \times \text{weight}_i + \text{bias}$$

Adding the bias term allows the neural network to model more complex functions and improve its ability to fit the training data. During training, just like weights, the bias term is also updated through optimization algorithms such as gradient descent to minimize the error between the predicted output and the actual output.

let's illustrate the difference using a simple example of a perceptron for binary classification. Suppose we have a perceptron with two inputs and no bias initially, and we want to classify points in a 2D space into two classes, class 0 and class 1.

Perceptron without Bias:

Let's define our perceptron without bias with the following parameters:

- Inputs: x1 and x2
- Weights: w1 and w2
- Activation function: Step function (outputs 0 if the weighted sum is negative, and 1 if positive)

Suppose we have a training dataset with the following points:

x_1	x_2	Class
1	2	0
-1	1	1
3	-2	0

Now let's compute the output of the perceptron for the first point:

$$\text{weighted sum} = (1 \times w_1) + (2 \times w_2)$$

And similarly for the other points. Then we apply the step function to determine the class.

Adding Bias:

Now, let's add a bias term b to the perceptron. So the new equation for the weighted sum becomes:

$$\text{weighted sum} = (1 \times w_1) + (2 \times w_2) + b$$

We'll set b = 1 for simplicity.

Effect of Bias:

Adding a bias term essentially allows the decision boundary to shift. Without a bias term, the decision boundary must pass through the origin (0,0). With a bias term, the decision boundary can be shifted in any direction.

This means that the addition of the bias term provides more flexibility to the perceptron to learn and classify data that may not necessarily pass through the origin.

By adjusting the bias term during training, the perceptron can learn to shift the decision boundary as needed to better separate the classes in the data space.

let's continue with the example and compute the output of the perceptron for both scenarios: without using bias and then adding bias.

Perceptron without Bias:

Given the inputs x1 and x2, and weights w1 and w2, the weighted sum is calculated as:

Let's say we have initial weights $w_1 = 0.5$ and $w_2 = -0.5$.

$$\text{weighted sum}_1 = (1 \times 0.5) + (2 \times (-0.5)) = 0.5 - 1 = -0.5$$
$$\text{weighted sum}_2 = (-1 \times 0.5) + (1 \times (-0.5)) = -0.5 - 0.5 = -1$$
$$\text{weighted sum}_3 = (3 \times 0.5) + (-2 \times (-0.5)) = 1.5 + 1 = 2.5$$

Now, let's apply the step function:

- For weighted $\text{sum}_1 = -0.5$: Step function output = 0
- For weighted $\text{sum}_2 = -1$: Step function output = 0
- For weighted $\text{sum}_3 = 2.5$: Step function output = 1

So, without using a bias term, the perceptron classifies the points as follows:
- Point (1, 2): Class 0
- Point (-1, 1): Class 0
- Point (3, -2): Class 1

Now, let's repeat the process with a bias term.

Adding Bias:

Let's b = 1. The new equation for the weighted sum becomes:

$$\text{weighted sum} = (x_1 \times w_1) + (x_2 \times w_2) + b$$

Perceptron with Bias:

$$\text{weighted sum}_1 = (1 \times 0.5) + (2 \times (-0.5)) + 1 = 0.5 - 1 + 1 = 0.5$$
$$\text{weighted sum}_2 = (-1 \times 0.5) + (1 \times (-0.5)) + 1 = -0.5 - 0.5 + 1 = 0$$
$$\text{weighted sum}_3 = (3 \times 0.5) + (-2 \times (-0.5)) + 1 = 1.5 + 1 + 1 = 3.5$$

Now, let's apply the step function:

- For weighted $\text{sum}_1 = 0.5$: Step function output = 1
- For weighted $\text{sum}_2 = 0$: Step function output = 0
- For weighted $\text{sum}_3 = 3.5$: Step function output = 1

So, with the addition of a bias term, the perceptron classifies the points as follows:
- Point (1, 2): Class 1
- Point (-1, 1): Class 0
- Point (3, -2): Class 1

As you can see, the introduction of the bias term has shifted the decision boundary, leading to a different classification for the points. This demonstrates how the presence of a bias term affects the output of a perceptron and allows it to learn more complex patterns in the data.

Advantages of ANN

- ANNs exhibits mapping capabilities, that is, they can map input patterns to their associated output pattern.
- The ANNs learn by examples. Thus, an ANN architecture can be trained with known example of a problem before they are tested for their inference capabilities on unknown instance of the problem. In other words, they can identify new objects previous untrained.
- The ANNs posses the capability to generalize. This is the power to apply in application where exact mathematical model to problem are not possible.

Advantages of ANN

متسامحة مع الاخطاء

- The ANNs are robust system and fault tolerant. They can therefore, recall full patterns from incomplete, partial or noisy patterns.
- The ANNS can process information in parallel, at high speed and in a distributed manner.
- Thus a massively parallel distributed processing system made up of highly interconnected (artificial) neural computing elements having ability to learn and acquire knowledge is possible.

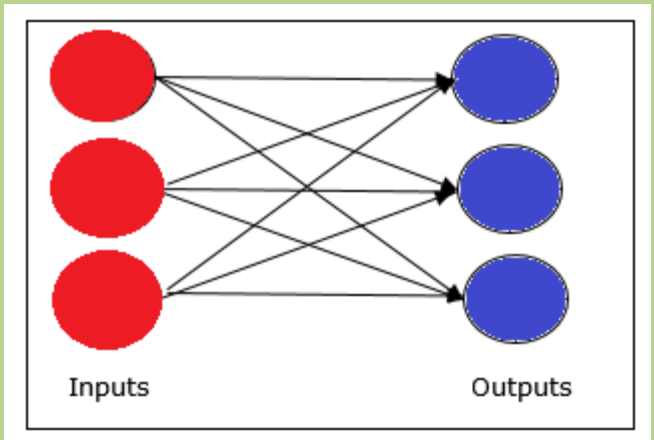
Processing of ANN depends upon the following three building blocks:

1. Network Topology
2. Adjustments of Weights or Learning
3. Activation Functions

1. Network Topology: A network topology is the arrangement of a network along with its nodes and connecting lines. According to the topology, ANN can be classified as the following kinds:

A. Feed forward Network: It is a non-recurrent network having processing units/nodes in layers and all the nodes in a layer are connected with the nodes of the previous layers. The connection has different weights upon them. There is no feedback loop means the signal can only flow in one direction, from input to output. It may be divided into the following two types:

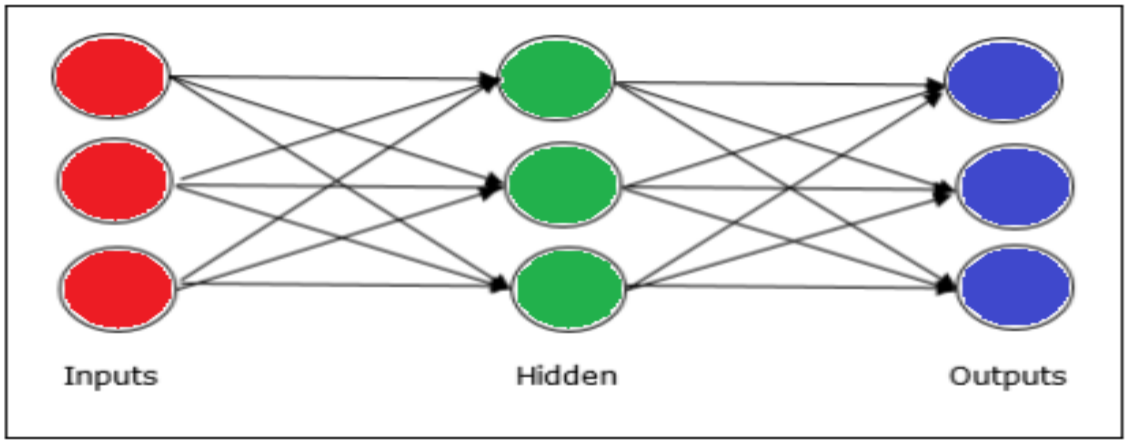
- **Single layer feed forward network:** The concept is of feed forward ANN having only one weighted layer. In other words, we can say the input layer is fully connected to the output layer.

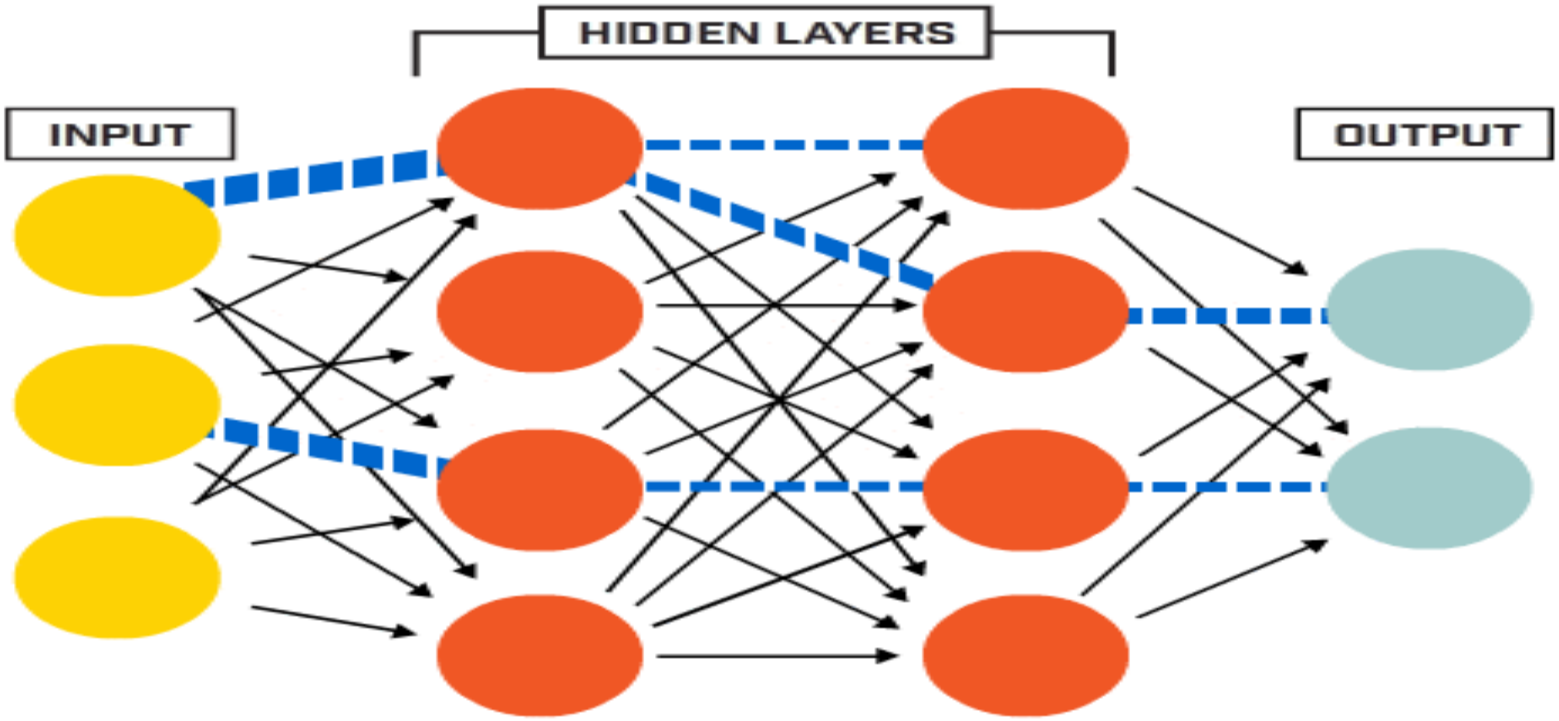


- They are limited to linearly separable problems and can only learn linear decision boundaries.

- Often used for binary classification tasks.

- **Multilayer feed forward network:** The concept is of feed forward ANN having more than one weighted layer. As this network has one or more layers between the input and the output layer, it is called hidden layers.





- They can learn complex non-linear relationships between inputs and outputs.

- Activation functions are typically applied to the output of each neuron to introduce non-linearity into the model.

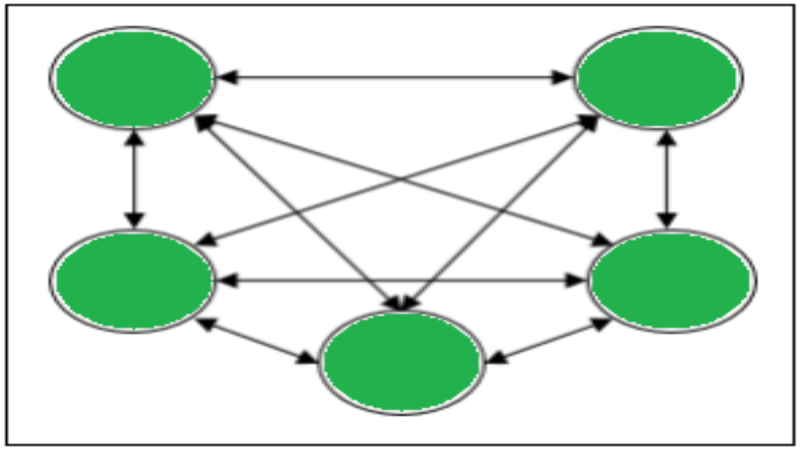
- MLPs are universal function approximators, meaning they can approximate any function given enough neurons and appropriate activation functions.

-Widely used for various tasks including regression, classification, and function approximation.

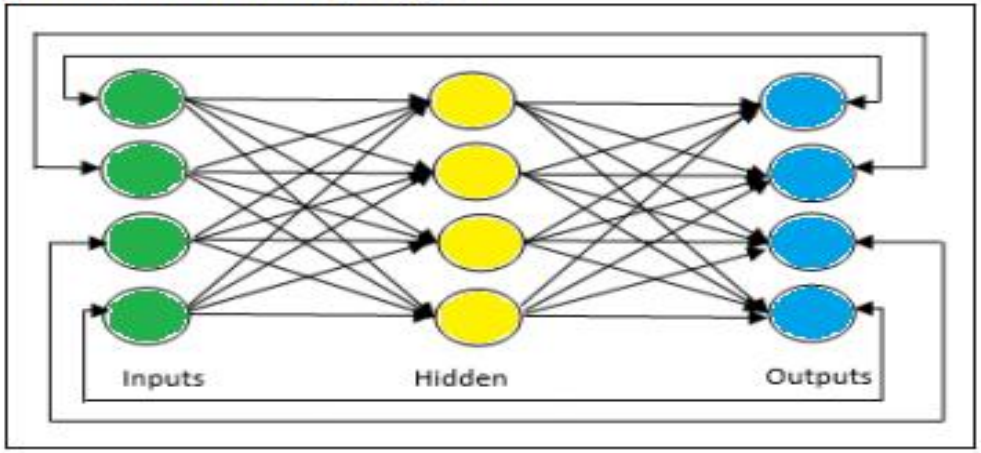
- يمكن استخدامها لتحديد إذا كانت صورة تحتوي على كلب أو قطة، لتصنيف البريد الإلكتروني إلى رسائل غير مرغوب فيها ورسائل هامة، أو لتقدير سعر منزل استنادًا إلى عدة متغيرات مثل المساحة والموقع وعمر المنزل.

B. Feedback Network: As the name suggests, a feedback network has feedback paths, which means the signal can flow in both directions using loops. This makes it a non-linear dynamic system, which changes continuously until it reaches a state of equilibrium. It may be divided into the following types:

- **Recurrent networks:** They are feedback networks with closed loops. Following are the two types of recurrent networks.
- **Fully recurrent network:** It is the simplest neural network architecture because all nodes are connected to all other nodes and each node works as both input and output.

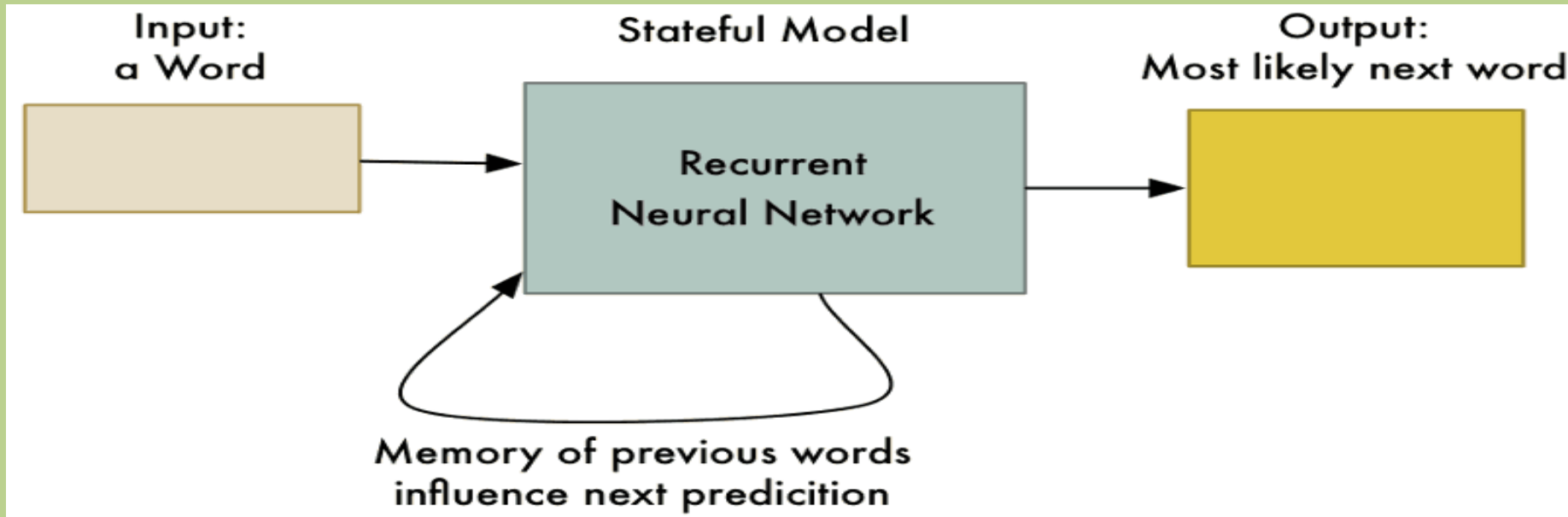


- **Jordan network** – It is a closed loop network in which the output will go to the input again as feedback as shown in the following diagram.

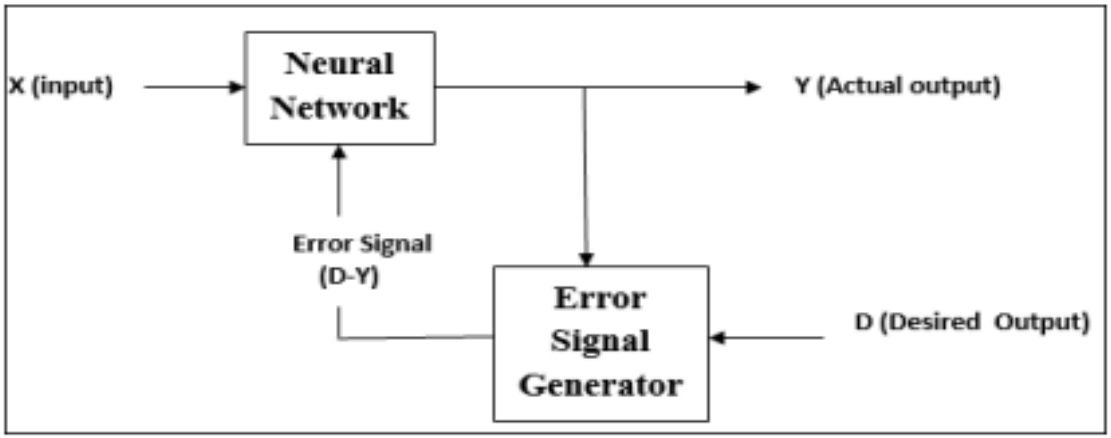


Recurrent Neural Networks (RNNs) are an important class of neural networks specialized in handling sequential data, where there is a temporal relationship between inputs. RNNs are particularly useful in tasks such as text analysis and machine translation, music composition, stock prediction, and many other applications where context changes over time.

RNNs typically consist of recurrent neural units connected to each other to form a loop or sequence. This means they can take current inputs along with the previous internal state (memory) and combine them together to produce the current output. This design allows RNNs to retain information about previous context and use it in predicting the future.

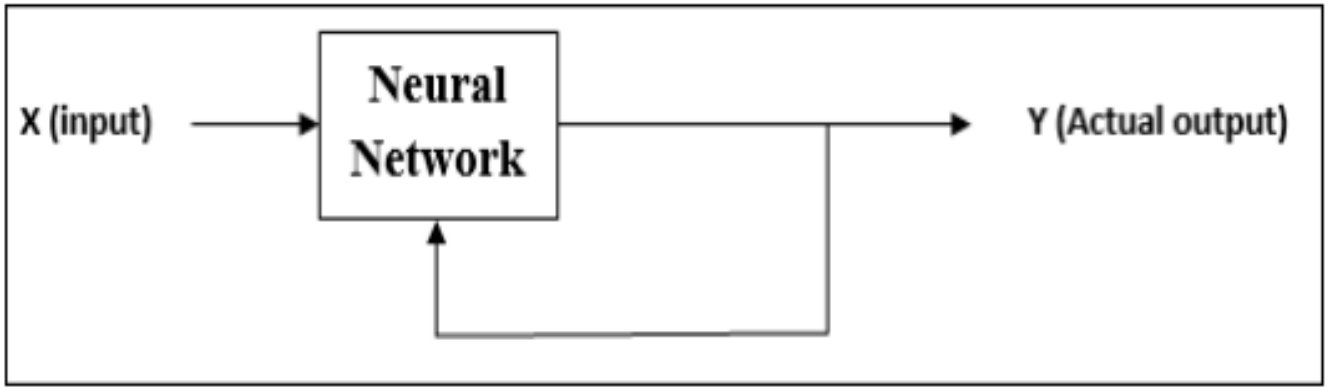


2. Adjustments of Weights or Learning: Learning, in artificial neural network, is the method of modifying the weights of connections between the neurons of a specified network. Learning in ANN can be classified into three categories namely supervised learning, unsupervised learning, and reinforcement learning.

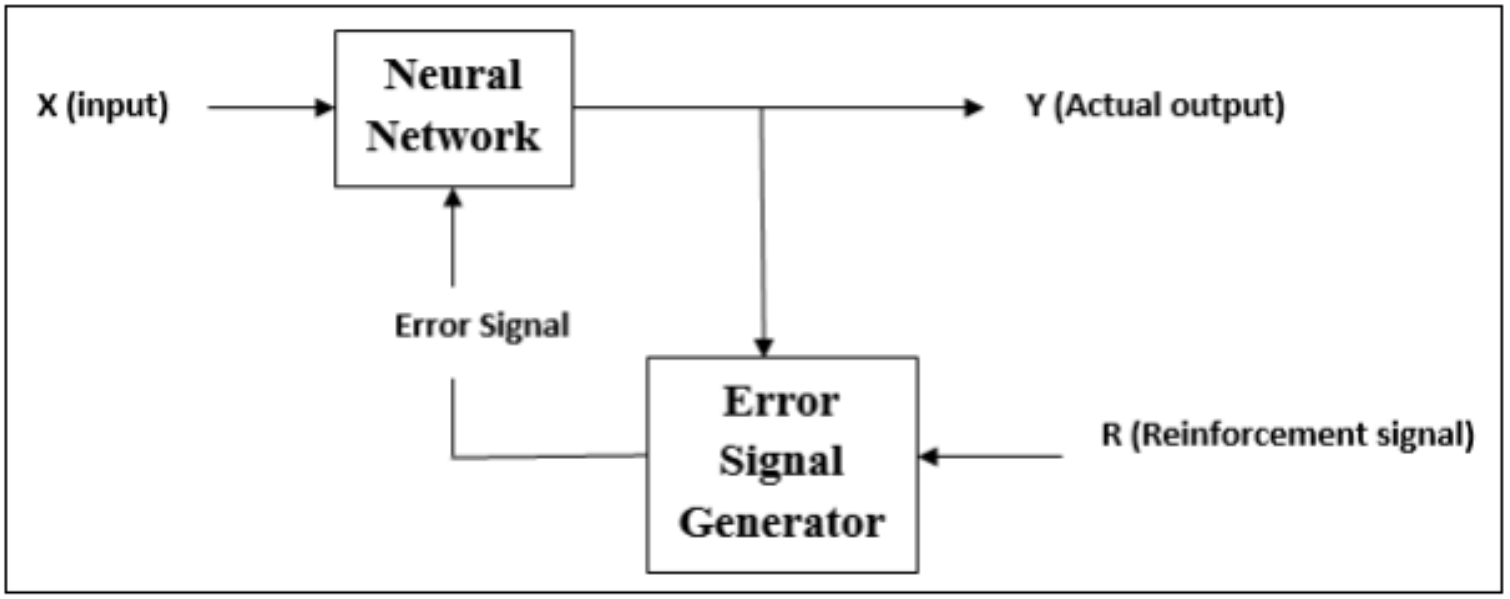


Supervised Learning: As the name suggests, this type of learning is done under the supervision of a teacher. This learning process is dependent. During the training of ANN under supervised learning, the input vector is presented to the network, which will give an output vector. This output vector is compared with the desired output vector. An error signal is generated, if there is a difference between the actual output and the desired output vector. On the basis of this error signal, the weights are adjusted until the actual output is matched with the desired output.

Unsupervised Learning: As the name suggests, this type of learning is done without the supervision of a teacher. This learning process is independent. During the training of ANN under unsupervised learning, the input vectors of similar type are combined to form clusters. When a new input pattern is applied, then the neural network gives an output response indicating the class to which the input pattern belongs. There is no feedback from the environment as to what should be the desired output and if it is correct or incorrect. Hence, in this type of learning, the network itself must discover the patterns and features from the input data, and the relation for the input data over the output.



Reinforcement Learning: As the name suggests, this type of learning is used to reinforce or strengthen the network over some critic information. This learning process is similar to supervised learning, however we might have very less information. During the training of network under reinforcement learning, the network receives some feedback from the environment. This makes it somewhat similar to supervised learning. However, the feedback obtained here is evaluative not instructive, which means there is no teacher as in supervised learning. After receiving the feedback, the network performs adjustments of the weights to get better critic information in future.



BOOLEAN FUNCTION

AND

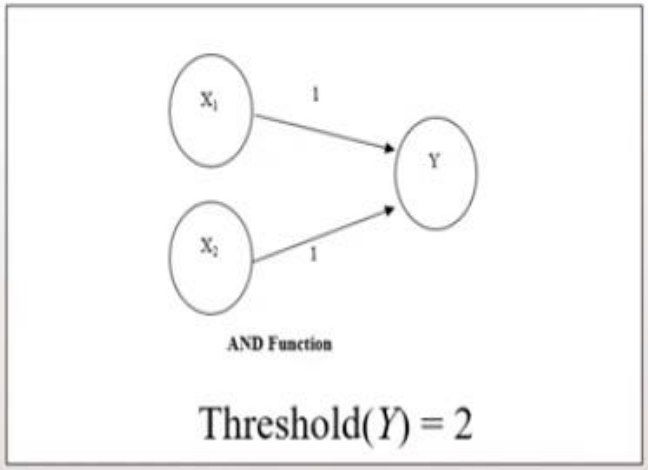
Input		Output
A	B	F = A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR

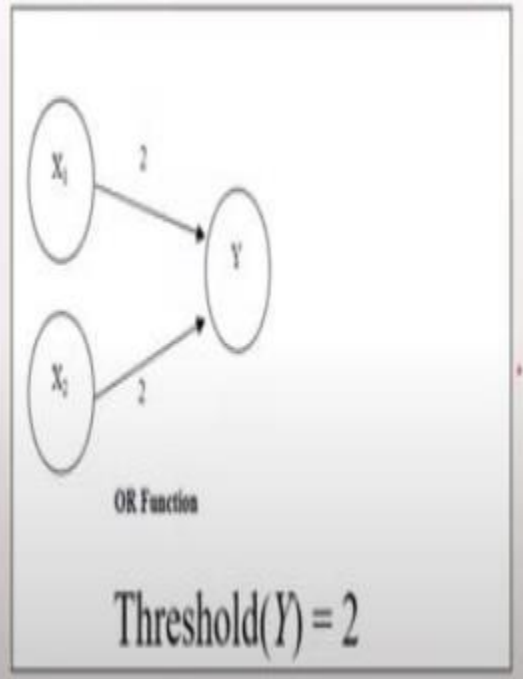
Input		Output
A	B	F = A+B
0	0	0
0	1	1
1	0	1
1	1	1

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

THE FIRST NEURAL NEURAL NETWORKS



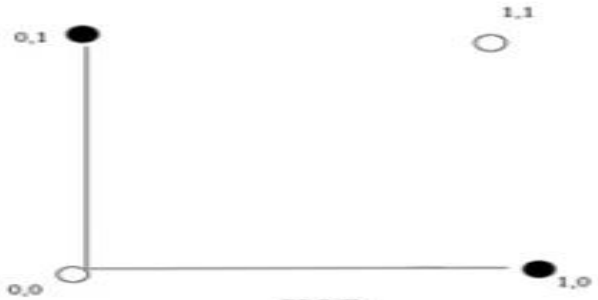
AND		
X1	X2	Y
1	1	1
1	0	0
0	1	0
0	0	0



OR		
X1	X2	Y
1	1	1
1	0	1
0	1	1
0	0	0

What can perceptrons represent?

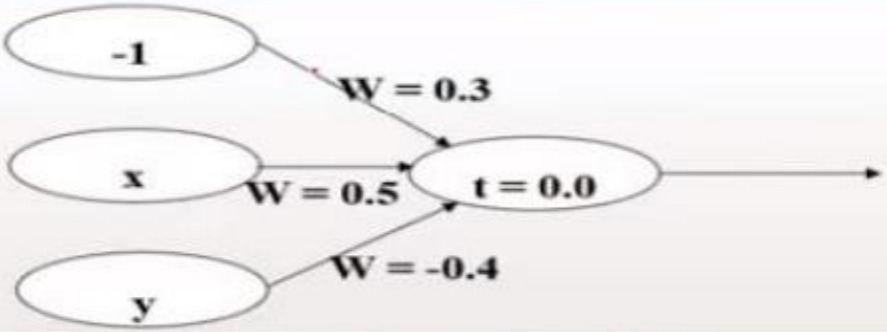
AND



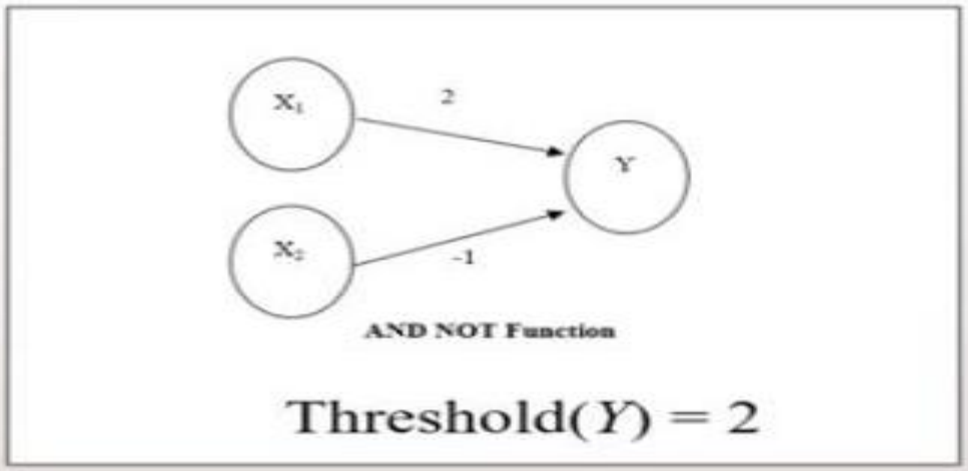
XOR

- **Functions which can be separated in this way are called *Linearly Separable***
- **Only linearly Separable functions can be represented by a perceptron**

TRAINING A PERCEPTRONS



I_1	I_2	I_3	Summation	Output
-1	0	0	(-1*0.3) + (0*0.5) + (0*-0.4) = -0.3	0
-1	0	1	(-1*0.3) + (0*0.5) + (1*-0.4) = -0.7	0
-1	1	0	(-1*0.3) + (1*0.5) + (0*-0.4) = 0.2	1
-1	1	1	(-1*0.3) + (1*0.5) + (1*-0.4) = -0.2	0



AND NOT		
X1	X2	Y
1	1	0
1	0	1
0	1	0
0	0	0

Example #1: Python code that trains to perform AND Gate table. It will continue training until all predictions are correct. And it will prints weights and bias at that point.

Note: The algorithms for training and testing are declared after the code

```
import numpy as np

# Define the AND gate 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.random.rand(2)
bias = np.random.rand(1)
```

```
# Define the learning rate and number of epochs
learning_rate = 0.1
epochs = 100
```

```python
# Training the perceptron
for epoch in range(epochs):
    all_correct = True  # Flag to check if all predictions are correct
    for i in range(len(X)):
        # Forward pass
        input_data = X[i]
        output = np.dot(input_data, weights) + bias

        # Calculate prediction
        prediction = 1 if output > 0 else 0

        # Update weights and bias if there's a misclassification
        if prediction != y[i]:
            error = y[i] - prediction
            weights += learning_rate * error * input_data
            bias += learning_rate * error
            all_correct = False  # Set the flag to False if there's a misclassification

    # Break the loop if all predictions are correct
    if all_correct:
        print(f"First correct prediction at epoch {epoch + 1}: Weight: {weights}, Bias: {bias}")
        break
```

```python
# Test the perceptron
print("\nTesting the perceptron:")
for i in range(len(X)):
    input_data = X[i]
    output = np.dot(input_data, weights) + bias
    prediction = 1 if output > 0 else 0
    print(f"Input: {input_data}, Predicted output: {prediction}")
```

```
weights=
[0.82535858 0.99263662]

bias=
[0.04716405]

First correct prediction at epoch 5: Weight: [0.52535858 0.59263662], Bias: [-0.75283595]

Testing the perceptron:
Input: [0 0], Predicted output: 0
Input: [0 1], Predicted output: 0
Input: [1 0], Predicted output: 0
Input: [1 1], Predicted output: 1
```

Pseudo code for Training Algorithm:

```
1. Start Training
2. Import Libraries (numpy)
3. Define Dataset (X, y)
4. Initialize Weights and Bias (weights, bias)
5. Define Hyperparameters (learning_rate, epochs)
6. Training Loop:
    7. For each Epoch:
        8. Initialize all_correct flag as True
        9. For each Data Point in Dataset:
            10. Forward Pass
            11. Make Prediction
            12. If Prediction is Incorrect:
                13. Update Weights and Bias
                14. Set all_correct to False
        15. If all_correct is True:
            16. Print First Correct Prediction and Weights/Bias
            17. Break Training Loop
18. End Training
```