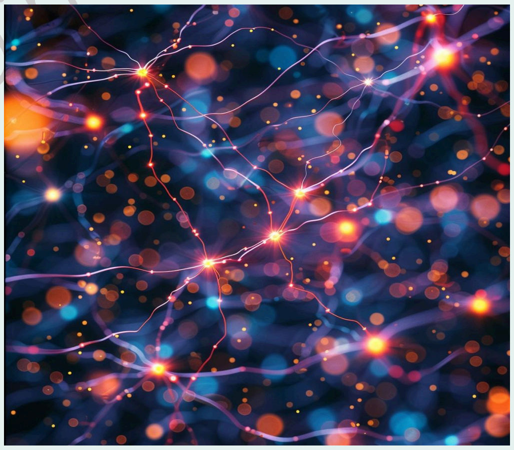
Notes on Artificial Neural Network: Part 1

Definition: A neural network is a method in artificial intelligence that teaches computers to process data in a way that is inspired by the human brain. It is a type of machine learning process, called deep learning, that uses interconnected nodes or neurons in a layered structure that resembles the human brain.

When we look at neural network there are three basic things that you will always hear:

1. Neuron
2. Weights
3. Bias

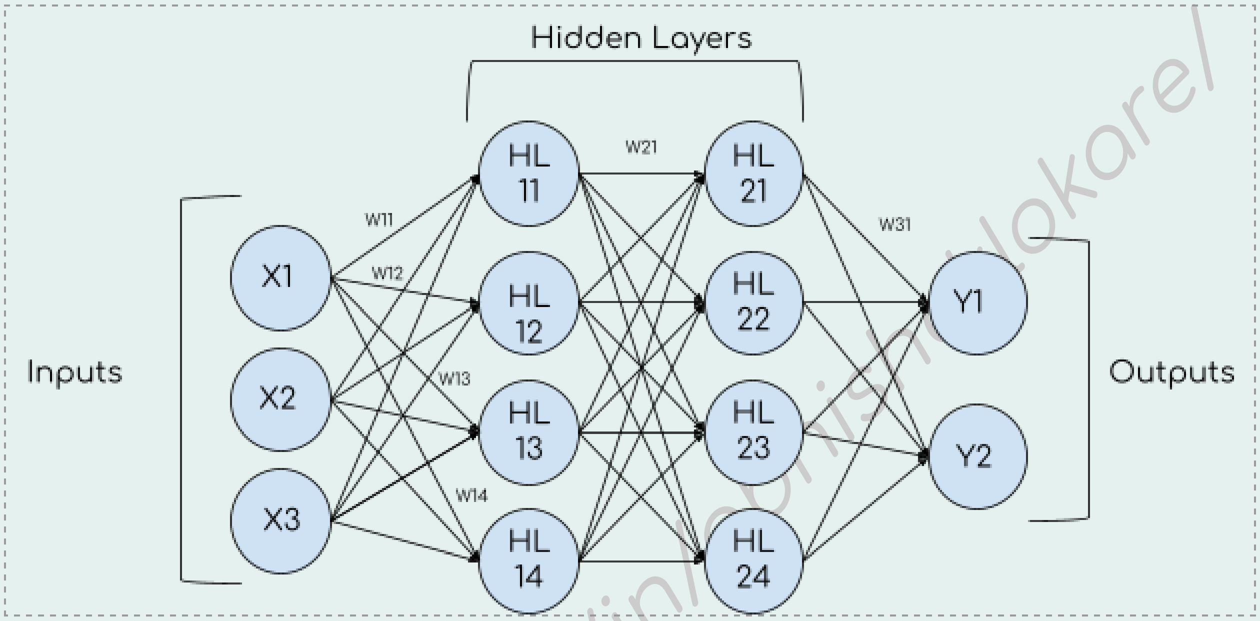
Neuron: Imagine a complex web network of interconnected nodes, each of these nodes represents the computational unit known as a neuron in a neural network.



Weights: Represent the strength of the connection between two neurons and the influence of one neuron's output on another neuron's input.

Bias: Each neuron in the neural network is different, and biases are offsets/constants associated with each neuron. They help in making the neural network flexible. They provide an offset in case the weighted sum is not sufficient.

Consider the neural network below with 3 inputs, 2 outputs and 2 hidden layers with 4 neurons each.



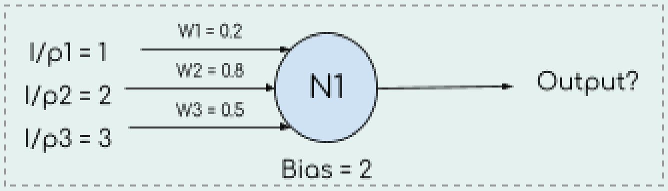
For the above neural network:

1. Total Weight => All the interconnections that are possible
   => 3 * 4 + 4 * 4 + 4 * 2 = 34
2. Total Biases => Addition of all neurons in the network
   => 3 + 4 + 4 + 2 = 13
3. Total Parameters => Total Weight + Biases
   => 34 + 13 = 47

Understanding how neurons work:

1. **Single neuron**

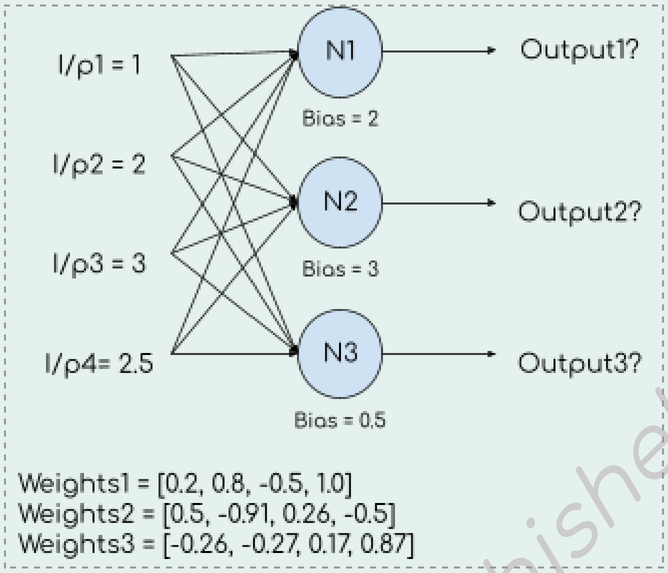
   Consider this example where a single neuron has to process 3 different inputs



   Output = Ip1 * W1 + Ip2 * W2 + Ip3 * W3 + Bias
   Output = 1 * 0.2 + 2 * 0.8 + 3 * 0.5 + 2 = **2.3**

2. **Multiple neurons/Layer**

Consider this example where there are 4 inputs feeding a neural network to get 3 outputs.


I/p1 = 1
I/p2 = 2
I/p3 = 3
I/p4= 2.5
N1
Bias = 2
N2
Bias = 3
N3
Bias = 0.5
Output1?
Output2?
Output3?
Weights1 = [0.2, 0.8, -0.5, 1.0]
Weights2 = [0.5, -0.91, 0.26, -0.5]
Weights3 = [-0.26, -0.27, 0.17, 0.87]

To calculate the output we apply the same formula that we saw in the example of a single neuron above.

Output = [1 * 0.2 + 2 * 0.8 + 3 * -0.5 + 2.5 * 1 + 2,
1 * 0.5 + 2 * -0.91 + 3 * 0.26 + 2.5 * -0.5 + 3,
1 * -0.26 + 2 * -0.27 + 3 * 0.17 + 2.5 * 0.87 + 0.5]

Output = **[4.8, 1.21, 2.385]**

<u>**Understanding shapes:**</u>

#	Example	Shape	Name in Python	Name in Numpy	Name in Maths
1	[1, 2, 3, 4]	(4,)	List	1D Array	Vector
2	[[1, 2, 3, 4], [5, 6, 7, 8]]	(2, 4)	List of List	2D Array	Matrix/List of Vectors
3	[[1, 2, 3, 4], [5, 6, 7, 8]] [[1, 2, 3, 4], [5, 6, 7, 8]] [[1, 2, 3, 4], [5, 6, 7, 8]]	(3, 2, 4)	List of List of List	3D Array	Matrices

<u>**Coding neurons using NumPy:**</u>

Above we saw the manual way of doing calculations, with Numpy you can use the dot product to do these calculations with ease.

1. **Single neuron**

   Let's consider a neuron with 4 Inputs and following weights and bias.

```
import numpy as np

inputs = [1.0, 2.0, 3.0, 2.5]
weights = [0.2, 0.8, -0.5, 1.0]
bias = 2.0

output = np.dot(inputs, weights) + bias
print(output) ##output = 4.8
```

   Internally it will perform similar calculations that we saw earlier:
   output = np.dot([1.0, 2.0, 3.0, 2.5], [0.2, 0.8, -0.5, 1.0]) + 2.0
   = 1.0 * 0.2 + 2.0 * 0.8 + 3.0 * -0.5 + 2.5 * 1.0 + 2.0
   = 4.8

2. Multiple neurons/Layer

   Let's consider a neural network layer with 4 Inputs and 3 outputs with the following weights and biases.

```
import numpy as np

inputs = [1, 2, 3, 2.5]

weights = [[0.2, 0.8, -0.5, 1.0],
          [0.5, -0.91, 0.26, -0.5],
          [-0.26, -0.27, 0.17, 0.87]]

biases = [2, 3, 0.5]

output = np.dot(inputs, np.array(weights).T) + biases ## output = [4.8   1.21  2.385]
```

   The weights must be transposed to do the dot product or, it will throw a shape mismatch error.
   Inputs shape = (4,1); Weights shape = (3, 4) → This combination will not work
   Inputs shape = (4,1); Weights shape = (4, 3) → This combination will work for dot product

   Internally it will perform similar calculations that we saw earlier:

   output = np.dot(inputs, np.array(weights).T) + biases
   = [np.dot(inputs, weights[0]), np.dot(inputs, weights[1]), np.dot(inputs, weights[2])] + biases
   = [2.8, 1.79, 1.885] + [2, 3, 0.5]

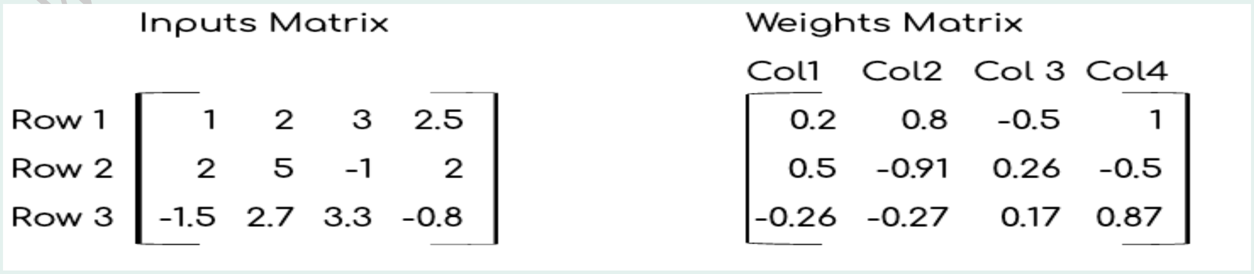
= [4.8   1.21 2.385]

Coding a neural network:

Remember, a neural network is a sequence of layers where one layer's output is an input to another.

```
import numpy as np

inputs = [[1, 2, 3, 2.5],
          [2.0, 5.0, -1.0, 2.0],
          [-1.5, 2.7, 3.3, -0.8]]

weights = [[0.2, 0.8, -0.5, 1.0],
           [0.5, -0.91, 0.26, -0.5],
           [-0.26, -0.27, 0.17, 0.87]]

biases = [2, 3, 0.5]

weights2 = [[0.1, -0.14, 0.5],
            [-0.5, 0.12, -0.33],
            [-0.44, 0.73, -0.13]]
biases2 = [-1, 2, -0.5]

layer1_output = np.dot(inputs, np.array(weights).T) + biases

layer2_output = np.dot(layer1_output, np.array(weights2).T) + biases2

print(layer2_output)

'''
layer2_output
[[ 0.5031  -1.04185 -2.03875]
 [ 0.2434  -2.7332  -5.7633 ]
 [-0.99314  1.41254 -0.35655]]
'''
```

Understanding the math behind dot product:


Inputs Matrix
Weights Matrix

$$\begin{array}{l|cccc|} & & & & \\ \text{Row 1} & 1 & 2 & 3 & 2.5 \\ \text{Row 2} & 2 & 5 & -1 & 2 \\ \text{Row 3} & -1.5 & 2.7 & 3.3 & -0.8 \end{array}$$

$$\begin{array}{cccc} \text{Col1} & \text{Col2} & \text{Col 3} & \text{Col4} \\ \hline 0.2 & 0.8 & -0.5 & 1 \\ 0.5 & -0.91 & 0.26 & -0.5 \\ -0.26 & -0.27 & 0.17 & 0.87 \end{array}$$

This matrix multiplication will not be possible as the Input matrix shape is (3, 4) and the Weights matrix shape is also (3, 4). Highlighted numbers don't match. Hence we transpose weights to do their successful multiplication.

Inputs Matrix

$$\begin{array}{l} \text{Row 1} \\ \text{Row 2} \\ \text{Row 3} \end{array} \begin{bmatrix} 1 & 2 & 3 & 2.5 \\ 2 & 5 & -1 & 2 \\ -1.5 & 2.7 & 3.3 & -0.8 \end{bmatrix}$$

Weights Matrix Transposed

$$\begin{array}{ccc} \text{Col1} & \text{Col2} & \text{Col 3} \end{array}$$
$$\begin{bmatrix} 0.2 & 0.5 & -0.26 \\ 0.8 & -0.91 & -0.27 \\ -0.5 & 0.26 & 0.17 \\ 1 & -0.5 & 0.87 \end{bmatrix}$$

Input matrix shape = (3, 4) and Weights matrix shape (4, 3))match, the final matrix will have the shape of (3, 3)

Output Matrix (3, 3)

Row1 * Col 1

$$\begin{array}{l} \text{Row1 * Col 1} \\ \text{Row2 * Col 1} \\ \text{Row3 * Col 1} \end{array} \begin{bmatrix} 2.8 & -1.79 & 1.885 \\ 6.9 & -4.81 & -0.3 \\ -0.59 & -1.949 & -0.474 \end{bmatrix} \begin{array}{l} \text{Row1 * Col 3} \\ \text{Row2 * Col 3} \\ \text{Row3 * Col 3} \end{array}$$

Row3 * Col 2

Output matrix cel 1 = row 1 of input matrix * col 1 of weights matrix. The other cells will be calculated similarly as shown in the image above.