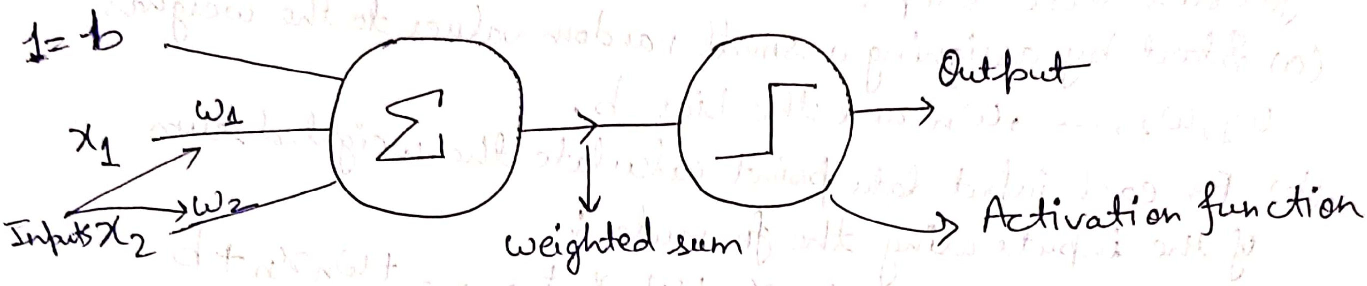
Perceptron

A perceptron is the simplest type of neural network model, it forms the basic building block for many deep learning models. A perceptron mimics a biological neuron, taking multiple inputs, applying a linear combination, and producing a binary output (either 0 or 1).



Structure of a Perceptron

① Input Layer: A perceptron takes a number of inputs $x_1, x_2, ..., x_n$. These can be features of data points in a dataset.

② Weights: Each input is assigned a weight $w_1, w_2, ..., w_n$, which indicates the input's importance in determining the output. These weights are initially random but are adjusted during training.

③ Weighted Sum (Linear Combination): The perceptron calculates the weighted sum of its inputs like—

$$\boxed{Z = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b}$$

where b is the bias, another learnable parameter that helps to shift the output of perceptron to fit the data better.

④ Activation function: After computing the weighted sum, the perceptron applies an activation function to decide the output. Originally, the activation function used in perceptron was a step function:

$$\text{output} = \begin{cases} 1, & \text{if } Z \geq 0 \\ 0, & \text{if } Z < 0 \end{cases}$$

This step f^n makes the perceptron output binary values, either 1 or 0.

The basic operation of the perceptron can be summarized as — (a) Compute a weighted sum of the inputs.
(b) Pass it through an activation (step) function.
(C) Output a binary result.

Training a Perceptron : Training a Perceptron involves adjusting its weights and bias based on the input data to make accurate predictions. The training process generally follows these steps →

(a) Start by assigning a small random values to the weights $\omega_1, \omega_2, \ldots, \omega_n$ and the bias b.

(b) For each input data point calculate the weighted sum of the inputs using the formula :

$$Z = \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3 + \ldots + \omega_n x_n + b$$

(c) Apply the step function (or another activation function) to produce the output.

(d) Compare the predicted output with the actual target value (label). If the perceptron's output is correct, no adjustment is made. If it's incorrect, calculate the error :

$$\text{erron} = \text{actual output} - \text{predicted output}$$

(e) Adjust the weights and bias based on the error using the Perceptron Learning Rule :

$$\omega_i \leftarrow \omega_i + \eta . \text{error} . x_i$$

$$b \leftarrow b + \eta . \text{error}$$

where η is the **learning rate**, a small positive number that controls how much to adjust the weights. The weights are updated in such a way that they push the output of the perceptron towards the correct class.

(f) Repeat the process, continue training for multiple epochs until the perceptron correctly classifies all the training data or until the error reaches an acceptable level.

Ques. How can we solve the AND gate problem using the perceptron?

Ans→ The truth table of an AND gate looks like—

x_1	x_2	T
0	0	0
0	1	0
1	0	0
1	1	1

We need to initialize a perceptron, so that it has two inputs x_1 & x_2, having some random weights lets say 0·1 for both x_1 & x_2 and a bias b initialized also with 0·1, we are going to use an activation function such that, the output y_{pred} is 1 if $z>0$, otherwise it's 0, i.e.,

$$y_{pred} = \begin{cases} 0, & \text{if } z \leq 0 \\ 1, & \text{if } z > 0 \end{cases}$$

Epoch 1: $w_1 = 0{\cdot}1,\ w_2 = 0{\cdot}1,\ b = 0{\cdot}1$ & $\eta = 0{\cdot}1$

for first row in truth table
$$z = w_1x_1 + w_2x_2 + b = 0{\cdot}1 * 0 + 0{\cdot}1 * 0 + 0{\cdot}1 = 0{\cdot}1$$
since, $z > 0 \Rightarrow y_{pred} = 1$

Error $= y_{True} - y_{pred} = 0 - 1 = -1$

Now, we need to update the weights using the perceptron learning rule.

$$w_1 = \underbrace{0{\cdot}1}_{w_1} + \underbrace{0{\cdot}1}_{\text{learning rate}} * \underbrace{(-1)}_{\text{error}} * \underbrace{0}_{x_1} = 0{\cdot}1$$

$$w_2 = 0{\cdot}1 + 0{\cdot}1 * (-1) * 0 = 0{\cdot}1$$

$$b = 0{\cdot}1 + 0{\cdot}1 * (-1) = 0$$

Hence, the updated weights are $\Rightarrow w_1 = 0{\cdot}1,\ w_2 = 0{\cdot}1$ & $b = 0$

for second row in truth table
$$z = 0{\cdot}1 * 0 + 0{\cdot}1 * 1 + 0 = 0{\cdot}1$$
since, $z > 0 \Rightarrow y_{pred} = 1$

Error $= y_{True} - y_{pred} = 0 - 1 = -1$

Now, again we need to update the weights as we made an error—

$$w_1 = 0{\cdot}1 + 0{\cdot}1 * (-1) * 0 = 0{\cdot}1$$

$$w_2 = 0{\cdot}1 + 0{\cdot}1 * (-1) * 1 = 0$$

$$b = 0 + 0{\cdot}1 * (-1) = -0{\cdot}1$$

Updated weights are $\Rightarrow w_1 = 0{\cdot}1,\ w_2 = 0$ & $b = -0{\cdot}1$

for third row in truth table
$$\begin{cases} z = 0.1*1 + 0*0 + (-0.1) = 0 \\ \text{since, } z = 0,\ y_{pred} = 0 \\ \text{Error} = y_{True} - y_{pred} = 0 - 0 = 0 \end{cases}$$

Thus, we do not need to update the weights since error is 0.

for fourth row in truth table
$$\begin{cases} z = 0.1*1 + 0*1 + (-0.1) = 0 \\ \text{since, } z = 0,\ y_{pred} = 0 \\ \text{Error} = y_{True} - y_{pred} = 1 - 0 = 1 \end{cases}$$

We again need to update the weights.

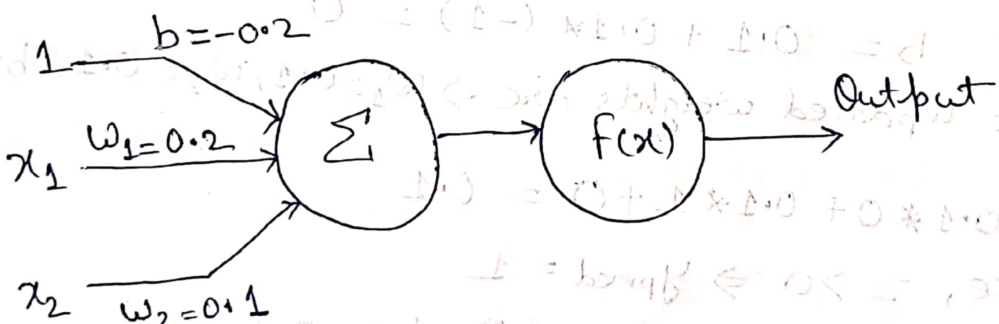
$$w_1 = 0.1 + 0.1*1*1 = 0.2$$
$$w_2 = 0 + 0.1*1*1 = 0.1$$
$$b = -0.1 + 0.1*1 = 0$$

Epoch 2:

first row in truth table
$$\begin{cases} z = 0.2*0 + 0.1*0 + 0 = 0 \\ \text{Since } z = 0,\ y_{pred} = 0 \\ \text{Error} = 0 - 0 = 0 \end{cases}$$

we need to continue the above steps till all rows give error 0, thus we will get optimal values for w_1, w_2 & b.

Since, Error = 0, we can say these values of w_1, w_2 & b are our optimal weights. For $w_1 = 0.2$, $w_2 = 0.1$ & $b = -0.2$, we will satisfy all the conditions of and gate.



Ques. 2 Implement the OR gate using the perceptron.

Ans→ Let's initialize $w_1 = 0.1$, $w_2 = 0.1$, $b = 0.1$ & $\eta = 0.1$

x_1	x_2	Y True
0	0	0
0	1	1
1	0	1
1	1	1

Epoch 1: For first row in truth table —

$$z = 0.1*0 + 0.1*0 + 0.1 = 0.1$$
$$\text{Since, } z > 0 \Rightarrow y_{pred} = 1$$
$$\text{Error} = Y_{true} - Y_{pred} = 0 - 1 = -1$$
$$w_1 = 0.1 + 0.1*(-1)*0 = 0.1$$
$$w_2 = 0.1 + 0.1*(-1)*0 = 0.1$$
$$b = 0.1 + (0.1)*(-1) = 0$$

For second row in truth table - $\left\{ \begin{array}{l} z = 0.1*0 + 0.1*1 + 0 = 0.1 \\ \text{Since } z>0, \; y_{pred} = 1 \\ \text{Error} = 1-1 = 0 \end{array} \right.$

For third row in truth table $\left\{ \begin{array}{l} z = 0.1*1 + 0.1*0 + 0 = 0.1 \\ y_{pred} = 1 \\ \text{Error} = 1-1 = 0 \end{array} \right.$

For fourth row in truth table $\left\{ \begin{array}{l} z = 0.1*1 + 0.1*1 + 0 = 0.2 \\ y_{pred} = 1 \\ \text{Error} = 0 \end{array} \right.$

If we continue the same process for next epoch, we will notice all the conditions of OR gate will be satified for weights $w_1 = 0.1$, $w_2 = 0.1$ & $b = 0$, thus these are our optimal weigts

Limitations of Perceptron

(1) A perceptron can only solve problems that are linearly separable, meaning it can only classify data points if they can be separated by a straight line (or a hyperplane) in higher dimensions. XOR problem cannot be solved by perceptron.

(2) Perceptron struggles with learning complex patterns like those found in non-linear problems, which limits their application to real world problems where data is rarely linearly separable.

(3) The original perceptron uses a step activation function, which only gives binary outputs. This is insufficient for problems requiring multi-class classification or continuous outputs.

Ques. What is the significance of **bias** term in perceptrons?

Ans→ The bias in perceptrons acts as a threshold that shifts the decision boundary, allowing the model to better fit the data. Without the bias the decision boundary would always pass through the origin in perceptrons, limiting the flexibility of model. The bias term ensures that even when the inputs are zero, the neuron can produce a non-zero output, enhancing the models ability to capture complex patterns and relationship in data.