Artificial Intelligence

Neural Network

Show the mathematical working of Artificial Neural Network by taking the case in figure below. First two columns are the input values for X1 and X2 and the third column is the desired output.

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

Learning rate = 0.2
Threshold = 0.5
Actual output = W1X1+W2X2

Next weight adjustment = Wn+ΔWn

Change in weight = ΔWn = learning rate * (desired output- output) * Xn

Show complete iterations for acquiring the desired output?

x1	x2	w1	w2	d	y	Δw1	Δw2

$y = x_1 w_1 + x_2 w_2$ | $\Delta w_1 = \eta(d-a) \cdot x_1$ | $w_{n1} = w_0 + \Delta w_n$ | $\eta = 0.2$ $\theta = 0.5$

$y = x_1 w_1 + x_2 w_2$ | $\Delta w_1 = \eta(d-a)\cdot x_1$ | $w_n = w_0 + \Delta w_n$ | $\eta = 0.2$ $\theta = 0.5$ R.W

day / date:

x_1	x_2	w_1	w_2	d	y	Δw_1	Δw_2	R.W
0	0	1	1	0	0	0	0	$0.2(0-1)\cdot 1$ -0.2
0	1	1	1	0	1	0	-0.2	$1.8 > 0.5 = 1$
1	0	1	0.8	1	1	0	0	
1	1	1	0.8	1	1	0	0	
0	0	1	0.8	0	0	0	0	$0.8 > 0.5$
0	1	1	0.8	0	1	0	-0.2	$0.2(0-1)\cdot 1$
1	0	1	0.6	1	1	0	0	-0.2
1	1	1	0.6	1	1	0	0	$1 + 0.6 = 1.6 > 0.5 = 1$
0	0	1	0.6	0	0	0	0	$0.6 > 0.5 = 1$
0	1	1	0.6	0	1	0	-0.2	
1	0	1	0.4	1	1	0	0	$1 > 0.5 = 1$
1	1	1	0.4	1	1	0	0	$1 + 0.4 = 1.4 > 0.5 = 1$
0	0	1	0.4	0	0	0	0	$0.4 < 0.5 = 0$
0	1	1	0.4	0	0	0	0	
1	0	1	0.4	1	1	0	0	$1 + .4 = 1.4 > 0.5 = 1$
1	1	1	0.4	1	1	0	0	

$(1-0)$ $y = 1 \times 1 + 0 \times 0.4 = 1 + 0 = 1 > 0.5 = 1$ $w_1 = 1$, $w_2 = 0.4$

$(0-1)$ $y = 0 \times 1 + 1 \times 0.4 = 0.4 < 0.5 = 0$

$(0-0)$ $y = 0 \times 1 + 0 \times 0.4 = 0$

$(1-1)$ $y = 1 \times 1 + 1 \times 0.4 = 1 + 0.4 = 1.4 > 0.5 = 1$

Q / Part No.

Perceptron learning rule:

dataset:

x_1	x_2	x_3	label
0	0	0	1
1	0	1	0

$$\begin{array}{cccc} w_0 & w_1 & w_2 & w_3 \\ b & x_1 & x_2 & x_3 \end{array}$$

weight vector: $[-2 \; 2 \; 1 \; 2]$

α = learning rate: 0.6

activation: step function: step functions means

$0 \quad x < 0$

$1 \quad x \geq 0$

weight adjustment $= w_n + \Delta w_n$

$\Delta w_n = \alpha * (\text{actual} - \text{predicted}) * \text{input}$

$y: \; b.w_0 + x_1 w_1 + x_2 w_2 + x_3 . w_3$

$= 1.(-2) + 0(2) + 0(1) + (0)(2)$

$= -2 + 0 + 0 + 0$

$\Sigma = -2$

act $(-2) = 0$

$b = 0$

$\Delta w_0 = 0.6 * (1 - 0) 1$

$\Delta w_0 = 0.6 * 1 = 1.6$

$w_{1new} = w_{1old} + \Delta w$

$= -2 + 1.6$

$= -0.4$

Q / Part No.

Rough Work

$\Delta w_1 = 0.6 * (1-0) . 0$

$\Delta w_1 = 0$

$w_{1new} = w_{1old} + \Delta w_1$

$= .2 + 0$

$= 2$

$\Delta w_2 = 0.6 * (1-0) . 1$

$= 0.6 * 1$

$= 0.6$

$w_{2new} = w_{2old} + \Delta w_2$

$= 1 + 0.6$

$= 1.6$

$\Delta w_3 = 0.6 . (1-0) 2$

$= 0.6 \times (2)$

$= 1.2$

$w_{3new} = w_{3old} + \Delta w_3$

$= 2 + 1.2$

$= 3.2$

using the updated weights for the 2nd sample

$b.w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$

$1.(-0.4) + 1(2) + 0(2) + 0(3.2)$

$= -0.4 + 2 + 0$

$= 1.6$

act $(1.6) = 1$

$w_0 = -0.4$

$w_1 = 2$

$w_2 = 1.6$

$w_3 = 3.2$

b, x_1, x_2

1 1 0

Q / Part No. Rough Work

Delta Rule

Dataset:

$$\begin{array}{ccc} x_1 & x_2 & x_3 \\ \left[\begin{array}{c}1\\0\end{array}\right. & \begin{array}{c}0\\0\end{array} & \left.\begin{array}{c}1\\0\end{array}\right] \end{array} \quad \begin{array}{c} b \text{ weight} \\ w_0 \\ \begin{bmatrix}2\end{bmatrix} \end{array} \quad \begin{array}{c} \text{label} \\ \begin{bmatrix}0\\1\end{bmatrix} \end{array}$$

unit step func
$x < 0$ 0
$x \geq 0$ 1

weight $\begin{array}{ccc} w_1 & w_2 & w_3 \\ -2 & 2 & 1 \end{array}$

$= b_1 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$

$= 1(2) + 1(-2) + 0(2) + 1(1)$

$= 2 - \cancel{2} + \cancel{2} + 1$

$= 3$

$\hat{y}_1 = act(3) = 1$

2nd sample

$b_2 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$

$= 1(2) + 0(-2) + 0(2) + 0(1)$

$= 2$

$act(2) = 1$

$\hat{y}_2 = 1$

$\boxed{\alpha = 0.2}$

for

$y_i - \hat{y}_i$	$(y-\hat{y}).b$	$(y-\hat{y}).x_1$	$(y-\hat{y}).x_2$	$(y-\hat{y}).x_3$
$0 - 1 = -1$	$1 \times (-1) = -1$	$-1 * 1 = -1$	$0 * -1 = 0$	$-1 * 1 = -1$
$1 - 1 = 0$	$1 \times 0 = 0$	$0 \times 0 = 0$	$0 * 0 = 0$	$0 * 0 = 0$
$\Sigma = -1$	$\Sigma = -1$	$\Sigma = -1$	$\Sigma\ 0$	$\Sigma = -1$

$w_{0_{new}} = w_0 + \Sigma\alpha(y_i - \hat{y}_i).b = 2 + 0.2(-1) = 1.2$

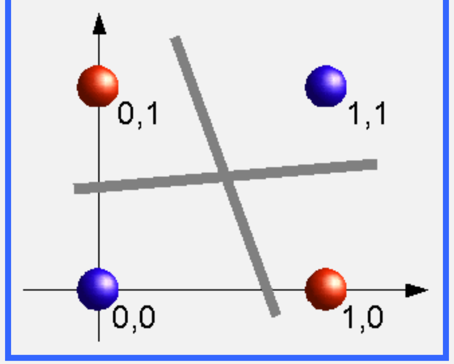
$w_{1_{new}} = w_1 + \Sigma\alpha(y_i - \hat{y}_i).x_1 = -2 + 0.2(-1) = -2.2$

$w_{2\,new} = w_2 + \Sigma\alpha(y_i - \hat{y}_i).x_2 = 2 + 0.2(0) = 2$

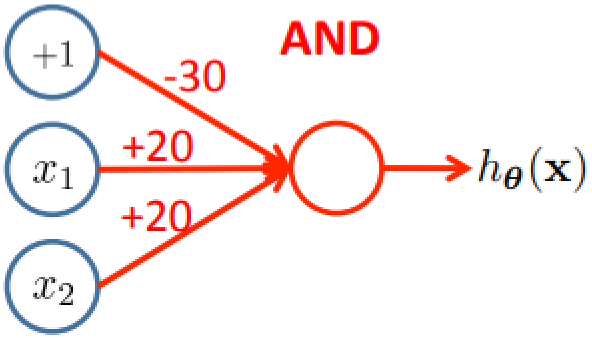
$w_{3\,new} = w_3 + \Sigma\alpha(y_i - \hat{y}_i).x_3 = 1 + 0.2(-1) = 0.8$

$$\begin{array}{c} w_0 \\ w_1 \\ w_2 \\ w_3 \end{array} = \begin{bmatrix} 1.2 \\ -2.2 \\ 2 \\ 0.8 \end{bmatrix}$$

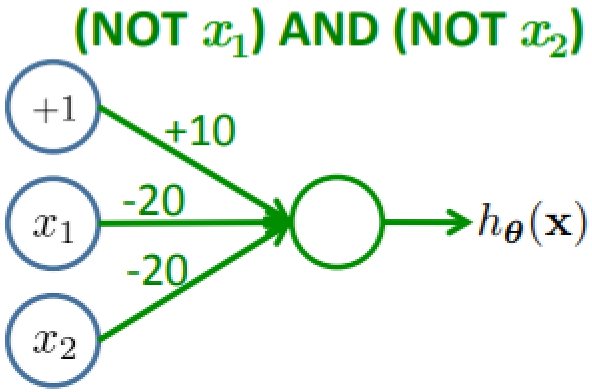
XOR Gate



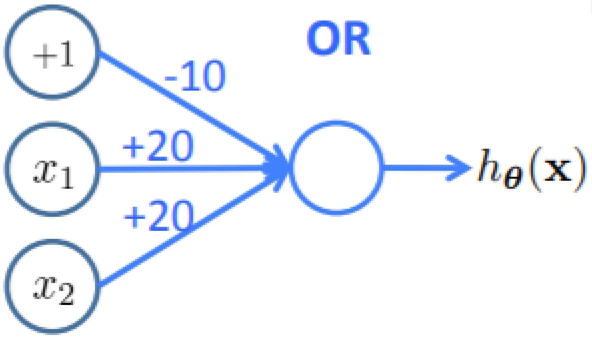
0,1
1,1
0,0
1,0



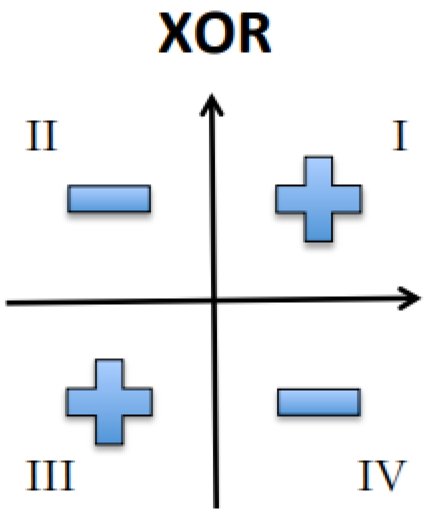
AND
$+1$
x_1
x_2
-30
+20
+20
$h_\theta(\mathbf{x})$



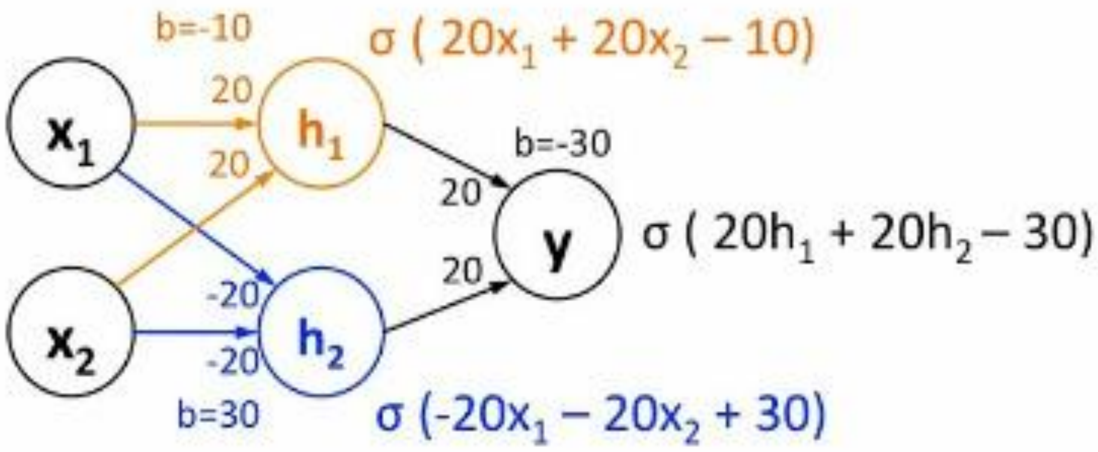
(NOT x_1) AND (NOT x_2)
$+1$
x_1
x_2
+10
-20
-20
$h_\theta(\mathbf{x})$



OR
$+1$
x_1
x_2
-10
+20
+20
$h_\theta(\mathbf{x})$

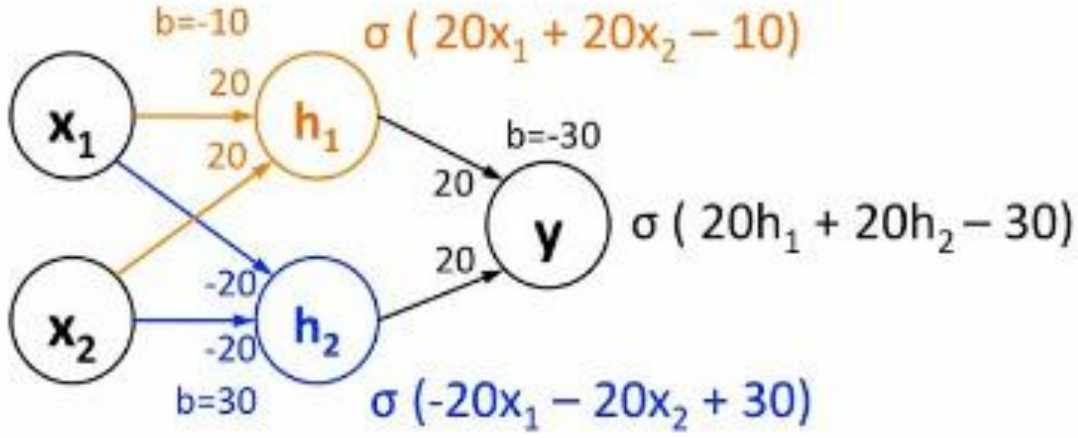


XOR
II
I
III
IV



b=-10
σ ($20x_1 + 20x_2 - 10$)
x_1
20
20
h_1
b=-30
20
y
σ ($20h_1 + 20h_2 - 30$)
20
x_2
-20
-20
h_2
b=30
σ ($-20x_1 - 20x_2 + 30$)

XOR Gate



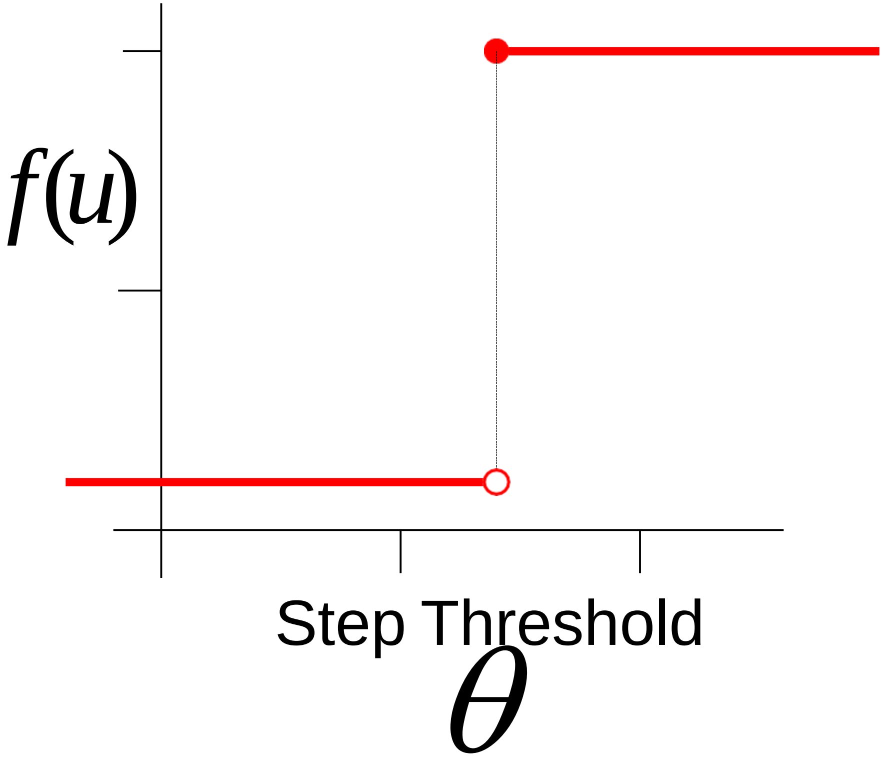
$\sigma(20*0 + 20*0 - 10) \approx 0$	$\sigma(-20*0 - 20*0 + 30) \approx 1$	$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$	$\sigma(-20*1 - 20*1 + 30) \approx 0$	$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*0 + 20*1 - 10) \approx 1$	$\sigma(-20*0 - 20*1 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$	$\sigma(-20*1 - 20*0 + 30) \approx 1$	$\sigma(20*1 + 20*1 - 30) \approx 1$

The Perceptron: Threshold Activation Function

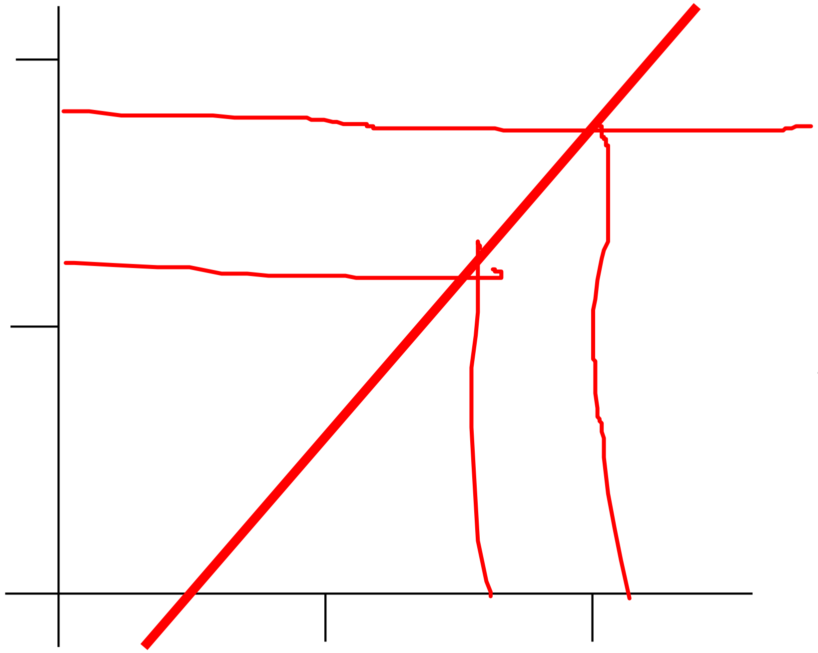
- Binary classifier functions
- Threshold activation function

$$u = \sum_{n=1}^{N} w_n x_n$$

$$f(u) = \begin{cases} 0 & u < \theta \\ 1 & u \geq \theta \end{cases}$$



Linear Activation functions



Linear

$$f(u) = u = \sum_{n=1}^{N} w_n x_n$$

Output is scaled sum of inputs

Nonlinear Activation Functions

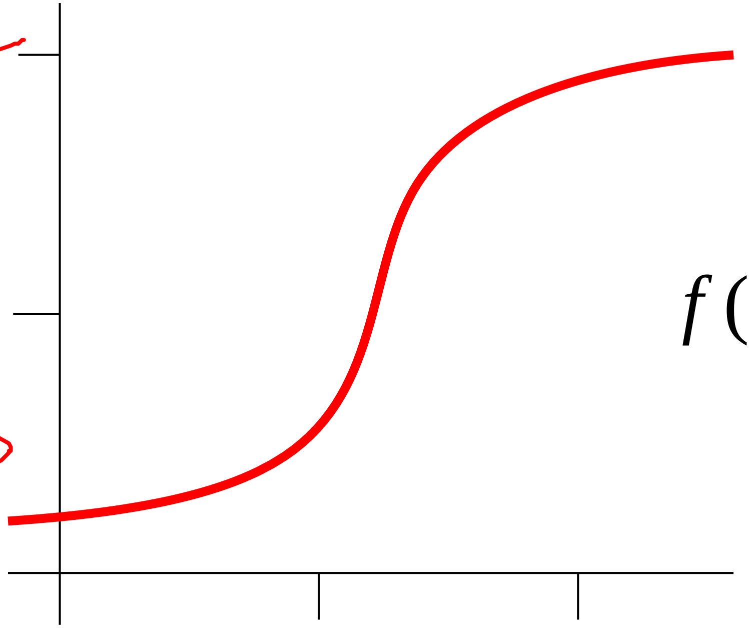
softmax

0.07

0.4

1

Sigmoid Neuron unit function



Sigmoid

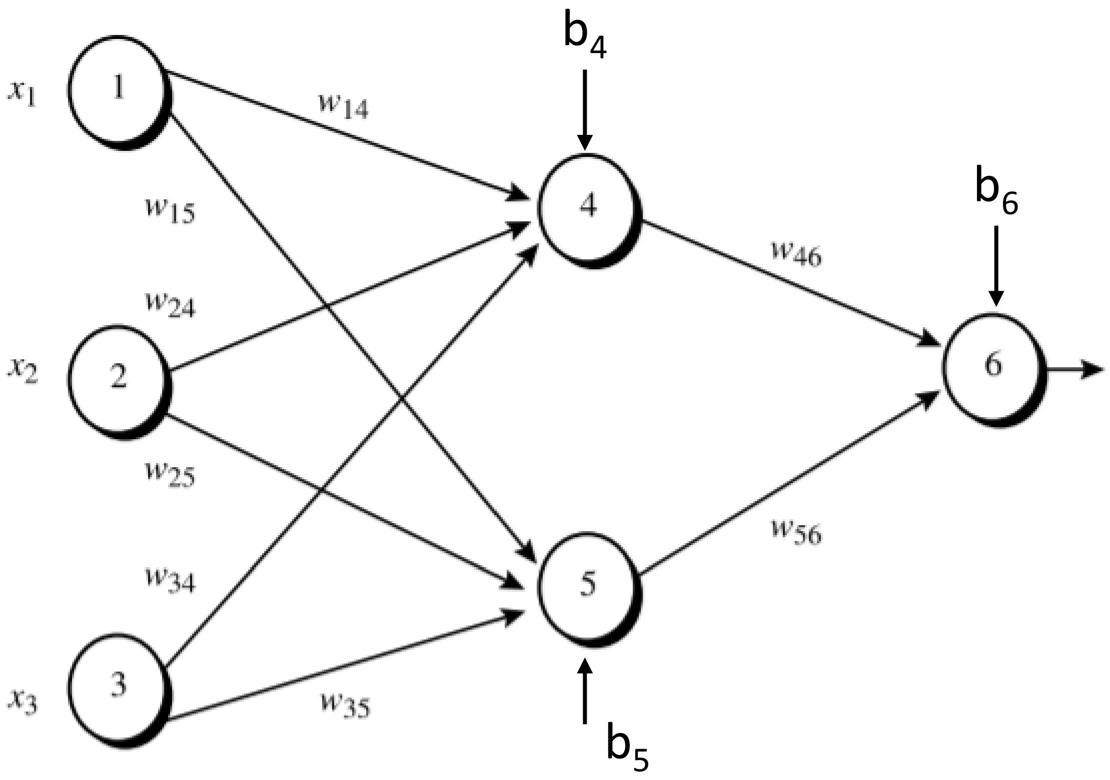
$$f(u) = \frac{1}{1+e^{-u}}$$

Artificial Neural Network Learning

- <u>How does a perceptron learn the appropriate weights?</u>

Example

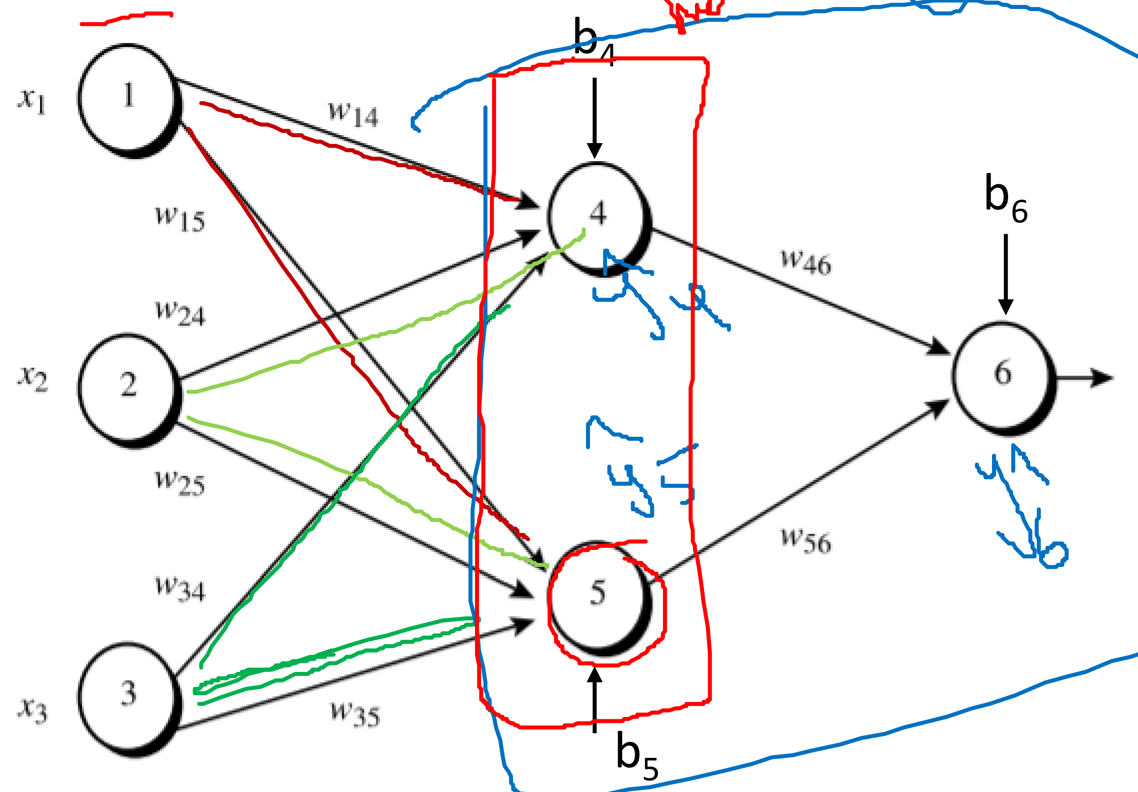
x1	x2	x3	w14	w15	w24	w25	w34	w35	w46	w56	b4	b5	b6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1



https://theneuralblog.com/forward-pass-backpropagation-example/

Example

x1	x2	x3	w14	w15	w24	w25	w34	w35	w46	w56	b4	b5	b6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1



Feed-Forward

$$\sigma = \frac{1}{1+e^{-y}}$$

$$y_4 = (x_1 \cdot w_{14} + x_2 \cdot w_{24} + x_3 \cdot w_{34}) + b_4$$

$$= 1 \cdot 0.2 + 0 \cdot 0.4 + 1 \cdot (-0.5) + (-0.4)$$

$$\hat{y}_4 = \sigma(-0.7) \Rightarrow 0.332$$

$$y_5 = x_1 \cdot w_{15} + x_2 \cdot w_{25} + x_3 \cdot w_{35} + b_5$$

$$= 0.1$$

$$\hat{y}_5 = \sigma(0.1) = 0.524$$

$$y_6 = \hat{y}_4 \cdot w_{46} + \hat{y}_5 \cdot w_{56} + b_6$$

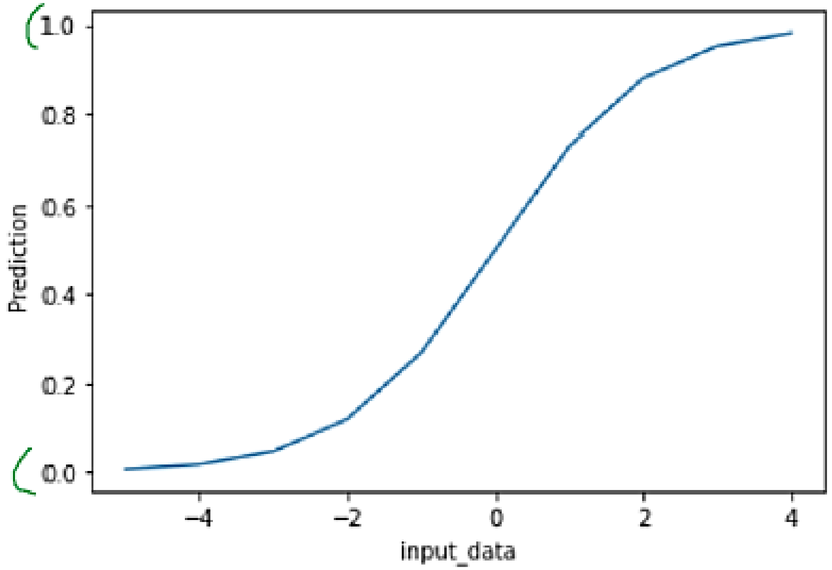
$$\hat{y}_6 = \sigma(0.60) = 0.5$$

- Error = ½ $(\text{Target} – \text{output})^2$
- Target =1
- Predicted= 0.5
- Error= ½ $(1 – 0.5)^2$

Activation Function

- Sigmoid [0,1]
  - Squashing Function

```
from math import exp
from matplotlib import pyplot as plt

def sigmoid(x):
    return 1.0 / (1.0 + exp(-x))
input_data=[]
for data in range(-5, 5):
    input_data.append(data)
prediction=[]
for data in input_data:
    prediction.append(sigmoid(data))
plt.xlabel('input_data')
plt.ylabel('Prediction')
plt.plot(input_data, prediction)
plt.show()
```

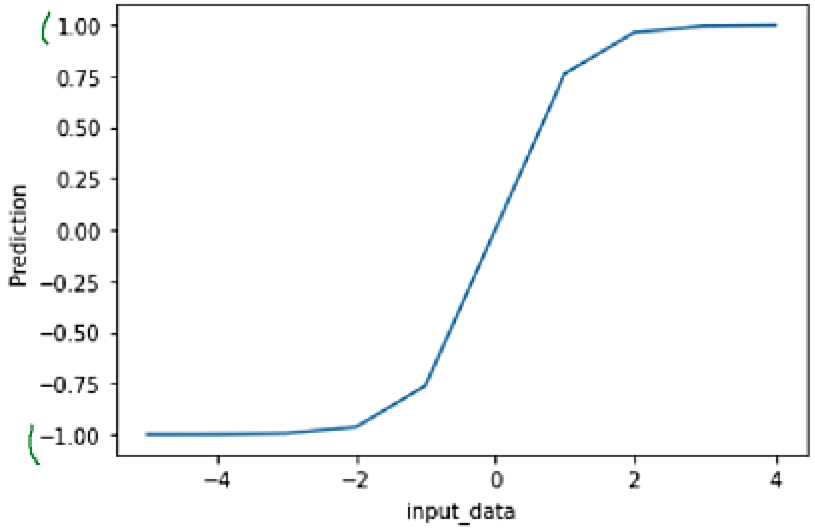


Used in 1990's

Activation Function

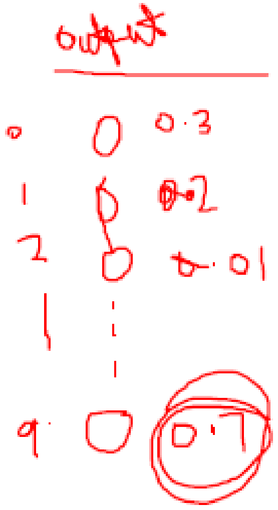
- Tanh
  - [-1,+1]

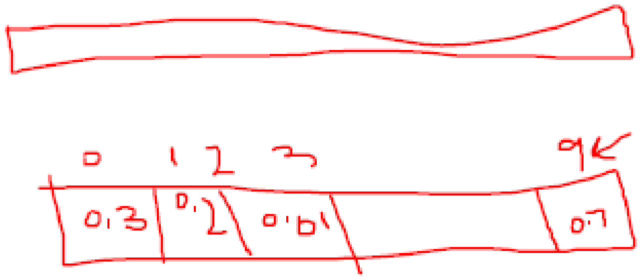
```
from math import exp
from matplotlib import pyplot as plt

def tanh(x):
    return (exp(x) - exp(-x)) / (exp(x) + exp(-x))
input_data=[]
for data in range(-5, 5):
    input_data.append(data)
prediction=[]
for data in input_data:
    prediction.append(tanh(data))
plt.xlabel('input_data')
plt.ylabel('Prediction')
plt.plot(input_data, prediction)
plt.show()
```



Used till 2010's

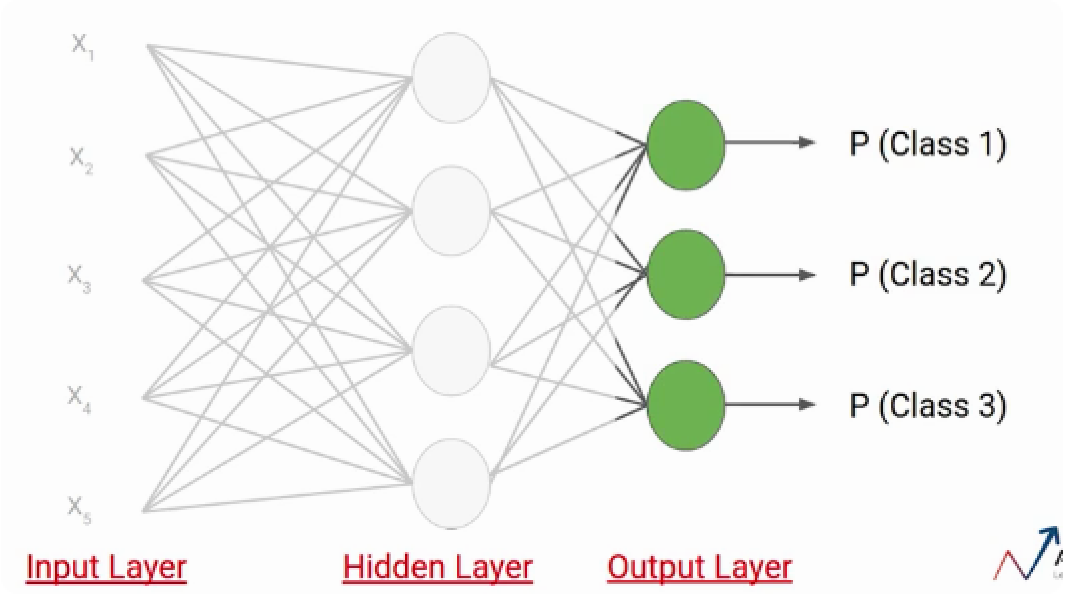
One-hot Encoding



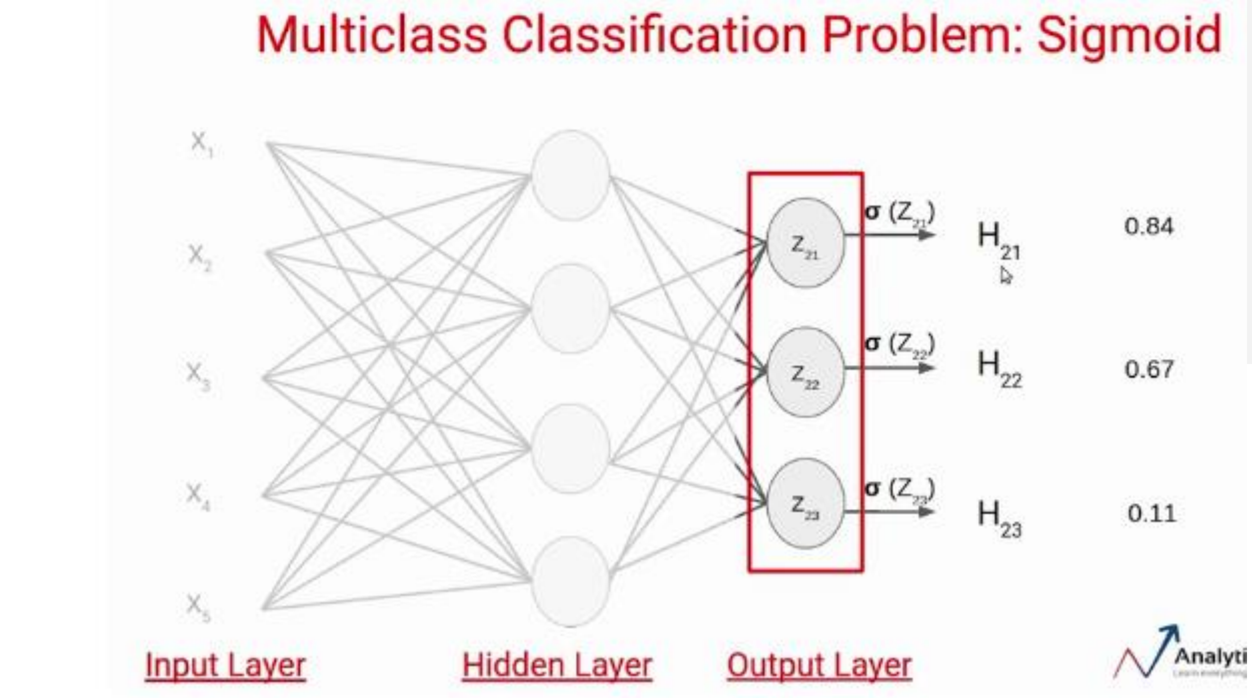


- To make the ground truth labels compatible with the softmax output of the neural network, you one-hot encode these labels.
- This means that you represent each ground truth label as a binary vector with a 1 in the position corresponding to the true class and 0s in all other positions.
- For example, if the ground truth label is “9" in the MNIST dataset, it is one-hot encoded as follows:
- Ground Truth Label "7" → One-Hot Encoding: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

Now let's create a simple neural network for this problem. Here, we have an Input layer with five neurons as we have five features in the dataset. Next, we have one hidden layer which has four neurons. Each of these neurons uses inputs, weights, and biases here to calculate a value which is represented as Zij here.



https://www.analyticsvidhya.com/blog/2021/04/introduction-to-softmax-for-neural-network/



There are two problems in this case-

First, if we apply a thresh-hold of say 0.5, this network says the input data point belongs to two classes. Secondly, these probability values are independent of each other. That means the probability that the data point belongs to class 1 does not take into account the probability of the other two classes.

This is the reason the sigmoid activation function is not preferred in multi-class classification problems.

Instead of using sigmoid, we will use the Softmax activation function in the output layer in the above example. The Softmax activation function calculates the relative probabilities. That means it uses the value of Z21, Z22, Z23 to determine the final probability value.

Let's see how the softmax activation function actually works. Similar to the sigmoid activation function the SoftMax function returns the probability of each class. Here is the equation for the SoftMax activation function.

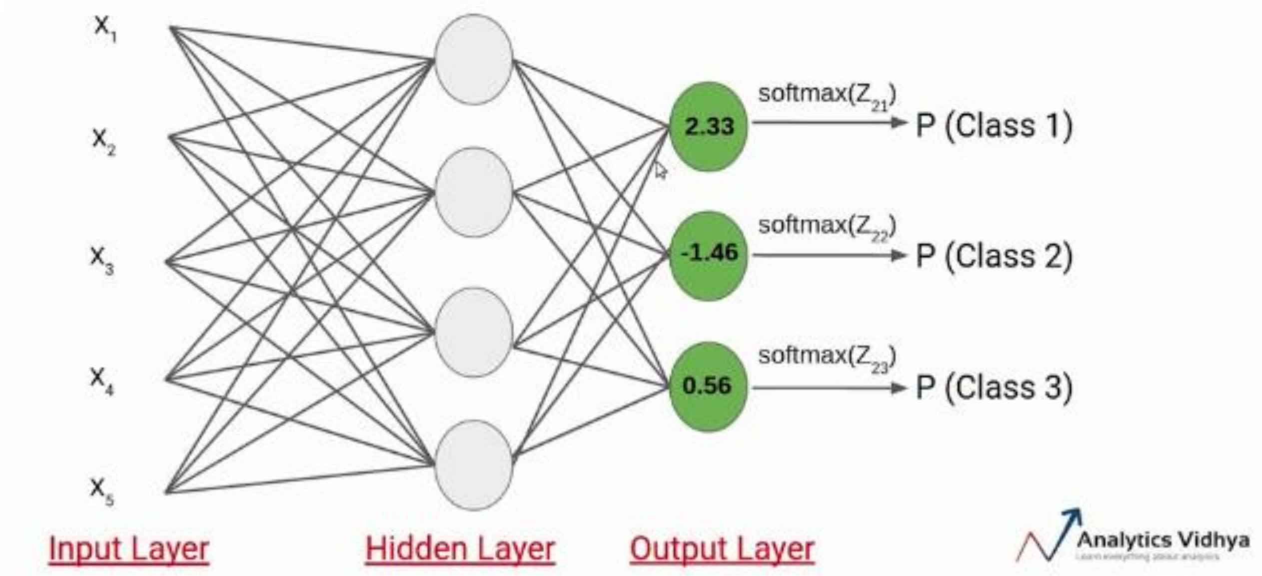
$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Here, the Z represents the values from the neurons of the output layer. The exponential acts as the non-linear function. Later these values are divided by the sum of exponential values in order to normalize and then convert them into probabilities.

Note that, when the number of classes is two, it becomes the same as the sigmoid activation function. In other words, sigmoid is simply a variant of the Softmax function. If you want to learn more about this concept, refer to this link.

Let's understand with a simple example how the softmax works, We have the following neural network.

Suppose the value of Z21, Z22, Z23 comes out to be 2.33, -1.46, and 0.56 respectively. Now the SoftMax activation function is applied to each of these neurons and the following values are generated.



These are the probability values that a data point belonging to the respective classes. Note that, the sum of the probabilities, in this case, is equal to 1.

Example :

$$2.33 \longrightarrow P\,(\text{Class } 1) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

$$-1.46 \longrightarrow P\,(\text{Class } 2) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \longrightarrow P\,(\text{Class } 3) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$