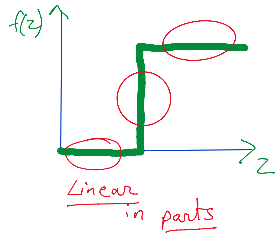
ANN Part-1

Tuesday, May 2, 2023 11:15 AM

* The threshold function in McCulloh and Pitts Model is a non-linear function.
* Activation function in Deep Learning can be both linear as well as non-linear, however while working in real-time we prefer non-linear activation function because linear activation functions are not that powerful.

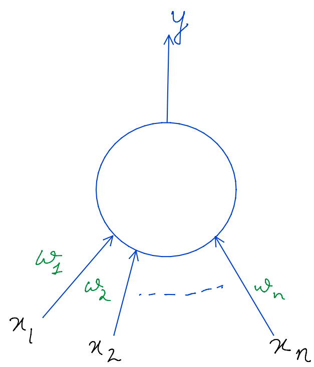
* The threshold function is overall non-linear, but is piece-wise linear



Note : Given a graph
- → which is Linear ⇒ it is also piece-wise linear
- → which is piece-wise Linear $\not\Rightarrow$ It will be Linear

---

* **Perceptron Model** → Inputs are Real Values $\{x_i \in R\}$ & $\{y_i \in \{0,1\}\}$



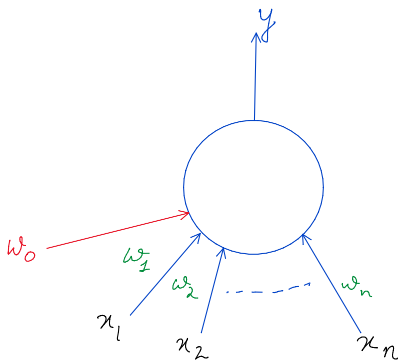
1.) Weighted Sum $= \sum_{i=1}^{n} w_i x_i$

2.) $$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i < T \end{cases}$$ [T → Threshold]

$$\Rightarrow y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i - T \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i - T < 0 \end{cases}$$

Lets define $W_0 = -T$

$$\Rightarrow y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} W_i x_i + W_0 \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} W_i x_i + W_0 < 0 \end{cases}$$



→ Instead of deciding T, we take W_0 as input and it is something that the architecture has to learn.

→ Since W_0 will be a part of input parameters, it won't be chosen by us rather it will be learnt by the model along with the other W_i

{Sometimes W_0 is also represented as 'b' which stands for 'bias'}

* So basically, now the model will select the best threshold value.

$$y = \begin{cases} 1 & \text{if } \underbrace{\sum_{i=1}^{n} W_i x_i + W_0}_{\text{This looks like an equation of line}} \geq 0 & \{\text{Right Side of the Line}\} \\ 0 & \text{if } \sum_{i=1}^{n} W_i x_i + W_0 < 0 & \{\text{Left side of the Line}\} \end{cases}$$

⇒ It is like a linear classifier.

Linear function V/S affine function

→ A general linear function, is like $\sum w_i x_i$. Linear function between vector spaces preserve the vector space structure so in particular they must fix the origin.

→ An affine function is the composition of a linear function with a translation, so while the linear part fixes the origin, the translation can map it somewhere else.

So if we have vector spaces 'v' & 'w' of dimensions 'm' and 'n' respectively and the function is $f: V \to W$

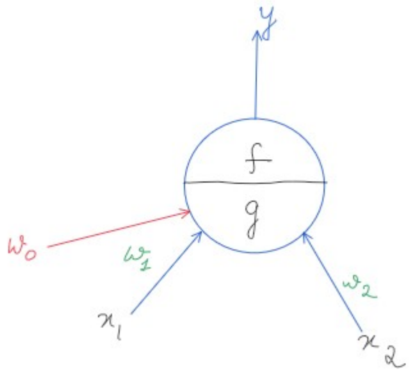
⇒ then f is linear if $f(v) = Av$ for some $n \times m$ matrix 'A'

⇒ the f is affine if $f(v) = Av + b$ for some matrix 'A' and vector b where co-ordinate representations are used with respect to the bases chosen.

OR Function

with Perceptron Model $\quad V \to OR$

x_1	x_2	$x_1 \vee x_2$	$w_1x_1 + w_2x_2 + w_0$	y
0	0	0	w_0	< 0
0	1	1	$w_2 + w_0$	≥ 0
1	0	1	$w_1 + w_0$	≥ 0
1	1	1	$w_1 + w_2 + w_0$	≥ 0



$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + w_0 \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i + w_0 < 0 \end{cases}$$

for $w_0 = -1$, $w_1 = 1$ & $w_2 = 1$ → the above conditions are satisfied.

{ w_0, w_1, w_2 can be found using perceptron learning algorithm or Gradient Descent }

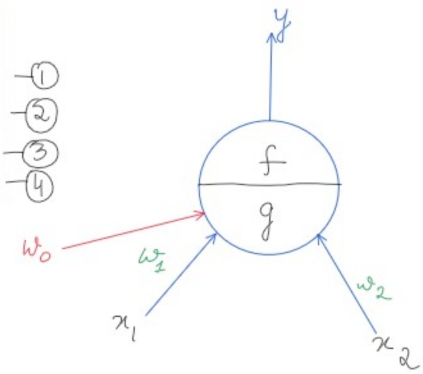
CONCLUSION: There exists a solution in perceptron which can implement OR Function.

↓ choice of weights and bias

XOR Function

with Perceptron Model $\quad \oplus \to XOR$

x_1	x_2	$x_1 \oplus x_2$	$w_1x_1 + w_2x_2 + w_0$	y	
0	0	0	w_0	< 0	— (1)
0	1	1	$w_2 + w_0$	≥ 0	— (2)
1	0	1	$w_1 + w_0$	≥ 0	— (3)
1	1	0	$w_1 + w_2 + w_0$	< 0	— (4)



$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + w_0 \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i + w_0 < 0 \end{cases}$$

Adding eq^n (2) & (3) $\Rightarrow$ $w_1 + w_2 + 2w_0 \geq 0$ — (5)

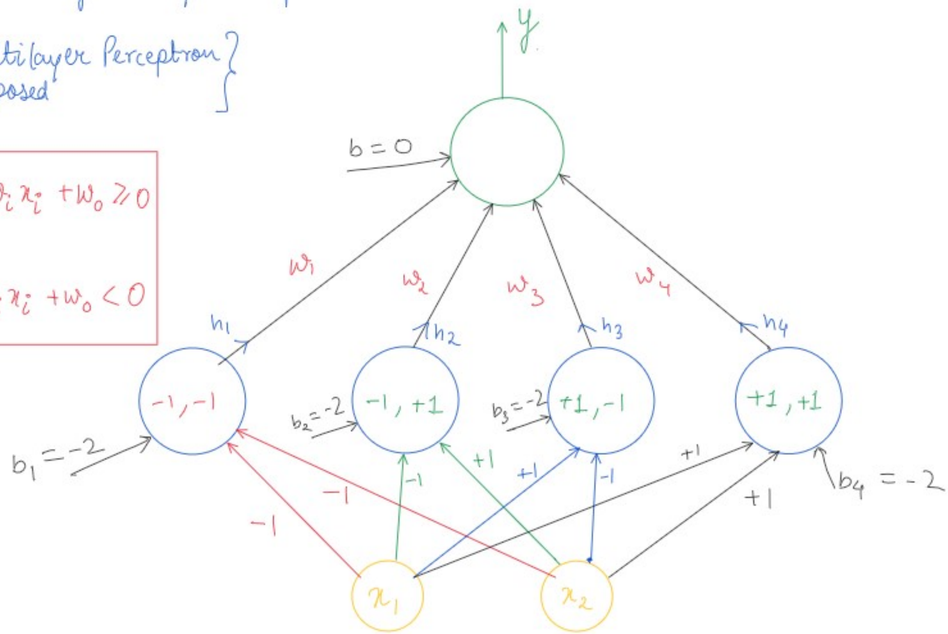
Considering eq^n (4) & (5), it is impossible to satisfy both the equations simultaneously.

XOR function → using Multilayer Perceptron

{This is the first multilayer Perceptron Architecture proposed}

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + w_0 \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i + w_0 < 0 \end{cases}$$

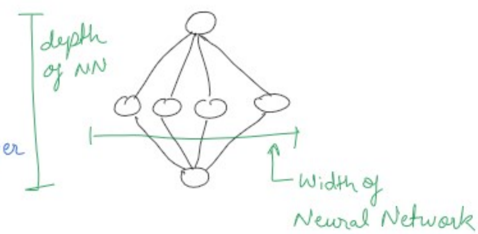
Threshold function



			aggregation funct^n O/P				Threshold f^n O/P				
x_1	x_2	y	h_1	h_2	h_3	h_4	h_1	h_2	h_3	h_4	$y = b + w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4$
-1	-1	0	0	-2	-2	-4	1	0	0	0	$w_1 < 0$
-1	1	1	-2	0	-4	-2	0	1	0	0	$w_2 \geq 0$
1	-1	1	-2	-4	0	-2	0	0	1	0	$w_3 \geq 0$
1	1	0	-4	-2	-2	0	0	0	0	1	$w_4 < 0$

∴ one possible solution → w_1, w_2, w_3, w_4: $-1, 1, 1, -1$

Universal Approximation Theorem



→ Neural Networks are known for being able to compute any function.

→ According to UAT, any feedforward Neural Network with 1 hidden layer and enough breadth can approximate any function.

There are however 2 caviats involved:

1. The neural network will not always give the exact answers but will be very close to the actual output. By increasing or decreasing the number of neurons we can control the function.

2. The functions that can be approximated in the way described must be continuous functions.

eg. for caviat (1) → given a cont. function $f(x)$, we want to compute it within some accuracy $\xi > 0$
UAT guarantees that by using enough neurons, we can always find a neural network whose output $g(x)$ satisfies:

$$|g(x) - f(x)| < \xi \quad \forall x$$

In other words, the approximation will be good to within the desired accuracy for every possible input.

"A feedforward network with a single layer is sufficient to represent any function, but the layer maybe infeasably large, and may fail to learn and generalize correctly."
~ Ian Goodfellow

References:

1) http://neuralnetworksanddeeplearning.com/chap4.html
2) https://towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6#:~:text=The%20Universal%20Approximation%20Theorem%20states,be%20able%20to%20approximate%20it.