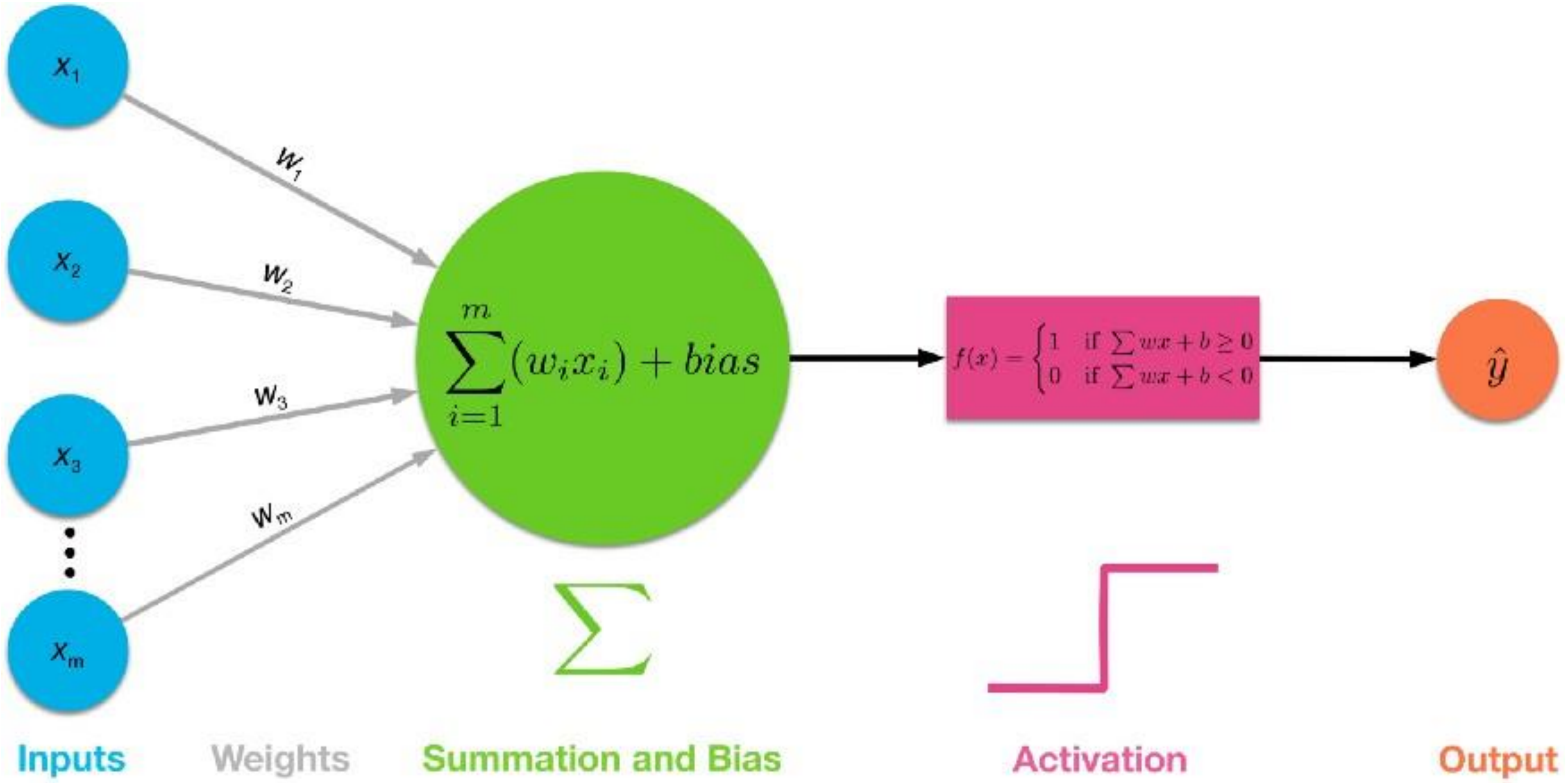
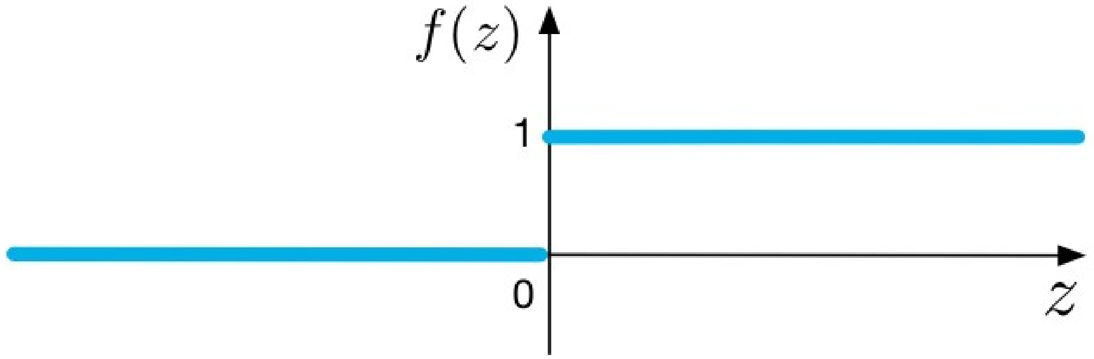
AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

Deep Learning

- Universal Approximation Theorem (UAT)
- Multi Layer Perceptron
- Feed forward Neural Network

Procedures of a Single-layer Perceptron Network


x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output



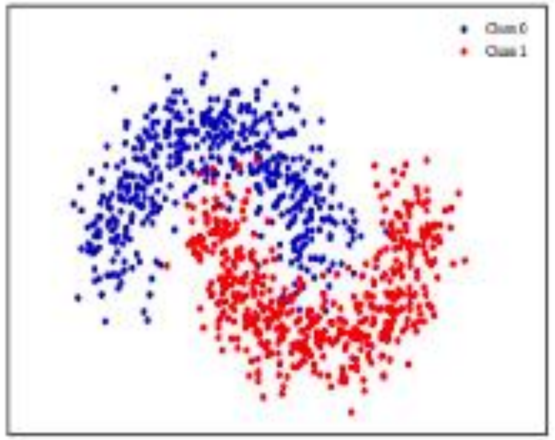
Step -Activation Function

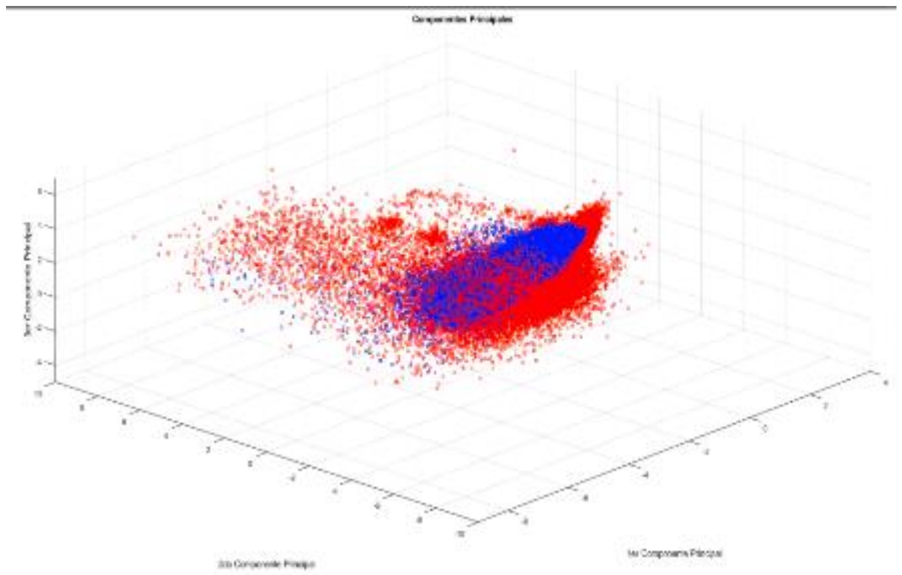
(Can be sigmoid as well)

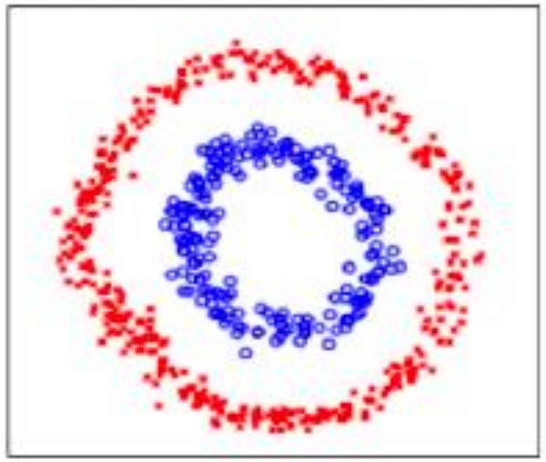
Courtesy: towardsdatascience.com

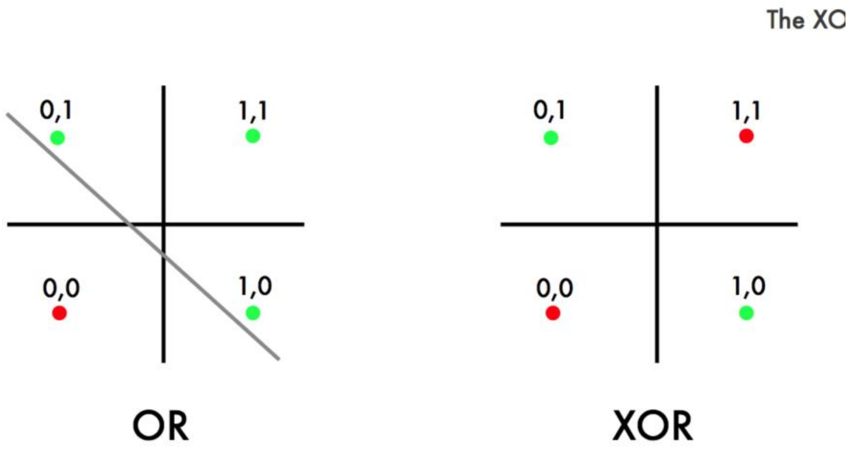
Non linearly separated data

XOR problem





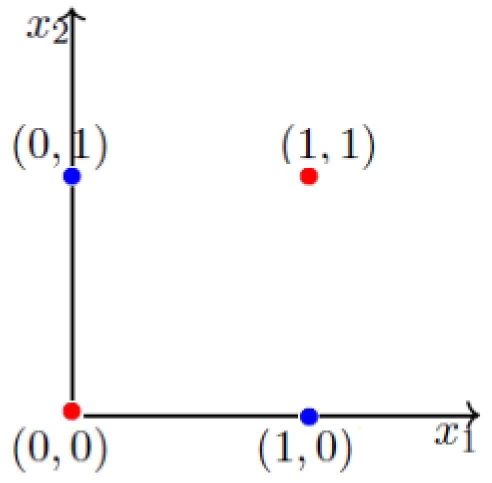




XOR Function — Can’t Do!- Non Linearly separated data

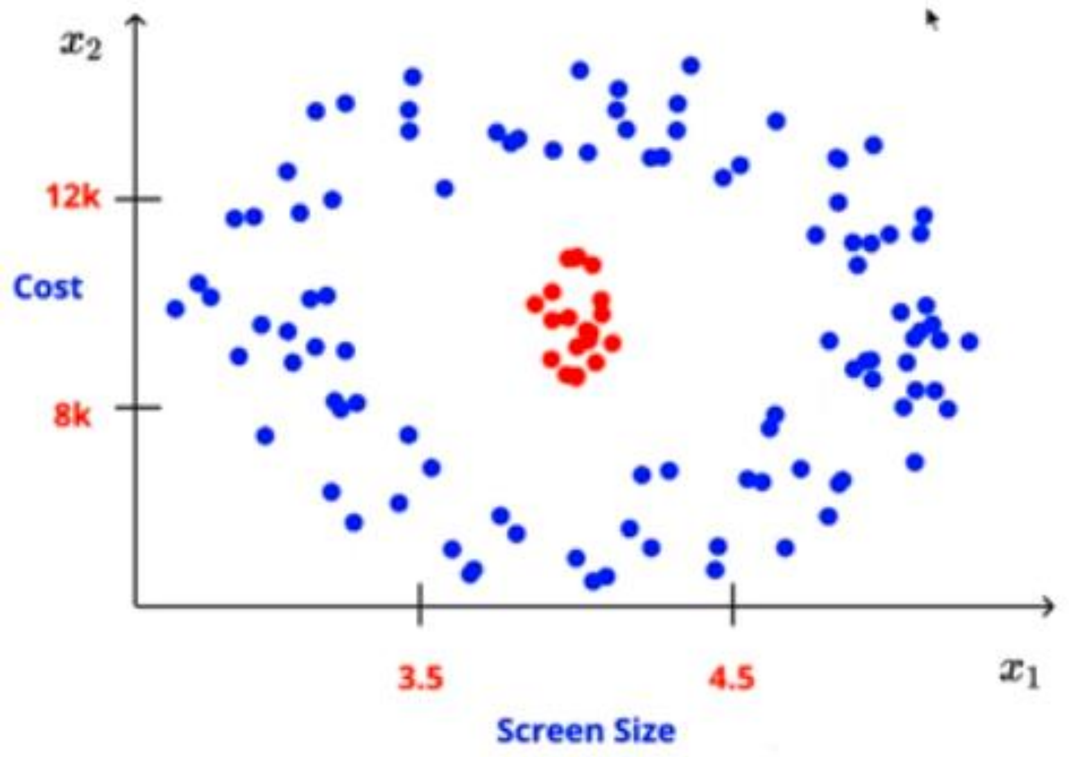
x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$

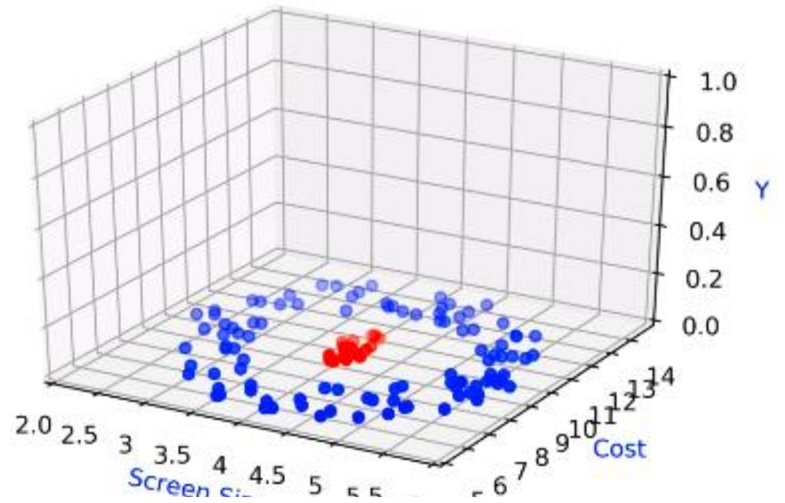
$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 < -w_0$$

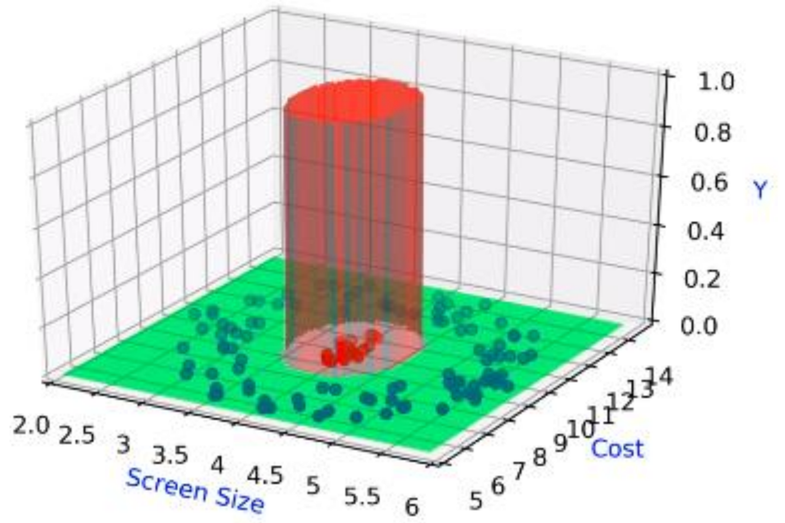


Notice that the fourth equation contradicts the second and the third equation. Point is, there are no *perceptron* solutions for non-linearly separated data. So the key take away is that a **single** *perceptron* cannot learn to separate the data that are non-linear in nature.

Non linearly separated data

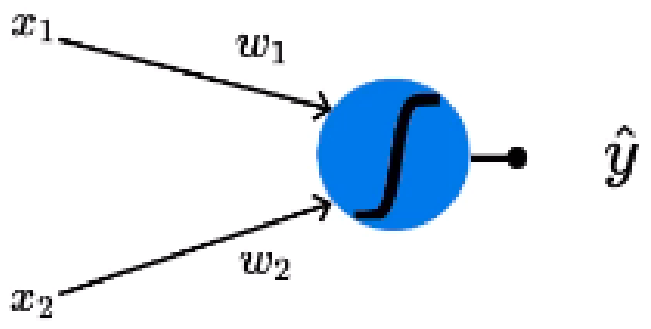


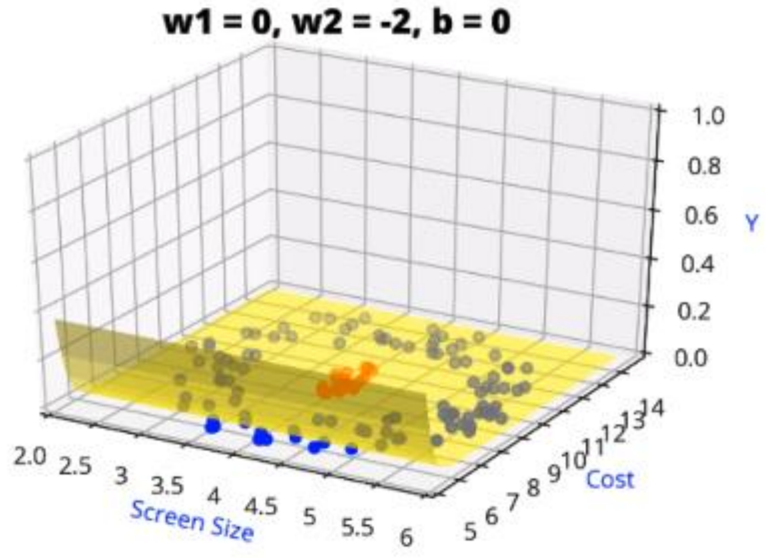




Will Sigmoid work ??

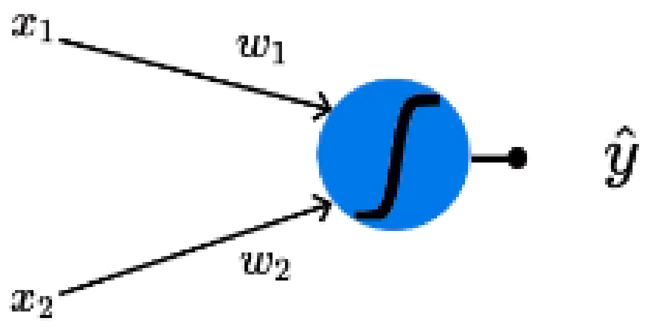
$$\hat{y} = \frac{1}{1+e^{-(w_1*x_1+w_2*x_2+b)}}$$

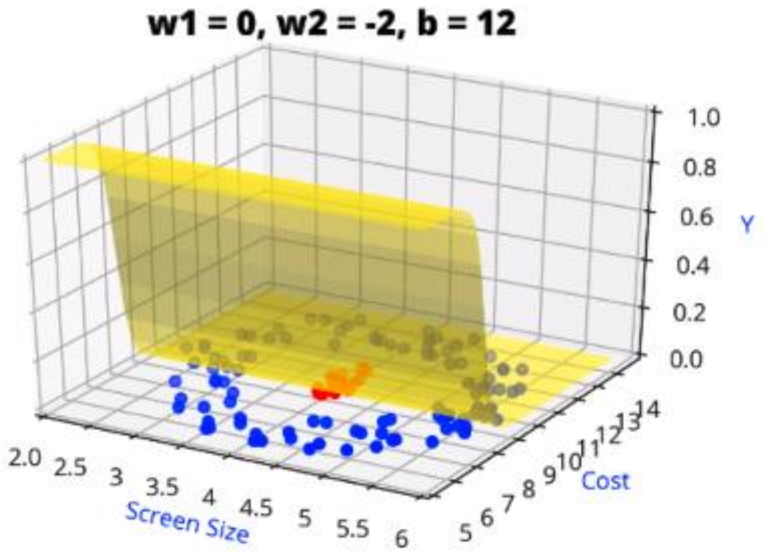




Will Sigmoid work ??

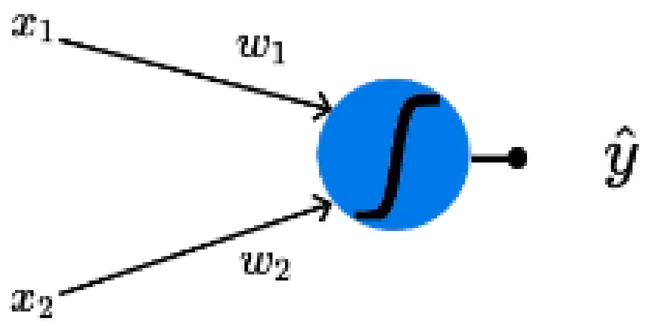
$$\hat{y} = \frac{1}{1+e^{-(w_1*x_1+w_2*x_2+b)}}$$

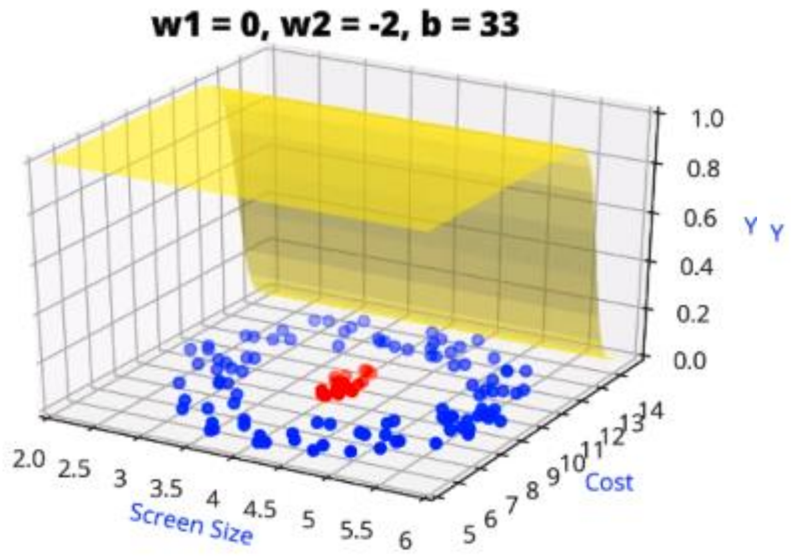




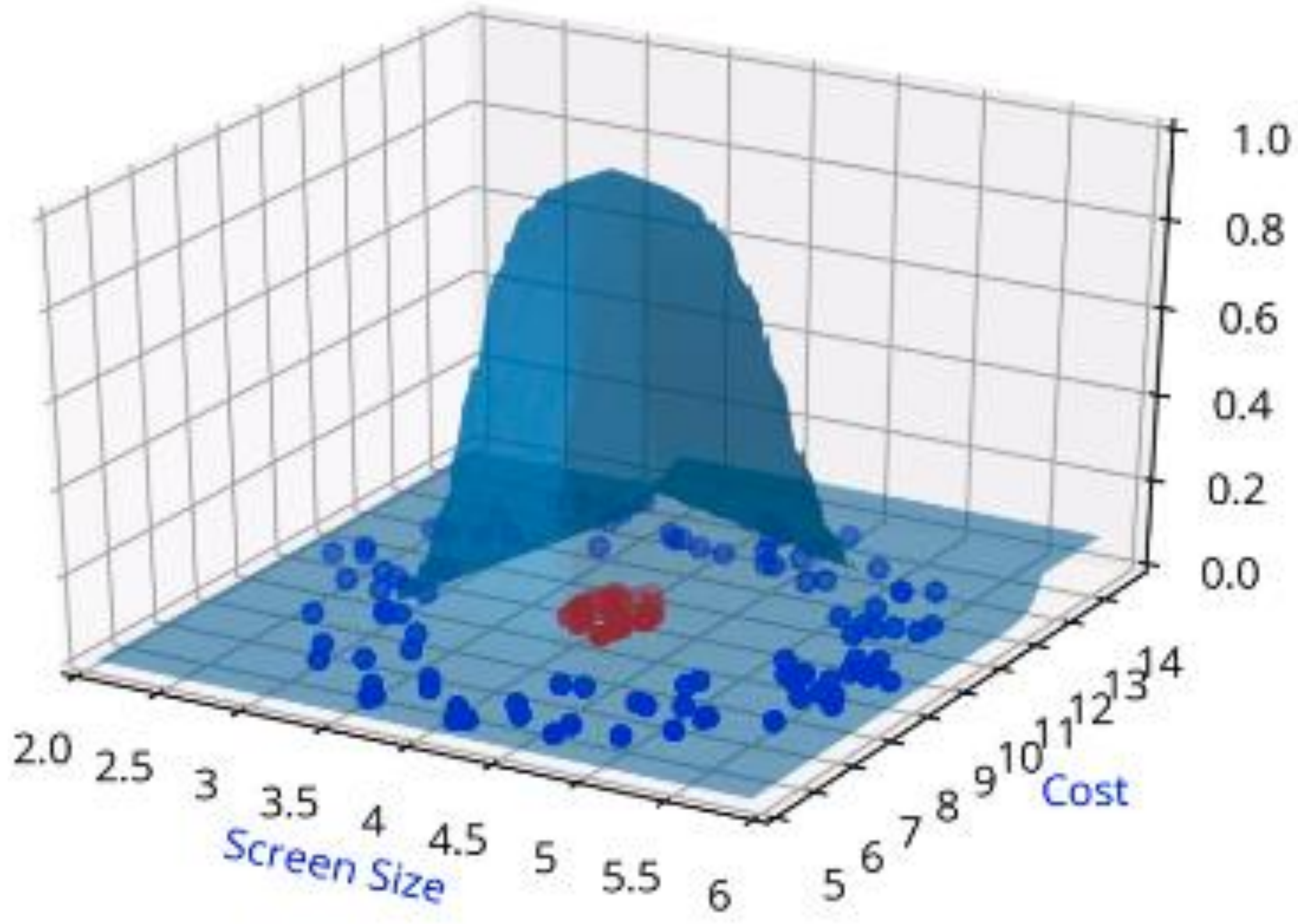
Will Sigmoid work ??

$$\hat{y} = \frac{1}{1+e^{-(w_1 * x_1 + w_2 * x_2 + b)}}$$

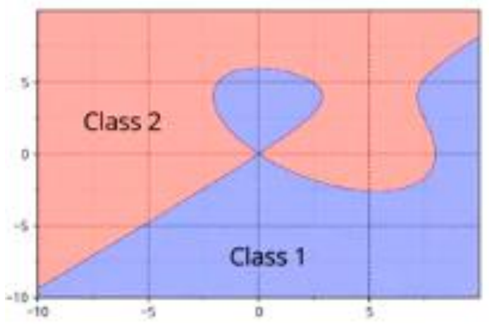


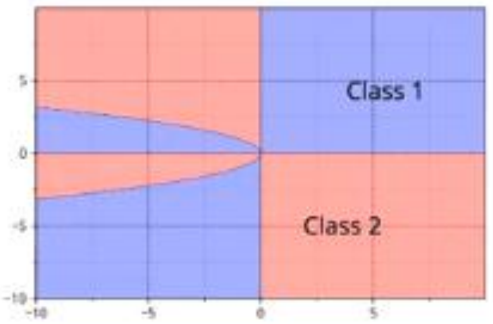


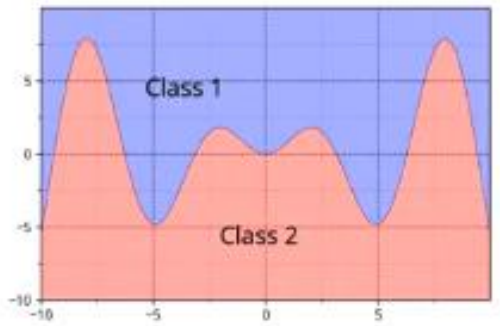
A complex function like this works

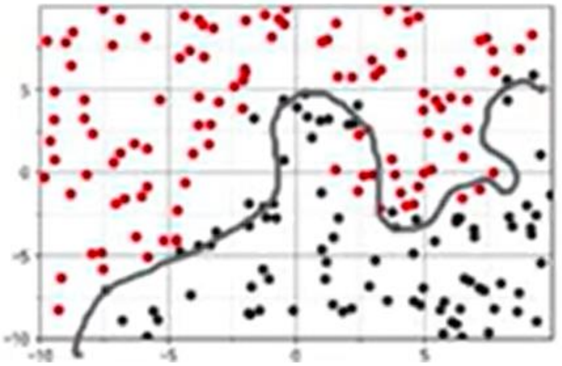


Need of DEEP ML models- complex real world functions









In real life, we deal with complex functions where the relationship between input and output might be complex, unlike the simple sigmoid neuron or perceptron.

It's hard to come up with such functions.
We need a simple approach!

Analogy

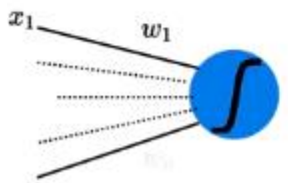
You can think of the house as a complex function, the way we will build a house is we start with a building a block i.e... brick in this case. First, we will lay the foundation and on top of that we will lay another layer and we will just keep continuing in this manner till we get very complex output i.e… house. In this process, we are combining the very simple building blocks in an effective way so that we can get this house.

Universal Approximation Theorem (UAT)

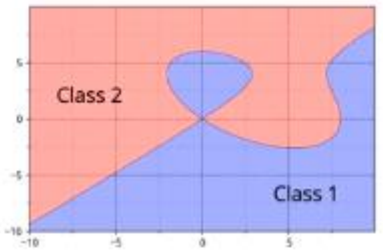
UAT says that — you can always come up with a deep neural network that will approximate any complex relation between input and output.

$$f(x_1, .., x_n) = \frac{1}{1+e^{-(w_1 * x_1 + ... + w_n * x_n + b)}}$$

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-(w * x + b)}}$$

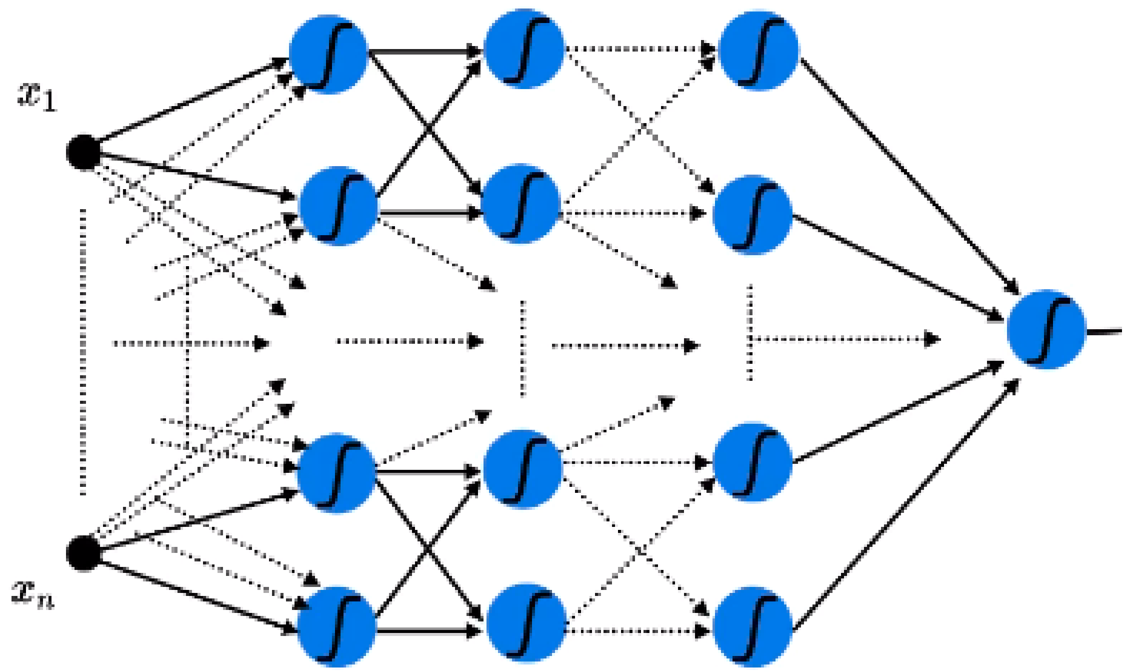


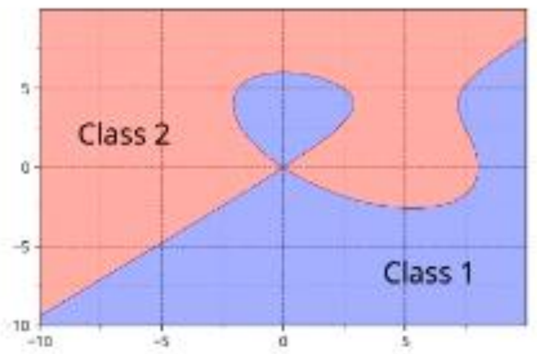
$\longrightarrow$

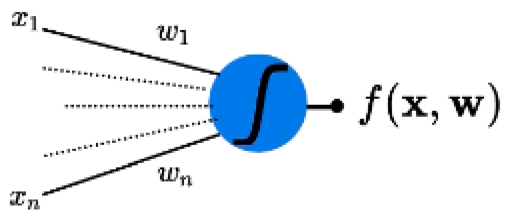


In the mathematical theory of artificial neural networks, the universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of Rn, under mild assumptions on the activation function.

Modelling Complex Functions

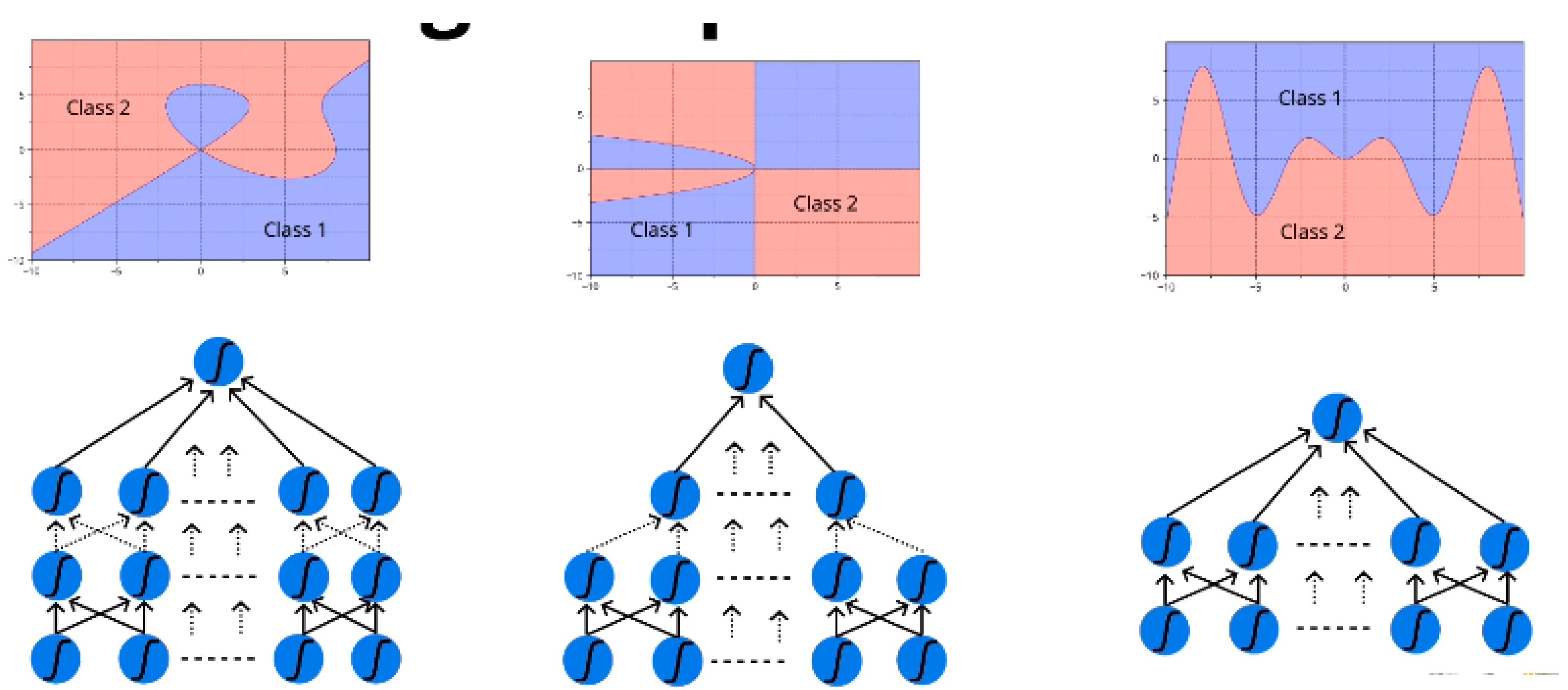






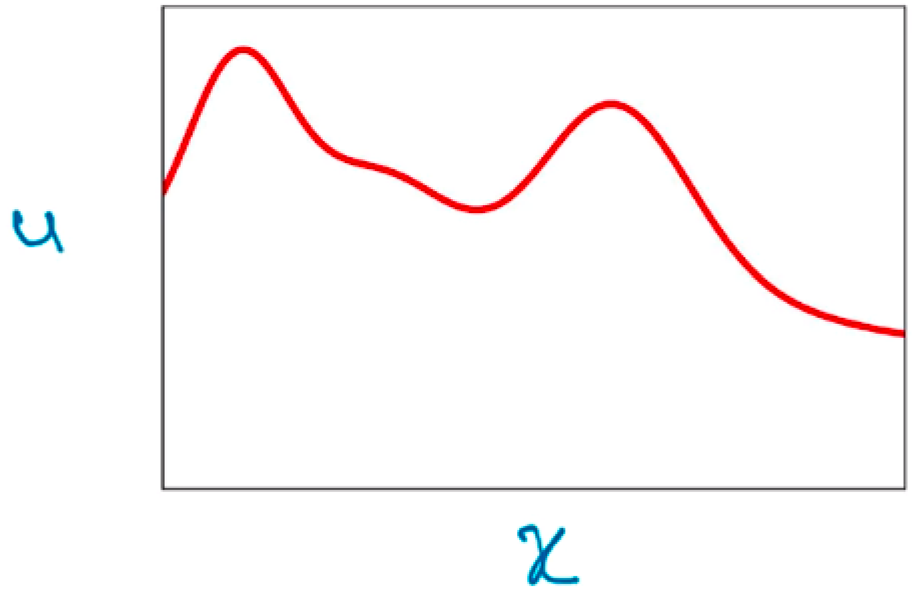
Deep neural network with a certain number of hidden layers should be able to approximate any function that exits between input and output.

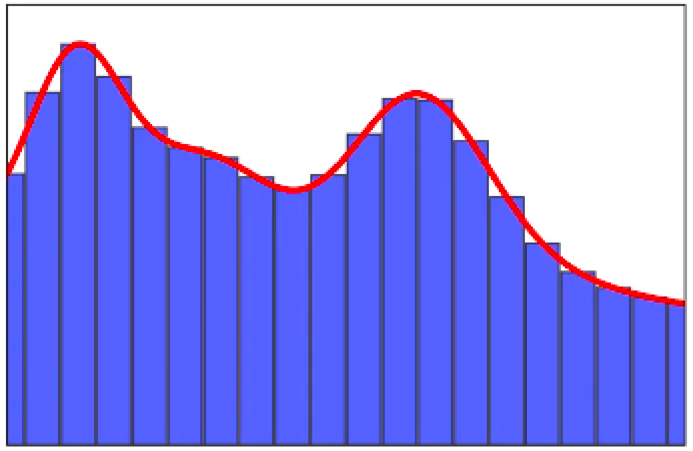
Universal Approximation Theorem (UAT)



$$f(x_1, .., x_n) = \frac{1}{1+e^{-(w_1 * x_1 + ... + w_n * x_n + b)}} \equiv \qquad f(\mathbf{x}, \mathbf{w}) = \frac{1}{1+e^{-(w * x + b)}} \equiv$$

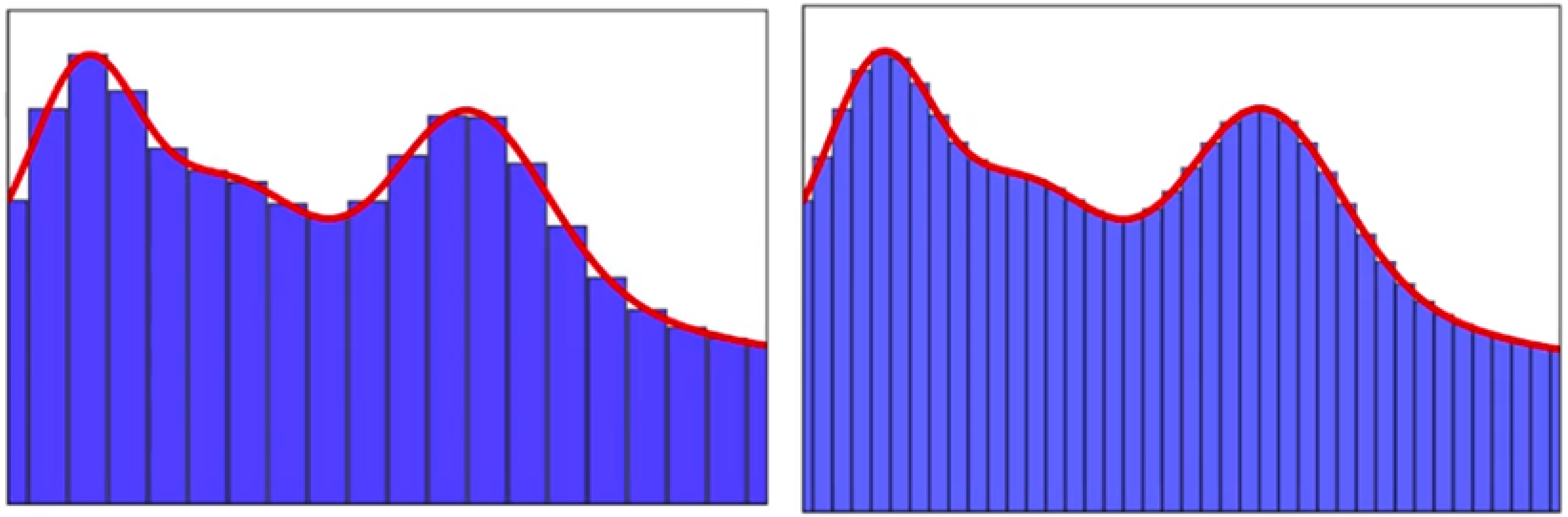
Illustrative proof of UAT





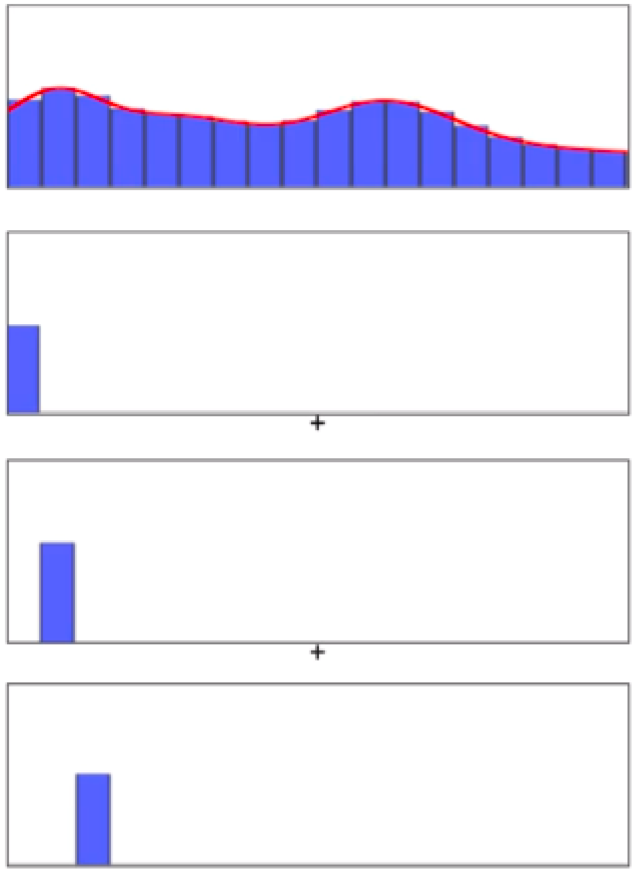
For this illustrative proof, we will consider a simple example where we have one-dimensional input **x** and output **y**, the graph shown above represents the true relationship between input and the output. Let’s assume that we don’t know what that function equation between **x** and **y** and we can’t come up with a representation for the equation.

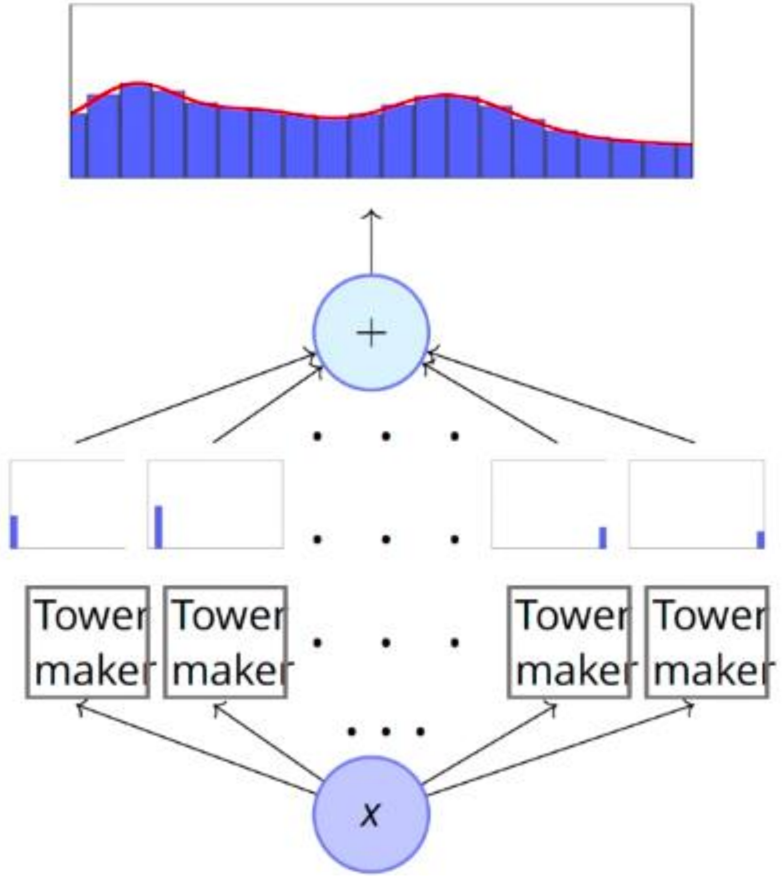
To solve this problem, we will break this function into multiple smaller parts so that each part is represented by a simpler function. By combining the series of smaller functions (rectangular bars/towers) I can approximate the relation between **x** and **y** as close to the true relationship possible.



The key point to note in this case is that we don’t have to worry about coming up with complex equations to represent the relationship between input and output. we can just come up with a simple function and use the combination of these functions to approximate the relationship to my true relationship. The more functions that I choose in this method the better will be my approximation.

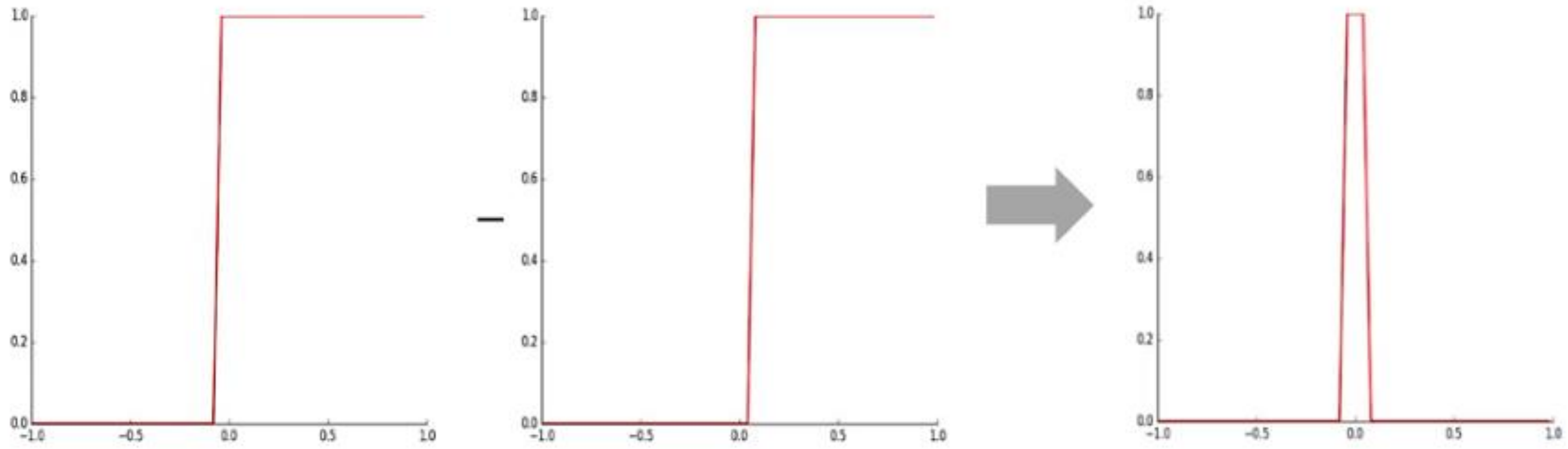
Illustrative proof of UAT





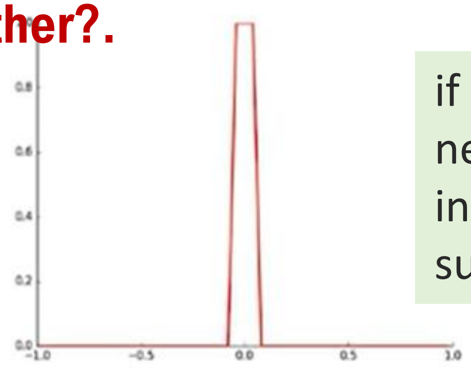
- We make a few observations
- All these "tower" functions are similar and only differ in their heights and positions on the x-axis
- Suppose there is a black box which takes the original input (x) and constructs these tower functions
- We can then have a simple network which can just add them up to approximate the function

How do we come up with these rectangles/towers and how it will tie back to the sigmoid neuron?

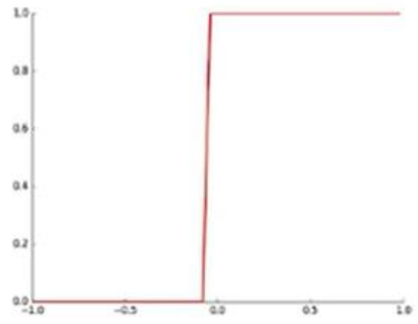


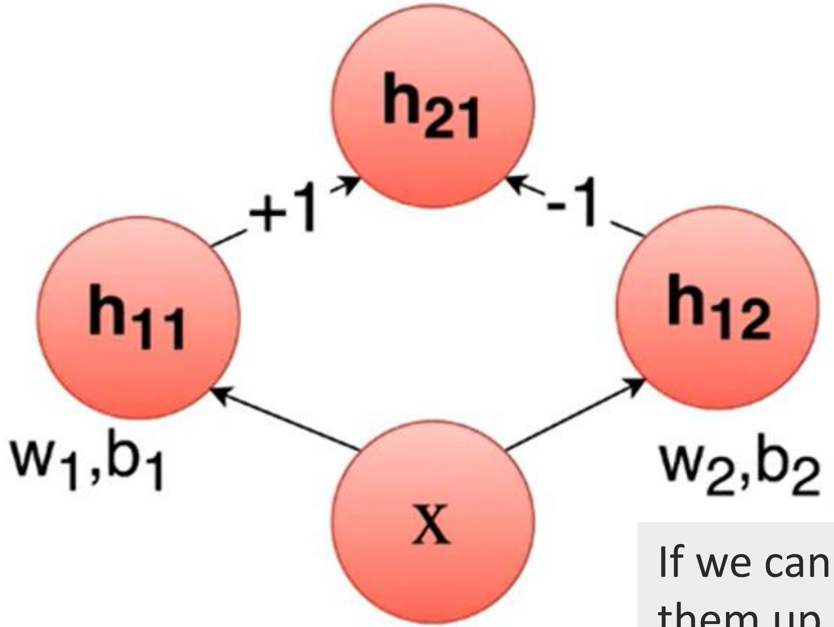
Let’s take two step functions having a very steep slope and notice that they have a different place at which they peak. The left sigmoid peaks just before zero and right sigmoid peaks just after zero. If we just subtract these two functions the net effect is going to be a tower (rectangular output).

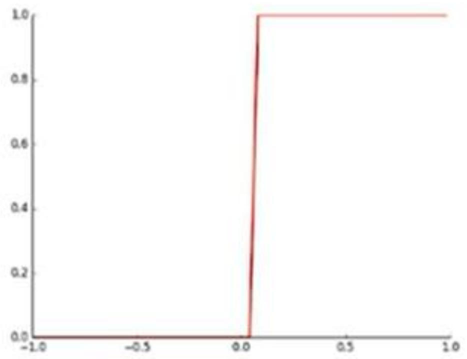
Neural Network-Can we come up with a neural network to represent this operation of subtracting one step/sigmoid function from another?.



if we have an input **x** and it is passed through the two sigmoid neurons and the output from these two neurons are combined in another neuron with weights +1 and -1 i.e… same is as subtracting these two outputs, then we will get our tower.





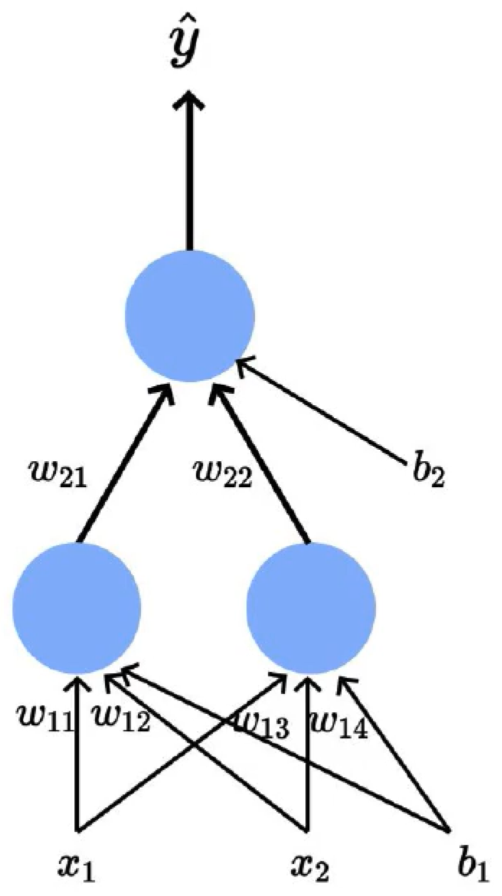


If we can construct many such building blocks and add all of them up, we can approximate any complex true relationship between input and output. This is called the ***representational power of deep neural networks***, to approximate any kind of relationship between input and output.

Feed Forward Neural Networks-Introduction

*Citation Note: The content, of this presentation were inspired by the awesome lectures and the material offered by Prof. **Mitesh M. Khapra** on NPTEL's Deep Learning course*

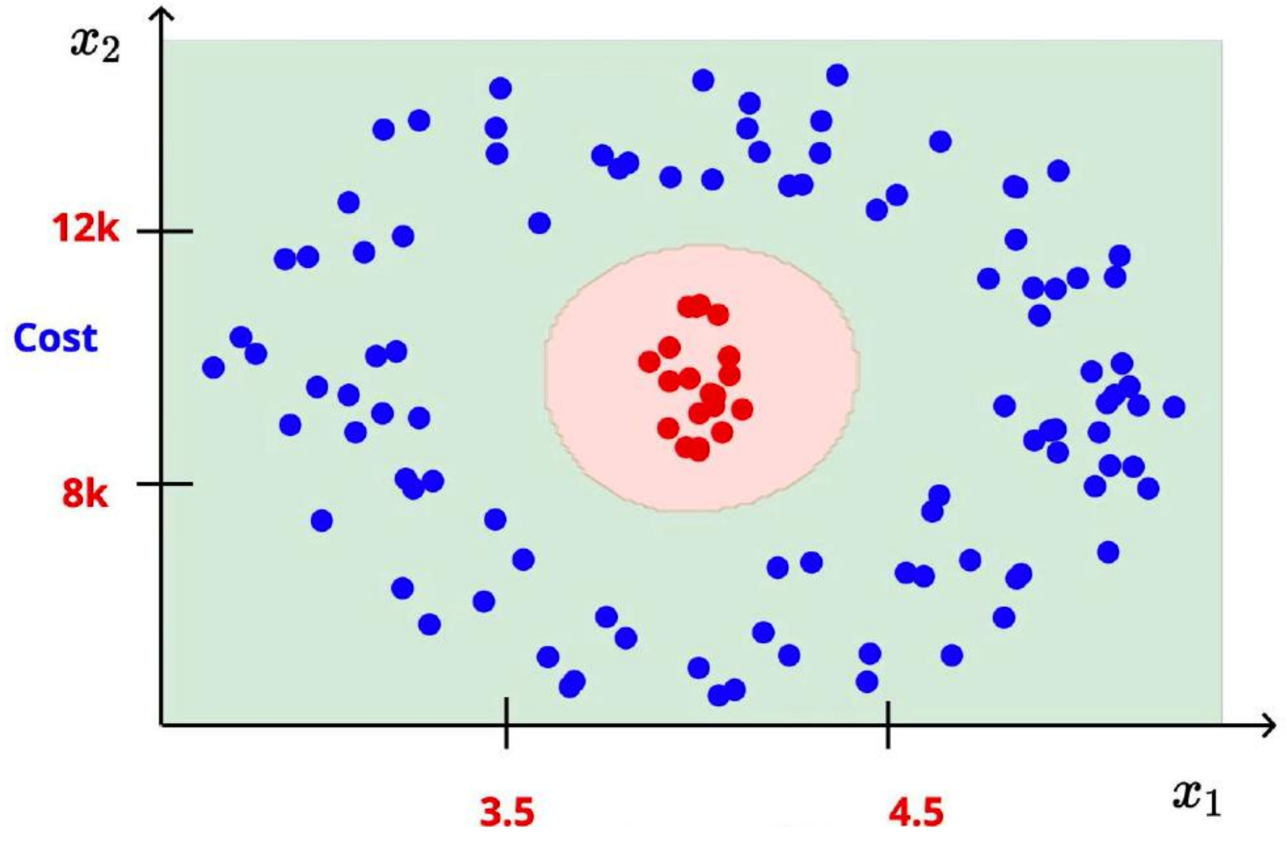
Multiple sigmoid Neurons



$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

$$\hat{y} = g(h_1, h_2)$$



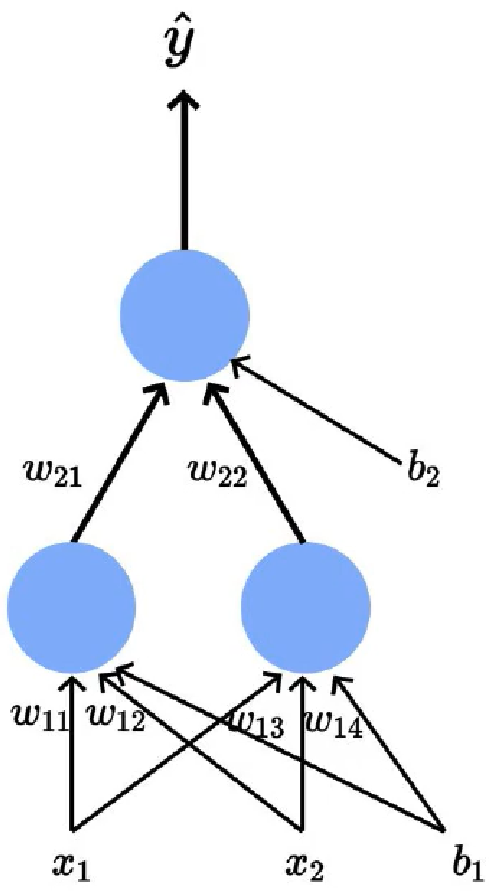
$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$

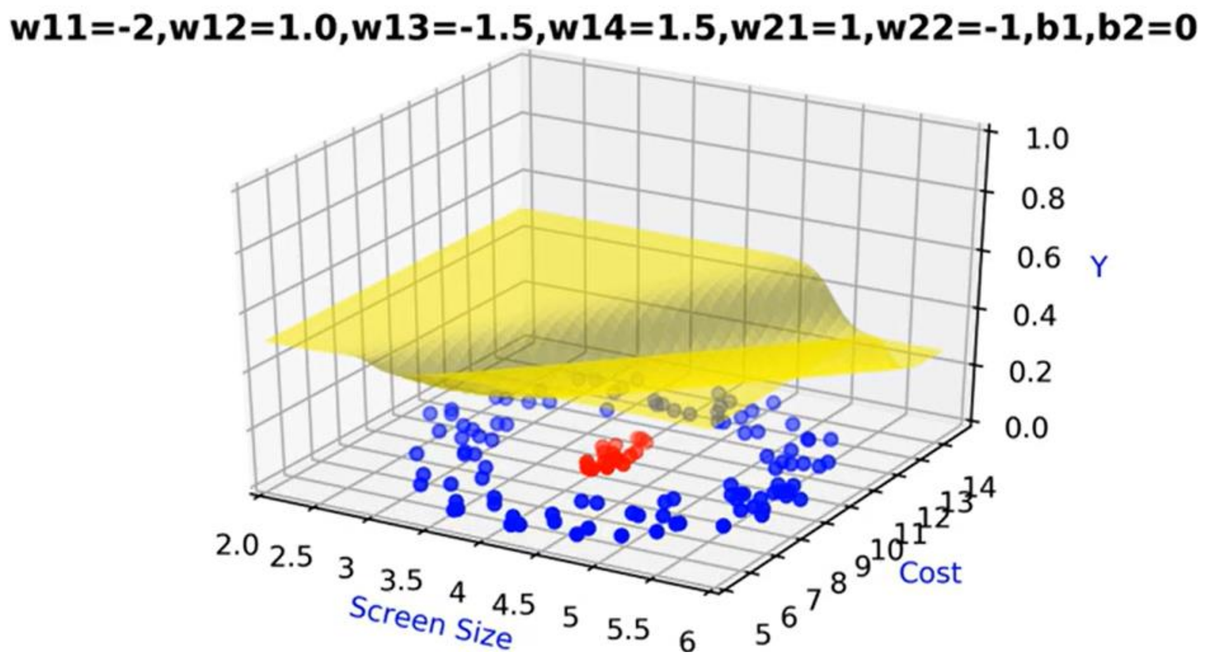
Multiple sigmoid Neurons



$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

$$\hat{y} = g(h_1, h_2)$$



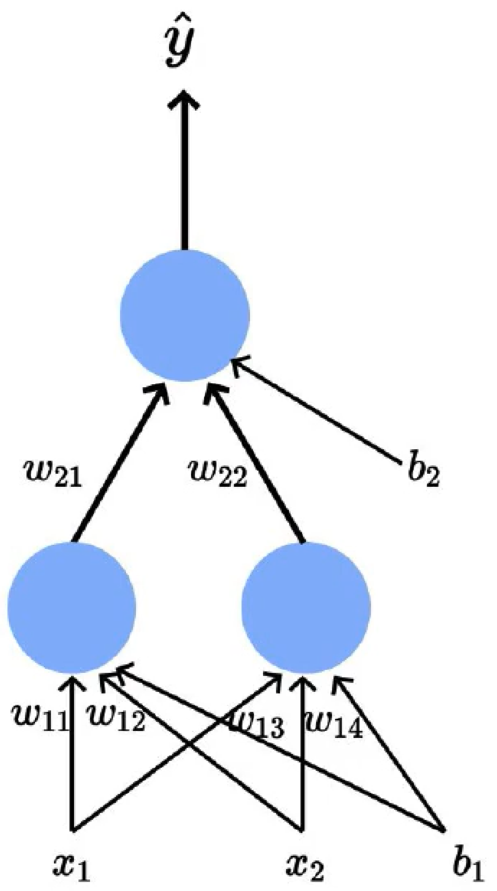
$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$

Multiple sigmoid Neurons



$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

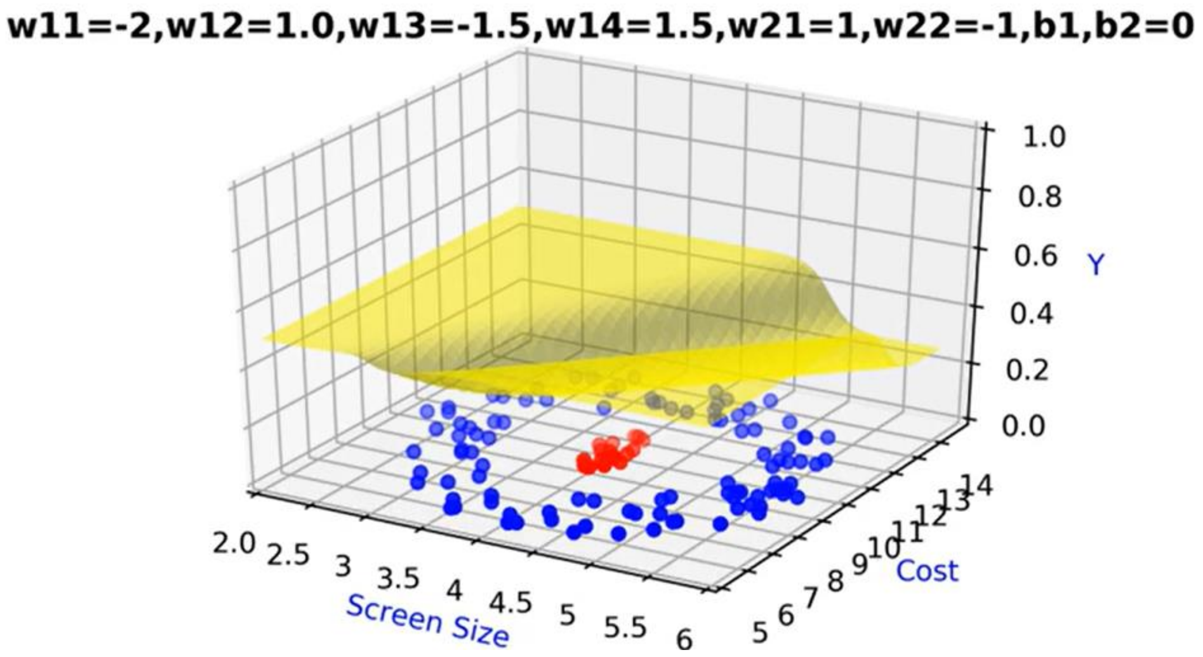
$$\hat{y} = g(h_1, h_2)$$

$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

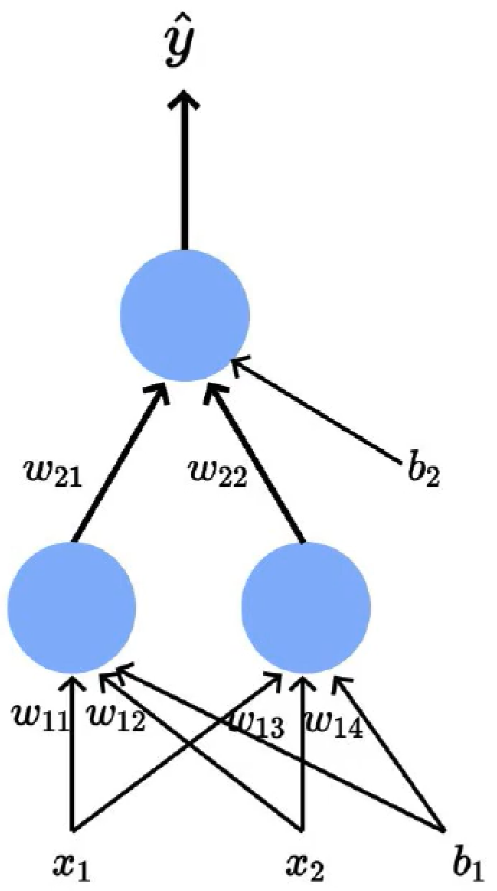
$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$



Multiple sigmoid Neurons



$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

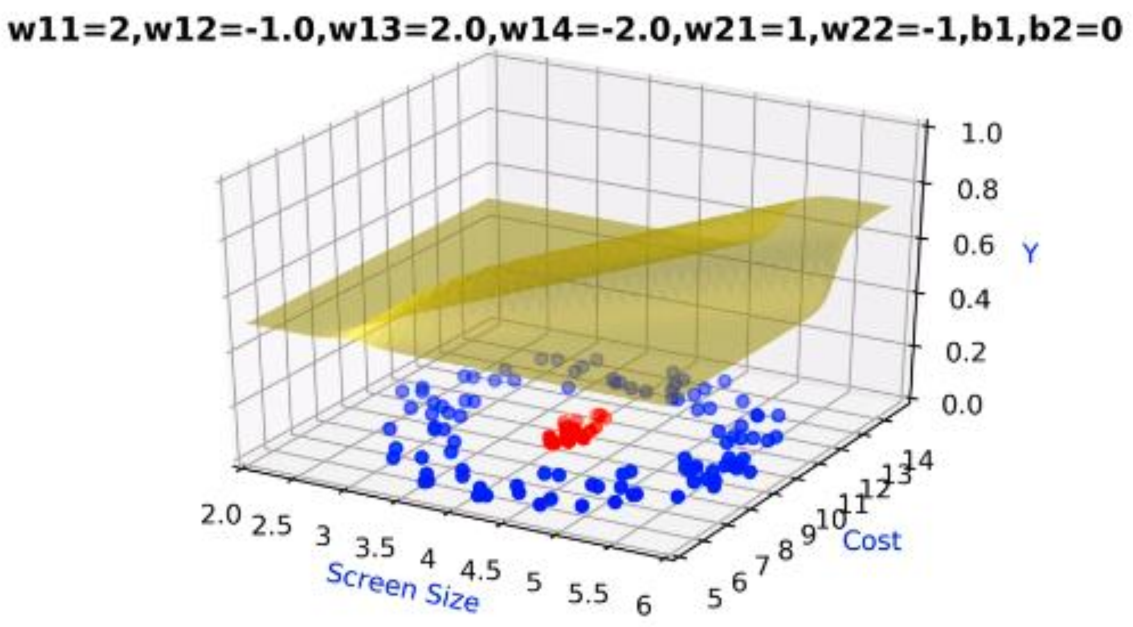
$$\hat{y} = g(h_1, h_2)$$

$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

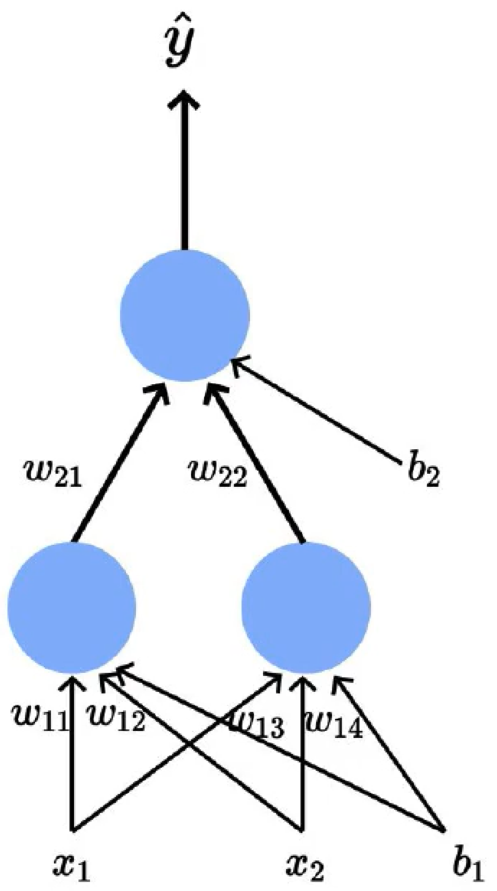
$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$



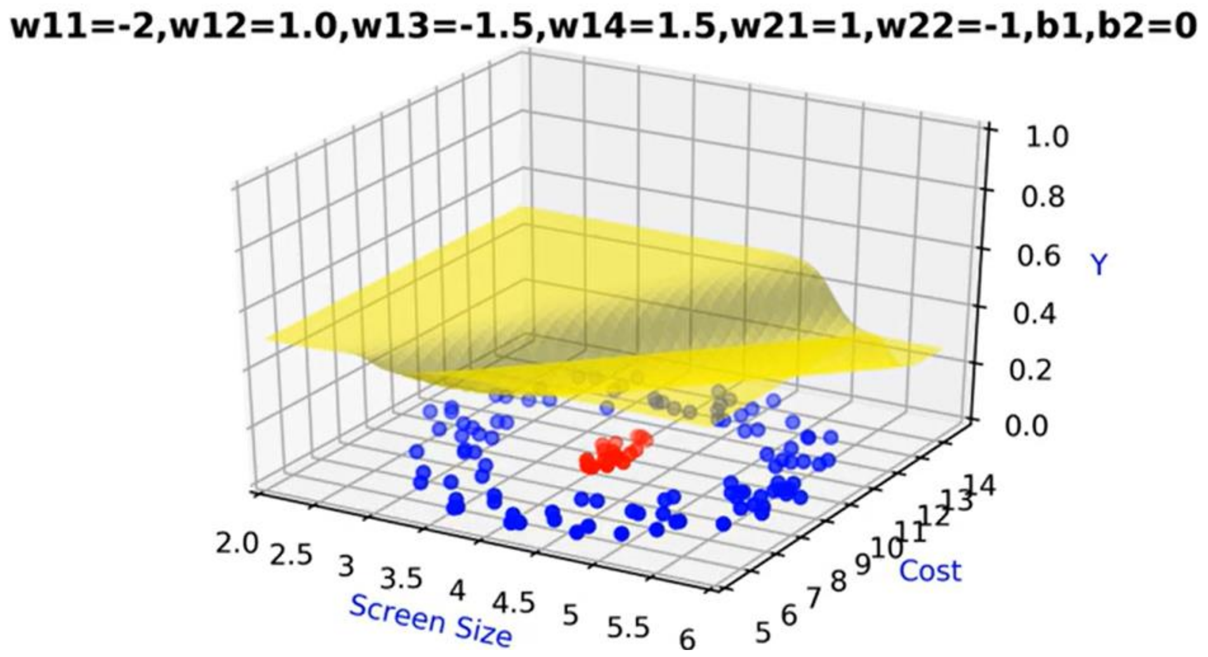
Multiple sigmoid Neurons



$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

$$\hat{y} = g(h_1, h_2)$$



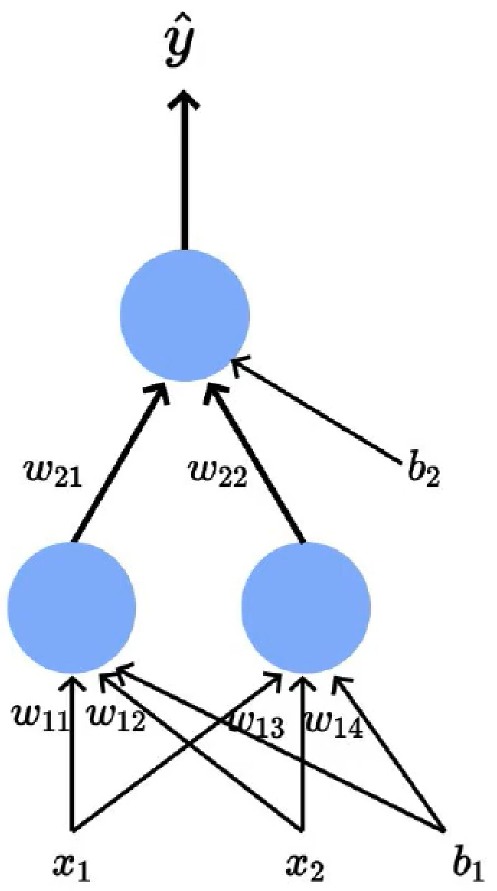
$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$

Multiple sigmoid Neurons



$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

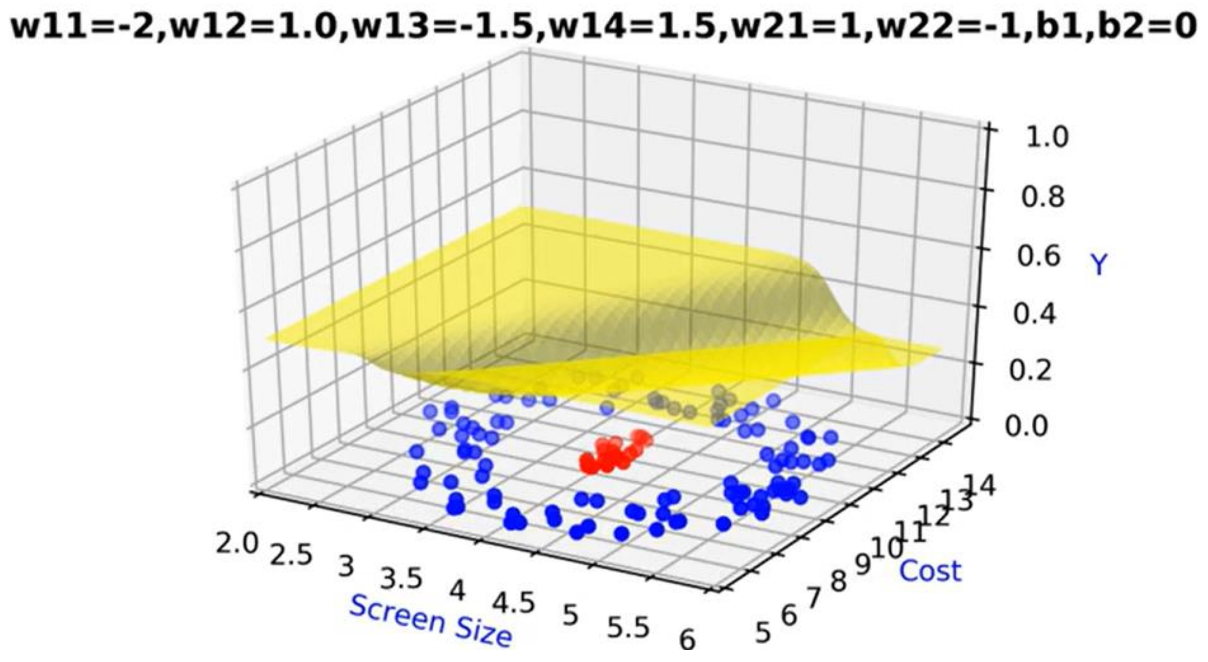
$$\hat{y} = g(h_1, h_2)$$

$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

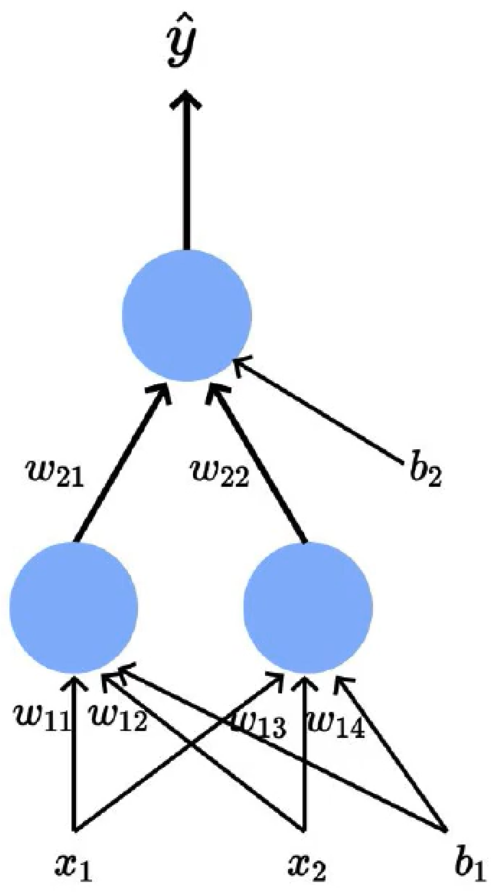
$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$



Multi Layer Neural Networks

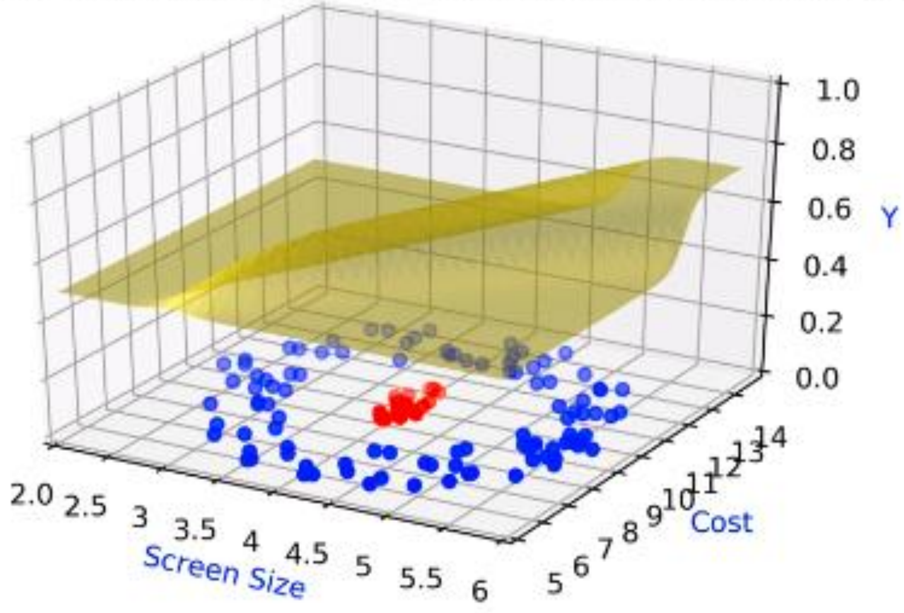


$$h_1 = f_1(x_1, x_2)$$

$$h_2 = f_2(x_1, x_2)$$

$$\hat{y} = g(h_1, h_2)$$

w11=2,w12=-1.0,w13=2.0,w14=-2.0,w21=1,w22=-1,b1,b2=0



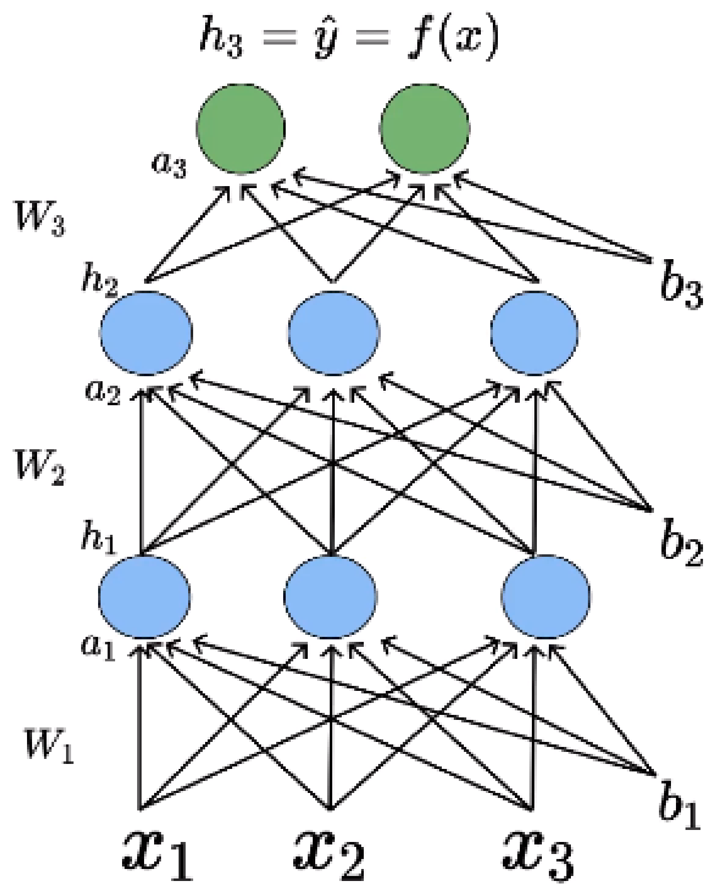
$$h_1 = \frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}}$$

$$h_2 = \frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}}$$

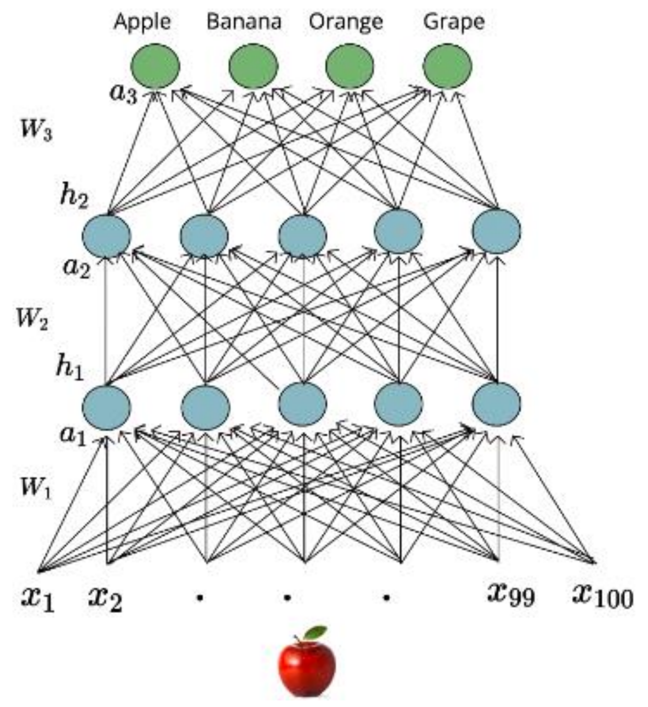
$$\hat{y} = \frac{1}{1+e^{-(w_{21}*h_1+w_{22}*h_2+b_2)}}$$

$$= \frac{1}{1+e^{-(w_{21}*(\frac{1}{1+e^{-(w_{11}*x_1+w_{12}*x_2+b_1)}})+w_{22}*(\frac{1}{1+e^{-(w_{13}*x_1+w_{14}*x_2+b_1)}})+b_2)}}$$

Feed Forward Neural Networks



- The pre-activation at layer 'i' is given by
  $$a_i(x) = W_i h_{i-1}(x) + b_i$$
- The activation at layer 'i' is given by
  $$h_i(x) = g(a_i(x))$$
  where 'g' is called as the activation function
- The activation at output layer 'L' is given by
  $$f(x) = h_L = O(a_L)$$
  where 'O' is called as the output activation function



Thank you