# Deep Learning

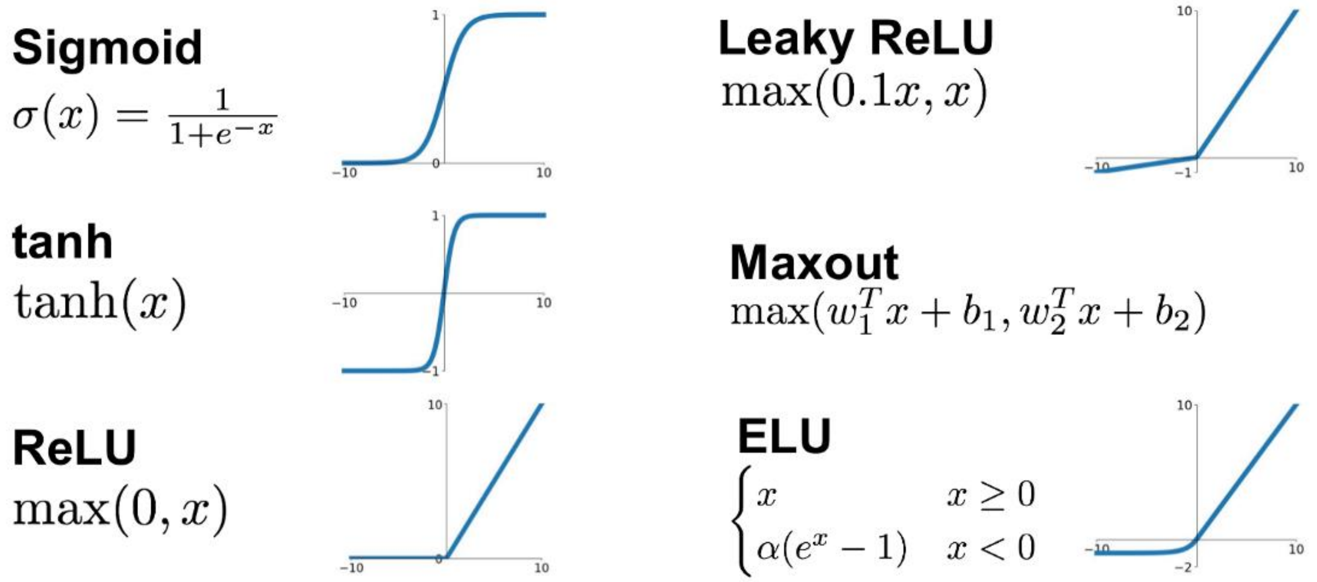
## Neural Networks : Activation Functions

Anita Budhiraja
Scientist D

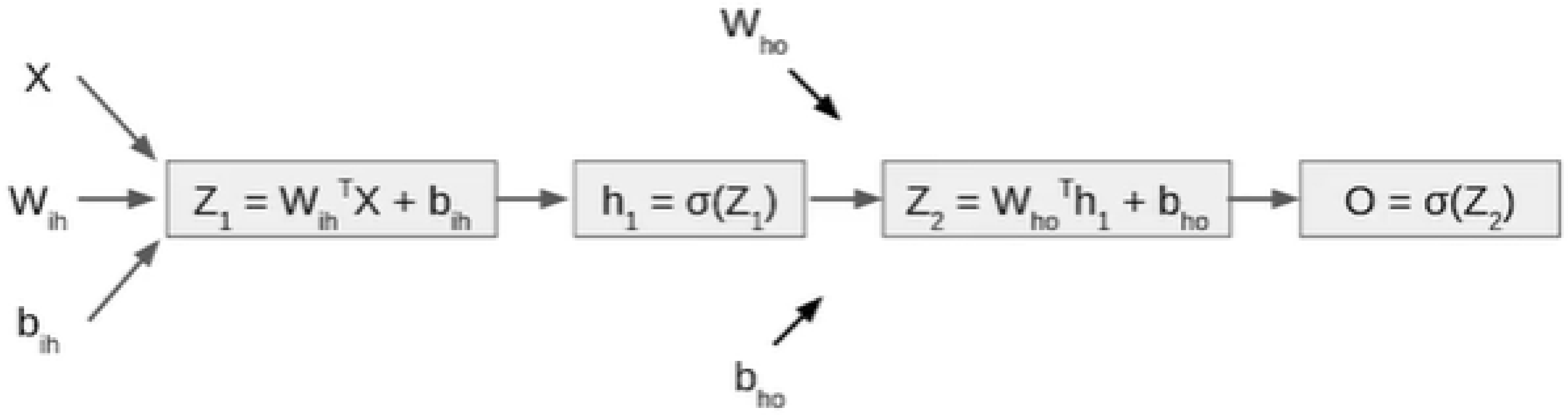
## Activation Function

- Activation functions live inside neural network layers and modify the data they receive before passing it to the next layer.
- Activation functions give neural networks their power — allowing them to model complex non-linear relationships.
- By modifying inputs with non-linear functions neural networks can model highly complex relationships between features.

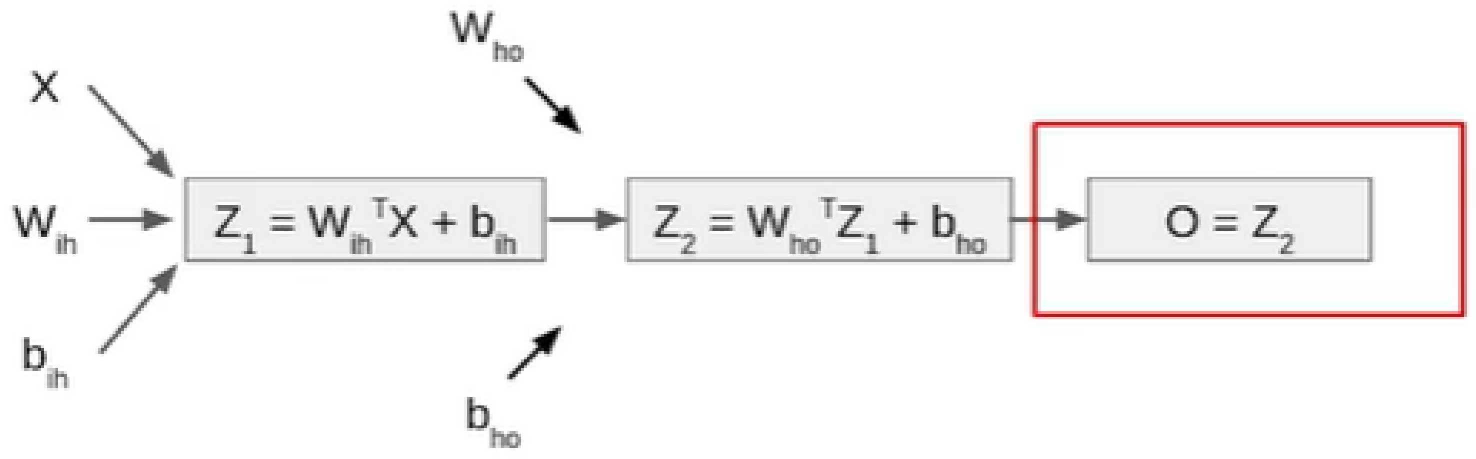
Popular activation functions



# Working of Neural Network



## Removing Activation functions



$$Z_1 = W_{ih}^T X + b_{ih}$$

$$Z_2 = W_{ho}^T Z_1 + b_{ho} \quad = W_{ho}^T (W_{ih}^T X + b_{ih}) + b_{ho}$$

This is linear combination of inputs and weights. This network cannot capture the comlex relationships in data. Check in Tensorflow playground.

# Why Activation function is needed

- The complex configuration of weights possessed by neural networks makes it difficult to solve the problem.
- The main reason lies in the concept of Non-Linearity

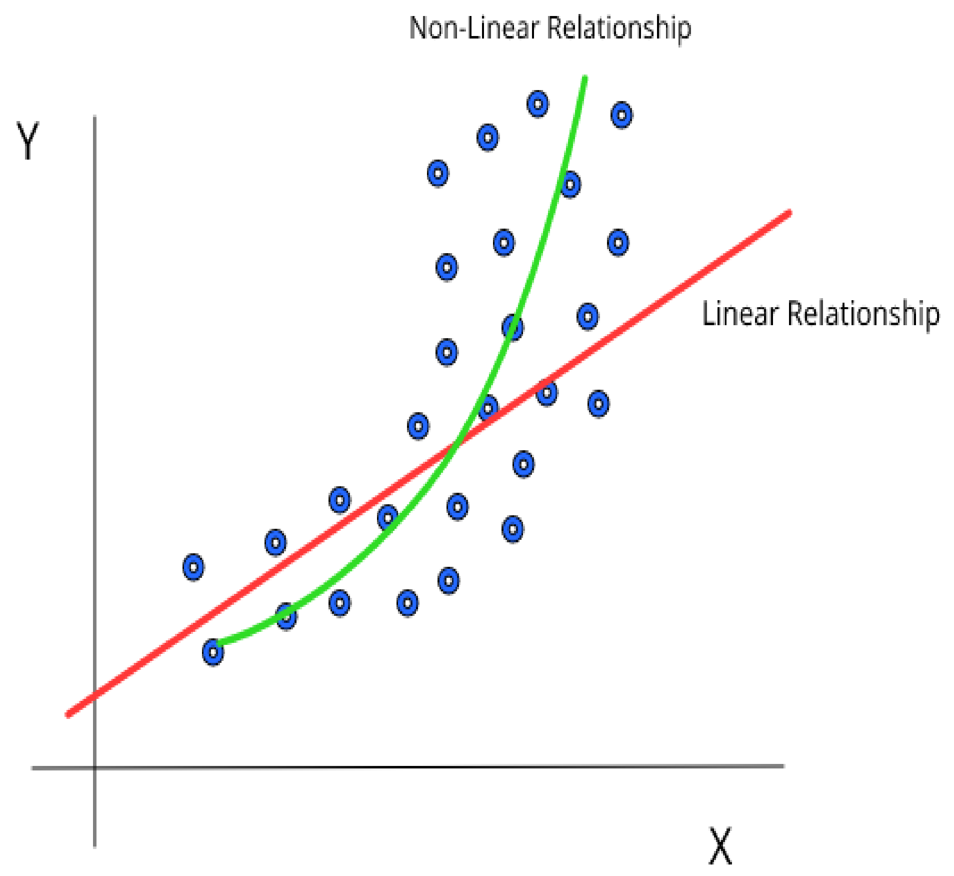
Linearity vs non-linearity

$Y = W_1 * X_1 + W_2 * X_2$ is a Linear function

- The above equation represents a **linear relationship** between Y and X1,X2. Regardless of what values W1 and W2 have, at the end of the day the change of value of X1 and X2 will result in a **linear** change in Y.
- **Activation functions are always differentiable.**

# Why Activation function is needed

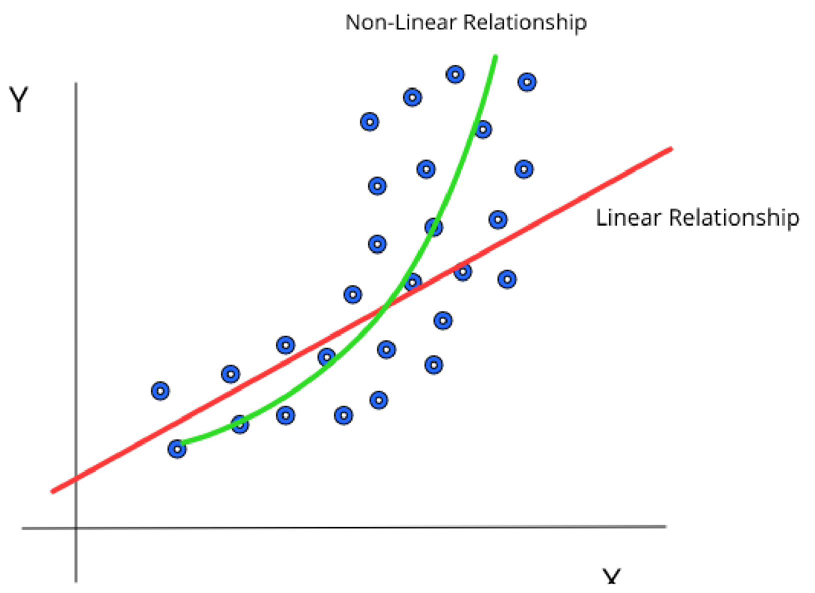
- If we do not apply the Activation function, then the output signal will only be a **Linear function.** Although a linear function is easy to solve, it will possess less functional complexity and mapping.
- Also without activation function, our Neural network would not be able to learn and model other complicated data such as images, videos, audios, speech etc.
- That is why we use Artificial Neural network techniques (**Deep learning) to make sense of something complicated, high dimensional, non-linear-big datasets, where the model has lots and lots of hidden layers.**



# Why Activation function is needed

- If the data scientist tries to fit in the linear relationship,  red is generated as an output, which is not accurate.
- But if **non linear** relationships is tried then green line is generated which is much better and near to the desired results.

✓ *So keep some non linear function as the activation function for each neuron and our neural network is now* ***capable*** *of fitting on non linear data.*



Without Activation Function

$$y = \sum_{i=0}^{n}(W_i * X_i) + B$$

With Activation Function

$$y = f(\sum_{i=0}^{n}(W_i * X_i) + B)$$

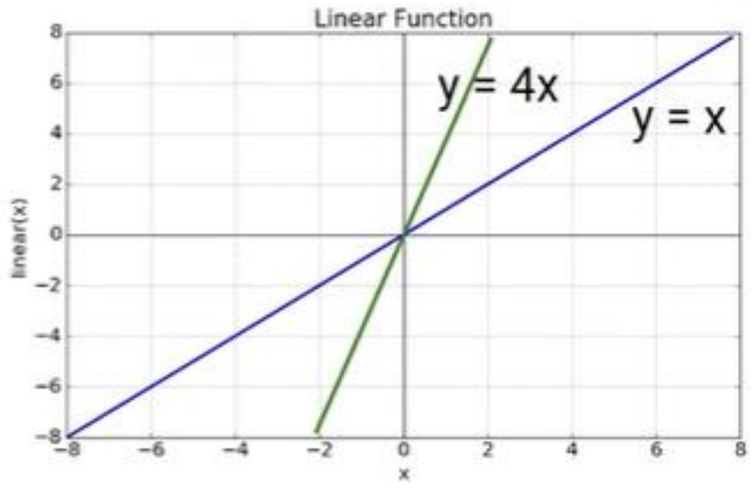
## Activation function - Properties

- Non-linear - In linear regression we're limited to a prediction equation that looks like a straight line
- But what if the patterns in our dataset were non-linear? (e.g. x2, sin, log). To model these relationships we need a non-linear prediction equation.[1]
- Activation functions provide this non-linearity.
- Continuously differentiable — To improve our model with gradient descent, we need our output to have a nice slope so we can compute error derivatives with respect to weights.
- If our neuron instead outputted 0 or 1 (perceptron), we wouldn't know in which direction to update our weights to reduce our error.

## Conditions for Activation Functions

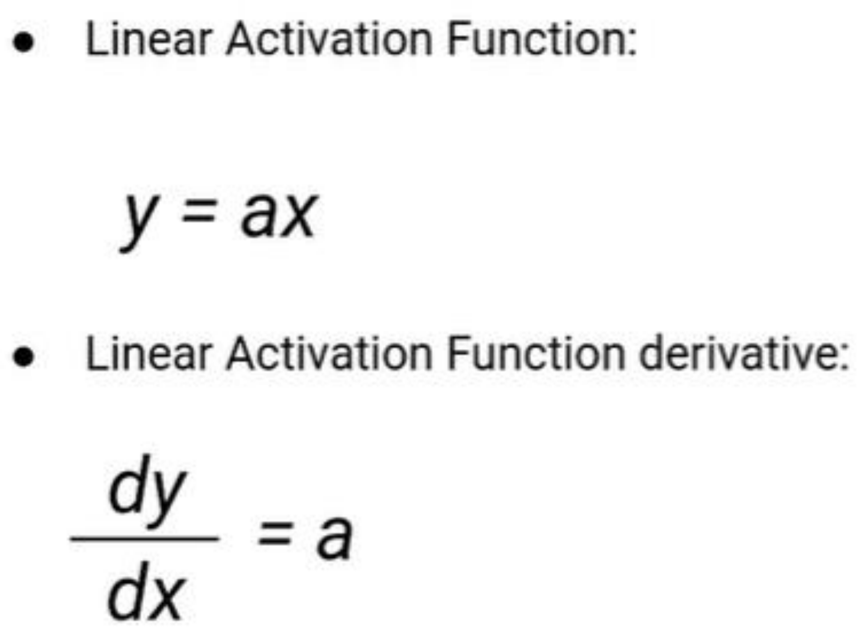
- Activation Function Must be Continuous (Infinity values)

  - Linear Activation Function: $y = ax$
  - Graph of Linear Function:
  - Input Range: $(-\infty$ to $\infty)$
  - Output Range: $(-\infty$ to $\infty)$



# Conditions for Activation Functions

2. Activation Function must be Differentiable at all points. In bak propagation we update the gradients in order to update the parameters. We also calculate the partial derivatives for the activation functions used in the network.



- Linear Activation Function:

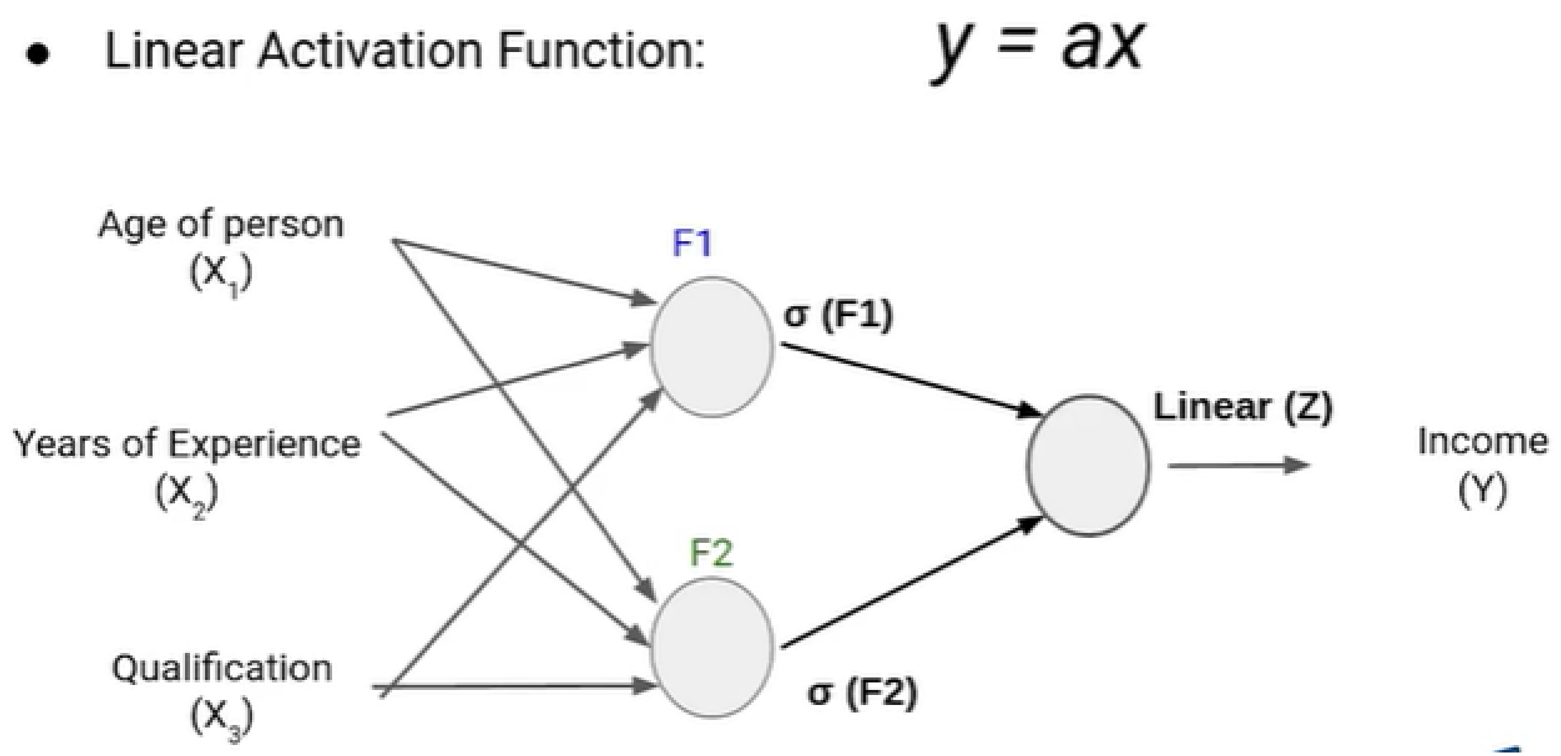
$$y = ax$$

- Linear Activation Function derivative:

$$\frac{dy}{dx} = a$$

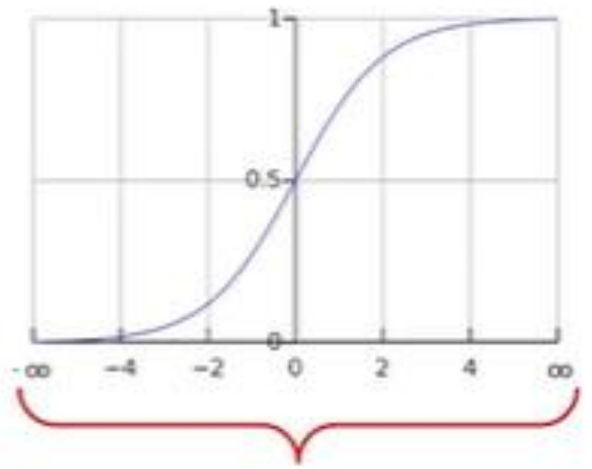
## Linear Function : Output Layer

- As Linear Activation Function Does not capture the non-linear relationships in the data. So it is used at output layer for regression problem.



# Sigmoid Activation Function

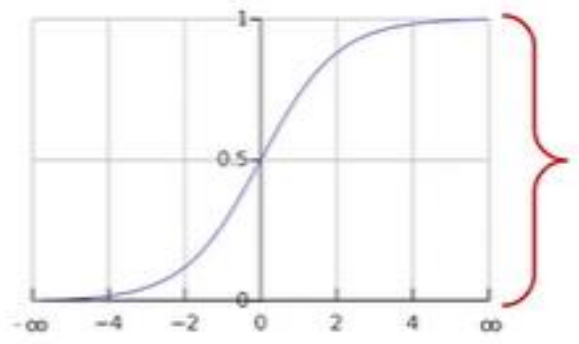
- Sigmoid Function Formula: $\sigma(x) = \frac{1}{1+e^{-(x)}}$
- Sigmoid Function Graph:
- Input Range: (- ∞ to ∞)



Can take any real values

# Sigmoid Activation Function

- Sigmoid Function Formula: $\sigma(x) = \dfrac{1}{1+e^{-(x)}}$
- Sigmoid Function Graph:
- Input Range: (- ∞ to ∞)
- Output Range: ( 0 to 1)



Restricts the output between 0 to 1. These values are probabilities and this function is continuous and differentiable at all times.

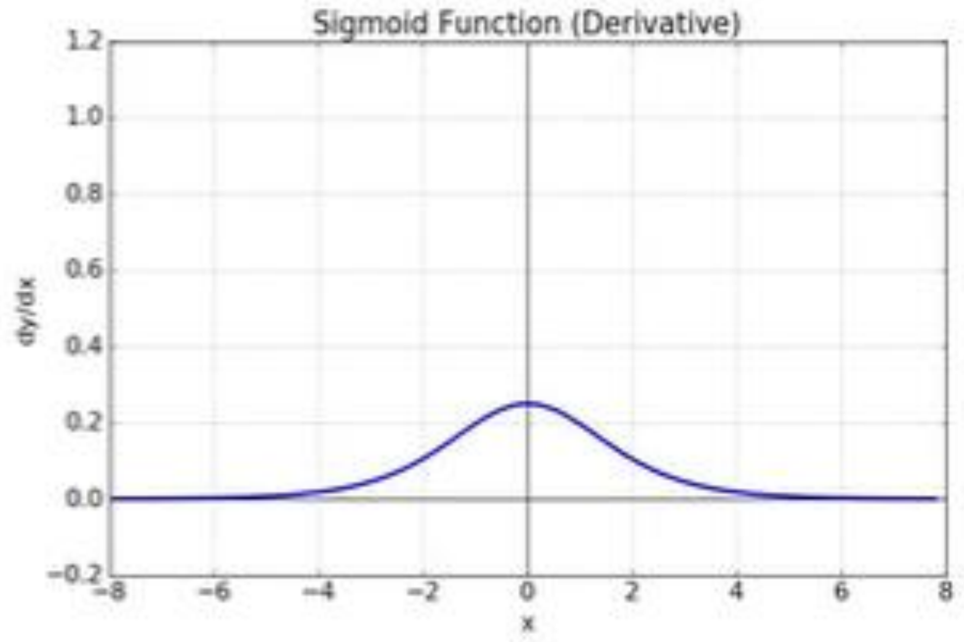
# Sigmoid Activation Function

- Sigmoid Function:

$$\sigma(x) = \frac{1}{1+e^{-(x)}}$$

- Sigmoid Function derivative:

$$\frac{d\sigma}{dx} = \sigma(x)\,(1-\sigma(x))$$



This curve is flat which means the derivatives or the gradient values for this activation function will be very small.

## Tanh Activation Function (Scale Sigmoid Function)

Tanh Activation Function: $$tanh(x) = \frac{\boxed{2}}{1+e^{-\boxed{2x}}} - 1$$

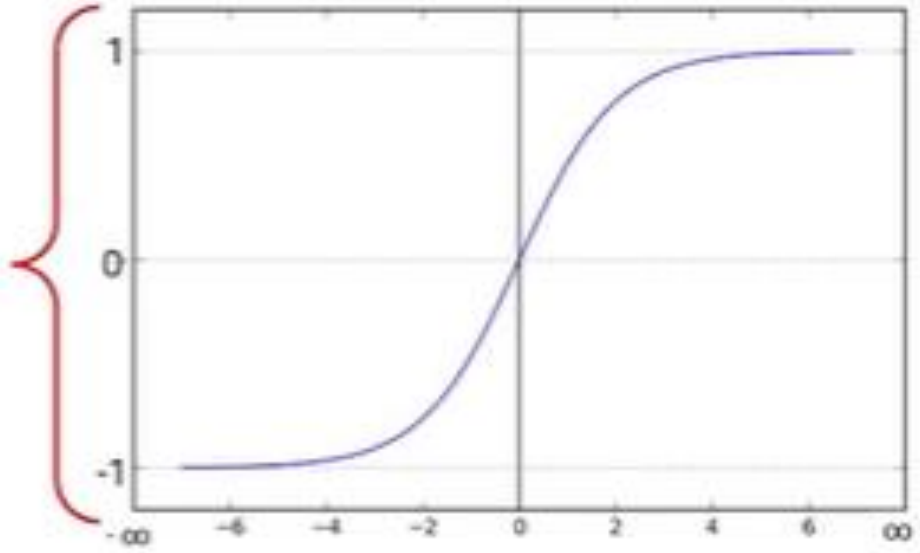
$$\sigma(x) = \frac{1}{1+e^{-(x)}}$$

$$tanh(x) = 2\,\sigma(2x) - 1$$

A factor of 2 and -1, that is why it is called scale activation function.

# Graph of Tanh

- Tanh Activation Function: $$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$
- Tanh Activation Function Graph:
- Input Range: (- ∞ to ∞)
- Output Range: ( -1 to 1)



# Comparison of Sigmoid and Tanh

- Tanh function is steeper at the centre and it is simply a scaled version of sigmoid.
- It is continuous differentiable at all points.

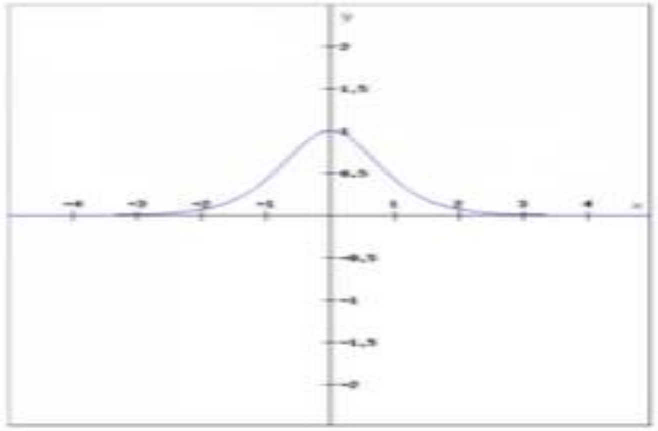
*Graph of Derivative of tanh*

- Tanh Activation Function:

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- Tanh Function derivative:

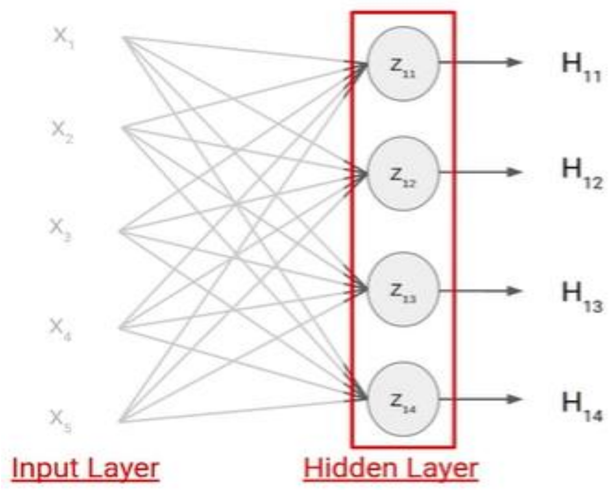
$$\frac{\mathrm{d}\,tanh}{\mathrm{d}\,x} = 1 - tanh^2(x)$$

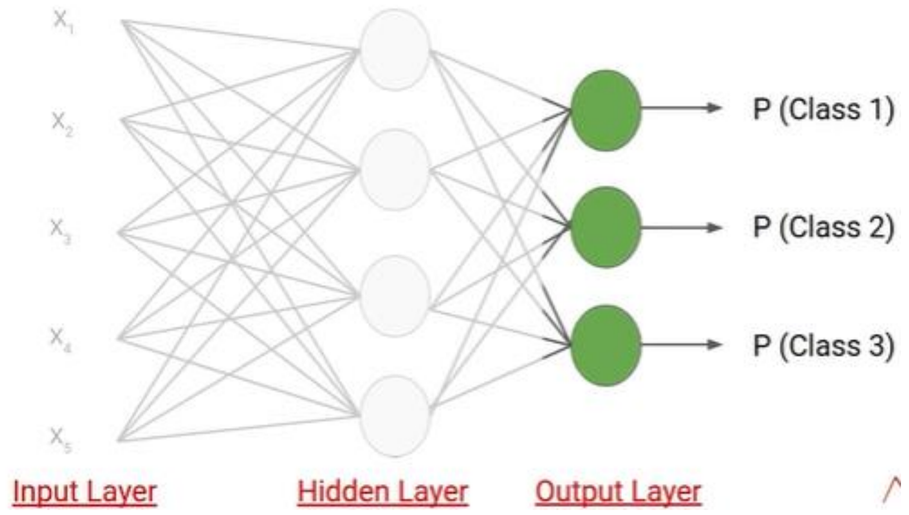


- In the graph, the values of tanh are comparatively larger. Hence the training in tanh is faster as compared to sigmoid since the gradient values would be larger and hence the update process would be faster. Check Tensorflow playground.

# Softmax Activation Function

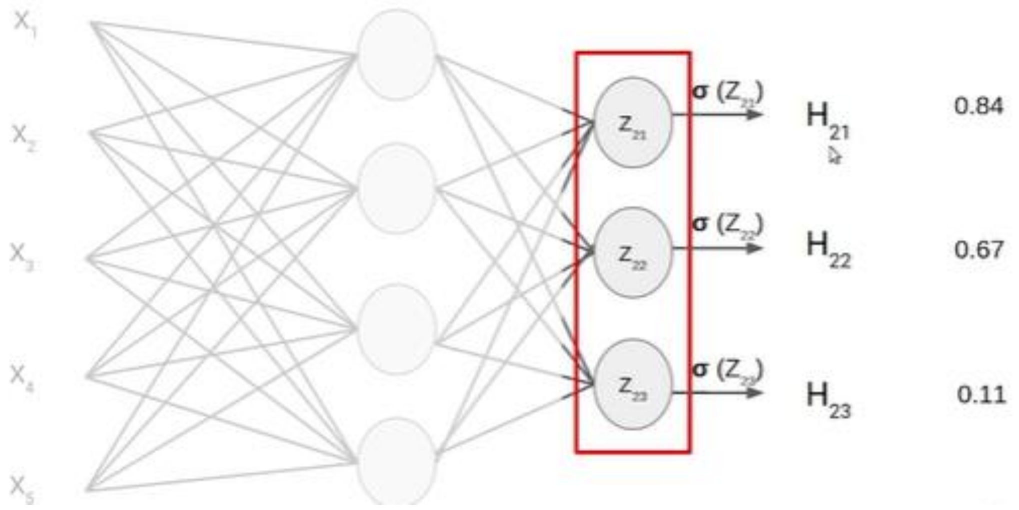
- Used for Multi-class Classification problems.
- Used in the output layer



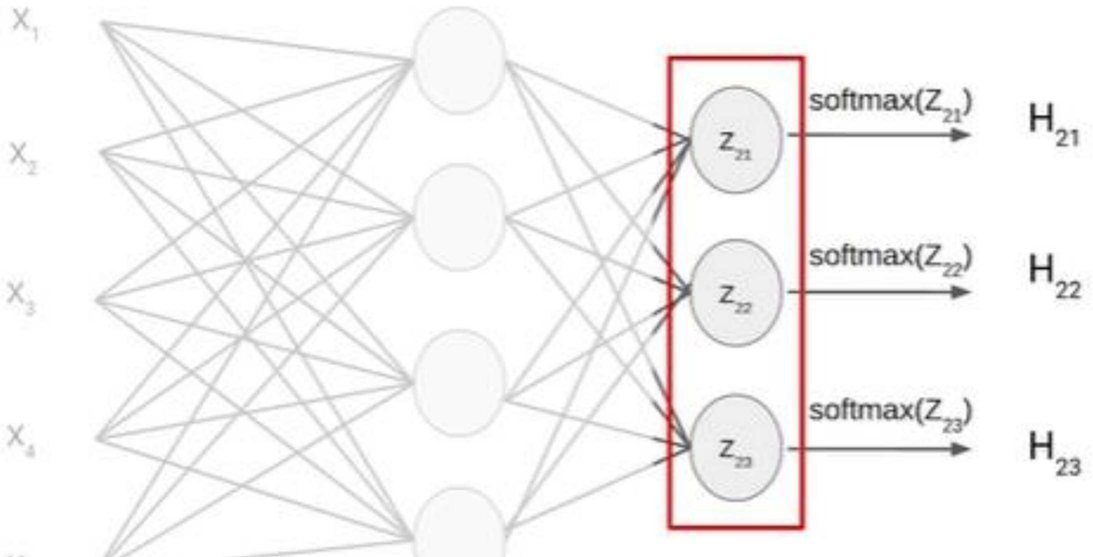


- The neurons in the output layer will be equal to the target classes. Each of individual neuron will give probability of the individual classes.

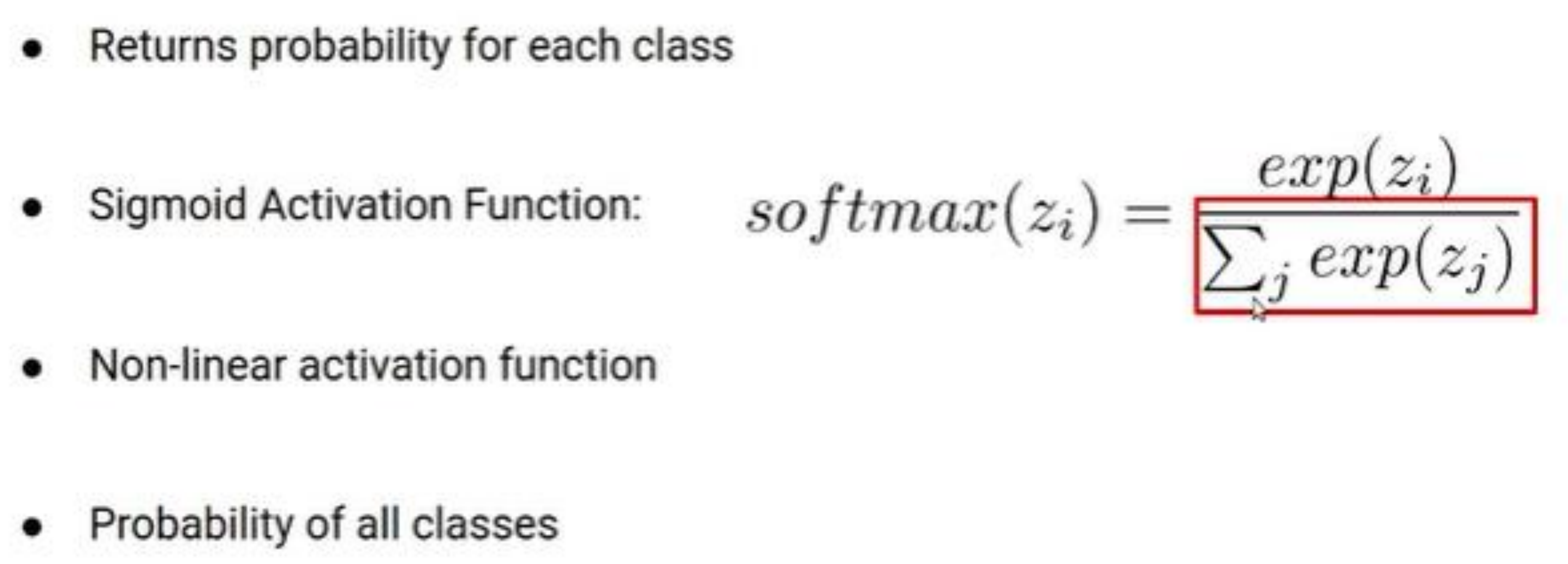
Sigmoid cannot be applied



Softmax is  applied : calculates relative probabilities.



# Softmax Activation Function



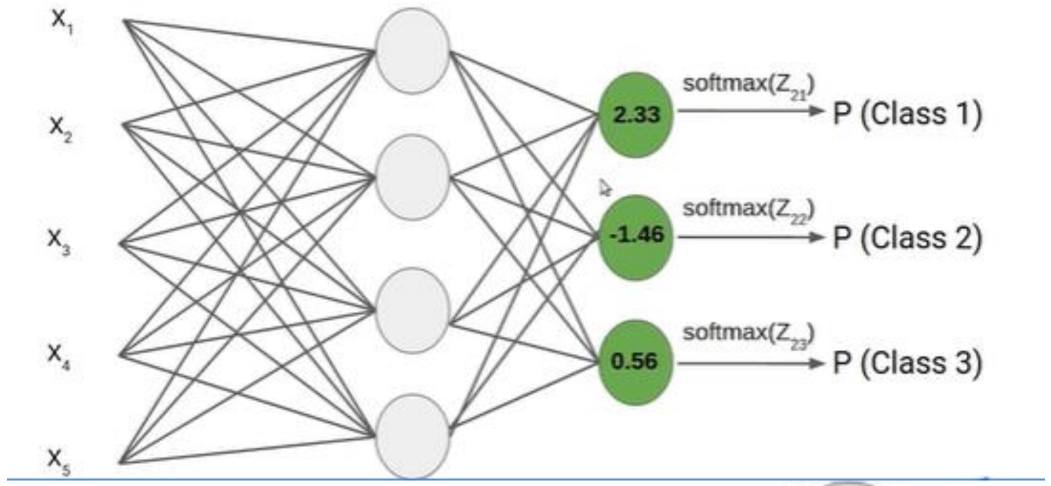
- Returns probability for each class
- Sigmoid Activation Function:

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

- Non-linear activation function
- Probability of all classes

Z values represent the values from the neurons in the output layer. If the number of neurons is 3, there will be 3 z values. Exp acts  as the non linear functions. These values are divided by the sum of exp values in order to normalize the values and  convert them into probabilities.  If the no of classes is 2, it becomes same as sigmoid function. So sigmoid is just a variant of Softmax.

# Softmax Example



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

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

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