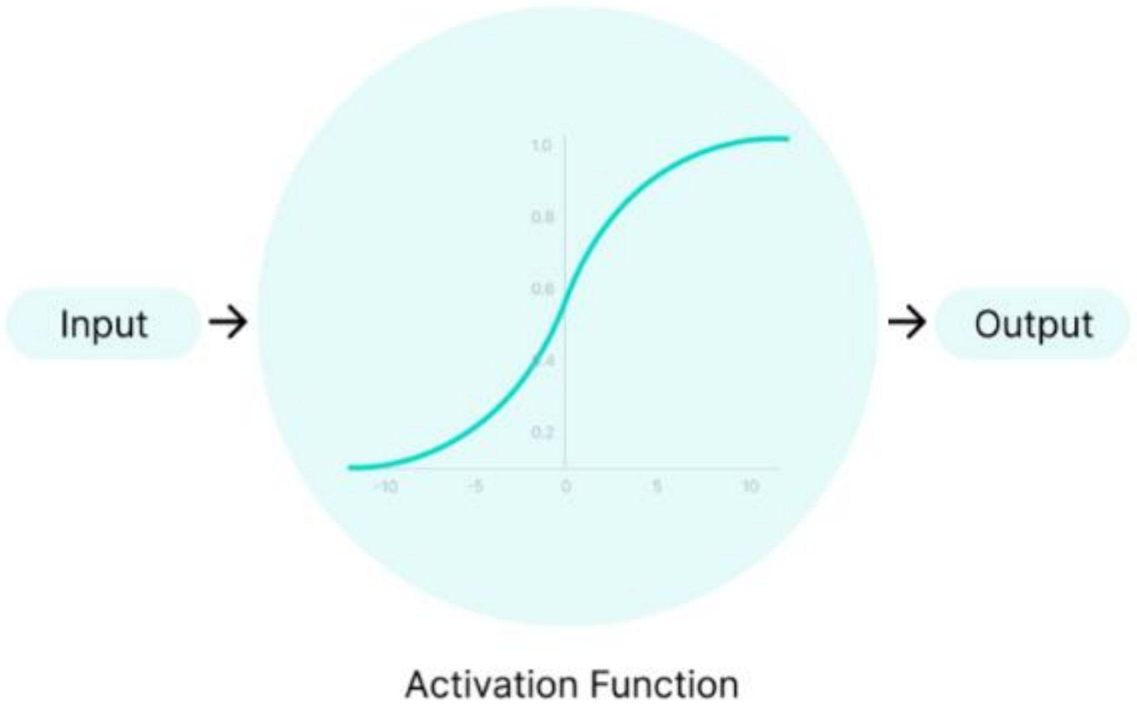
# ANN AND DEEP LEARNING (CS636)

BY: Nidhi S. Periwal,
Teaching Assistant,
COED, SVNIT, Surat

# Activation Function

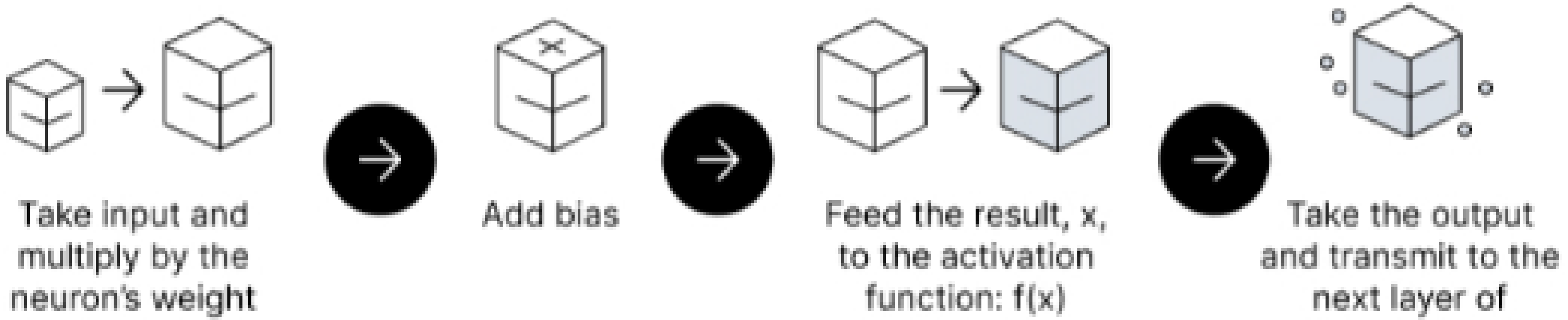
- **An Activation Function/Threshold function** decides whether a neuron should **be activated or not.**
- it will decide whether the neuron’s input to the network is **important or not** in the process of prediction using simpler mathematical operations.
- **Activation Function** helps the neural network to use important information while **suppressing irrelevant data points.**
- The role of the Activation Function **is to derive output from a set of input values fed to a node** (or a layer)

# Activation Function



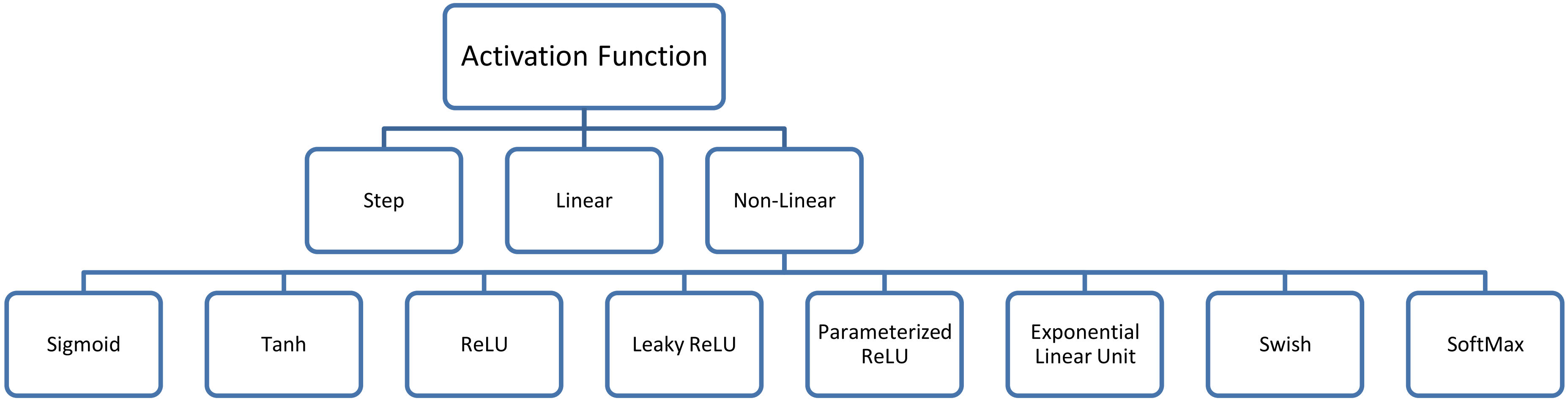
# Activation Function

- In the feedforward propagation, the Activation Function is a mathematical "gate" in between the input feeding the current neuron and its output going to the next layer.


Take input and multiply by the neuron's weight
Add bias
Feed the result, x, to the activation function: f(x)
Take the output and transmit to the next layer of

## Activation Function

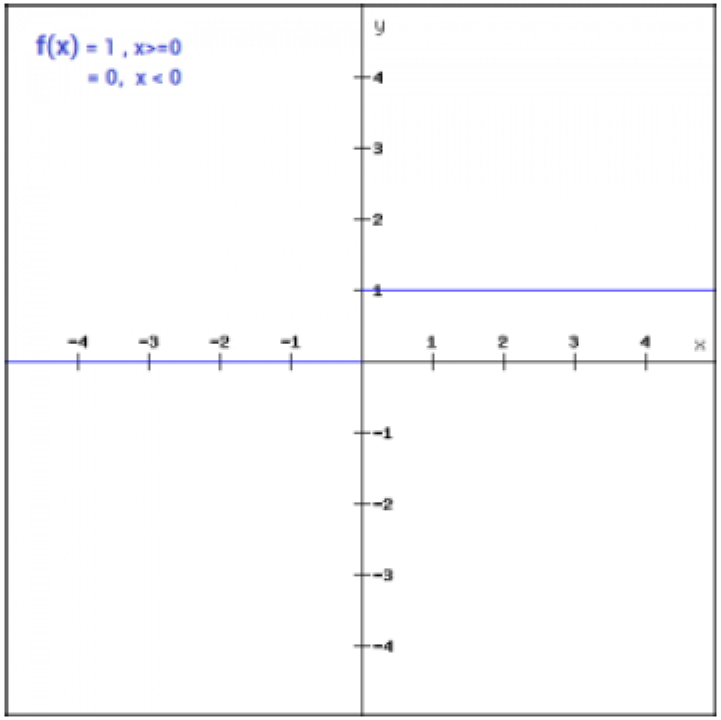
- Activation functions introduce an additional step at each layer during the **forward propagation**,
- Necessity :
  - If not used, Every neuron will only be performing a **linear transformation** on the inputs using the weights and biases.
  - Hence, our model would be just a **linear regression model.**
  - Used to enable a **limited amplitude** of the output of a neuron and enabling it in a **limited range** is known as **squashing functions. (**A squashing function squashes the amplitude of output signal into a finite value)


Activation Function
Step
Linear
Non-Linear
Sigmoid
Tanh
ReLU
Leaky ReLU
Parameterized ReLU
Exponential Linear Unit
Swish
SoftMax

# Activation Function

**Binary Step Function**

- It depends on a threshold value that decides **whether a neuron should be activated or not**.
- The input fed to the activation function is compared to a certain **threshold**; if the input is greater than it, **then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.**
- It cannot provide **multi-value outputs—for** example, it cannot be used for multi-class classification problems.



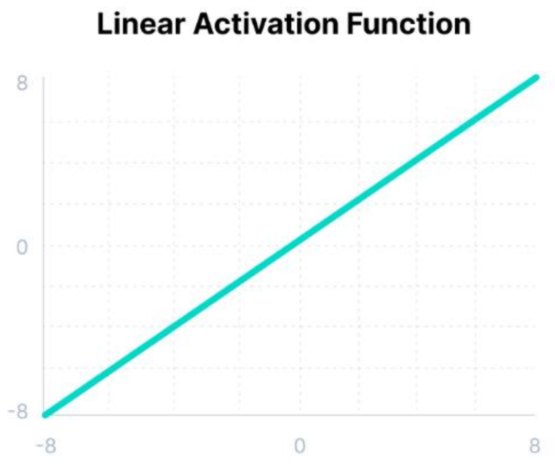
*Binary step*

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

# Activation Function

**Linear Activation Function**

- It is directly proportional to the input.
- The final layer of the **Neural Network will be working as a linear function of the first layer**.
- **Does not** allow the model to create **complex mappings** between the network's inputs and outputs.
- Here the derivative of the function f(x) is equal to the **value of constant used**.
- There **isn't much benefit of using linear function because the neural network would not improve the error due to the same value of gradient** for every iteration.
- Linear functions are ideal where interpretability is required and for simple tasks.



*f(x) = ax + c*

## Activation Function

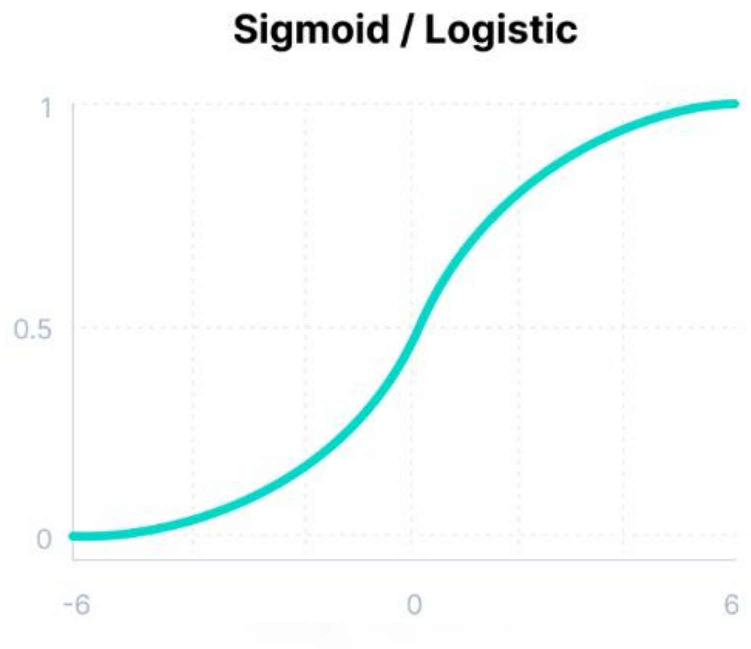
**Non-Linear Activation Functions**

- They allow **backpropagation** because the derivative function would be related to the input and it’s possible to go back and understand which **weights in the input neurons can provide a better prediction**.
- They allow the **stacking** of **multiple** layers of neurons as the output would now be a non-linear combination of input passed through multiple layers.

# Activation Function

**Sigmoid / Logistic Activation Function**

- This function takes any real value as input and outputs values in the range of 0 to 1.
- S-shape
- The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0
- Suitable for Binary Classification

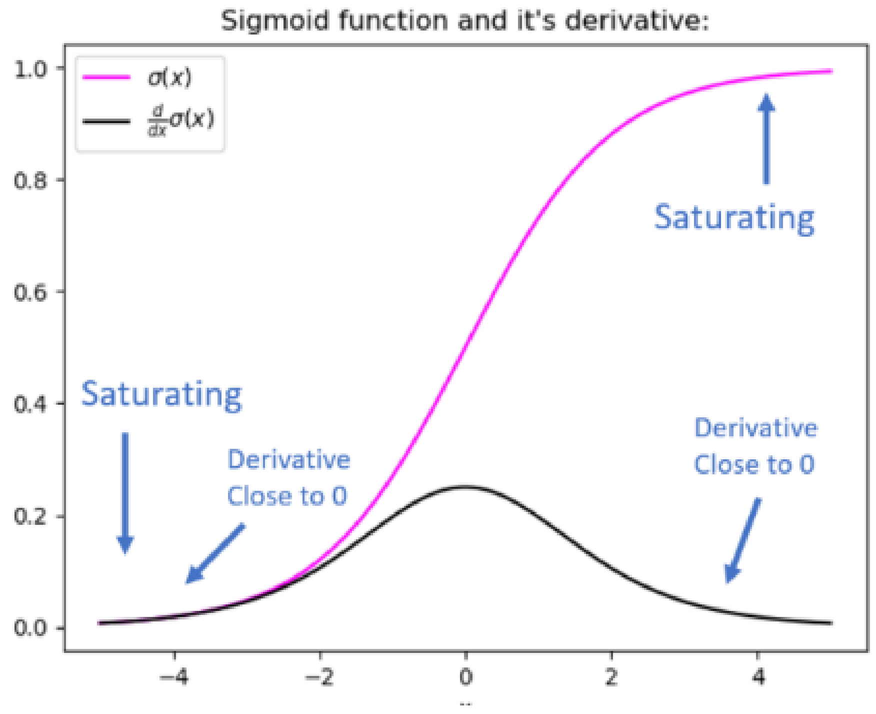


$$\phi(z) = \frac{1}{1 + e^{-z}}$$

# Activation Function

**Sigmoid / Logistic Activation Function**

- Sigmoids saturate and kill gradients.
- (See derivative in figure)Top and bottom level of sigmoid functions the curve changes slowly, if we calculate slope(gradients) it is zero.
- **Hence,** When the x value is small or big the slope is zero->then there is no learning, which leds to Slow convergence (**Vanishing gradients**).
- The output of the logistic function is not symmetric around zero. So the output of all the neurons will be of the same sign. This makes the training of the neural network more difficult and unstable



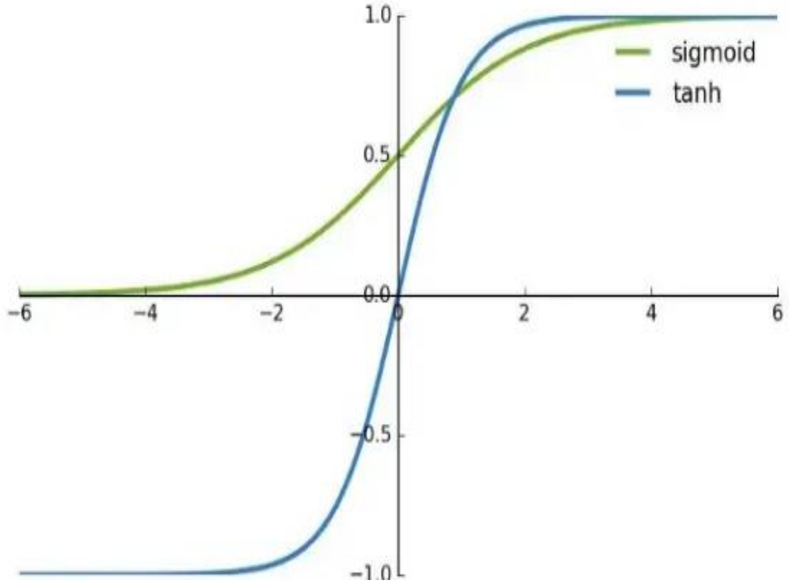
- **Gradients are used during training to update the network weights**
- **Vanishing gradients:** This occurs when the gradient is too small. As we move backwards during backpropagation, **the gradient continues to become smaller, causing the earlier layers in the network to learn more slowly than later layers.** When this happens, the weight parameters update until they become insignificant—i.e. 0—resulting in an algorithm that is no longer learning
- Eg: A large change in the input of the sigmoid function will cause a small change in the output.
- Hence, the derivative becomes small**. For shallow networks with only a few layers that use these activations, this isn't a big problem. However, when more layers are used, it can cause the gradient to be too small for training to work effectively**.

# Activation Function

**Tanh Activation Function**

- Tanh function is very similar to the **sigmoid/logistic activation function,** and even has the **same S-shape with the difference in output range of -1 to 1.**
- In Tanh, the larger the input (more positive), the closer the output value will be to **1.0**, whereas the smaller the input (more negative), the closer the output will be to **-1.0.**
- It is bipolar in nature
- It is continuous activation function

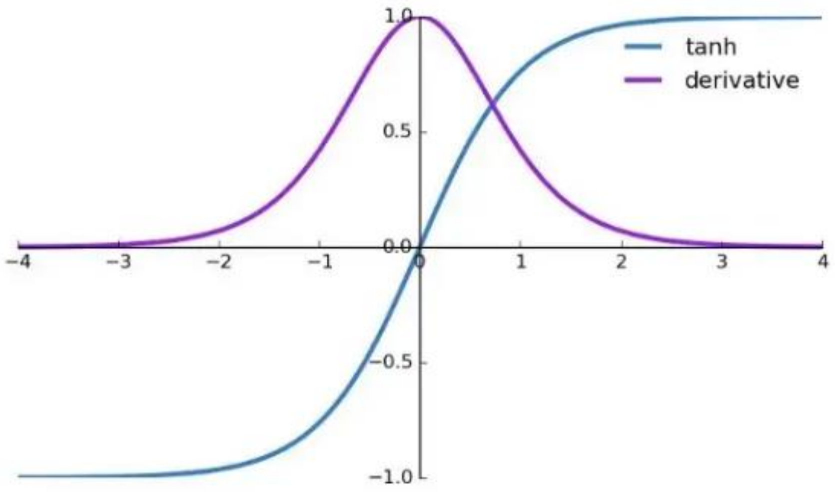
$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$



# Activation Function

**Tanh Activation Function**

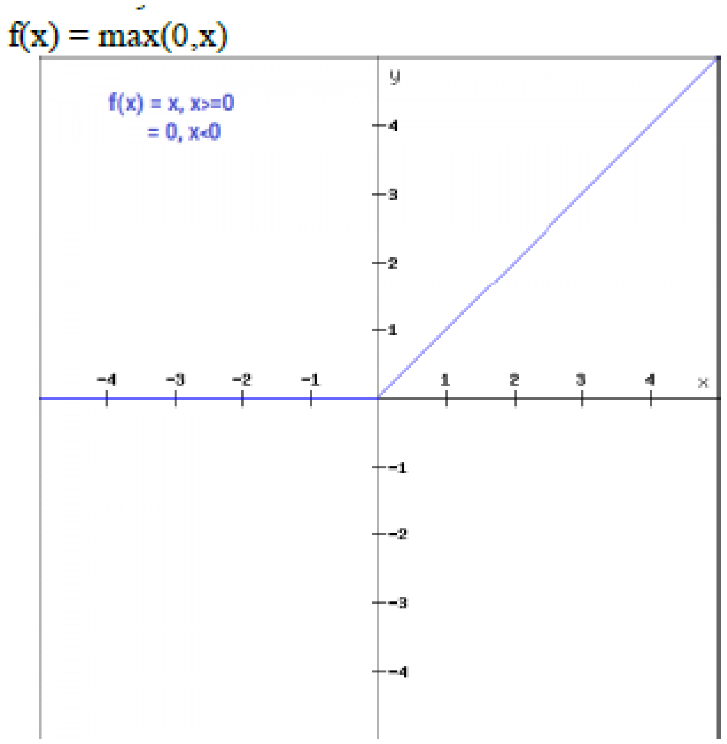
- Here, output is zero centered because its range in between -1 to 1 i.e -1 < output < 1. Hence, preferred more as compared to sigmoid.
- Usually used in **hidden layers** of a neural network as it's values lies between**-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.
- As per derivative, it also suffers from **Vanishing gradients**



# Activation Function

**RELU Activation Function**

- ReLU stands for rectified linear unit and is a non-linear activation function which is widely used in neural network in hidden layers.
- ***f(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- Linear for x greater or equal to 0, non-linear otherwise.
- **Value Range :-** [0, inf)
- Referred as piece–wise linear or hinge function.

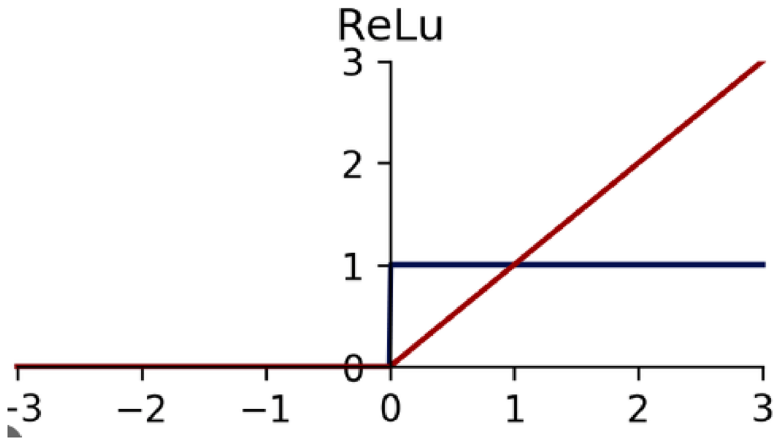


- *Monotonic means something that does not vary or change.*
- *Non-Monotonic means something which can vary according to the situation or condition.*

## Activation Function

**RELU Activation Function**

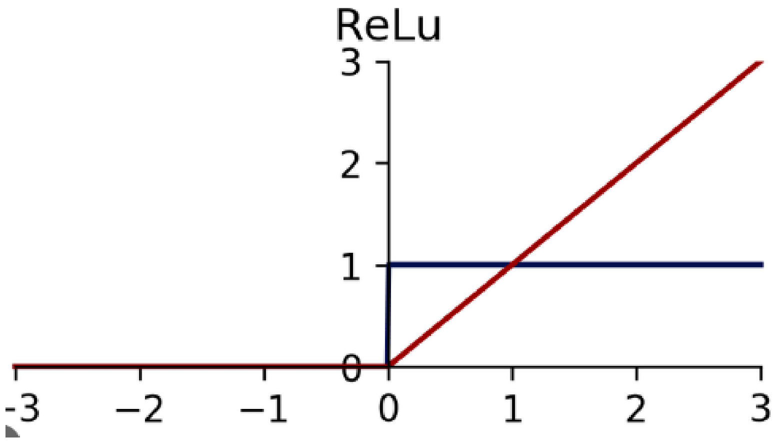
- The function and its derivative (in blue in figure) **both are monotonic.**
- ReLU has a derivative function and allows for **backpropagation** while simultaneously making it computationally efficient. It **converges faster** as compared to sigmoid and tanh.
- It involves simple calculation as compared to sigmoid and tanh, hence **Computation is faster.**



# Activation Function

**RELU Activation Function**

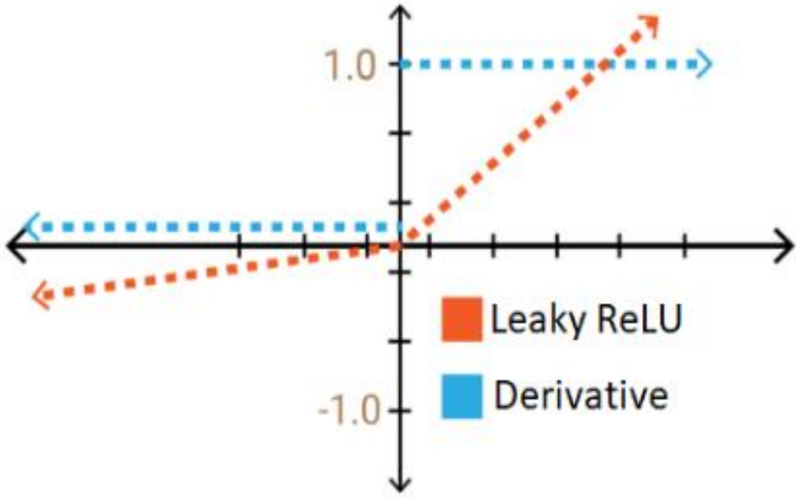
- **Issue**: **(Dying ReLU)** Any negative input given to the ReLU activation function turns the value into **zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately**. Hence, decreases the model’s ability to fit or train from the data properly.
- **Moreover,** Some gradients can be fragile during training and can die. It can cause a **weight update which will makes it never activate on any data point again.** Simply saying that ReLu could result in **Dead Neurons.**



# Activation Function

**Leaky RELU Activation Function**

- Leaky ReLU is **an improvised version of ReLU function where for negative values of x,** instead of defining the ReLU functions' value as zero, it is defined as **extremely small linear component of x.**
- **Range** of the Leaky ReLU is (-infinity to infinity)

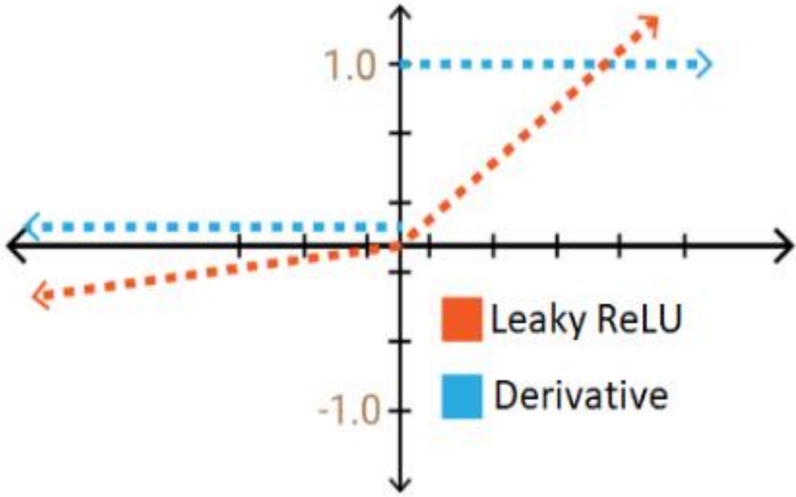


$$f(x) = 0.01x,\ x < 0$$
$$f(x) = x,\ x >= 0$$

# Activation Function

**Leaky RELU Activation Function**

- **It** enables backpropagation, even for negative input values
- The gradient of the left side of the graph comes out to be a non-zero value. Therefore, we would **no longer encounter dead neurons in that region.**
- Limitations :
  - The predictions may not be consistent for negative input values.
  - The gradient for negative values is a small value that makes the learning of model parameters time-consuming

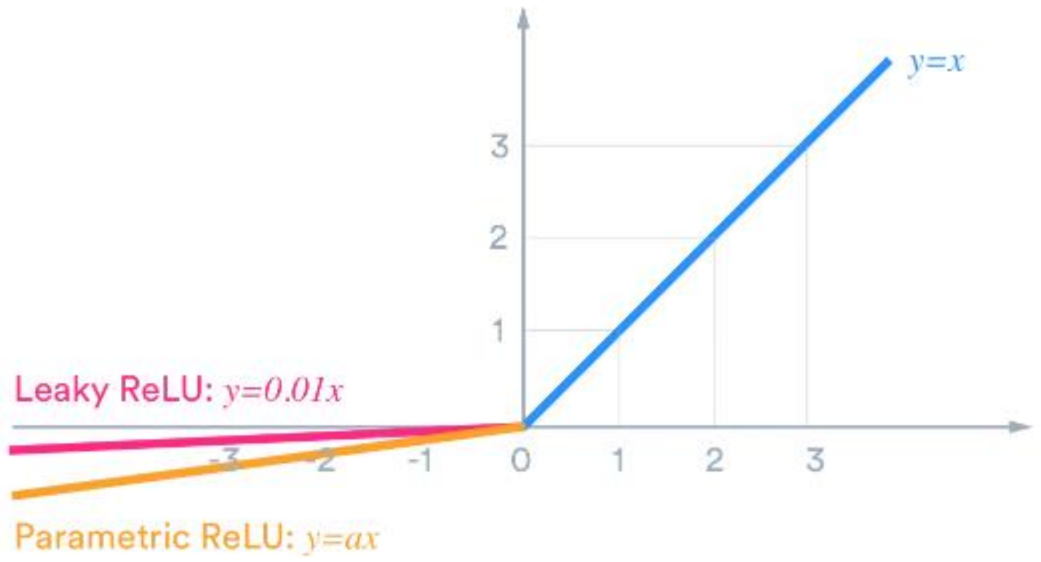


$$f'(x) = g(x) = 1,\ x>=0$$
$$= 0.01,\ x<0$$

# Activation Function

**Parameterized RELU Activation Function**

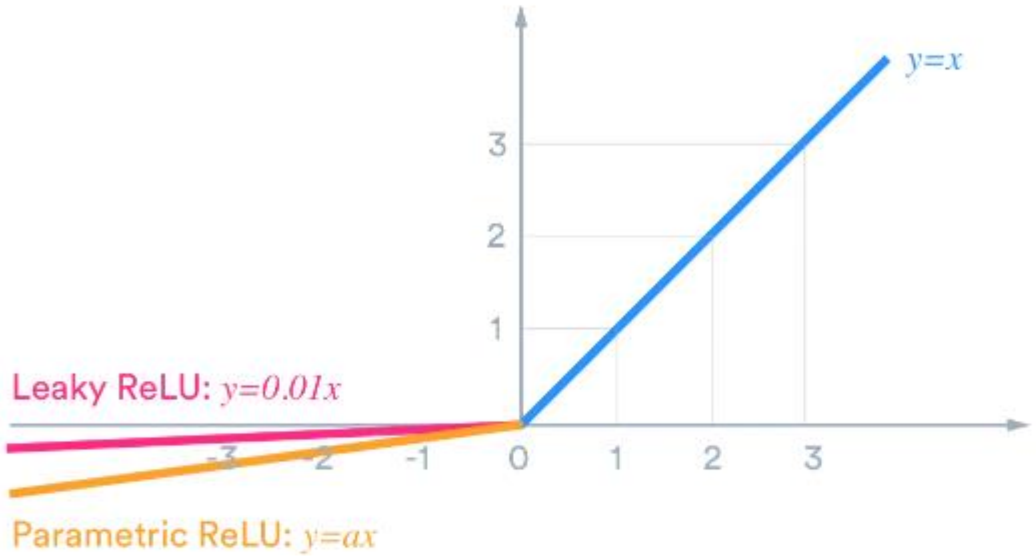
- This function provides the slope of the negative part of the function as an argument ***a***.
- When **a is not 0.01** then it is called **Parameterized ReLU**
- By performing backpropagation, the most appropriate value of *a* is learnt.



# Activation Function

**Parameterized RELU Activation Function**

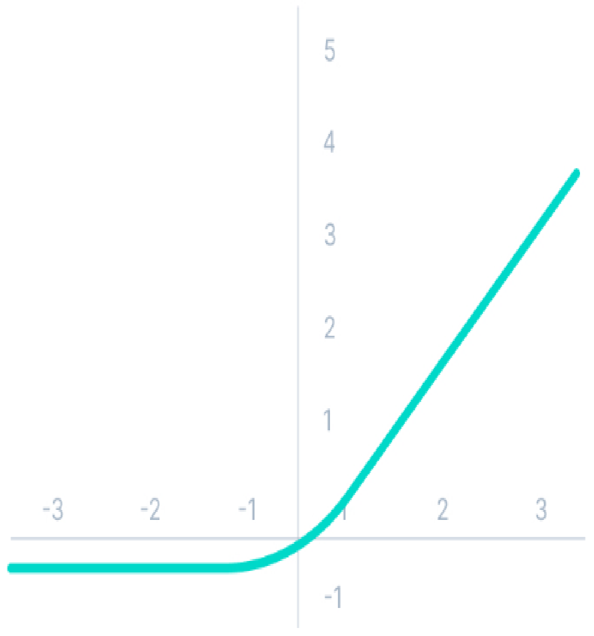
- The parameterized ReLU function is used when the **leaky ReLU function still fails at solving the problem of dead neurons,** and the relevant information is not successfully passed to the next layer.
- This function's limitation is that it may perform differently for different problems depending upon the value of slope parameter ***a***



# Activation Function

**ELU Activation Function**

- Exponential Linear Unit or ELU is also a variant of Rectified Linear Unit. ELU introduces a parameter slope for the negative values of x. It uses a log curve for defining the negative values
- Avoids **dead ReLU problem by introducing log curve for negative values of input**. It helps the network nudge weights and biases in the **right direction.**
- Issue: It increases the computational time because of the exponential operation included



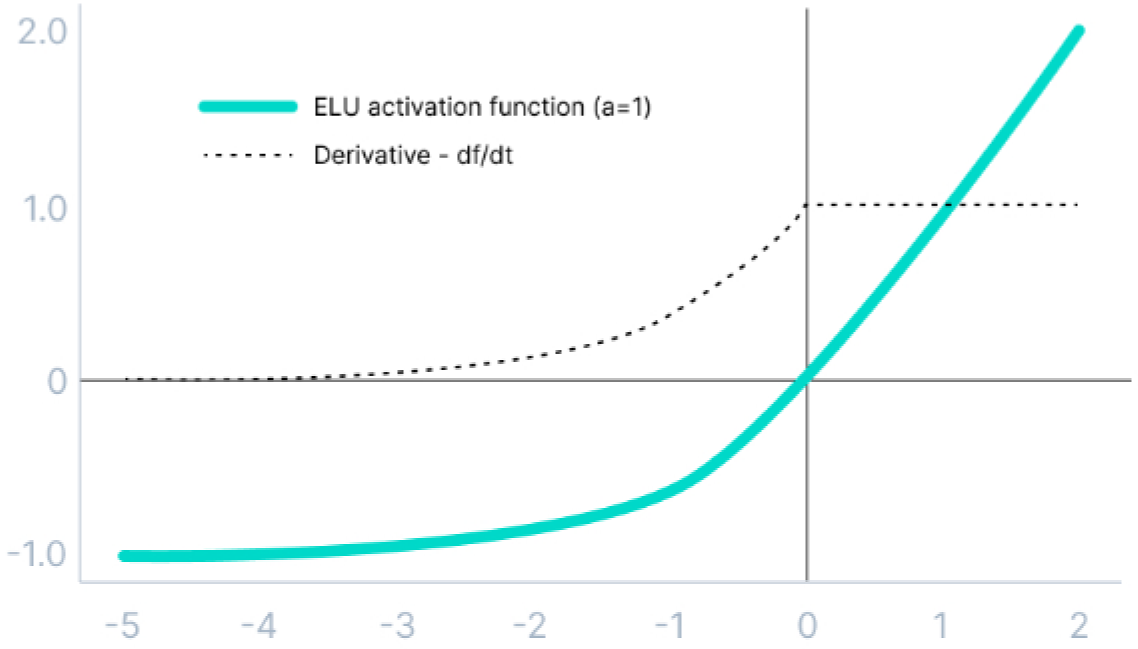
*ELU*

$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha\left(e^{x}-1\right) & for\ x < 0 \end{cases}$$

# Activation Function

## ELU Activation Function

- **Exploding Gradient Problem**



- Exploding gradients are **a problem when large error gradients accumulate and result in very large updates to neural network model weights during training**.
- An unstable network can result when there are exploding gradients, and the learning cannot be completed.
- The values of the weights can also become so large as to overflow.

## Activation Function

**Softmax**

- It is most commonly used as an activation function for the last layer of the neural network in the case of **multi-class classification.**
- It **generates probability vector to indicate the probability of each of the classes**
- Softmax function is a combination of multiple sigmoid functions.
- The function, for every data point of all the **individual classes, returns the probability**.

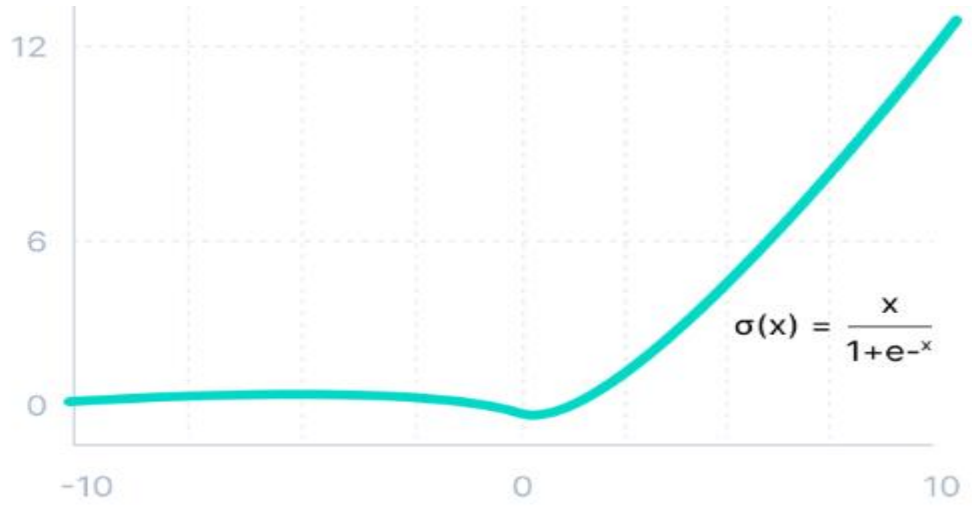
**SOFTMAX FUNCTION**

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

# Activation Function

**Swish**

- Swish consistently matches or outperforms ReLU activation function on deep networks applied to various challenging domains such as image classification, machine translation etc.
- This function is bounded below but unbounded above i.e. *Y* approaches to a constant value as *X* approaches negative infinity but *Y* approaches to infinity as *X* approaches infinity.



$f(x) = x * \text{sigmoid}(x)$

$f(x) = x/(1 - e^{-x})$

## Activation Function

**Swish**

- Swish is a smooth function that means that it does not abruptly change direction like ReLU does near x = 0. Rather, it smoothly bends from 0 towards values < 0 and then upwards again.
- Small negative values were zeroed out in ReLU activation function. However, those negative values may still be relevant for capturing patterns underlying the data.
- **Large negative values are zeroed out for reasons of sparsity making it a win-win situation.**

Thank You!