# ML

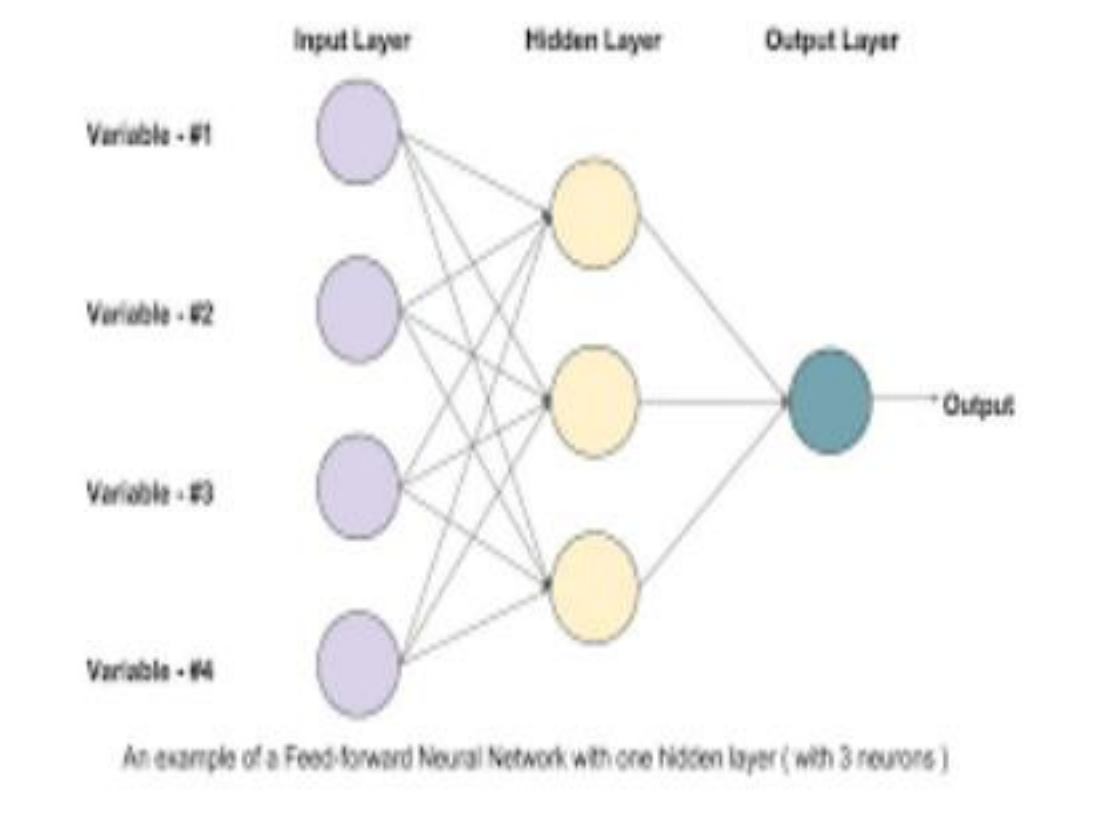
## LECTURE-22

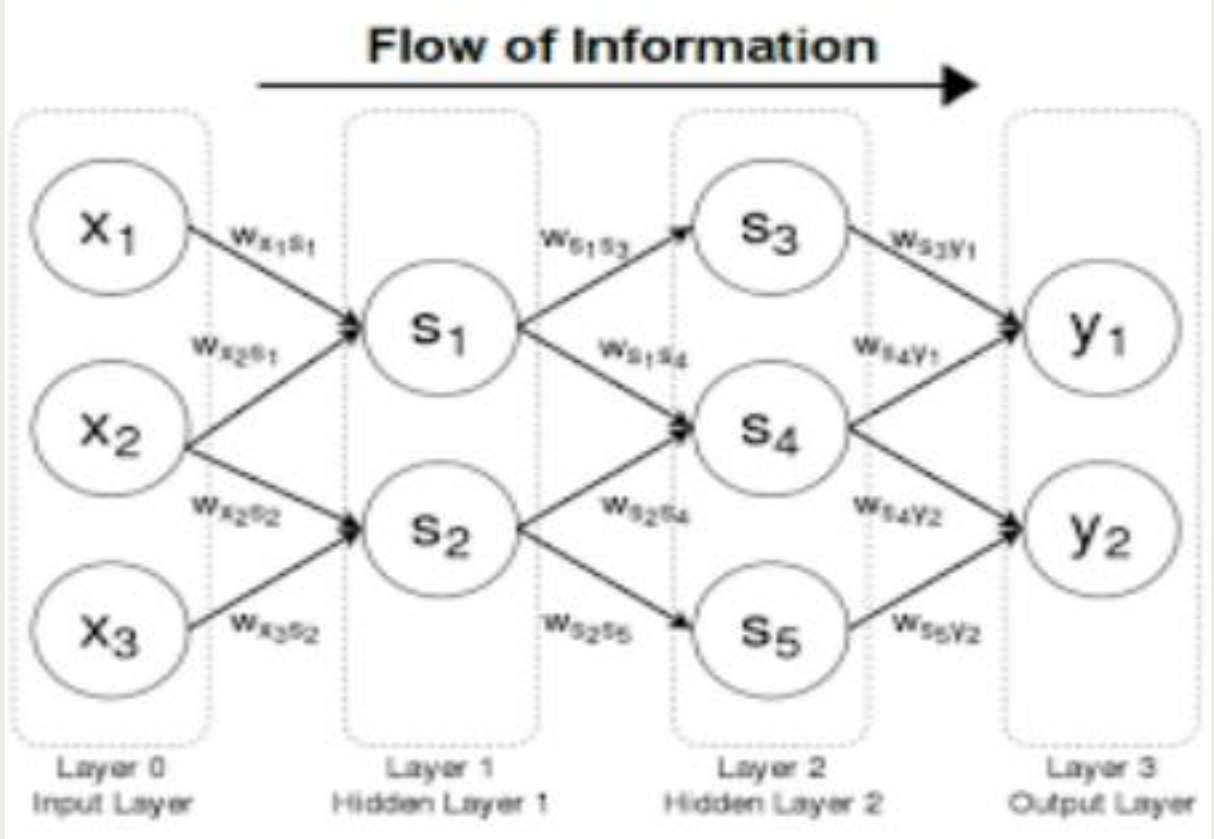
BY
Dr. Ramesh Kumar Thakur
Assistant Professor (II)
School Of Computer Engineering

# What is Feed-forward Neural Networks?

- A feedforward neural network is a type of artificial neural network in which **nodes' connections do not form a loop.**
- Often referred to as a multi-layered network of neurons, feedforward neural networks are so named because **all information flows in a forward manner only.**
- The **data enters the input nodes, travels through the hidden layers, and eventually exits the output nodes.**
- The network is devoid of links that would allow the information exiting the output node to be sent back into the network.
- The **purpose of feedforward neural networks is to approximate functions.**
- There is a **classifier using the formula y = f* (x).**
- **This assigns the value of input x to the category y.**
- **The feedforward network will map y = f (x; θ). It then memorizes the value of θ that most closely approximates the function.**

# Diagram of Feed-forward Neural Networks?



# Components of Feedforward Neural Network

- **Input layer**
- It **contains the neurons that receive input.**
- The data is subsequently passed on to the next layer.
- The **input layer's total number of neurons is equal to the number of variables in the dataset.**
- **Hidden layer**
- This is the **intermediate layer, which is concealed between the input and output layers.**
- This layer has a **large number of neurons that perform alterations on the inputs.**
- They then communicate with the output layer.
- **Output layer**
- It is the **last layer and is depending on the model's construction**.
- Additionally, **the output layer is the expected feature**, as you are aware of the desired outcome.
- **Neurons weights**
- Weights are used to describe the **strength of a connection between neurons**.
- The **range of a weight's value is from 0 to 1.**

# Cost Function in Feedforward Neural Network

- The cost function is an important factor of a feedforward neural network.
- Generally, minor adjustments to weights and biases have little effect on the categorized data points.
- Thus, to determine a method for improving performance by making minor adjustments to weights and biases using a smooth cost function.
- The mean square error cost function is defined as follows:

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

- Where,
- w = weights collected in the network
- b = biases
- a = output vectors
- x = input
- ‖v‖ = usual length of vector v.

# Loss Function in Feedforward Neural Network

**Cross Entropy Loss:**

$$L(\Theta) = \begin{cases} -log(\hat{y}) & \text{if } y = 1 \\ -log(1-\hat{y}) & \text{if } y = 0 \end{cases}$$

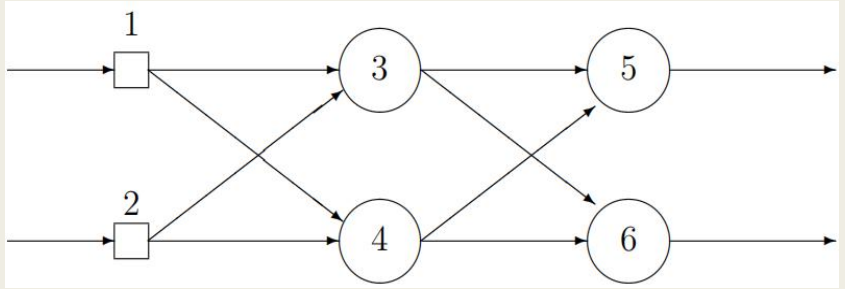
- The cross-entropy loss associated with multi-class categorization is as follows:

**Cross Entropy Loss:**

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

# Numerical on Feedforward Neural Network

- The following diagram represents a feed-forward neural network with one hidden layer:




- A weight on connection between nodes i and j is denoted by $w_{ij}$ , such as $w_{13}$ is the weight on the connection between nodes 1 and 3. The following table lists all the weights in the network:

| | |
|---|---|
| $w_{13} = -2$ | $w_{35} = 1$ |
| $w_{23} = 3$ | $w_{45} = -1$ |
| $w_{14} = 4$ | $w_{36} = -1$ |
| $w_{24} = -1$ | $w_{46} = 1$ |

- Each of the nodes 3, 4, 5 and 6 uses the following activation function:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- where v denotes the weighted sum of a node. Each of the input nodes (1 and 2) can only receive binary values (either 0 or 1). Calculate the output of the network ($y_5$ and $y_6$) for each of the input patterns:

| Pattern: | $P_1$ | $P_2$ | $P_3$ | $P_4$ |
|---|---|---|---|---|
| Node 1: | 0 | 1 | 0 | 1 |
| Node 2: | 0 | 0 | 1 | 1 |

# Numerical on Feedforward Neural Network

- **Answer:**
- In order to find the output of the network it is necessary to calculate weighted sums of hidden nodes 3 and 4:
- $v_3 = w_{13}x_1 + w_{23}x_2$ , $v_4 = w_{14}x_1 + w_{24}x_2$
- Then find the outputs from hidden nodes using activation function $\phi$:
- $y_3 = \phi(v_3)$ , $y_4 = \phi(v_4)$ .
- Use the outputs of the hidden nodes $y_3$ and $y_4$ as the input values to the output layer (nodes 5 and 6), and find weighted sums of output nodes 5 and 6:
- $v_5 = w_{35}y_3 + w_{45}y_4$ , $v_6 = w_{36}y_3 + w_{46}y_4$
- Finally, find the outputs from nodes 5 and 6 (also using $\phi$):
- $y_5 = \phi(v_5)$ , $y_6 = \phi(v_6)$

# Numerical on Feedforward Neural Network

$P_1$: *Input pattern* $(0, 0)$

$$v_3 = -2 \cdot 0 + 3 \cdot 0 = 0, \qquad y_3 = \varphi(0) = 1$$
$$v_4 = 4 \cdot 0 - 1 \cdot 0 = 0, \qquad y_4 = \varphi(0) = 1$$
$$v_5 = 1 \cdot 1 - 1 \cdot 1 = 0, \qquad y_5 = \varphi(0) = 1$$
$$v_6 = -1 \cdot 1 + 1 \cdot 1 = 0, \qquad y_6 = \varphi(0) = 1$$

*The output of the network is* $(1, 1)$.

$P_2$: *Input pattern* $(1, 0)$

$$v_3 = -2 \cdot 1 + 3 \cdot 0 = -2, \qquad y_3 = \varphi(-2) = 0$$
$$v_4 = 4 \cdot 1 - 1 \cdot 0 = 4, \qquad y_4 = \varphi(4) = 1$$
$$v_5 = 1 \cdot 0 - 1 \cdot 1 = -1, \qquad y_5 = \varphi(-1) = 0$$
$$v_6 = -1 \cdot 0 + 1 \cdot 1 = 1, \qquad y_6 = \varphi(1) = 1$$

*The output of the network is* $(0, 1)$.

# Numerical on Feedforward Neural Network

$P_3$: *Input pattern* $(0, 1)$

$$v_3 = -2 \cdot 0 + 3 \cdot 1 = 3, \qquad y_3 = \varphi(3) = 1$$
$$v_4 = 4 \cdot 0 - 1 \cdot 1 = -1, \qquad y_4 = \varphi(-1) = 0$$
$$v_5 = 1 \cdot 1 - 1 \cdot 0 = 1, \qquad y_5 = \varphi(1) = 1$$
$$v_6 = -1 \cdot 1 + 1 \cdot 0 = -1, \qquad y_6 = \varphi(-1) = 0$$

*The output of the network is* $(1, 0)$.

$P_4$: *Input pattern* $(1, 1)$

$$v_3 = -2 \cdot 1 + 3 \cdot 1 = 1, \qquad y_3 = \varphi(1) = 1$$
$$v_4 = 4 \cdot 1 - 1 \cdot 1 = 3, \qquad y_4 = \varphi(3) = 1$$
$$v_5 = 1 \cdot 1 - 1 \cdot 1 = 0, \qquad y_5 = \varphi(0) = 1$$
$$v_6 = -1 \cdot 1 + 1 \cdot 1 = 0, \qquad y_6 = \varphi(0) = 1$$

*The output of the network is* $(1, 1)$.

# Activation Functions

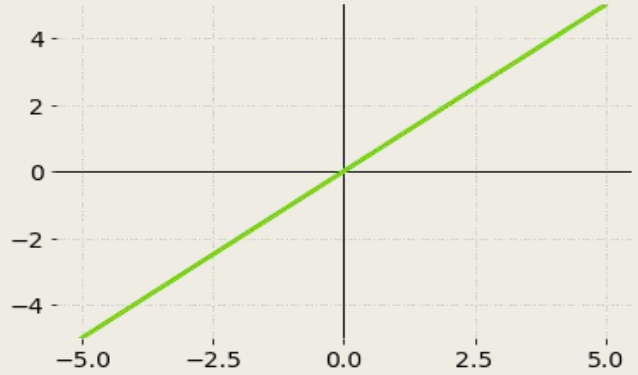
- In artificial neural networks, **each neuron forms a weighted sum of its inputs and passes the resulting scalar value through a function referred to as an activation function**.

- **Why do we need activation functions?**
- An activation function **determines if a neuron should be activated or not activated.**
- This implies that **it will use some simple mathematical operations to determine if the neuron's input to the network is relevant or not relevant in the prediction process.**
- The ability to **introduce non-linearity to an artificial neural network and generate output from a collection of input values fed to a layer** is the purpose of the activation function.

- **Types of Activation functions**
- Activation functions can be divided into three types:
- **Linear Activation Function**
- **Binary Step Function**
- **Non-linear Activation Functions**

# Linear Activation Function

- The linear activation function, often called the **identity activation function**, is **proportional to the input.**
- The range of the linear activation function will be (-∞ to ∞).
- The linear activation function simply adds up the weighted total of the inputs and returns the result.



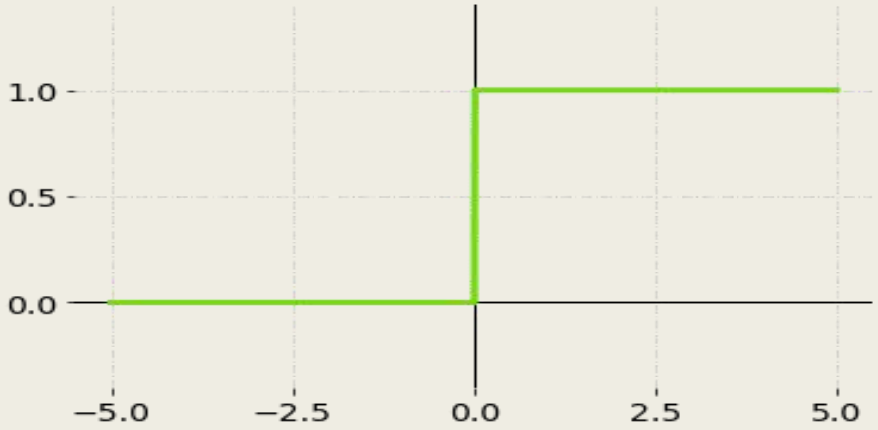

- Mathematically, it can be represented as:

$$f(x) = x$$

- **<u>Pros and Cons</u>**
- It is not a binary activation because the linear activation function only delivers a range of activations.
- We can surely connect a few neurons together, and if there are multiple activations, we can calculate the max (or soft max) based on that.
- The derivative of this activation function is a constant. That is to say, the gradient is unrelated to the x (input).

# Binary Step Activation Function

- A **threshold value determines whether a neuron should be activated or not activated** in a binary step activation function.
- The activation function compares the input value to a threshold value.
- If the input value is greater than the threshold value, the neuron is activated.
- It's disabled if the input value is less than the threshold value, which means its output isn't sent on to the next or hidden layer.




- Mathematically, the binary activation function can be represented as:

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

- **Pros and Cons**
- It cannot provide multi-value outputs—for example, it cannot be used for multi-class classification problems.
- The step function's gradient is zero, which makes the back propagation procedure difficult.

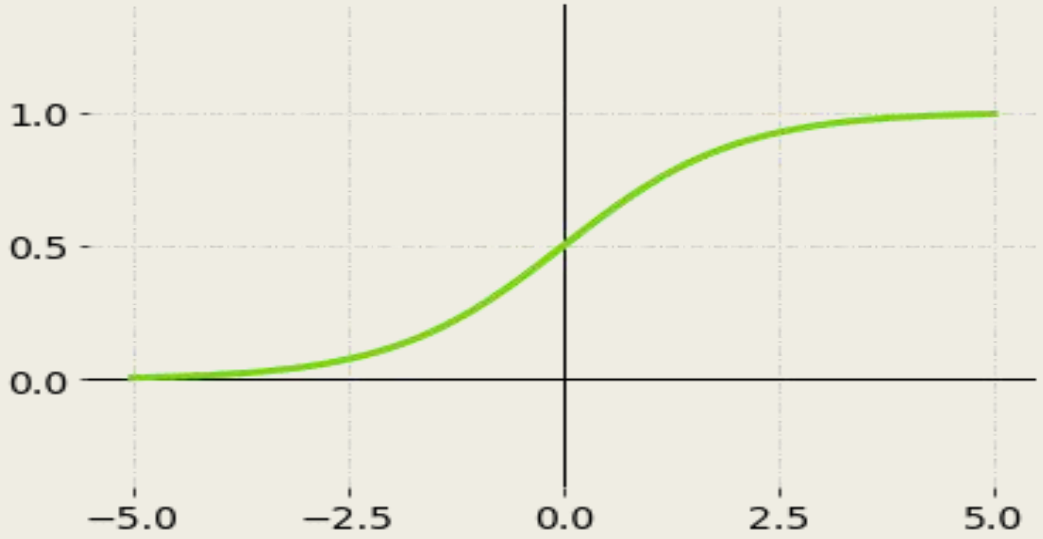
# Non - linear Activation Functions

- The non-linear activation functions are the **most-used activation functions.**
- They make it uncomplicated for an artificial neural network model to adapt to a variety of data and to differentiate between the outputs.
- Non-linear activation functions **allow the stacking of multiple layers of neurons, as the output would now be a non-linear combination of input** passed through multiple layers.
- Any output can be represented as a functional computation output in a neural network.
- These activation functions are **mainly divided basis on their range and curves.**
- Some of the most used non-linear activation functions are:-

1. **Sigmoid**
2. **TanH**
3. **ReLU**
4. **Leaky ReLU**
5. **ELU**
6. **Softmax**
7. **Swish**

# Sigmoid (Logistic Activation Function)

- Sigmoid **accepts a number as input and returns a number between 0 and 1.**
- It's simple to use and has all the desirable qualities of activation functions: nonlinearity, continuous differentiation, monotonicity, and a set output range.
- This is **mainly used in binary classification problems.**
- This sigmoid function gives the probability of an existence of a particular class.




- Mathematically, it can be represented as:

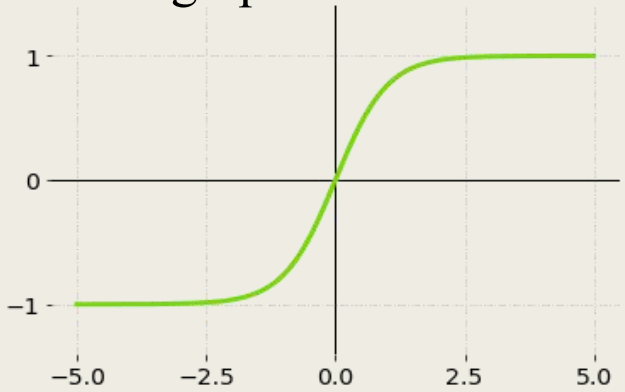
$$f(x) = \frac{1}{1 + e^{-x}}$$

# Sigmoid (Logistic Activation Function)

- **Pros and Cons**
- It is non-linear in nature.
- Combinations of this function are also non-linear, and it will give an analogue activation, unlike binary step activation function.
- It has a smooth gradient too, and It's good for a classifier type problem.
- The output of the activation function is always going to be in the range (0,1) compared to $(-\infty, \infty)$ of linear activation function. As a result, we've defined a range for our activations.
- Sigmoid function gives rise to a problem of "**Vanishing gradients**" and Sigmoids **saturate and kill gradients.**
- Its output isn't zero centred, and it makes the gradient updates go too far in different directions.
- The output value is between zero and one, so it makes optimization harder.
- The network either refuses to learn more or is extremely slow.

# TanH (Hyperbolic Tangent)

- TanH compress a real-valued number to the range [-1, 1].
- It's non-linear, But it's different from Sigmoid,and its output is zero-centered.
- The main advantage of this is that the negative inputs will be mapped strongly to the negative and zero inputs will be mapped to almost zero in the graph of TanH.



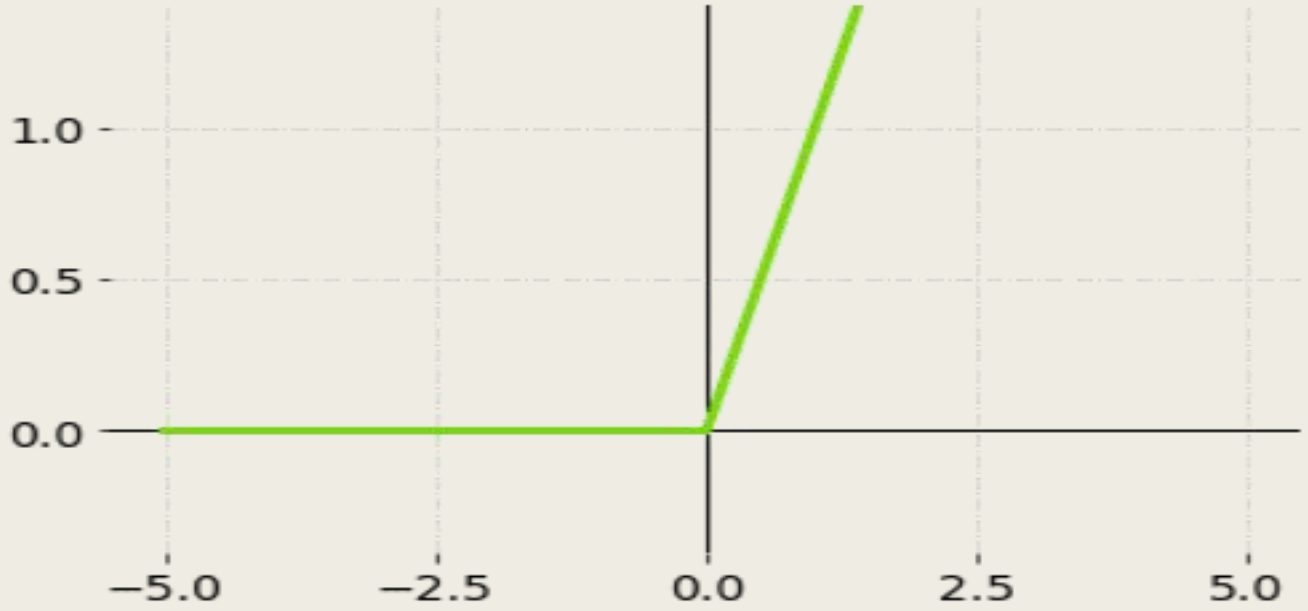

- Mathematically, TanH function can be represented as:

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

- **Pros and Cons**
- TanH also has the **vanishing gradient problem**, but the gradient is stronger for TanH than sigmoid (derivatives are steeper).
- TanH is zero-centered, and gradients do not have to move in a specific direction.

# ReLU (Rectified Linear Unit)

- ReLU stands for Rectified Linear Unit and is one of the most commonly used activation function in the applications.
- It's solved the problem of vanishing gradient because the maximum value of the gradient of ReLU function is one.
- It also solved the problem of saturating neuron, since the slope is never zero for ReLU function. The range of ReLU is between 0 and infinity.




- Mathematically, it can be represented as:

$$f(x) = max\,(0, x)$$

# ReLU (Rectified Linear Unit)

- **Pros and Cons**
- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and TanH functions.
- ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.
- One of its limitations is that it should only be used within hidden layers of an artificial neural network model.
- Some gradients can be fragile during training.
- In other words, For activations in the region ($x<0$) of ReLu, the gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons, which go into that state will stop responding to variations in input (simply because the gradient is 0, nothing changes.) This is called the **dying ReLu problem**.

# Leaky ReLU

- Leaky ReLU is an upgraded version of the ReLU activation function to **solve the dying ReLU problem**, as it has a **small positive slope in the negative area.**
- But, the consistency of the benefit across tasks is presently ambiguous.

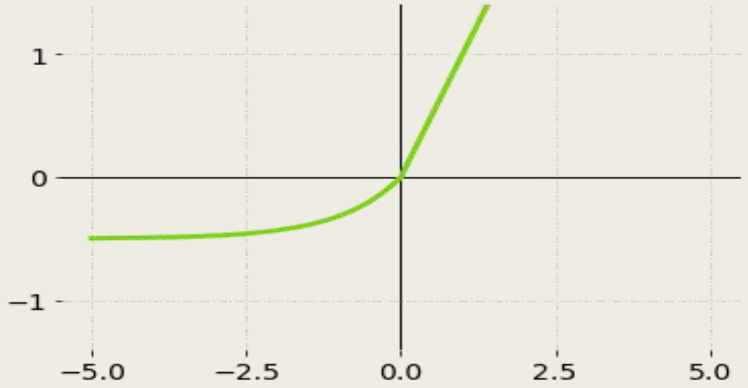

- Mathematically, it can be represented as,

$$f(x) = max\,(0.1x, x)$$

- **<u>Pros and Cons</u>**
- The advantages of Leaky ReLU are the same as that of ReLU, in addition to the fact that **it does enable back propagation, even for negative input values.**
- Making minor modification of negative input values, the gradient of the left side of the graph comes out to be a real (non-zero) value. As a result, there would be no more dead neurons in that area.
- The predictions may not be steady for negative input values.

# ELU (Exponential Linear Units)

- ELU is also one of the variations of ReLU which also solves the dead ReLU problem.
- ELU, just like leaky ReLU also considers negative values by introducing a new alpha parameter and multiplying it will another equation.
- ELU is slightly more computationally expensive than leaky ReLU, and it's very similar to ReLU except negative inputs. They are both in identity function shape for positive inputs.



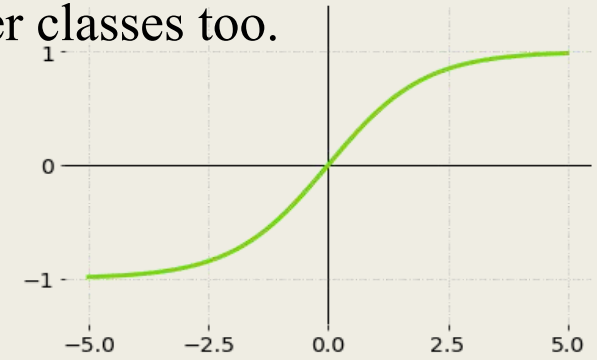

- Mathematically, it can be represented as:

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

- **Pros and Cons**
- ELU is a strong alternative to ReLU. Different from the ReLU, ELU can produce negative outputs.
- Exponential operations are there in ELU, So it increases the computational time.
- No learning about the 'a' value takes place, and exploding gradient problem.

# Softmax

- A **combination of many sigmoids** is referred to as the Softmax function. It determines **relative probability.** Similar to the sigmoid activation function, the Softmax returns the probability of each class.
- In **multi-class classification, softmax activation function is most commonly used for the last layer of the neural network.**
- The **softmax function gives the probability of the current class with respect to others**. This means that it also considers the possibility of other classes too.




- Mathematically, it can be represented as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- **<u>Pros and Cons</u>**
- It mimics the one encoded label better than the absolute values.
- We would lose information if we used absolute (modulus) values, but the exponential takes care of this on its own.
- The softmax function **should be used for multi-label classification and regression task as well.**

# Swish

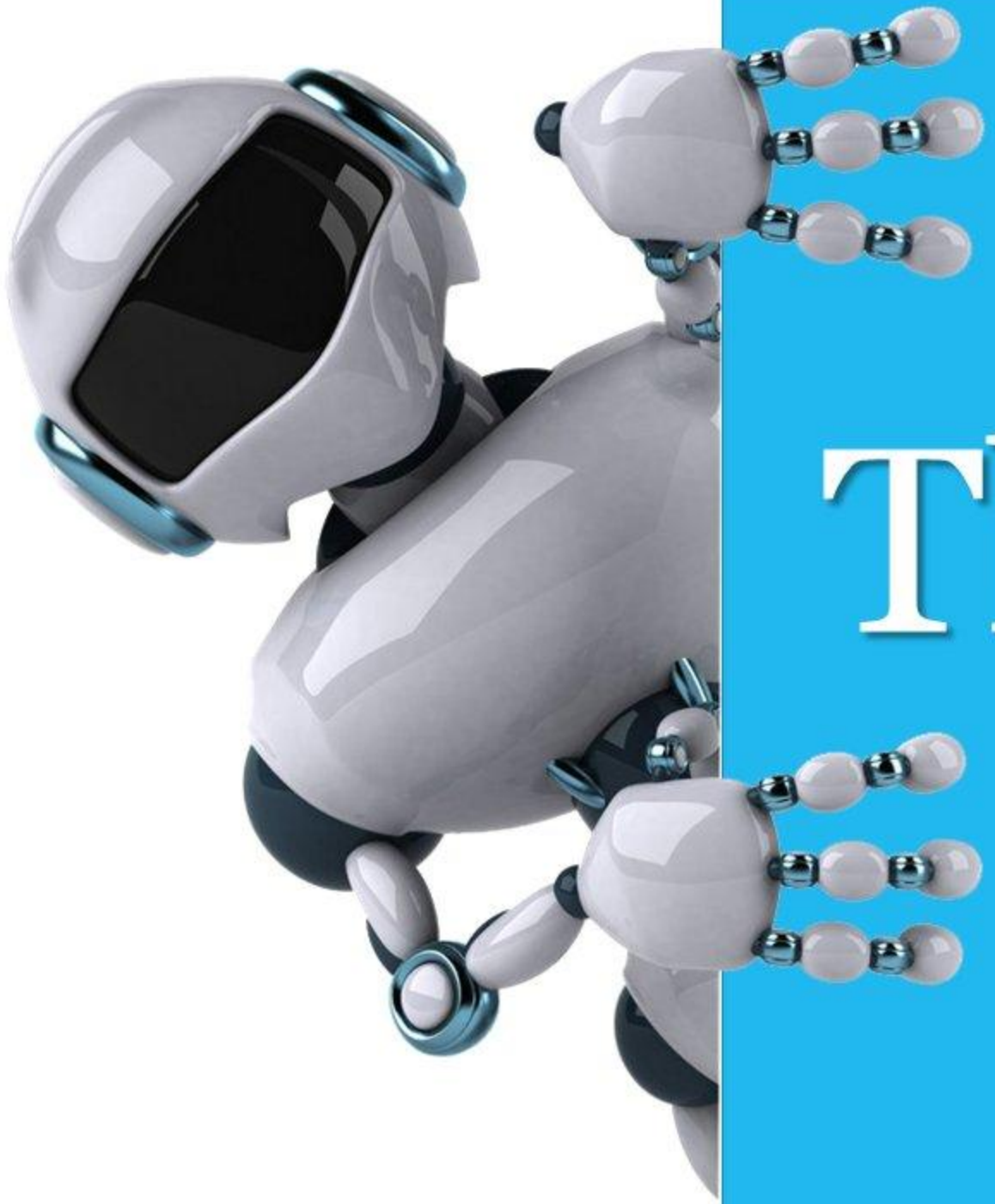
- Swish **allows for the propagation of a few numbers of negative weights**, whereas ReLU sets all non-positive weights to zero.
- This is a crucial property that determines the success of non-monotonic smooth activation functions, such as Swish's, in progressively deep neural networks.
- It's a **self-gated activation function created by Google researchers.**


- Mathematically, it can be represented as:

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

# Swish

- **Pros and Cons**
- Swish is a smooth activation function that means that it **does not suddenly change direction like ReLU does near x equal to zero.**
- Rather, **it smoothly bends from 0 towards values < 0 and then upwards again.**
- Non-positive values were zeroed out in ReLU activation function.
- Negative numbers, on the other hand, may be valuable for detecting patterns in the data.
- Because of the sparsity, large negative numbers are wiped out, resulting in a win-win situation.
- The swish activation function being non-monotonous enhances the term of input data and weight to be learnt.
- Slightly more computationally expensive and more problems with the algorithm will probably arise given time.



# Thank you