Neural Networks

- A neural network is a method in artificial intelligence that teaches computers to process data inspired by the human brain.
- A machine learning process called deep learning that uses interconnected nodes or neurons.
- These nodes are arranged in a layered structure that resembles the human brain.

Why are neural networks important?

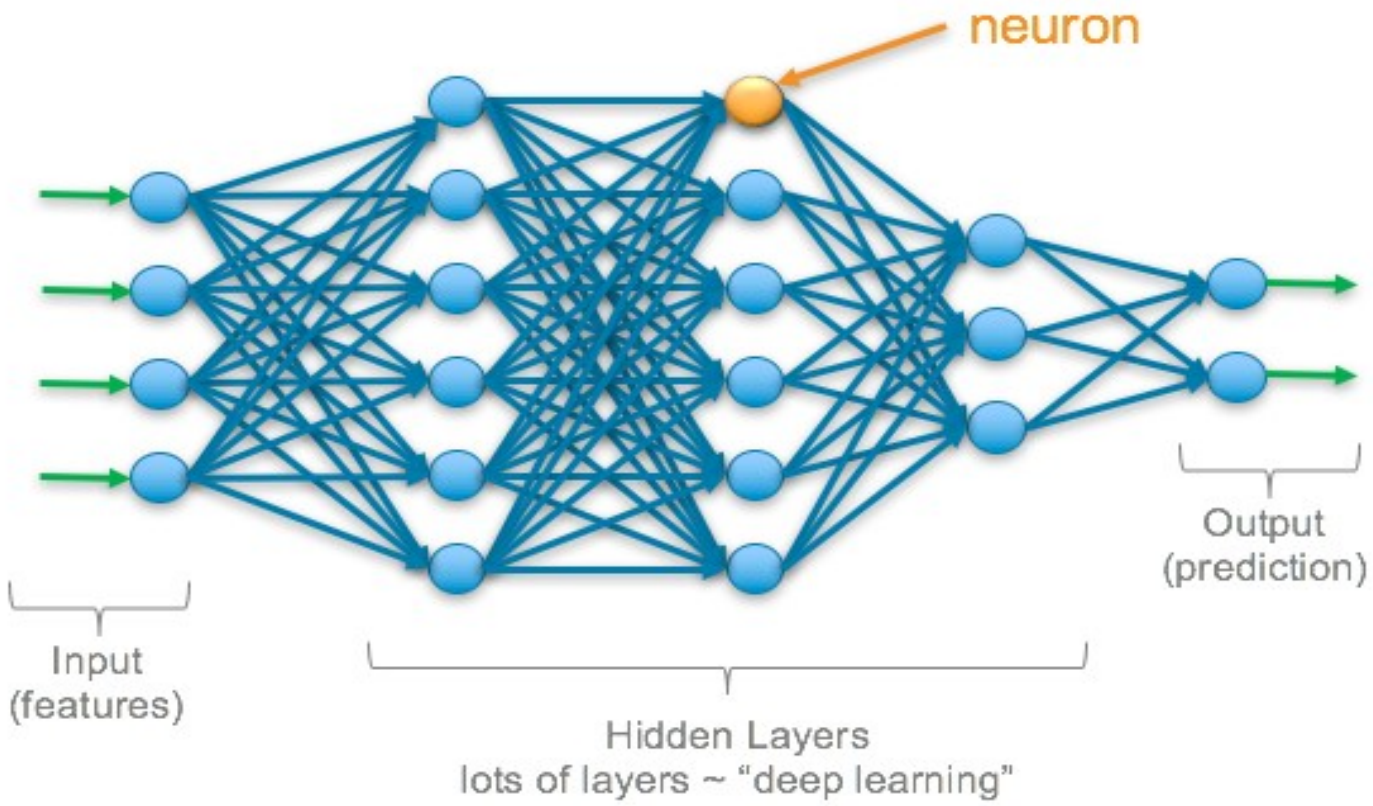
- Neural networks can help computers make intelligent decisions with limited human assistance.
- They can learn and model the relationships between input and output data that are nonlinear and complex.

How do neural networks work?

- Neurons (human brain cells) send electrical signals to each other to help humans process information.
- An artificial neural network is composed of artificial neurons that work together to solve problems.
- Artificial neurons are software modules called nodes.
- Artificial neural networks are software programs or algorithms.
- At their core, neural networks use computing systems to solve mathematical calculations.

Components of a Neural Network

1. **Neurons (Nodes):** The fundamental units of a neural network that process input data and pass the information to other neurons.

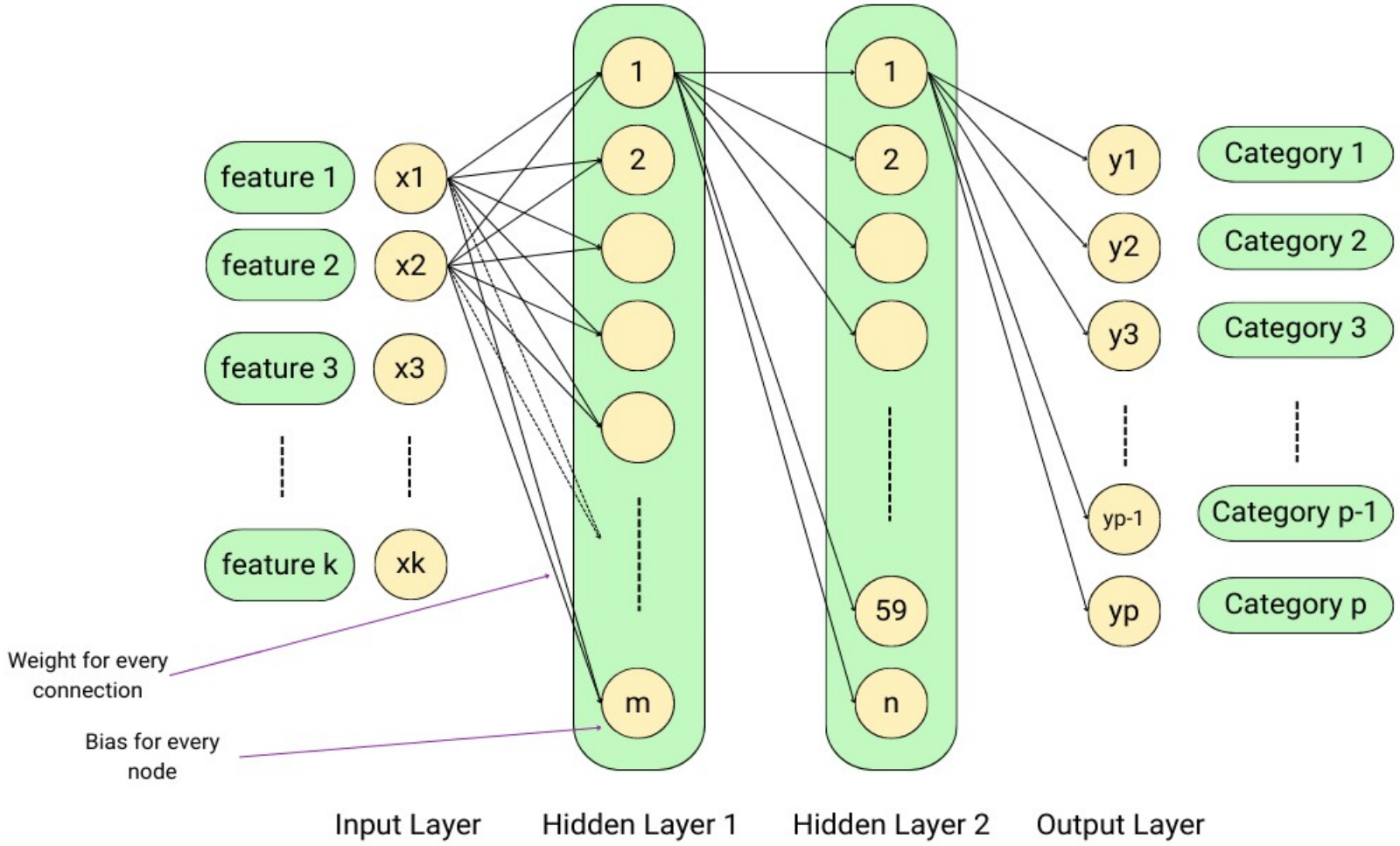


2. Layers:

- **Input Layer:** Information from the outside world enters the artificial neural network through the input layer. Input nodes process the data, analyze or categorize it, and pass it on to the next layer.

- **Hidden Layer:** Hidden layers take their input from the input layer or other hidden layers. Each hidden layer analyzes the output from the previous layer, processes it further, and passes it on to the next layer.

- **Output Layer:** The output layer gives the final result of all the data processing by the artificial neural network. It can have single or multiple nodes. For instance, if we have a binary (yes/no) classification problem, the output layer will have one output node, which will give the result as 1 or 0. However, if we have a multi-class classification problem, the output layer might consist of more than one output node.



3. Weights: Parameters within the network that adjust as learning proceeds, influencing the strength of the signal from one neuron to the next. Each weight represents the strength of the connection between the two nodes it connects.

4. **Biases:** The constant that is added to the product of features and weights. It is used to offset the result and helps the models to shift the activation function towards the positive or negative side.

5. **Activation Functions:** Mathematical functions applied to the output of each neuron, introducing non-linearity into the network.

   Examples include:

   a. Linear Function
   b. Heaviside Step Function
   c. ReLU (Rectified Linear Unit)
   d. Sigmoid Function
   e. Tanh Function
   f. SoftMax Function

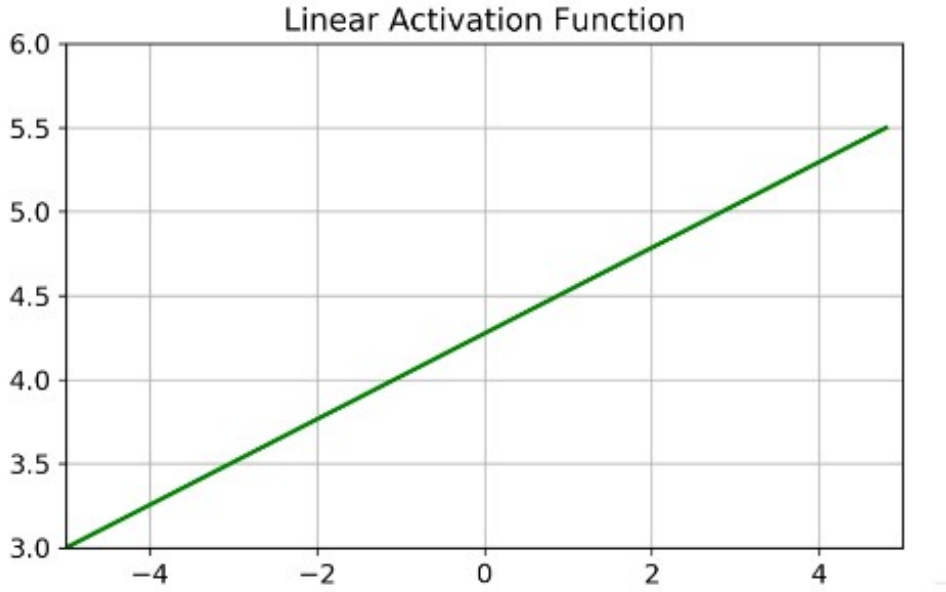
Linear Function/No activation/Identity function

- The linear activation function is where the activation is proportional to the input i.e. the weighted sum from neurons.
- The function doesn't do anything to the weighted sum of the input, it simply spits out the value it was given.
- When? Usually used in Regression Problems, if all layers are linear in nature the final will absolutely be Linear one

$$f(x) = a + x$$
$$f(x) = a + \sum x_i \, w_i$$

Where

$a \rightarrow bias$



Heaviside/Unit Step Function

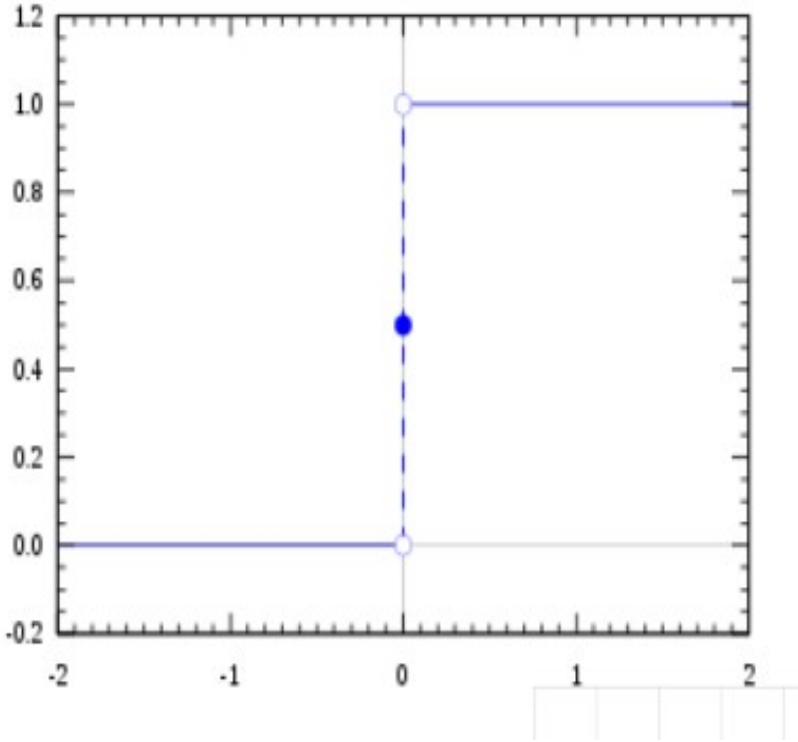
- The Heaviside step function H(x) is a discontinuous function, whose value is zero for negative arguments x < 0 and one for positive arguments x > 0.

- When? Usually used for Binary Classification, If threshold is passed, it is 1.

$$f(x) = \begin{cases} 1, & if\ x \geq a \\ 0, & Else \end{cases}$$

Where

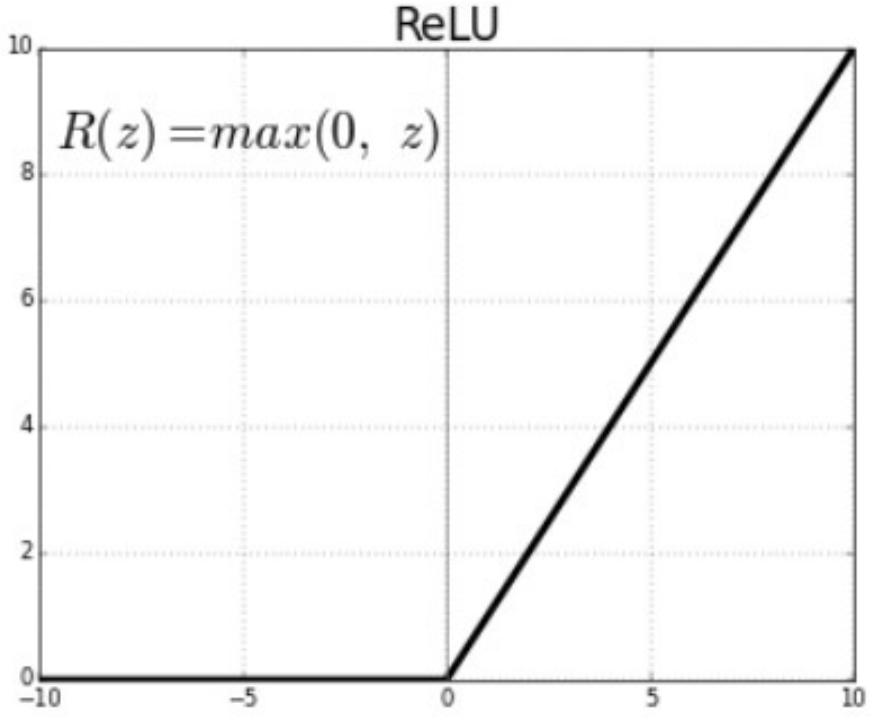
$$a \rightarrow bias \mid threshold$$



ReLU (Rectified Linear Unit)

- The ReLU is the most commonly used activation function in deep learning models.
- The function returns 0 if it receives any negative input, but for any positive value x it returns that value back.
- It can be written as f(x)=max(0, x).

$$f(x) = \begin{cases} x_i, & if\ x \geq 0 \\ 0, Else\ |if\ x < 0 \end{cases}$$



Range is 0 to + Inf

Nature :- Non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.

Uses ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

Sigmoid Activation Function

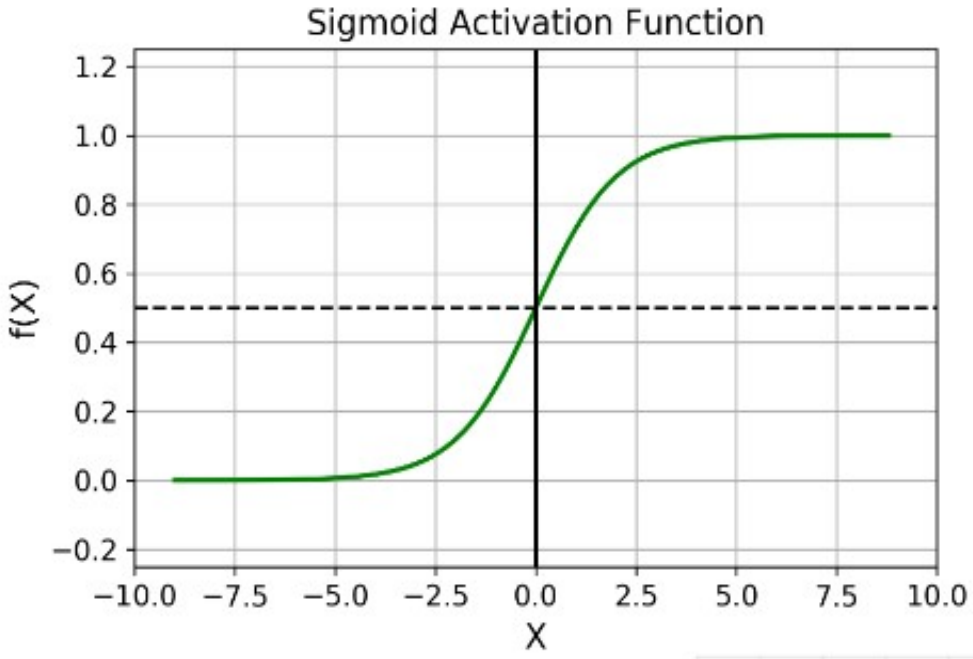
- This function maps input values to an output range between 0 and 1, providing a smooth gradient useful for neural network training.

- It is particularly advantageous for binary classification tasks due to its probabilistic interpretation.

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

Where

$e \rightarrow$ Euler's Constant



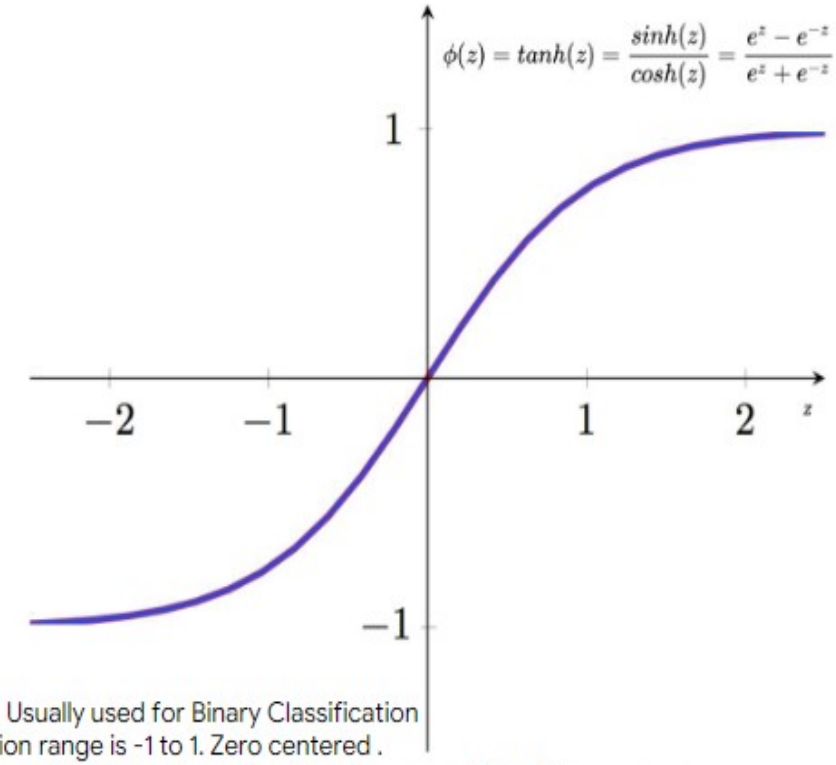
Tanh/Hyperbolic Function

- This function is a mathematical function commonly used in artificial neural networks for their hidden layers.
- It transforms input values to produce output values between -1 and 1.

$$f(x) = \tanh(x)$$

$$f(x) = \frac{\sinh(x)}{\cosh(x)}$$

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



When? Usually used for Binary Classification
Prediction range is -1 to 1. Zero centered .
Usually used in hidden layers, for bringing mean of the data Close to 0.

SoftMax Function

- This function is a mathematical function that converts a vector of real numbers into a probability distribution.
- It exponentiates each element, making them positive, and then normalizes them by dividing by the sum of all exponentiated values.

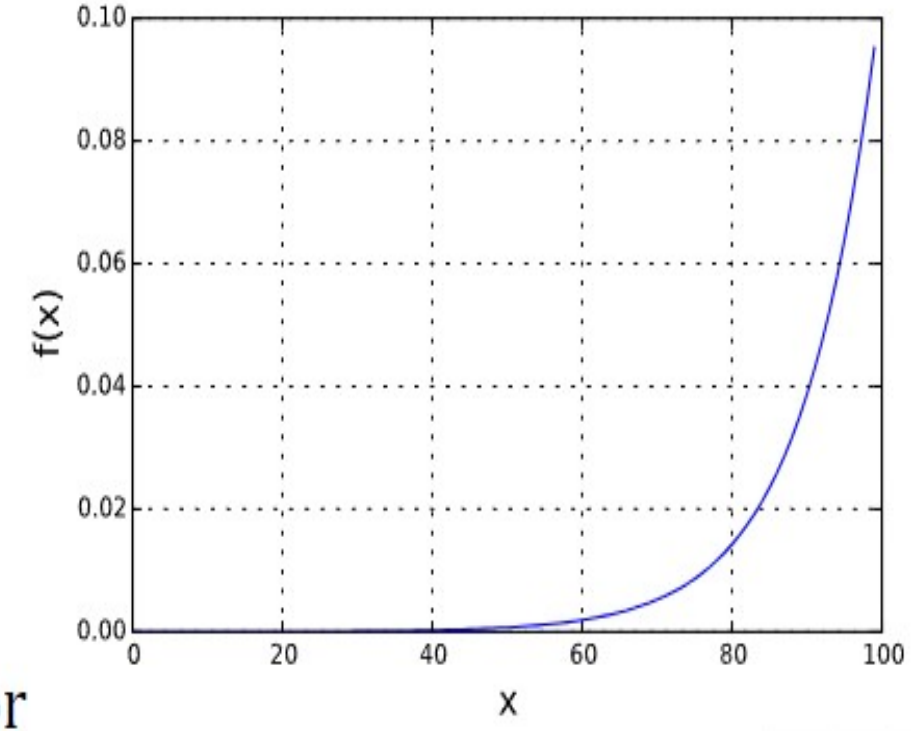
$$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Where,

$K \rightarrow$ *Number of Classes*

$e^{x_i} \rightarrow$ Exp fn for input vector

$e^{x_j} \rightarrow$ Exp fn for output vector



Choosing the right Activation Function

The basic rule of thumb is if you really don't know what activation function to use, then simply use *RELU* as it is a general activation function and is used in most cases these days.
If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.

Foot Note :-

The **activation function** does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

6. **Loss Function (Cost Function):** A function that measures the difference between the predicted output and the actual output, guiding the optimization process.

7. **Optimizer:** Algorithms that adjust the weights and biases to minimize the loss function. Common optimizers include Gradient Descent, Adam, and RMSprop.

8. **Learning Rate:** A hyperparameter that controls how much the model's parameters are adjusted with respect to the loss gradient during training. Or a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving towards a minimum of a loss function.

9. **Epochs:** One complete pass through the entire training dataset during the learning process.

10. **Batch Size:** The number of training examples utilized in one iteration before updating the model's parameters.