Everything About Activation Function & Bias

Course Title: Artificial Intelligence

Course Code: cse-403

Khandaker Jannatul Ritu, Lecturer, Dept. of CSE [BAIUST]

Outlines:

- What is an activation function and why use them?
- Why do we need Non-linear activation function?
- Types of functions
    - ✓ **Sigmoid function**
    - ✓ **Tanh Function**
    - ✓ **SoftMax Function**
    - ✓ **ReLU**
    - ✓ Leaky ReLU
    - ✓ Linear / Identity Activation Function
    - ✓ Ramp Activation Function
    - ✓ Swish Function
- What is SoftMax Activation Function?
- Need for the Softmax activation function
- Why is Softmax Used in the Last Layer?
- **Why Not Sigmoid activation function is not preferred in multi-class classification problems?**
- Using Softmax Activation Function in Output
- Why is Softmax Function Useful in CNN?
- Why is Softmax used in CNN?
- **When to Use Softmax Activation Function vs ReLU?**
- Choosing the right Activation Function
- Bias

❑ Activation functions in Neural Networks

✓ What is an activation function and why use them?

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

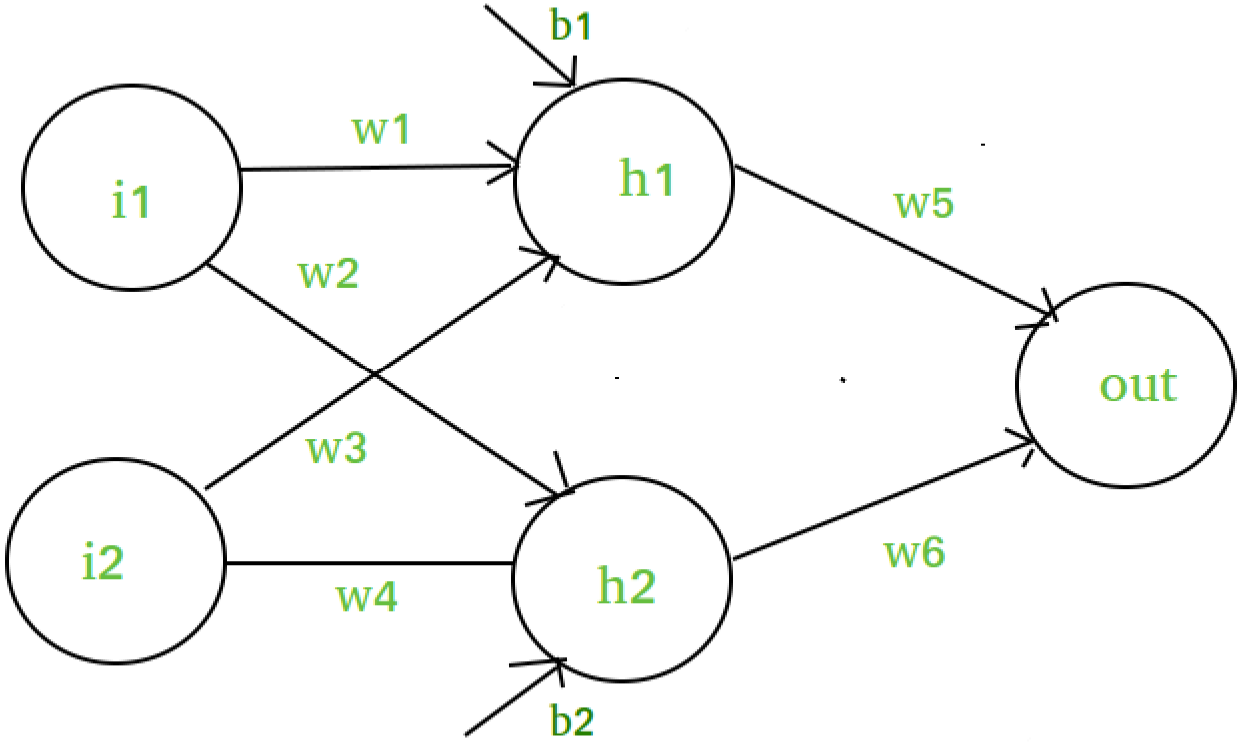
Explanation: We know, the neural network has neurons that work in correspondence with *weight*, *bias*, and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as ***back-propagation***. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

✓ *Why do we need Non-linear activation function?*

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

Mathematical proof

Suppose we have a Neural net like this :-



Elements of the diagram are as follows:

Hidden layer i.e. layer 1:

$z(1) = W(1)X + b(1)\ a(1)$

Here,

$z(1)$ is the vectorized output of layer 1
$W(1)$ be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4
X be the vectorized input features i.e. i1 and i2
b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2
$a(1)$ is the vectorized form of any linear function.

Layer 2 i.e. output layer :-
Note : Input for layer 2 is output from layer 1
$z(2) = W(2)a(1) + b(2)$
$a(2) = z(2)$

Calculation at Output layer
$z(2) = (W(2) * [W(1)X + b(1)]) + b(2)$
$z(2) = [W(2) * W(1)] * X + [W(2)*b(1) + b(2)]$

Let,
$[W(2) * W(1)] = W$
$[W(2)*b(1) + b(2)] = b$
Final output : $z(2) = W*X + b$
which is again a linear function

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn’t matter how many hidden layer we attach in neural net, all layers will behave same way because ***the composition of two linear function is a linear function itself***. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. **Hence we need an activation function.**

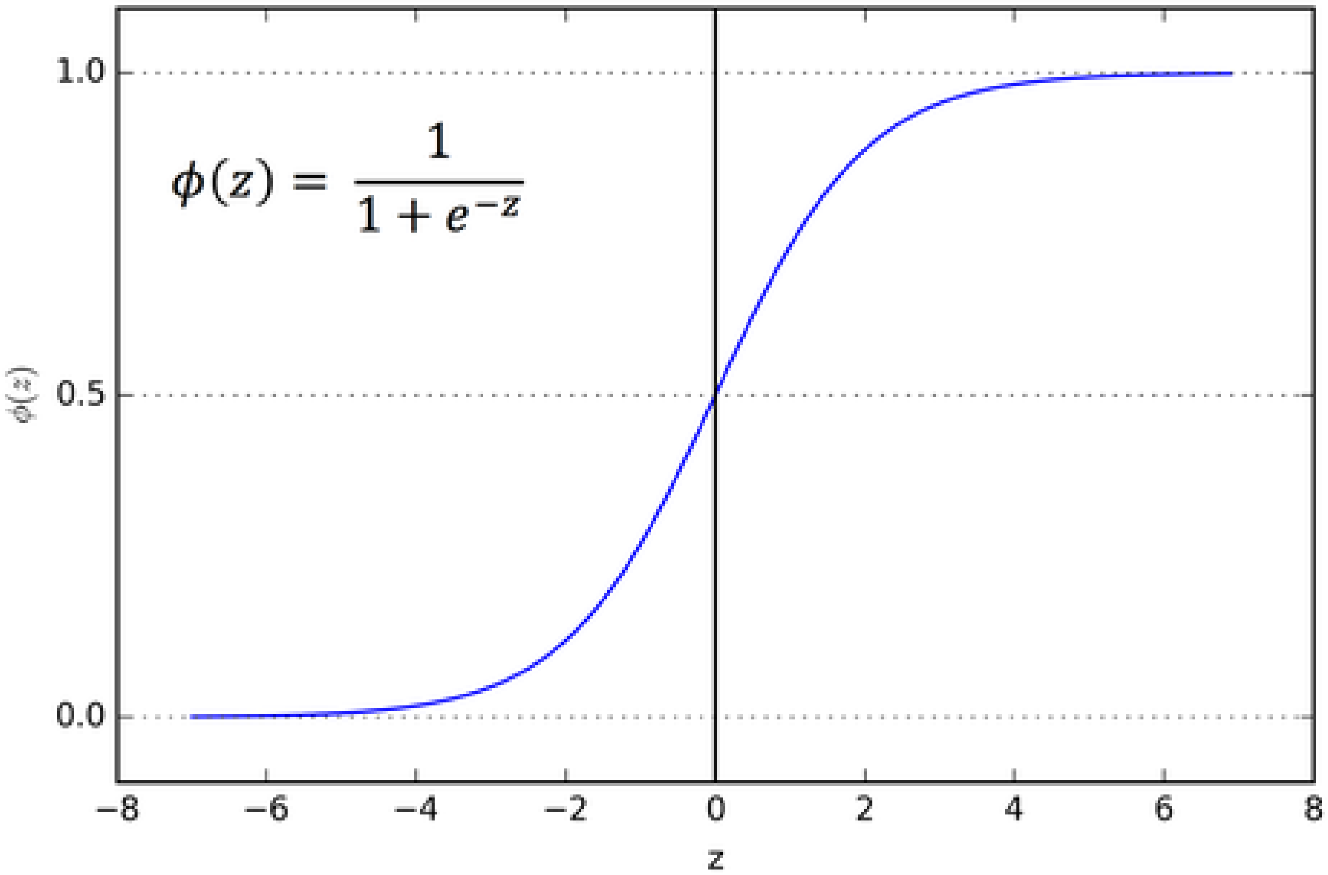
Types of functions

❑ Sigmoid function:

Sigmoid function refers to a function that is projected as S - shaped graph. It is two types:- s

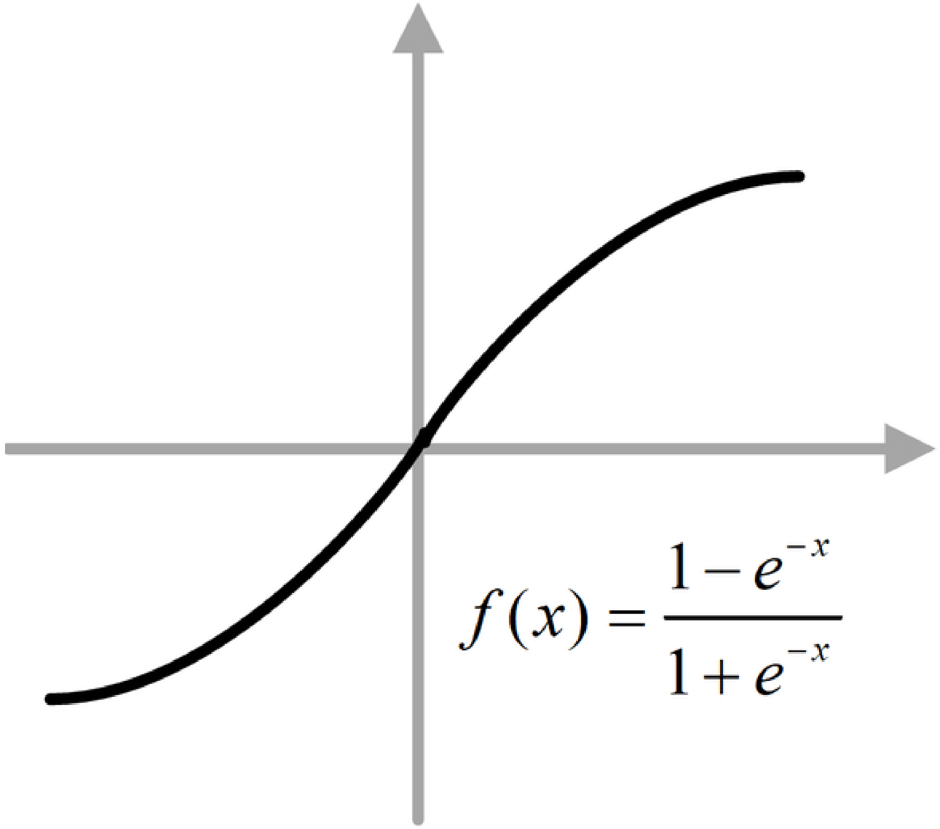
1) Binary sigmoid activation function:

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



2) Bipolar sigmoid activation function:

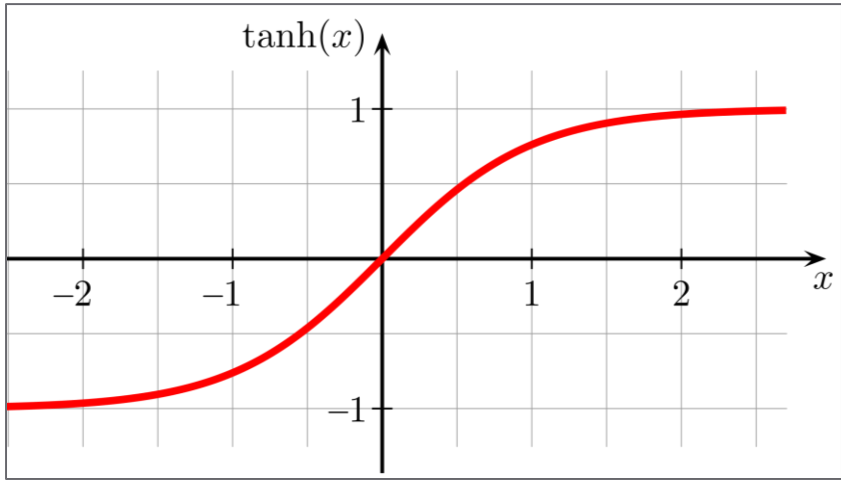
$$\mathbf{f(y)} = \frac{\mathbf{2}}{\mathbf{1+e^{-y}}} - \mathbf{1}$$



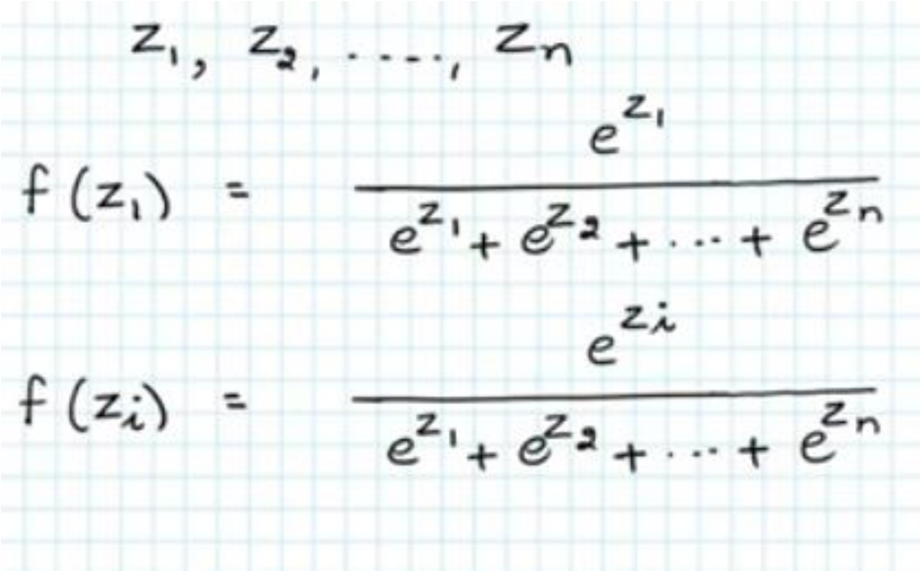
❑ Tanh Function:

The activation function, which is more efficient than the sigmoid function is Tanh function. Tanh function is also known as **Tangent Hyperbolic Function**. It is a mathematical updated version of the sigmoid function.

$$f(y) = tanH(y) = \frac{2}{1+e^{-2y}} - 1$$



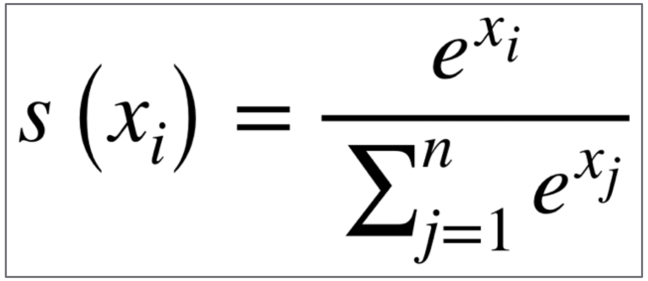
❑ SoftMax Function:



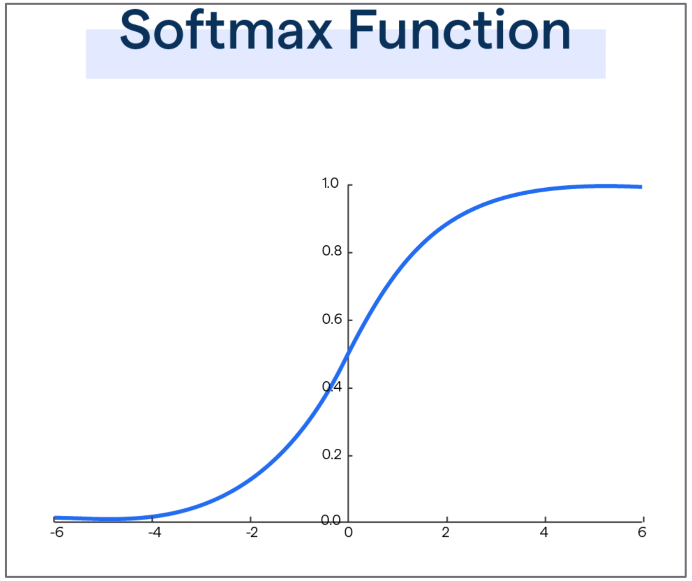
$$z_1, z_2, \ldots, z_n$$

$$f(z_1) = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_n}}$$

$$f(z_i) = \frac{e^{z_i}}{e^{z_1} + e^{z_2} + \cdots + e^{z_n}}$$



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



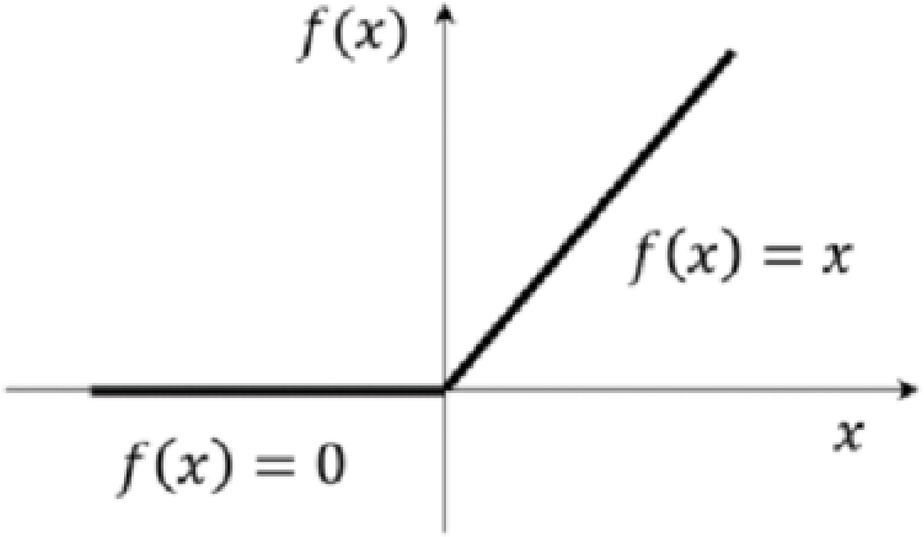
Used frequently when managing several classes. In the output nodes of image classification issues, the softmax was typically present. The softmax function would split by the sum of the outputs and squeeze all outputs for each category between 0 and 1.

❑ ReLU(Rectified Linear Unit) Function:

Chiefly implemented in *hidden layers* of Neural network.

•**Equation :-** ***A(x) = max(0, x)***.

$$f(x) = \begin{cases} 0 & \text{if } z < 0 \\ x & \text{if } z \geq 0 \end{cases}$$



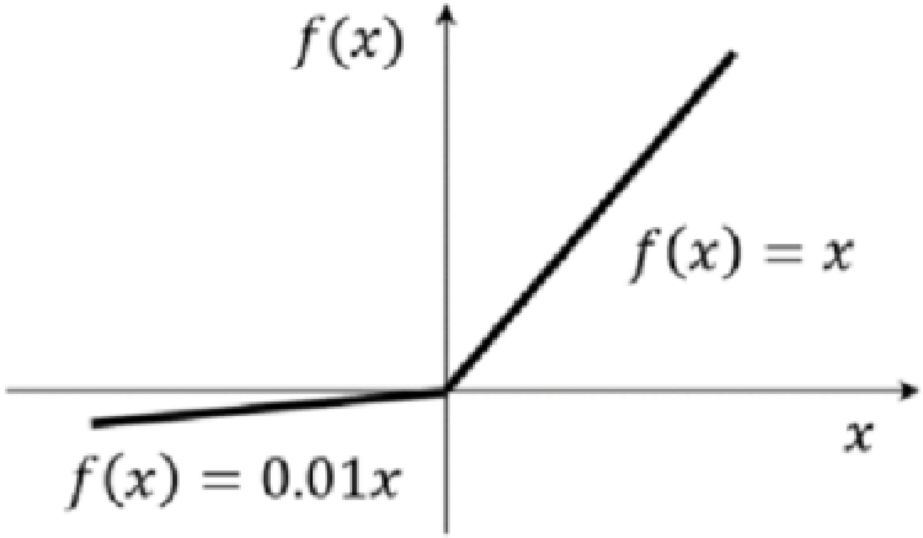
ReLU activation function

❑ Leaky ReLU(Leaky Rectified Linear Unit) Function:

Chiefly implemented in *hidden layers* of Neural network.

•**Equation :-** ***A(x) = max(ax, x)***.

$$f(x) = \begin{cases} ax & \text{if } z < 0 \\ x & \text{if } z \geq 0 \end{cases}$$



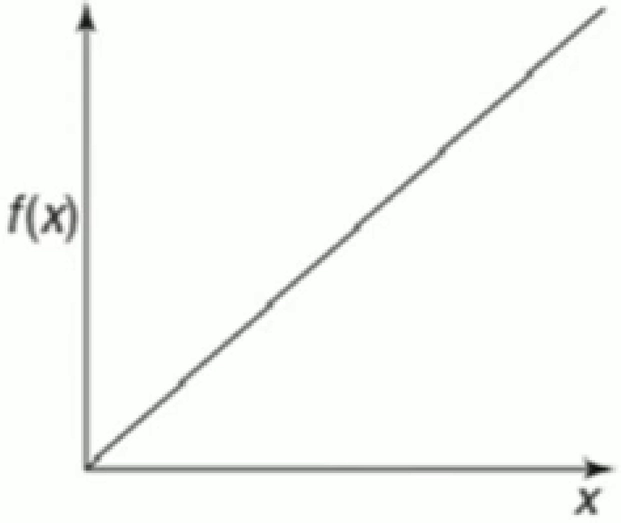
LeakyReLU activation function

□ Linear / Identity Activation Function:

The equation of the linear activation function is the same as the equation of a straight line **Y= MX+ C**

Linear activation function can be used at only one place that is the output layer.

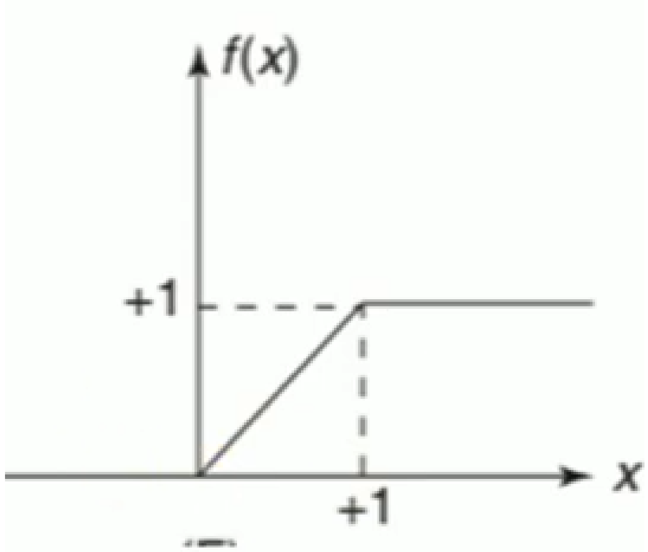
It is a linear function and can be defined as : f(x) = x for all x



□ Ramp Activation Function:

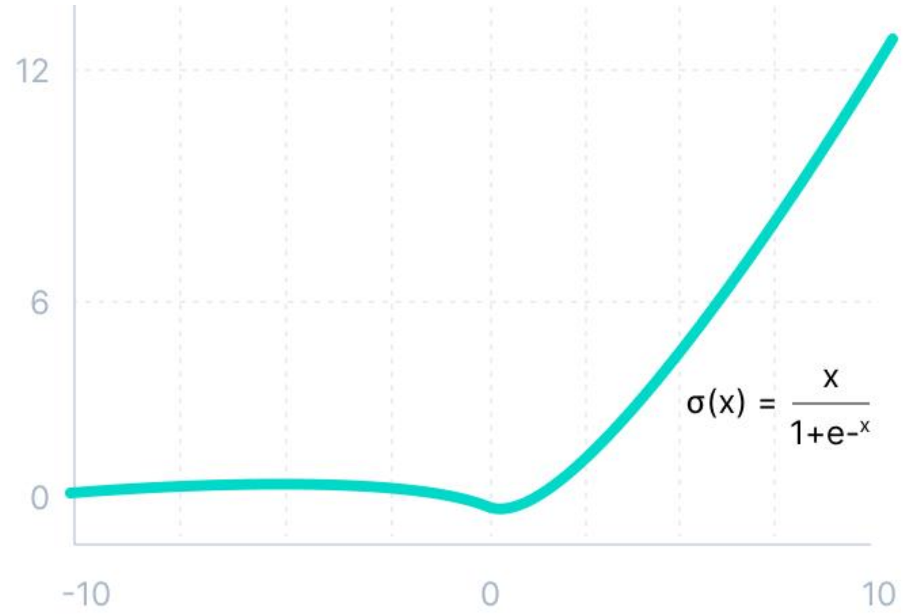
The ramp function is defined as

$$f(x) = \left\{ \begin{array}{ll} 1 & \text{if } x > 1 \\ x & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } x < 0 \end{array} \right\}$$



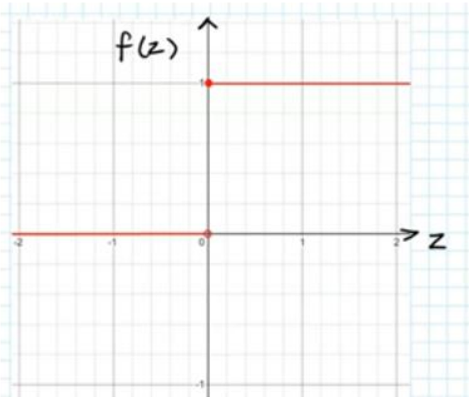
□ Swish Function:

$$f(x) = x*sigmoid(x) \qquad \sigma(x) = \frac{x}{1+e^{-x}}$$



❑ **Step Activation Function:** The step function is defined as

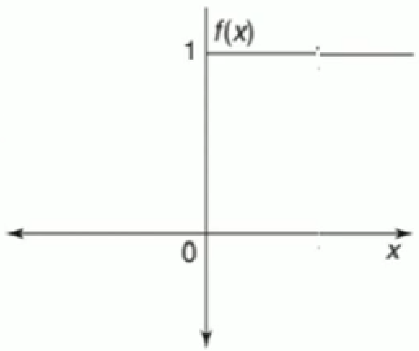
$$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$$



❑ **Binary Step Activation Function:** The step function is defined as

Where **ϴ** = threshold value

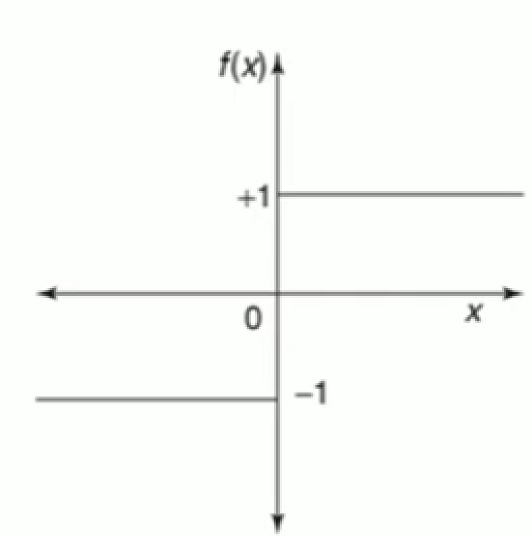
$$f(x) = \begin{cases} 1 \text{ if } x \geq \Theta \\ 0 \text{ if } x < \Theta \end{cases}$$



❑ **Bipolar Step Activation Function:** The step function is defined as

Where **ϴ** = threshold value

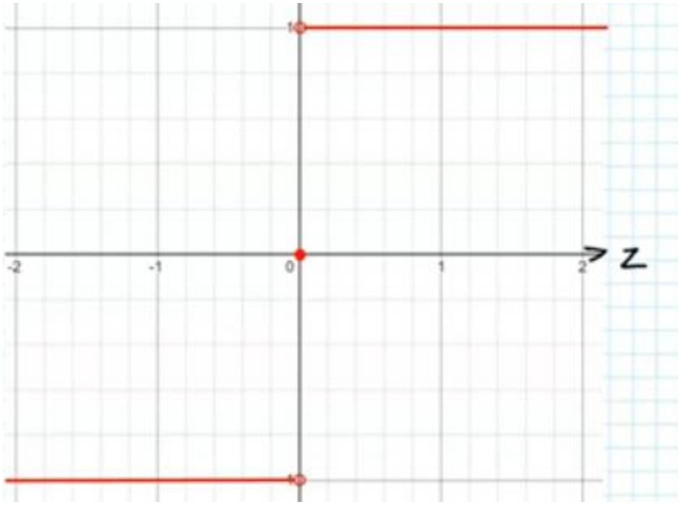
$$f(x) = \begin{cases} 1 \text{ if } x \geq \Theta \\ -1 \text{ if } x < \Theta \end{cases}$$



❑ **Signum(sgn or sign) Activation Function:** The step function is defined as

Where **ϴ** = threshold value

$$f(x) = \begin{cases} 1 \text{ if } x > \Theta \\ -1 \text{ if } x < \Theta \end{cases}$$



✓ What is SoftMax Activation Function?

The SoftMax activation function is commonly used in machine learning, particularly in neural networks for classification tasks. An activation function converts a vector of raw prediction scores (logits) into probabilities.

Key Characteristics of the SoftMax Function:

1.Normalization: The SoftMax activation function normalizes the input values into a probability distribution, ensuring that the sum of all output values is 1. This makes it suitable for classification problems where the output needs to represent probabilities over multiple classes.

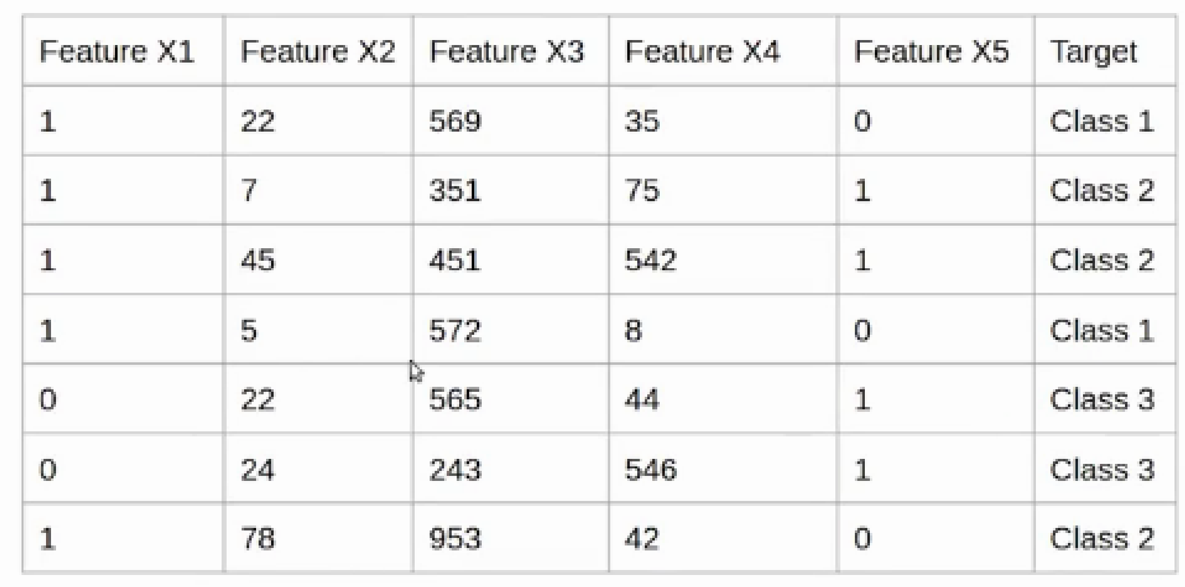
2.Exponentiation: By exponentiating the inputs, the SoftMax function in machine learning amplifies the differences between the input values, making the largest value more pronounced in the output probabilities.

3.Differentiability: The SoftMax function is differentiable and essential for backpropagation in neural networks.

Example

Suppose we have the following dataset: For every observation, we have five features from FeatureX1 to FeatureX5, and the target variable has three classes.

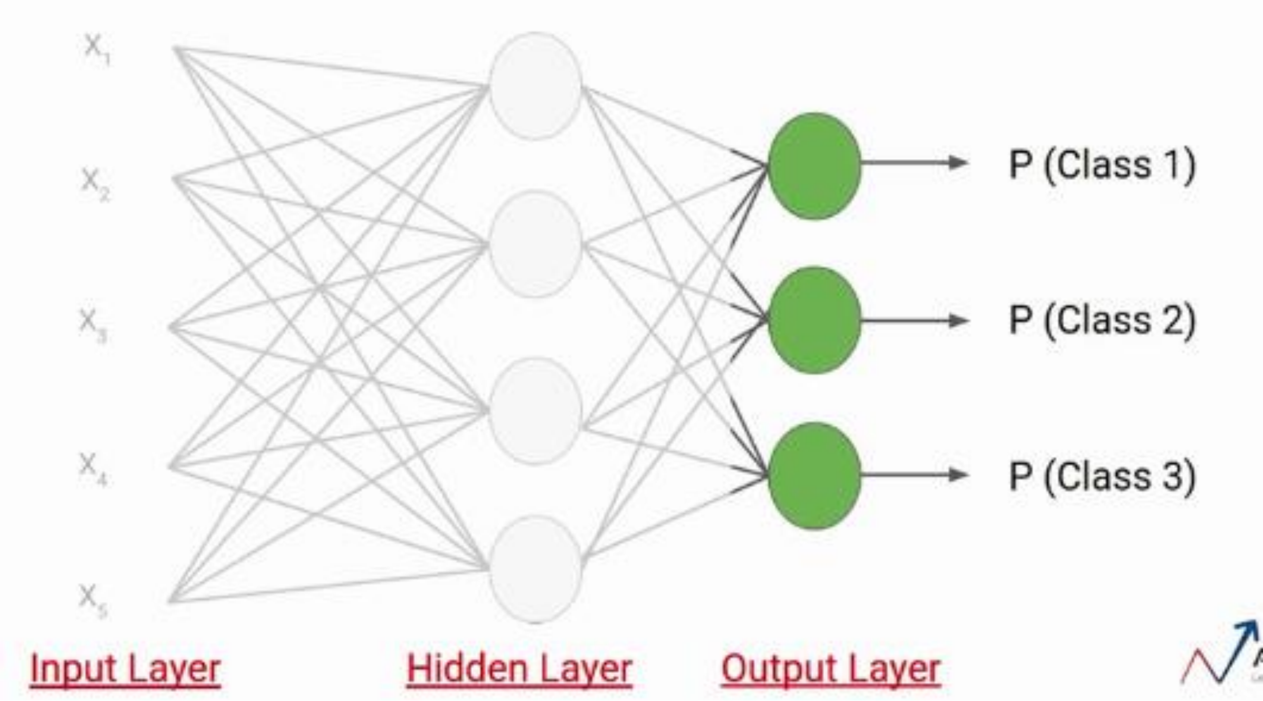
Now, let's create a simple neural network to solve this problem. Here, we have an Input layer with five neurons, as we have five features in the dataset. Next, we have one hidden layer with four neurons. Each of these neurons uses inputs, weights, and biases to calculate a value, which is represented as Zij here.



Feature X1	Feature X2	Feature X3	Feature X4	Feature X5	Target
1	22	569	35	0	Class 1
1	7	351	75	1	Class 2
1	45	451	542	1	Class 2
1	5	572	8	0	Class 1
0	22	565	44	1	Class 3
0	24	243	546	1	Class 3
1	78	953	42	0	Class 2

For example, the first neuron of the first layer is represented as Z11 Similarly the second neuron of the first layer is represented as Z12, and so on.

We apply the activation function, let's say a tanh activation function, to these values and send the values or result to the output layer.
The number of neurons in the output layer depends on the number of classes in the dataset. Since we have three classes in the dataset, we will have three neurons in the output layer. Each of these neurons will give the probability of individual classes.



This means the first neuron will give you the probability that the data point belongs to class 1. Similarly, the second neuron will give you the probability that the data point belongs to class 2.

✓ Need for the Softmax activation function

Consider a case where for a ternary classification problem we have preactivation outputs as [2.4, -0.6, 1.2] in the output layer. Preactivation output can be any real value.
Now if we directly calculate the probability for each class by taking one vale and dividing it by sum of all the values. Then we get probabilities [0.57, -0.14, 0.12]. Since probability can never be negative so here the predicted probability for class 2 is not accepted. Therefore we use the *softmax* activation function in the output layer for multiclass classification problem.
If the input value is negative then also *softmax* returns positive value because e^x always gives positive values.

✓ Why is Softmax Used in the Last Layer?

Here’s how the softmax function in machine learning works in the last layer of a neural network :

Input: The softmax activation function takes a vector of real numbers (z) as input. These values typically represent the outputs from the final hidden layer of the neural network, often accessed via an API.

Exponentiation: Each element in the input vector z is exponentiated using the mathematical constant e (approximately 2.718). This step ensures all the values become positive. The derivative of this step is crucial for backpropagation.

Normalization: After exponentiation, all the elements are summed up. This is a key step for ensuring that the probabilities add up to 1.

Probability Calculation: Each exponentiated value from step 2 is then divided by the sum obtained in step 3. This process normalizes the values, forcing them between 0 and 1. The cross-entropy loss function often uses these probabilities to measure a classifier’s performance.

Output: The result is a new vector the same size as the input vector z. However, each element in the output vector now represents a probability between 0 and 1. The argmax function is typically used to select the index of the highest probability, determining the predicted class generalization.

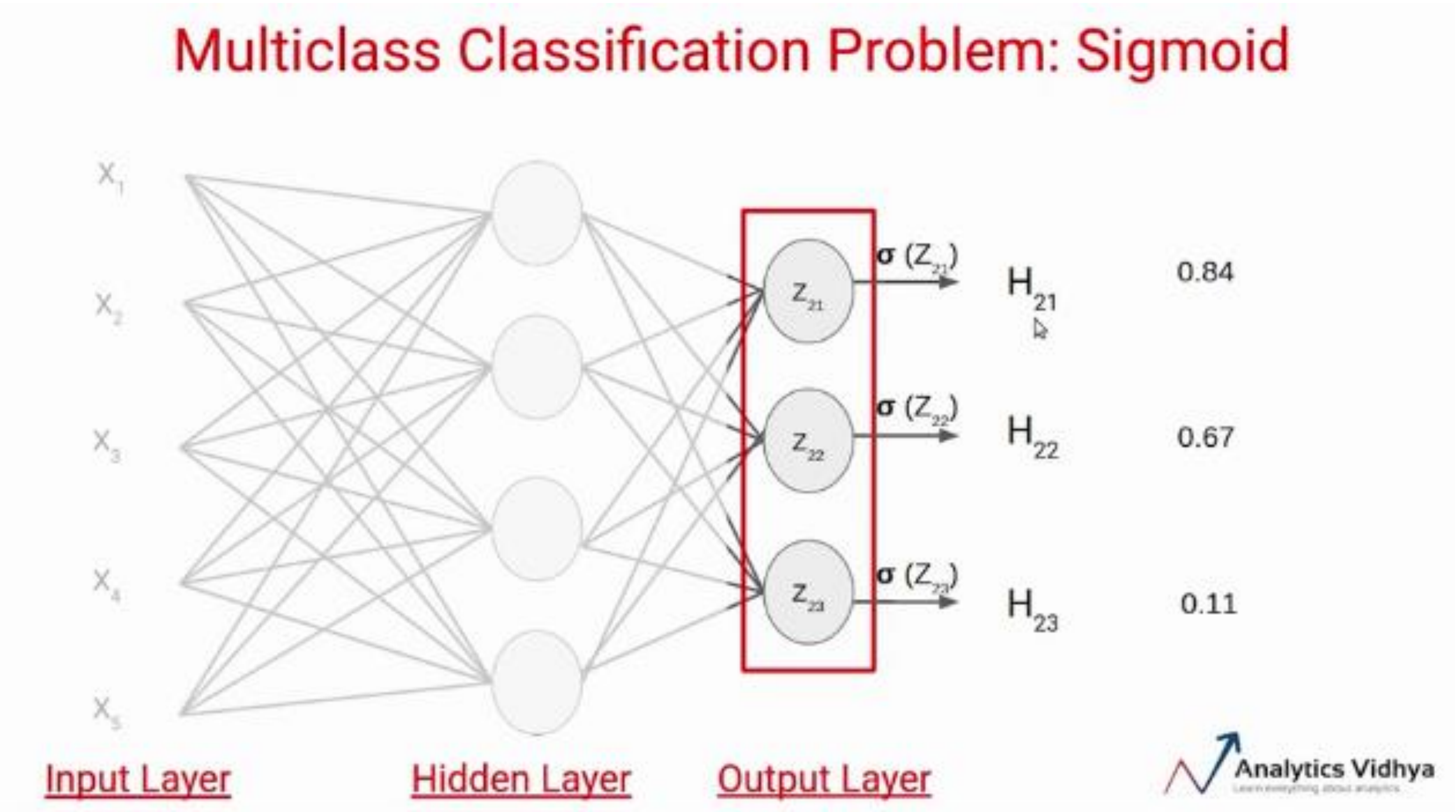
✓ Why Not Sigmoid activation function is not preferred in multi-class classification problems?

Suppose we calculate the Z value using weights and biases of this layer and apply the sigmoid activation function over these values. We know that the sigmoid activation function gives the value between 0 and 1 suppose these are the values we get as output.

There are two problems in this case-

1. First, if we apply a threshold of 0.5, this network says the input data point belongs to two classes.
2. Secondly, these probability values are independent of each other. That means the probability that the data point belongs to class 1 does not consider the probability of the other two classes.

The sigmoid activation function is not preferred in multi-class classification problems.



✓ Using Softmax Activation Function in Output

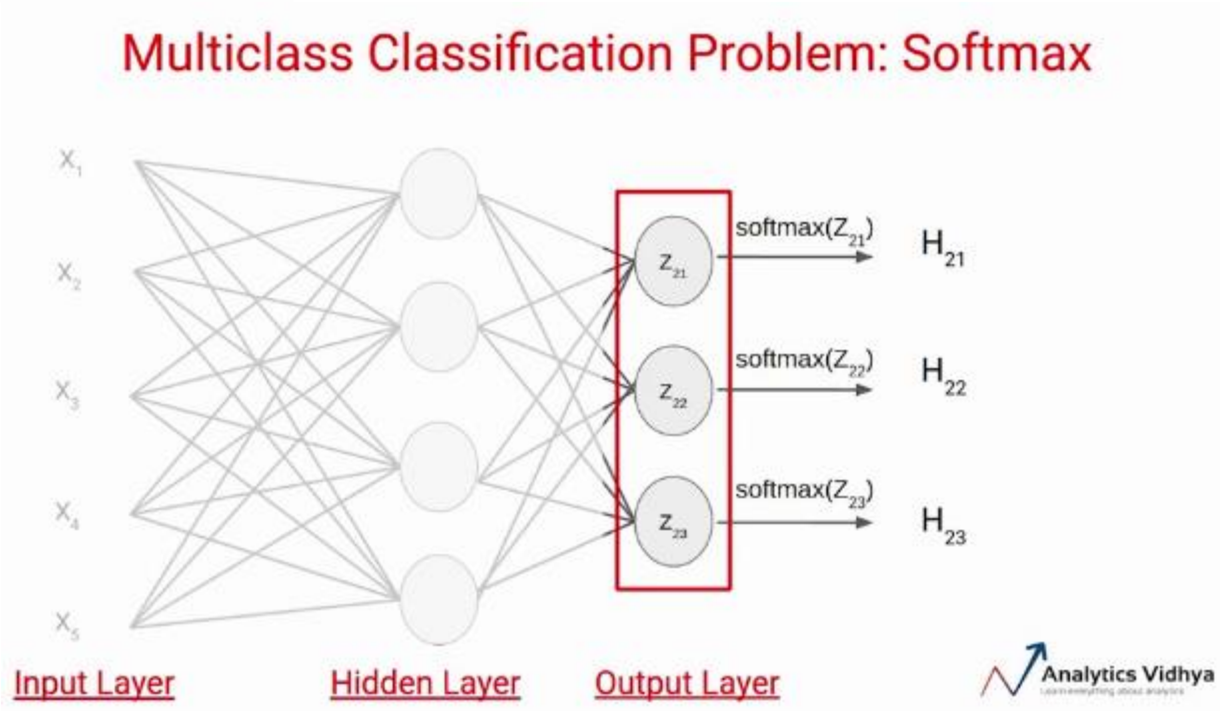
In the above example, we will use the Softmax activation function in the output layer instead of sigmoid. The Softmax activation function calculates relative probabilities. That means it uses the values of Z21, Z22, and Z23 to determine the final probability value.

Let’s see how the softmax activation function works. Like the sigmoid activation function, the SoftMax function in machine learning returns the probability of each class. Here is the equation for the SoftMax activation function.

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

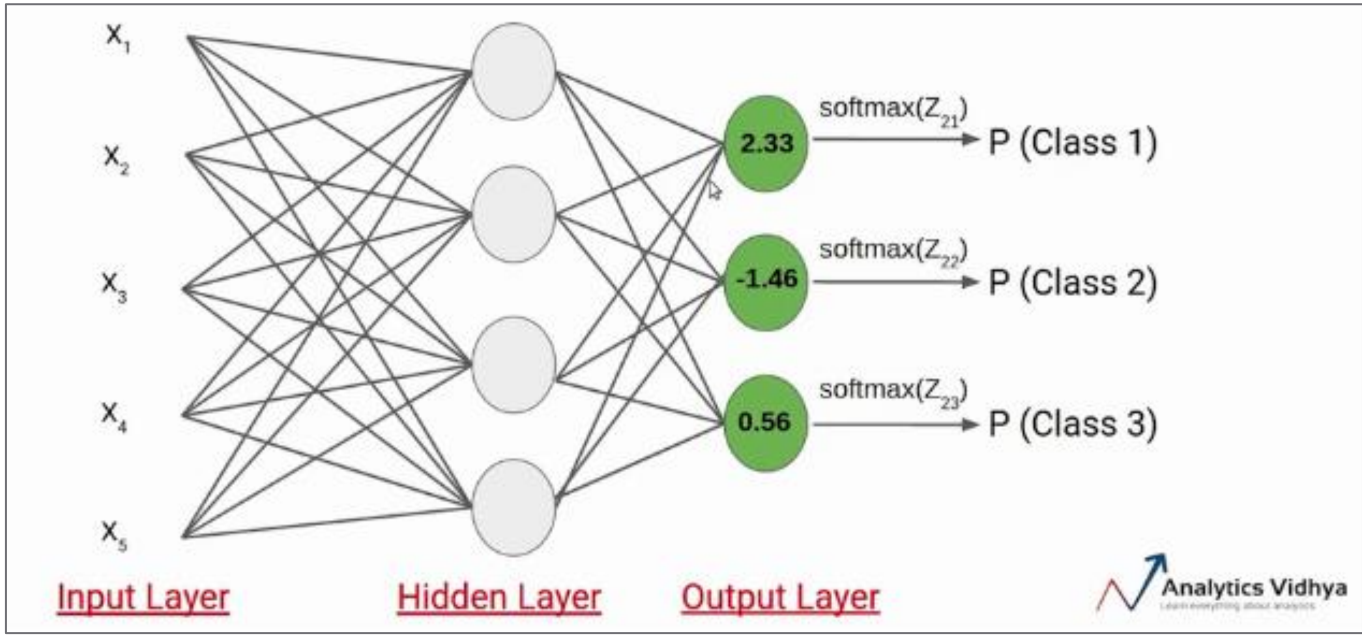
Here, Z represents the values from the neurons of the output layer. The exponential acts as the non-linear function. Later, these values are divided by the sum of exponential values to normalize them and then convert them into probabilities.

Note that, when the number of classes is two, it becomes the same as the sigmoid activation function. In other words, sigmoid is simply a variant of the Softmax function.

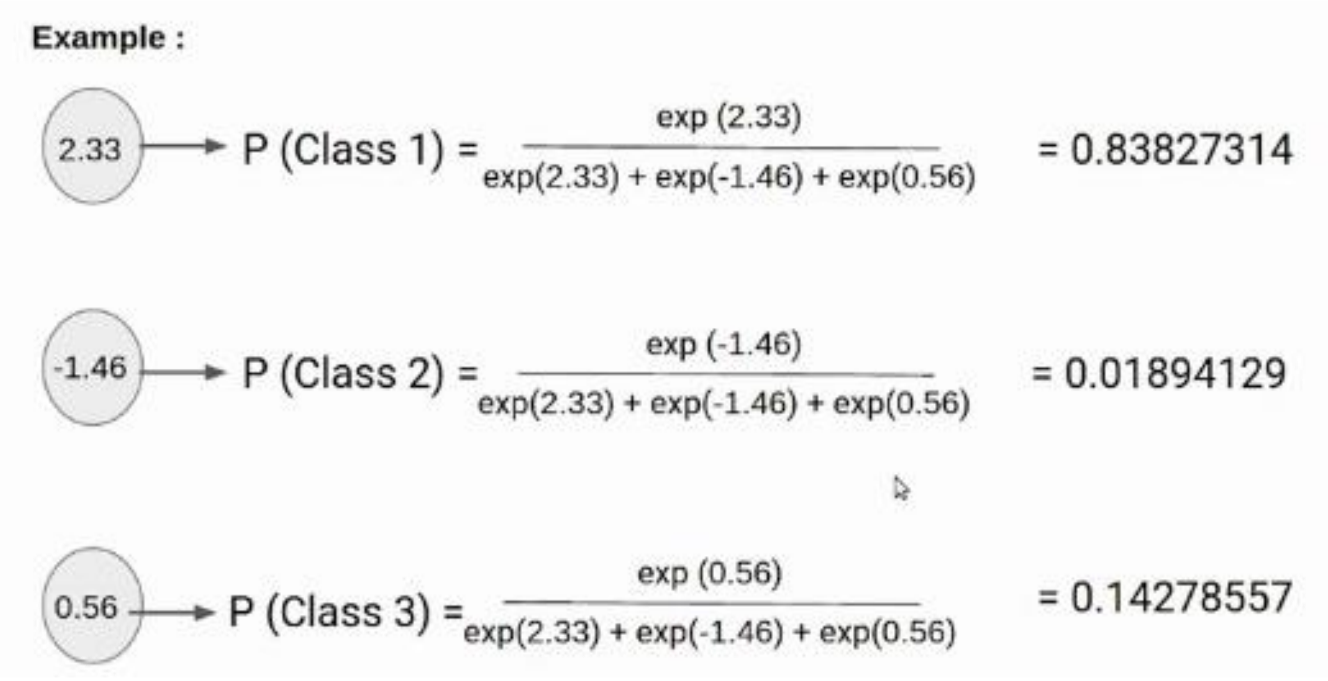


Let’s understand with a simple example how the softmax function works. We have the following neural network.

Suppose the values of Z21, Z22, and Z23 are 2.33, -1.46, and 0.56, respectively. Now, the SoftMax activation function is applied to each of these neurons, and the following values are generated.



These are the probability values that a data point belonging to the respective classes. Note that, in this case, the sum of the probabilities is equal to 1.



Example :

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

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

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

In this case, the input belongs to class 1. So, if the probability of any of these classes is changed, the probability value of the first class would also change.

✓ Why is Softmax Function Useful in CNN?

•The Softmax function allows CNNs to output a probability distribution over the possible classes. This is important because it will enable CNN to make more accurate predictions.

•The softmax activation function in machine learning first normalizes the input vector so that all numbers in the vector are equal to 1. Then, it exponentiates each number in the vector and divides by the sum of all the exponentiated numbers. This results in a vector of probabilities, where each probability is between 0 and 1 and represents the probability that the input belongs to a particular class.

•The probability distribution output by the softmax function can then be used to make a more accurate prediction about the class of an input image. For example, if the CNN predicts whether an image contains a cat or a dog, the probability distribution can indicate how likely the image contains a cat and how likely it is that the image contains a dog.

✓ Why is Softmax used in CNN?

•**CNN Processes Image:** The CNN takes an image as input and performs various convolutional and pooling operations to extract features.

•**Final Layer Generates Logits:** After processing, the final layer of the CNN outputs a set of numbers called logits. These logits represent the raw scores or activation levels for each class the CNN can classify. There will be one logit for each class.

•**Softmax Takes Over:** The Softmax function takes these logits as input.

•**Exponentiation:** Softmax activation function applies an exponent function (often $exex$) to each logit value. This emphasizes the differences between the logits, making the higher-scoring classes stand out more.

•**Normalization:** Softmax function in machine learning then divides each exponentiated value by the sum of all the exponentiated values, ensuring the final outputs add up to 1.

•**Probability Distribution:** The result is a vector of numbers between 0 and 1, representing probabilities. Each value corresponds to the probability of the image belonging to a specific class.

•**Decision and Interpretation:** The class with the highest probability value is CNN’s predicted class. This probability value also reflects CNN’s confidence level in its prediction.

✓ When to Use Softmax Activation Function vs ReLU?

Softmax Function is typically used in the last layer of a neural network to predict the class of an input image. It is also used in other applications, such as natural language processing and machine translation.

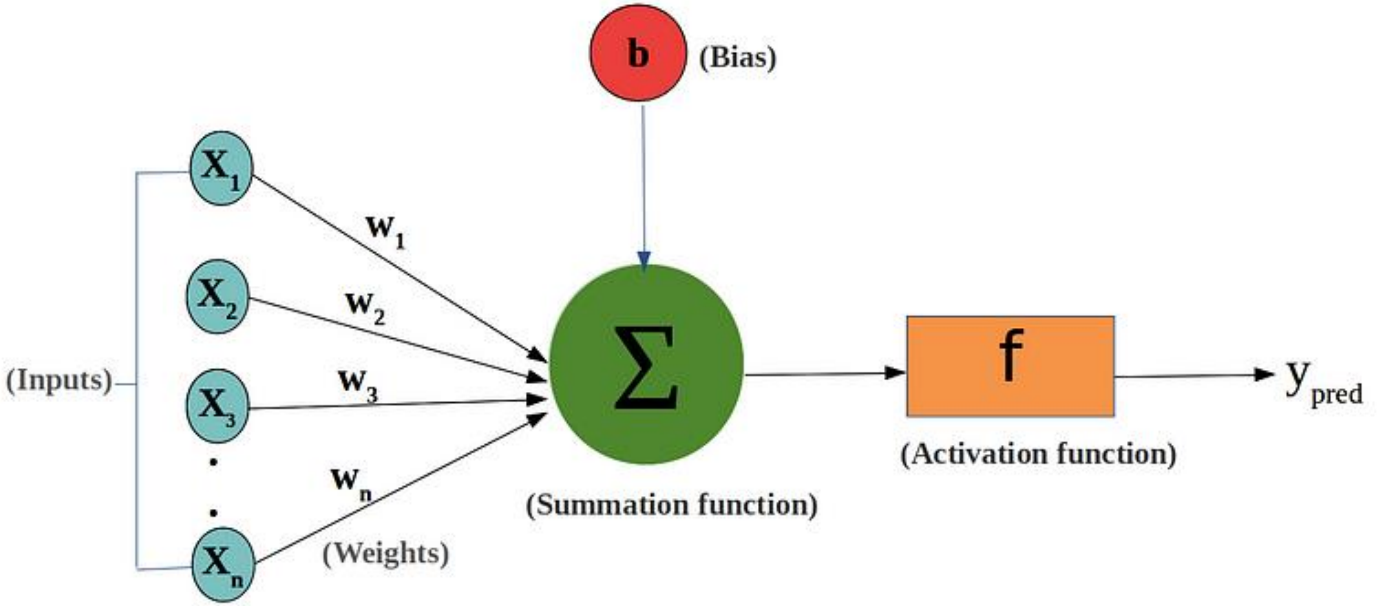
ReLU is typically used in the hidden layers of a neural network to add non-linearity. It is efficient and can help neural networks learn more complex relationships between the input and output data.

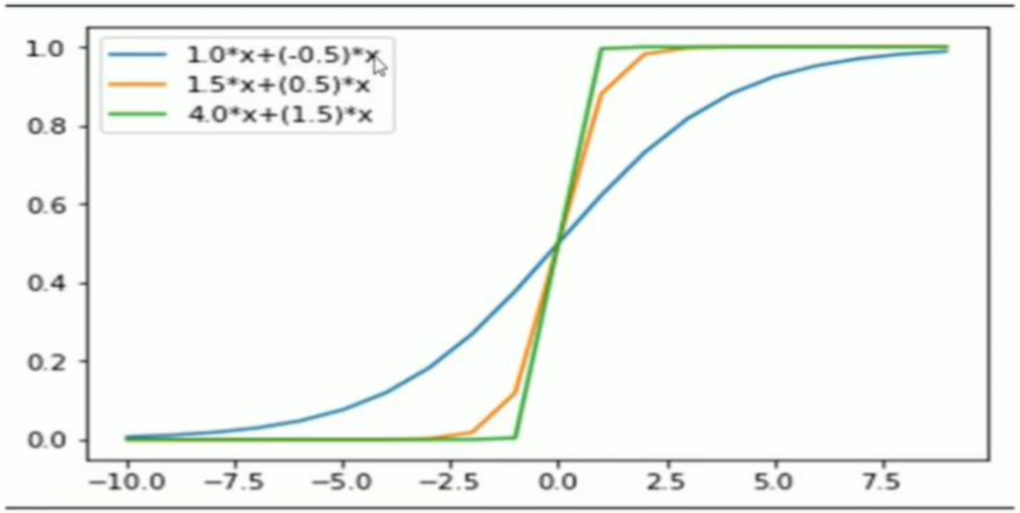
✓ Choosing the right Activation Function

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- ReLU function is a general activation function and is used in most cases these days
- If we encounter a case of dead neurons in our networks the leaky ReLU function is the best choice
- Always keep in mind that ReLU function should only be used in the hidden layers
- As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn’t provide with optimum results

WHAT IS BIAS..??

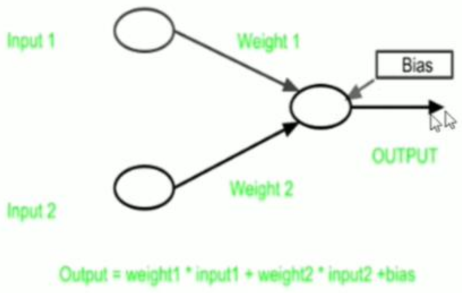
- **Bias** is one of the important terminologies.
- Often we add bias while creating any model in the artificial neural network.
- In a Neural network, **increase in weight** increases the **steepness of activation function.**
- Whereas **bias is used to delay the triggering of the activation function.**

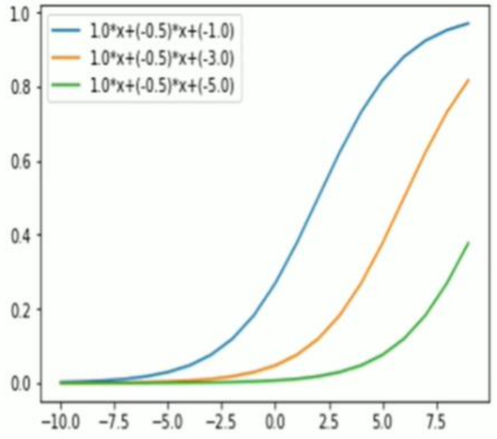




Therefore it can be inferred that:

More the weight , the earlier activation function will trigger.

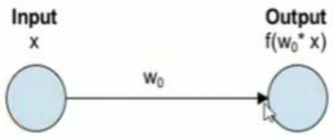




BIAS IS USED TO DELAY THE TRIGGERING OF THE ACTIVATION FUNCTION.

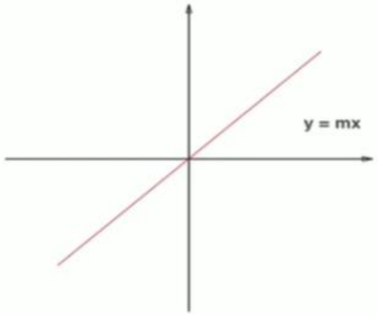
BIAS HELPS IN CONTROLLING THE VALUE AT WHICH ACTIVATION FUNCTION WILL TRIGGER.

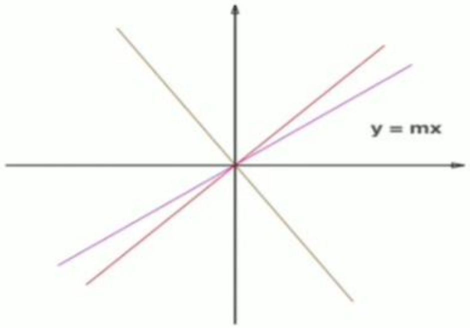
No Bias



Say, model is y = mx

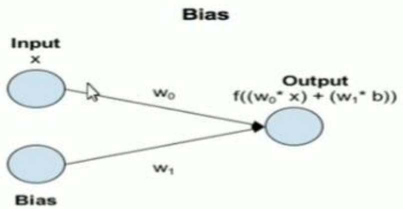
```
m = weight
```





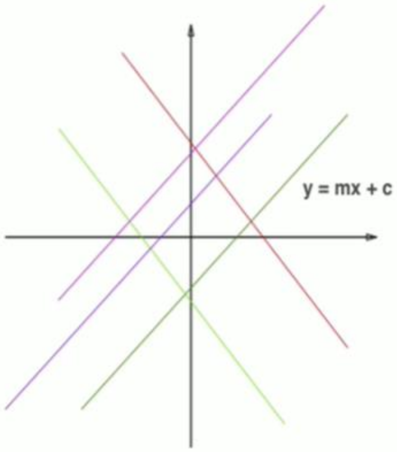
Here, the model is having constraint to train itself and find a line which passes only through the origin.

Many times for the given data, it is impossible for the algorithm to **fit the model so that it passes through the origin**.



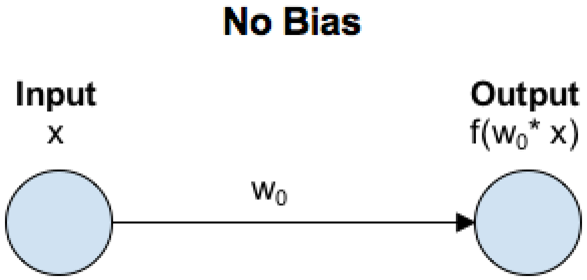
```
Instead of y=mx,

Change the model to y = mx+c

 m = weight   c = bias
```

- Now, it is having the full freedom to train itself and find a model that fits the best for the given data.
- **Bias** is a constant which helps the model in a way that it can fit best for the given data.
- In other words, **Bias** is a constant which gives freedom to perform the model at its best.

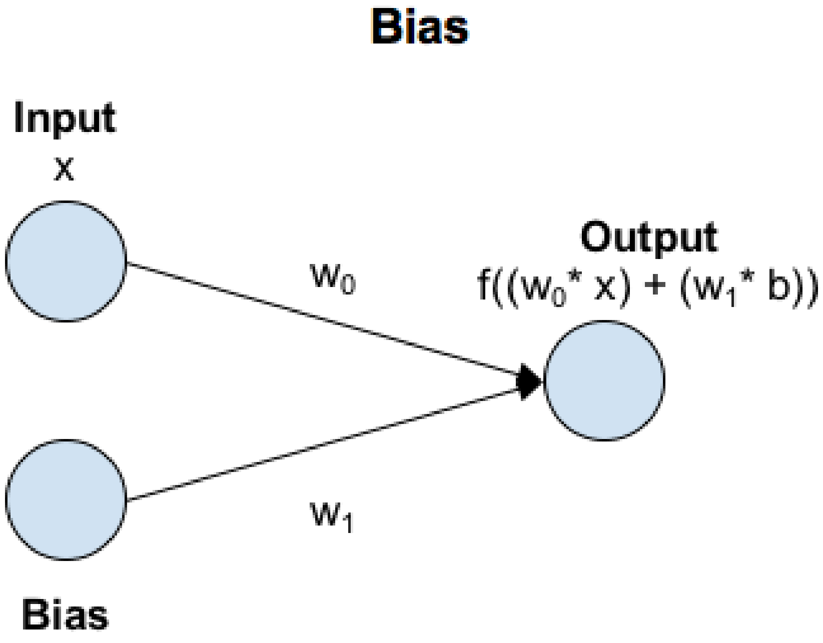


✓ The role of bias in Neural Networks

The activation function in Neural Networks takes an input 'x' multiplied by a weight 'w'. Bias allows you to shift the activation function by adding a constant (i.e. the given bias) to the input. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value.



In a scenario with no bias, the input to the activation function is 'x' multiplied by the connection weight 'w_0'.



In a scenario with bias, the input to the activation function is 'x' times the connection weight 'w_0' plus the bias times the connection weight for the bias 'w_1'. This has the effect of shifting the activation function by a constant amount (b * w_1).