Activation functions in Deep Learning

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

An activation function is a mathematical function applied to the output of a neuron. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns in the data. Without this non-linearity feature, a neural network would behave like a linear regression model, no matter how many layers it has.

The activation function decides whether a neuron should be activated by calculating the weighted sum of inputs and adding a bias term. This helps the model make complex decisions and predictions by introducing non-linearities to the output of each neuron.

Why is Non-Linearity Important in Neural Networks?

Neural networks consist of neurons that operate using **weights**, **biases**, and **activation functions**.

In the learning process, these weights and biases are updated based on the error produced at the output—a process known as **backpropagation**. Activation functions enable backpropagation by providing gradients that are essential for updating the weights and biases.

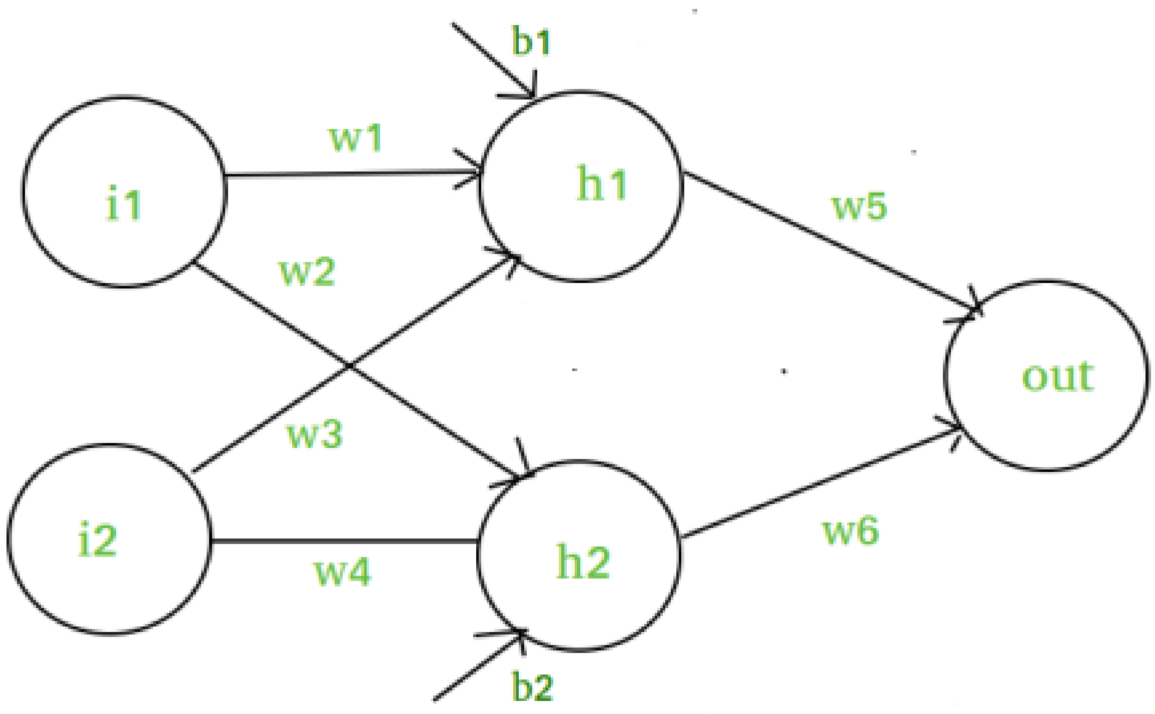
Without non-linearity, even deep networks would be limited to solving only simple, linearly separable problems. Activation functions empower neural networks to model highly complex data distributions and solve advanced deep learning tasks. Adding non-linear activation functions introduce flexibility and enable the network to learn more complex and abstract patterns from data.

Mathematical Proof of Need of Non-Linearity in Neural Networks

To illustrate the need for non-linearity in neural networks with a specific example, let's consider a network with two input nodes (i_1 and i_2), a single hidden layer containing one neuron (h_1), and an output neuron (out). We will use w_1, w_2 as weights connecting the inputs to the hidden neuron, and w_5 as the weight connecting the hidden neuron to the output. We'll also include biases (b_1 for the hidden neuron and b_2 for the output neuron) to complete the model.

Network Structure

1. **Input Layer**: Two inputs $i_1$and i_2.
2. **Hidden Layer**: One neuron h_1.
3. **Output Layer**: One output neuron.



Mathematical Model Without Non-linearity

Hidden Layer Calculation:

The input to the hidden neuron h1 is given as:

$$z_{h1} = w_1 \cdot i_1 + w_2 \cdot i_2 + b_1$$

Similarly, the input to the hidden neuron h2h_2h2 (though not shown explicitly in your description) would be:

$$z_{h2} = w_3 \cdot i_1 + w_4 \cdot i_2 + b_2$$

If no activation function is applied (i.e., the linear activation $h_1 = z_{h1}$ and $h_2 = z_{h2}$), the output neuron receives the following input:

Output Layer Calculation

The output neuron receives input from both hidden neurons h_1 and h_2:

$$output = w_5 \cdot h_1 + w_6 \cdot h_2 + b_3$$

Substituting $h_1 = z_{h1}$ and $h_2 = z_{h2}$, we get:

$$output = w_5 \cdot (w_1 \cdot i_1 + w_2 \cdot i_2 + b_1) + w_6 \cdot (w_3 \cdot i_1 + w_4 \cdot i_2 + b_2) + b_3$$

Expanding:

$$output = w_5 \cdot w_1 \cdot i_1 + w_5 \cdot w_2 \cdot i_2 + w_5 \cdot b_1 + w_6 \cdot w_3 \cdot i_1 + w_6 \cdot w_4 \cdot i_2 + w_6 \cdot b_2 + b_3$$

Grouping terms:

$$output = (w_5 \cdot w_1 + w_6 \cdot w_3) \cdot i_1 + (w_5 \cdot w_2 + w_6 \cdot w_4) \cdot i_2 + (w_5 \cdot b_1 + w_6 \cdot b_2 + b_3)$$

Conclusion

The above equation shows that the output is a **linear combination of the inputs** i1 and i2. Therefore, the network, despite having multiple layers, is effectively performing a linear transformation. This behavior is equivalent to a **single-layer perceptron**.

Introducing Non-Linearity in Neural Network

To introduce non-linearity, let’s use a non-linear activation function σ for the hidden neuron. A common choice is the ReLU function, defined as

σ(x)=max(0,x)

Updated Hidden Layer Calculation:

$$h_1 = \sigma(z_{h_1}) = \sigma(w_1 i_1 + w_2 i_2 + b_1)$$

Output Layer Calculation with Non-linearity:

$$\text{output} = w_5 \sigma(w_1 i_1 + w_2 i_2 + b_1) + b_2$$

Effect of Non-linearity

The inclusion of the ReLU activation function sigma allows h_1 to introduce a non-linear decision boundary in the input space. This non-linearity enables the network to learn more complex patterns that are not possible with a purely linear model, such as:

- Modeling functions that are not linearly separable.
- Increasing the capacity of the network to form multiple decision boundaries based on the combination of weights and biases.

Code: `activation_functions_neural_network.ipynb`

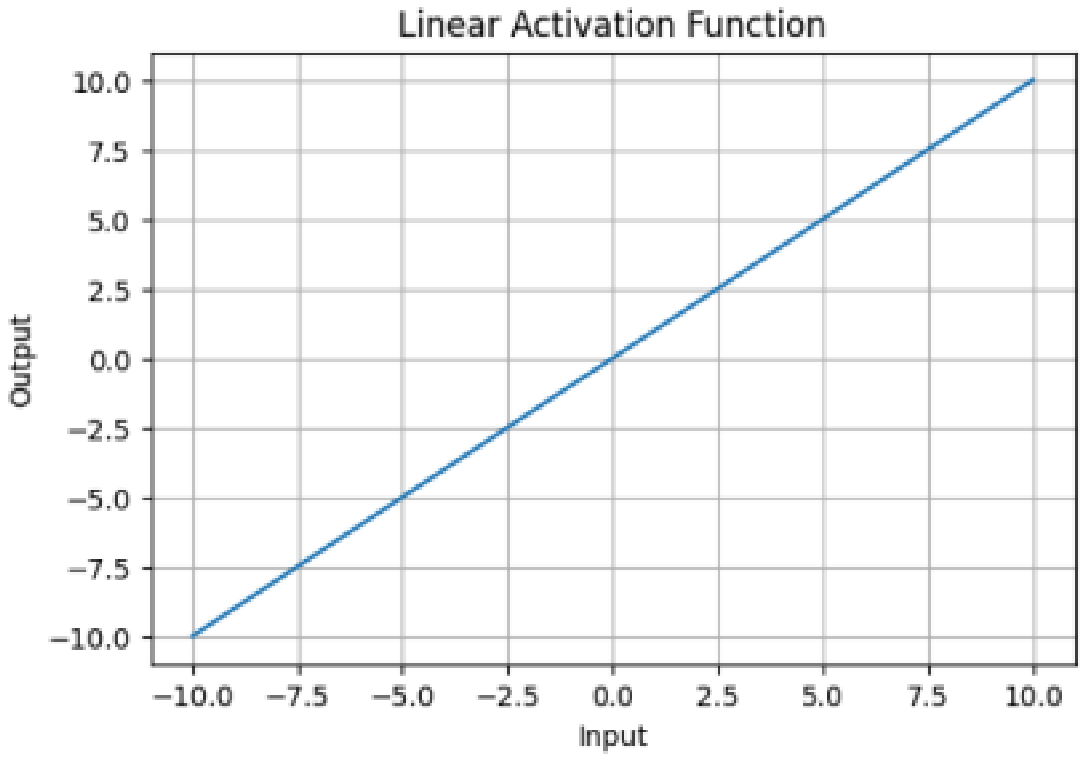
Types of Activation Functions in Deep Learning

1. Linear Activation Function

Linear Activation Function resembles straight line define by y=x. No matter how many layers the neural network contains, if they all use linear activation functions, the output is a linear combination of the input.

- The range of the output spans from
- $(-\infty \text{ to } +\infty)$
- $(-\infty \text{ to } +\infty)$.
- **Linear activation function** is used at just one place i.e. output layer.
- Using linear activation across all layers makes the network's ability to learn complex patterns limited.

Linear activation functions are useful for specific tasks but must be combined with non-linear functions to enhance the neural network's learning and predictive capabilities.



Linear Activation Function or Identity Function returns the input as the output

2. Non-Linear Activation Functions

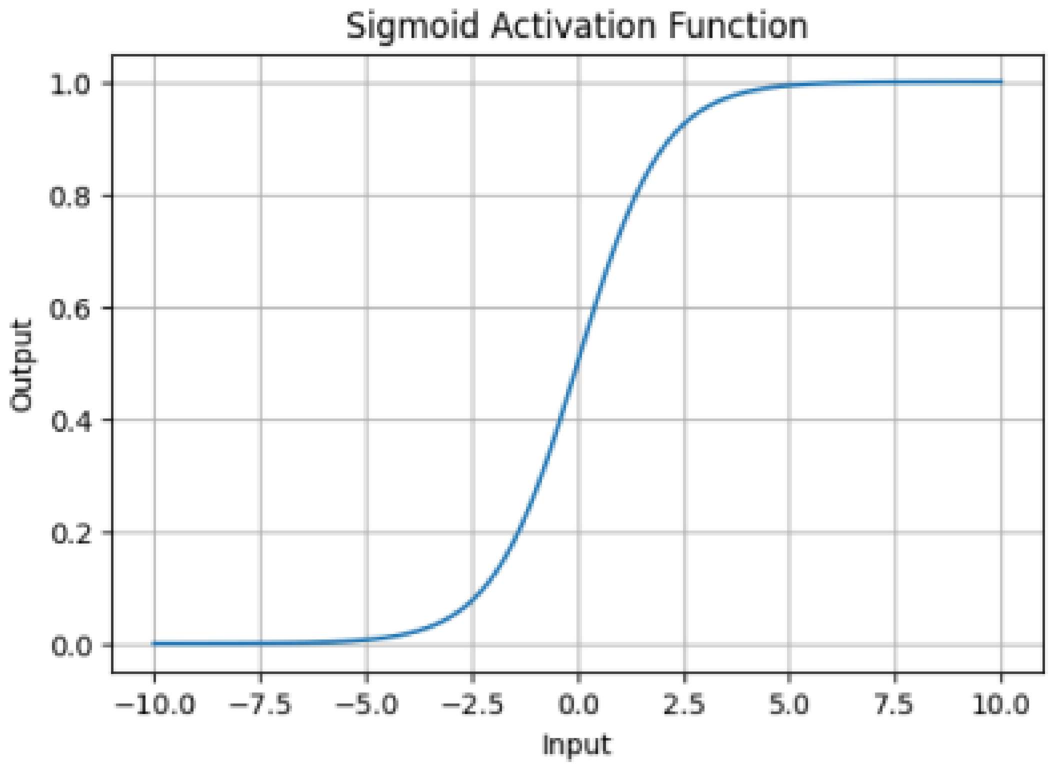
1. Sigmoid Function

Sigmoid Activation Function is characterized by 'S' shape. It is mathematically defined as

$$A = \frac{1}{1+e^{-x}}.$$

This formula ensures a smooth and continuous output that is essential for gradient-based optimization methods.

- It allows neural networks to handle and model complex patterns that linear equations cannot.
- The output ranges between 0 and 1, hence useful for binary classification.



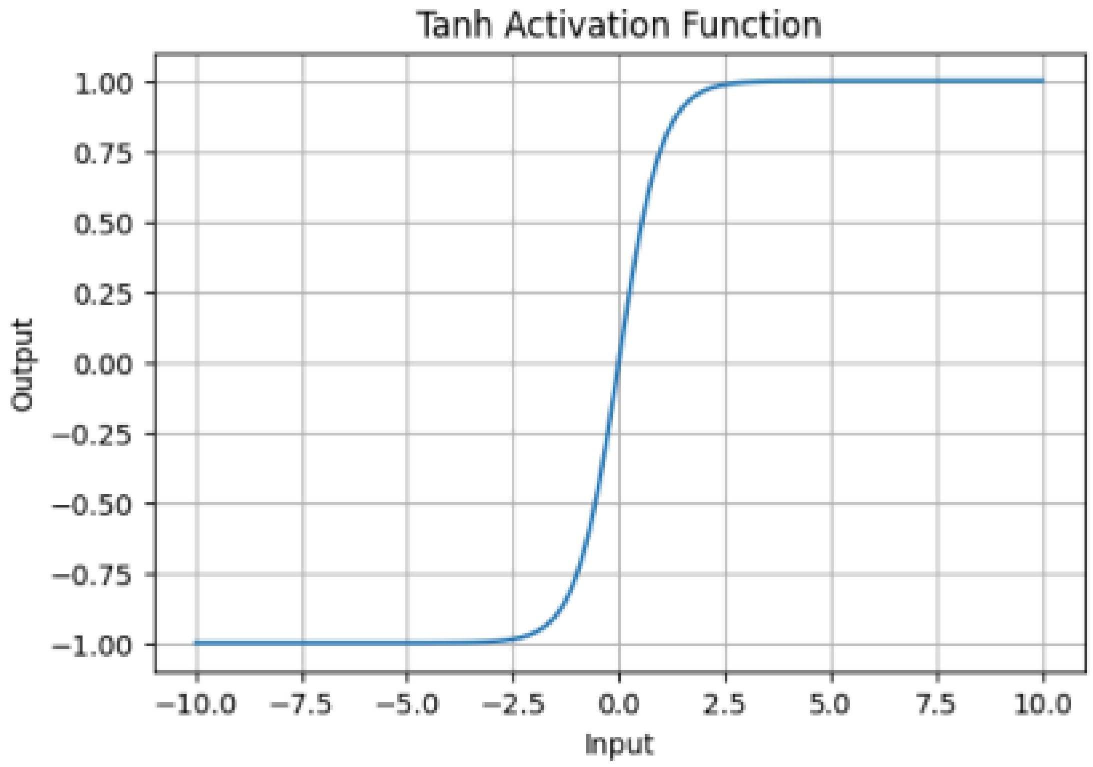
Sigmoid or Logistic Activation Function Graph

2. Tanh Activation Function

Tanh function or hyperbolic tangent function, is a shifted version of the sigmoid, allowing it to stretch across the y-axis. It is defined as:

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

- **Value Range**: Outputs values from -1 to +1.
- **Non-linear**: Enables modeling of complex data patterns.
- **Use in Hidden Layers**: Commonly used in hidden layers due to its zero-centered output, facilitating easier learning for subsequent layers.



Tanh Activation Function

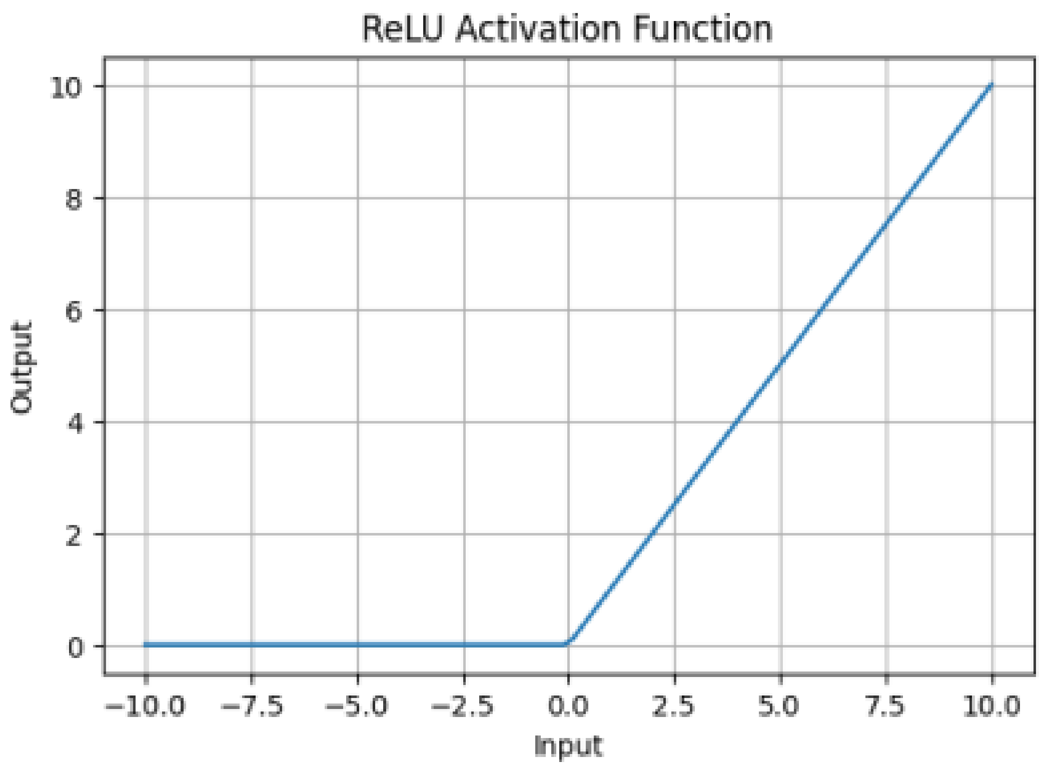
3. ReLU (Rectified Linear Unit) Function

ReLU activation is defined by

$A(x)=\max(0,x)$, this means that if the input x is positive, ReLU returns x, if the input is negative, it returns 0.

- **Value Range**: $[0,\infty)$, meaning the function only outputs non-negative values.
- **Nature**: It is a **non-linear** activation function, allowing neural networks to learn complex patterns and making backpropagation more efficient.
- **Advantage over other Activation:** 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.



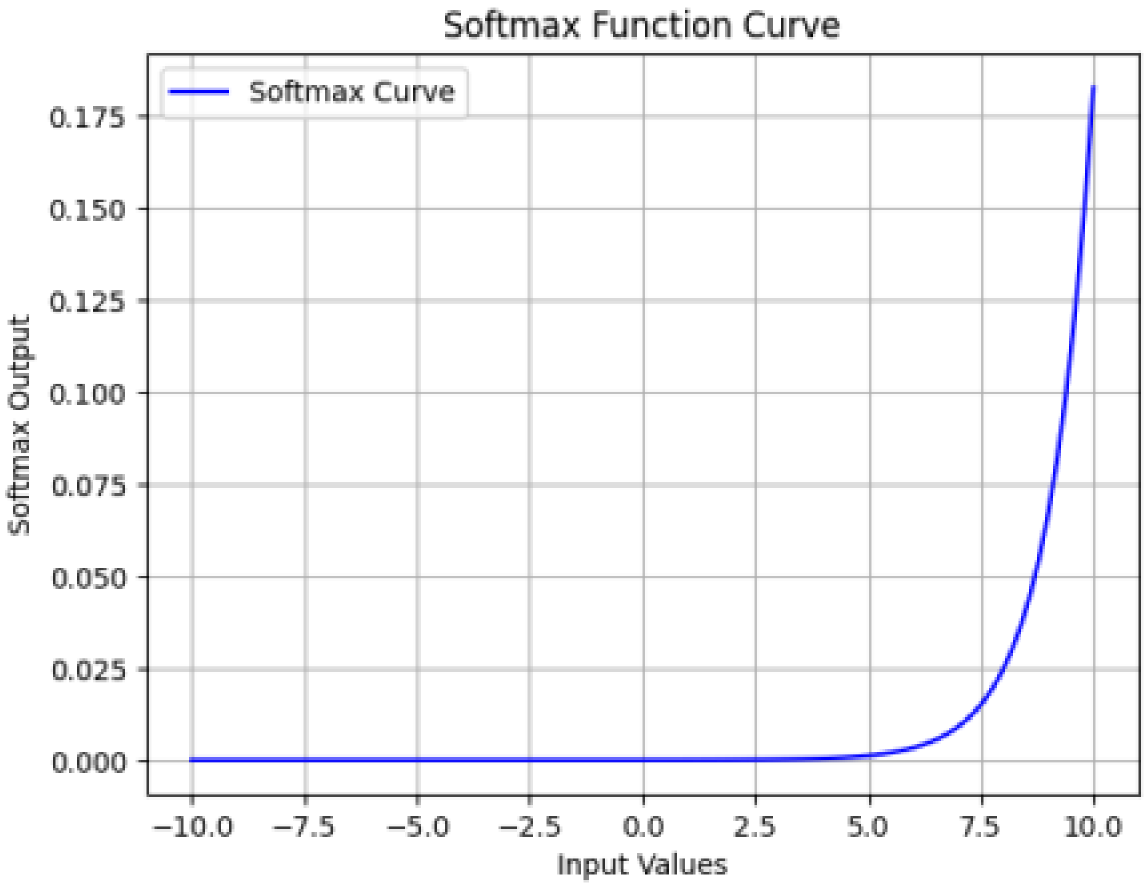
ReLU Activation Function

3. Exponential Linear Units

1. Softmax Function

Softmax function is designed to handle multi-class classification problems. It transforms raw output scores from a neural network into probabilities. It works by squashing the output values of each class into the range of 0 to 1, while ensuring that the sum of all probabilities equals 1.

- Softmax is a **non-linear** activation function.
- The Softmax function ensures that each class is assigned a probability, helping to identify which class the input belongs to.



Softmax Activation Function