Activation Functions in Neural Networks

What are Activation Functions?

Activation functions are mathematical equations that determine the output of a neural network. They basically decide to deactivate neurons or activate them to get the desired output, thus the name, *activation functions*.

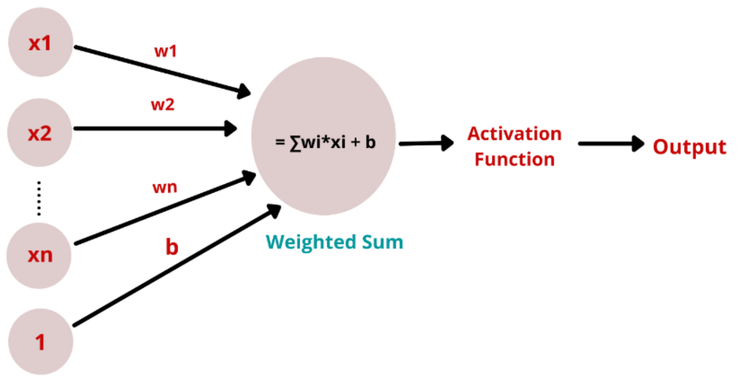
In a neural network, the weighted sum of inputs is passed through the activation function.

Y = Activation function(∑ (weights*input + bias))

- **Function:** Decide whether a neuron should be "fired" based on its weighted input and bias.
- **Output:** Smaller for small inputs, larger for larger inputs (like a threshold gate).
- **Purpose:** Introduce non-linearity into the network, enabling it to learn complex patterns.
- **Without activation functions**: Network becomes a simple linear model, limiting its capability.
- Different activation functions have different properties and are suitable for different tasks.
- Choosing the right activation function is crucial for optimizing network performance.
- Activation functions are applied after each layer's weighted sum and bias are calculated.
- 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.

Benefits:

- ✓ Enables learning non-linear relationships.
- ✓ Allows for backpropagation for error correction and weight adjustments.
- ✓ Makes the network more expressive and powerful.



A Simple Neural Network

Types of activation function:

1. Identity
2. Binary Step
3. Sigmoid
4. Tanh
5. ReLU
6. Leaky ReLU
7. Softmax

The two main categories of activation functions are:

- Linear Activation Function
- Non-linear Activation Functions

Linear Activation Function:

Equation: y = x, similar to a straight line.

Behaviour: Applies only a direct scaling to the input, no non-linearity added.

Impact on network:

- Single layer network: Works like a regular linear regression model.
- Multi-layer network: All layers become equivalent, collapsing into just one effective layer.

Range: (-∞, +∞).

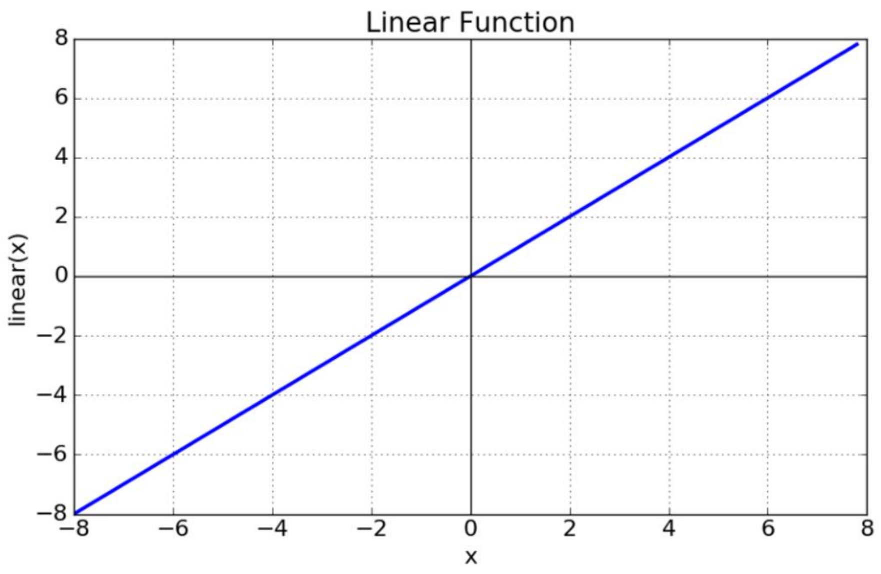
Use cases:

- Primarily in the output layer for regression problems with continuous, unbounded outputs (e.g., house price prediction).

Limitations:

- Cannot learn complex relationships due to missing non-linearity.
- Backpropagation ineffective as derivative is constant (1), preventing weight updates.
- Reduces network expressiveness and limits its usefulness for many tasks.

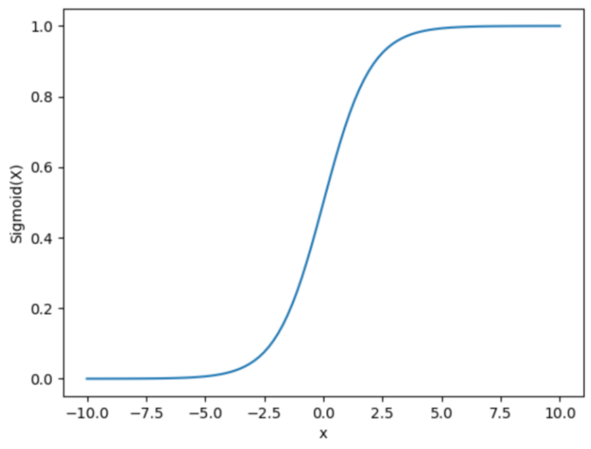
Example: Predicting house price with a linear activation at the output. This works because house price can range continuously from any small to large value. However, the hidden layers still need non-linear activation functions to capture complex relationships that influence the price.



Non-Linear Function

Sigmoid Function

- It is a function which is plotted as ‘S’ shaped graph.
- **Equation :** A = 1/(1 + e-x)
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be 1 if value is greater than 0.5 and 0 otherwise.



Pros and Cons

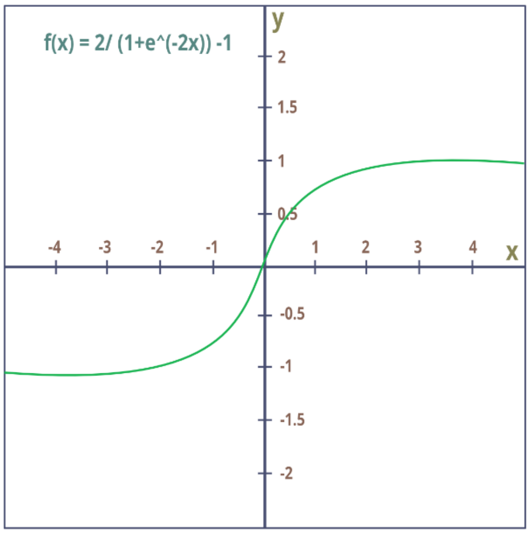
- It is non-linear in nature. Combinations of this function are also non-linear, and it will give an analogue activation, unlike binary step activation function. It has a smooth gradient too, and It's good for a classifier type problem.
- The output of the activation function is always going to be in the range (0,1) compared to (-∞, ∞) 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 Function

- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**

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

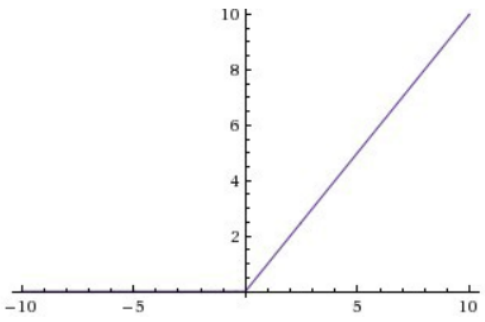
- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.



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 Function



- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, 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.

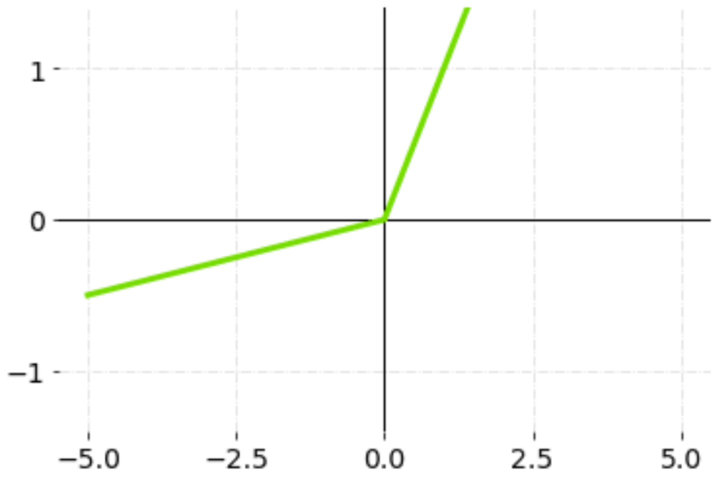
In simple words, RELU learns *much faster* than sigmoid and Tanh function.

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**.

4.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.



Leaky ReLU Activation Function—Graph

Mathematically, it can be represented as,

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

Leaky ReLU Activation Function—Equation

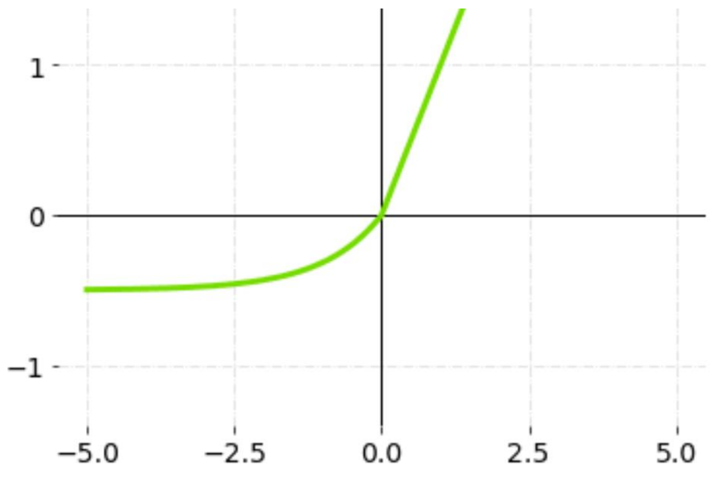
Pros and Cons

- 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.

5.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:

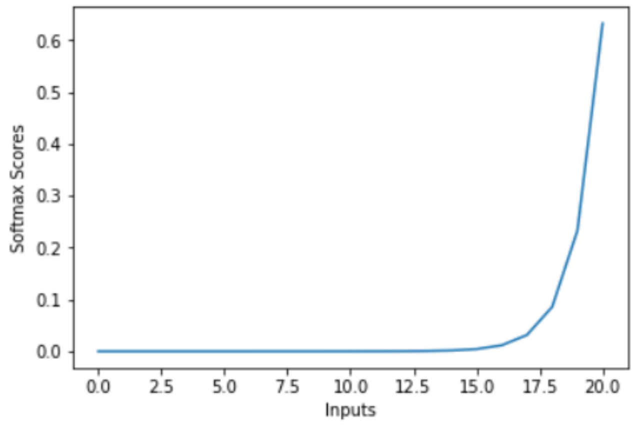
$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha\left(e^{x}-1\right) & for\ x < 0 \end{cases}$$

ELU Activation Function—Equation

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 Function



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi-class classification problems.

- **Nature :-** non-linear
- **Uses :-** Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
- **Output:-** The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
- 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 in hidden layers 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.
- If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Mathematically, it can be represented as:

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

Softmax Activation Function — Equation

Pros and Cons

- 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.

Choosing the right Activation Function

However depending upon the properties of the problem we might be able to make a better choice for easy and quicker convergence of the network.

- 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

- All hidden layers generally use the same activation functions. **ReLU** activation function **should only** be used in the hidden layer for better results.
- Sigmoid and TanH activation functions **should not be utilized** in hidden layers due to the vanishing gradient, since they make the model more susceptible to problems during training.
- Swish function is used in artificial neural networks having a depth **more than 40 layers.**
- Regression problemsshould uselinear activation functions
- Binary classification problems should use the sigmoid activation function
- Multiclass classification problemsshold use the softmax activation function

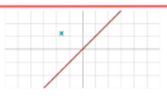
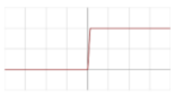
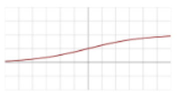
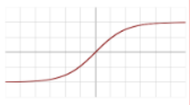
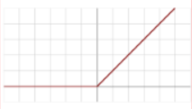
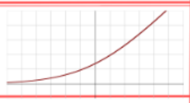
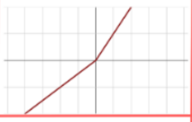
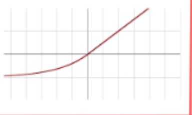
Neural network architecture and their usable activation functions,

- Convolutional Neural Network (CNN): ReLU activation function
- Recurrent Neural Network (RNN): TanH or sigmoid activation functions

While choosing the proper activation function, the following problems and issues must be considered:

Vanishing gradient is a common problem encountered during neural network training. Like a sigmoid activation function, some activation functions have a small output range (0 to 1). So a huge change in the input of the sigmoid activation function will create a small modification in the output. Therefore, the derivative also becomes small. These activation functions are only used for shallow networks with only a few layers. When these activation functions are applied to a multi-layer network, the gradient may become too small for expected training.

Exploding gradients are situations in which massive incorrect gradients build during training, resulting in huge updates to neural network model weights. When there are exploding gradients, an unstable network might form, and training cannot be completed. Due to exploding gradients, the weights' values can potentially grow to the point where they overflow, resulting in loss in **NaN** values.

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	$(-\infty, \infty)$
Binary Step		f(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		f(x) = $\sigma(x) = \frac{1}{1+e^{-x}}$	f'(x) = $f(x)(1-f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		f(x) = $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	f'(x) = $1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		f(x) = $\begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		f(x) = $\ln(1+e^x)$	f'(x) = $\frac{1}{1+e^{-x}}$	$(0, 1)$
Leaky ReLU		f(x) = $\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		f(x) = $\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	f'(x)= $\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$