AI Essentials: Activation Functions

crackbitswilp.in

Content

What is an activation function?

Activation Functions are used to compute the output values of neurons in hidden layers in a neural network. In other words, a node's input value $\boldsymbol{x}$ is transformed by applying a function g, which is called an activation function.
Activation functions can regulate the outputs of nodes and add a level of complexity that neural network without activation functions - or neural networks using just linear activation functions - cannot achieve.
One can also consider an activation function as a mathematical 'gate'. Each neuron of an hidden layer receives inputs $(x_i)_{i=1,\dots,n}$ from previous layers for which a linear combination with associated weights $(w_i)_{i=1,\dots,n}$ is calculated, i.e., $\sum_{i=1}^{n} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$. A bias term w_0 is added to the result and then passed to the 'gate', i.e, the activation function $g(w_0 + \sum_{i=1}^{n} w_i x_i)$ transforms the result. This defines the output of a neuron and is transmitted to the neurons of the next hidden layer.

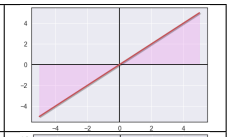
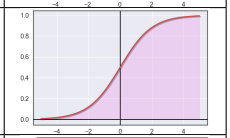
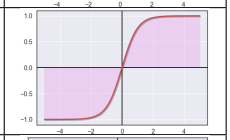
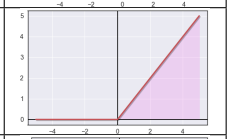
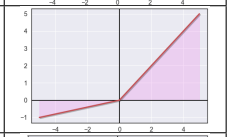
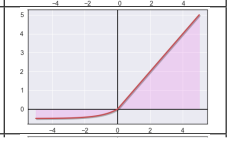
Why using non-linear activation functions?

Non-linear activation functions allow the model to create a more complex mapping between the inputs and the outputs. That is an important issue when analyzing complex and high dimensional data, like images, video, audio. The criteria for an activation function is that it has to be continuous and (almost everywhere) differentiable such that it can be used in the process of backpropagation. Comparing linear activation functions with non-linear activation functions, the latter ones can address the following problems:

- Their derivative is related to the inputs being important when updating the weights of the neural network (backpropagation). For linear functions the derivative is a constant and has no relation to the input.

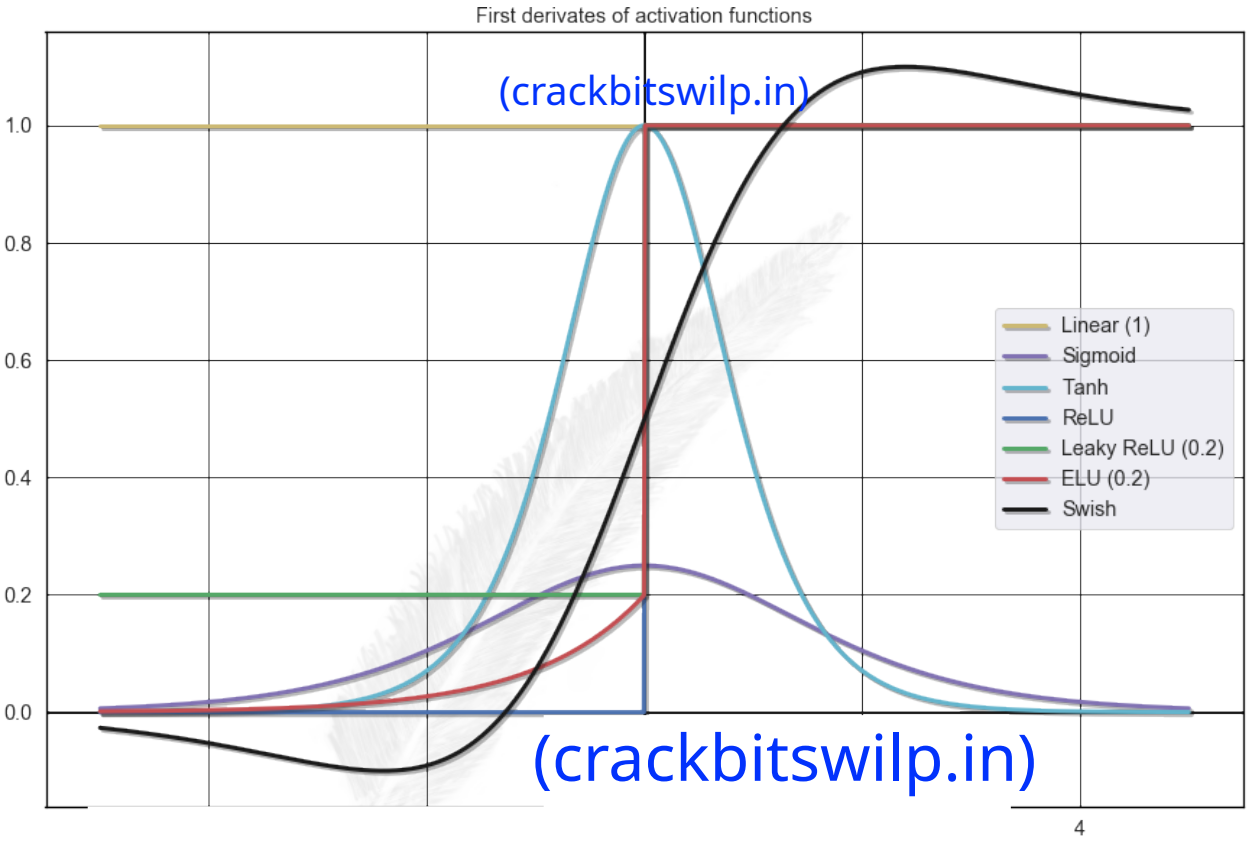
List of Activation Functions (crackbitswilp.in)

The following table shows a list of activation functions:

Name	Definition	Derivative	Plot
Linear	$g(x) = \alpha x$	$\frac{\partial g}{\partial x}(x) = \alpha$	
Sigmoid	$g(x) = \frac{1}{1+e^{-x}} = sig(x)$	$\frac{\partial g}{\partial x}(x) = g(x)(1 - g(x))$	
Tanh	$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 1 - \frac{2}{e^{2x}+1}$	$\frac{\partial g}{\partial x}(x) = 1 - g(x)^2$	
Rectified Linear Unit (ReLU)	$g(x) = max(0, z)$	$\frac{\partial g}{\partial x}(x) = \begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{for } x > 0 \end{cases}$	
Leaky ReLU	$g(x) = \begin{cases} \alpha x, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$	$\frac{\partial g}{\partial x}(x) = \begin{cases} \alpha, & \text{for } x < 0 \\ 1, & \text{for } x > 0 \end{cases}$	
Exponential Linear Unit (ELU)	$g(x) = \begin{cases} \alpha(e^x - 1), & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$	$\frac{\partial g}{\partial x}(x) = \begin{cases} g(x) + \alpha, & \text{for } x < 0 \\ 1, & \text{for } x > 0 \end{cases}$	

Note for functions including non-differentiable points.
As being (almost everywhere) differentiable for an activation function is a crucial part for learning a model, it may seem that functions with non-differentiable points are not eligible for being used in the optimization process., e.g., ReLU is not differentiable at $x = 0$.
From a practical point of view, these functions still perform well because neural network only approximates a local minimum of the cost function and does not hit it. Therefore, the minimum of the cost function can correspond to points with an undefined gradient. Nevertheless, for the case that we find ourselves in such a situation, then in modern software implementations it is either reported the left-hand or right-hand derivative of f at a point a instead of yielding that the derivative at a is not defined which might end up in an error. Therefore, it can be reconciled with our conscience that the non-differentiability of an activation function in only a small number of points can be disregarded.

While learning a neural network the derivative of an activation function play an important role in the process of backpropagation. In this process it is traced back from the output of the model, through the neurons having been involved in generating that output, to the input layer. Each weight being associated between two neurons is then adapted results in a more accurate prediction. Therefore, the following illustration shows the first derivative of each of the activation functions being listed above - the number within the brackets indicate the value for α for functions where it has to be set:



When to use which activation function? - Pros & Cons (crackbitswilp.in)

Name	Advantages	Disadvantages
Linear	• Activation values are real values; no binary activation	• No non-linearity can be learned • Derivative is constant - there is no relationship to an input x • With a constant gradient, the changes made by backpropagation is also constant
Sigmoid	• Smooth gradient, preventing 'jumps' in output values • Output values in range $[0, 1]$, normalizing the output of each neuron • Clear predictions: Low (High) values are very close to 0 (1)	• Vanishing gradient - for very high (low) values, there is almost no change to the prediction (known as vanishing gradient. This can lead to a slow learning or in the worst case there is no learning at all • Outputs are not zero centered making the optimization harder • Computationally expensive
Tanh	• Zero centered - modeling of inputs having strongly negative, neutral and positive values is facilitated	• Vanishing gradient - saturated activations kill the gradient
ReLU	• Learning procedure converges much faster than one with sigmoid/tanh activation functions • Computationally efficient - less mathematical operations • Avoids the vanishing gradient problem	• Dying ReLU Problem - when inputs approach zero or are negative, then the gradient is zero, hence, the network cannot perform backpropagation and there is no learning, leading to dead neurons Range of ReLU is $[0, \infty)$ • Output is not zero-centered
Leaky ReLU	• Learning procedure converges much faster than one with sigmoid/tanh activation functions • Computationally efficient • Prevents the Dying ReLU Problem - a small positive slope in the negative domain enables backpropagation even for negative input values	• Results are not consistent being a consequence of the additional hyperparameter α regularizing the slope in the negative domain

ELU	• Advantage over ReLU, ELU can produce negative outputs,hence, no dead neurons • Closer to zero mean outputs	• Computation requires calling $exp(\cdot)$
Swish	• No dying ReLU problem • Has shown to have a slightly increase in accuracy over ReLU	• Slightly more expensive in terms of mathematical operations being necessary compared to ReLU