Deep Learning and Applications
UEC630

Activation Functions and choice

By

Dr. Ashu 

Activation Function

- An activation function in the context of neural networks is a **mathematical function** applied to the output of a neuron
- The activation function **decides** whether a neuron should be activated or not.
- The purpose of the activation function is to introduce **non-linearity** into the output of a neuron.
- A neural network without an activation function is essentially just a linear regression model.

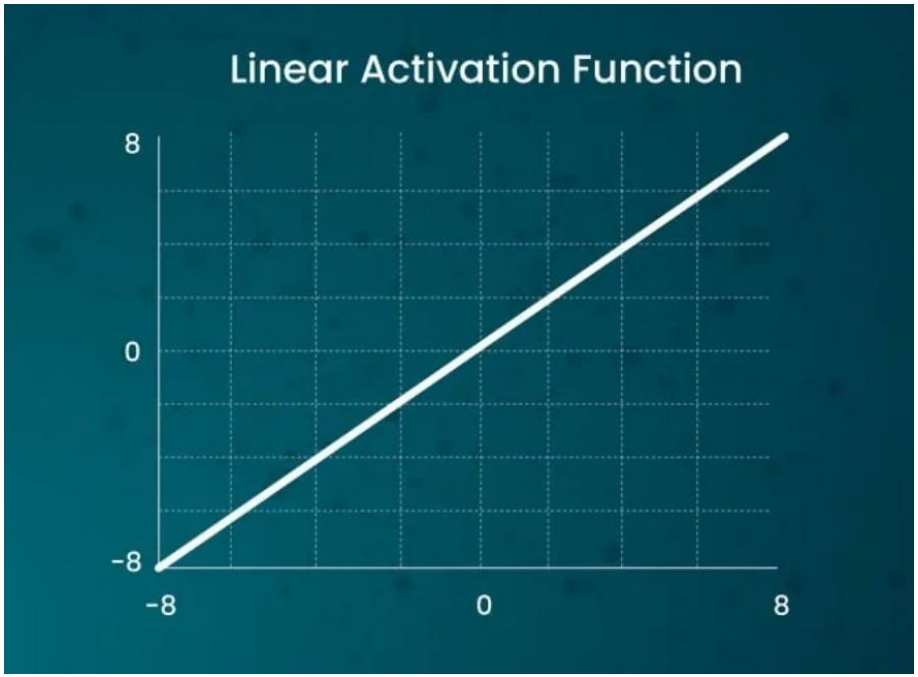
Why do we need activation functions?

- Basically an activation function is used to map the input to the output.
- This activation function helps a neural network to learn complex relationships and patterns in data.
- Now the question is what if we don't use any activation function and allow a neuron to give the weighted sum of inputs as it is as the output. Well in that case computation will be very difficult as the weighted sum of input doesn't have any range and depending upon input it can take any value. Hence one important use of the activation function is to keep output restricted to a particular range.

Playground Link

Types of activation Function

- Linear
- Non-Linear(Preferred)



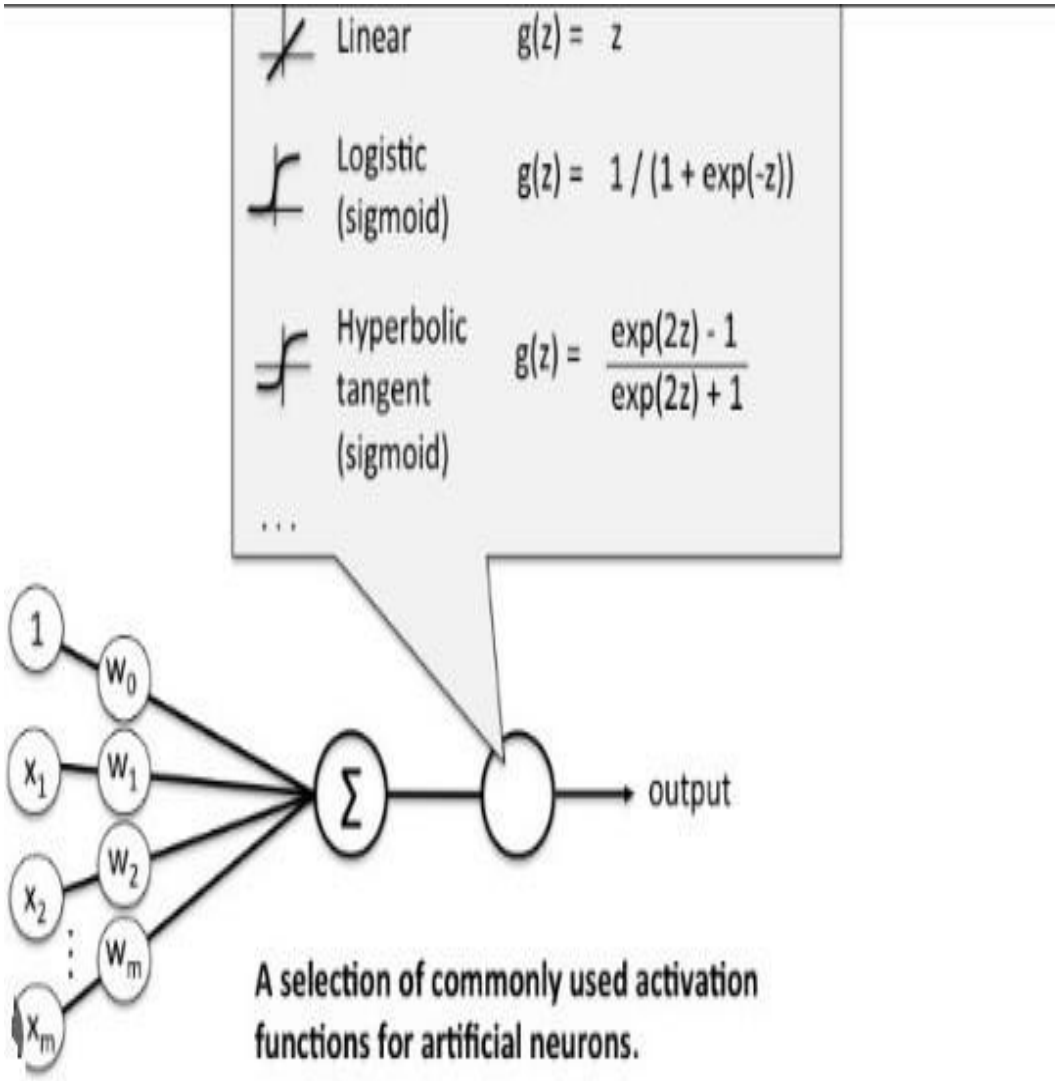
Why do we always choose non-linear activation functions in a neural network?

These are one of the most widely used activation function. It helps the model in generalizing and adapting any sort of data in order to perform correct differentiation among the output.

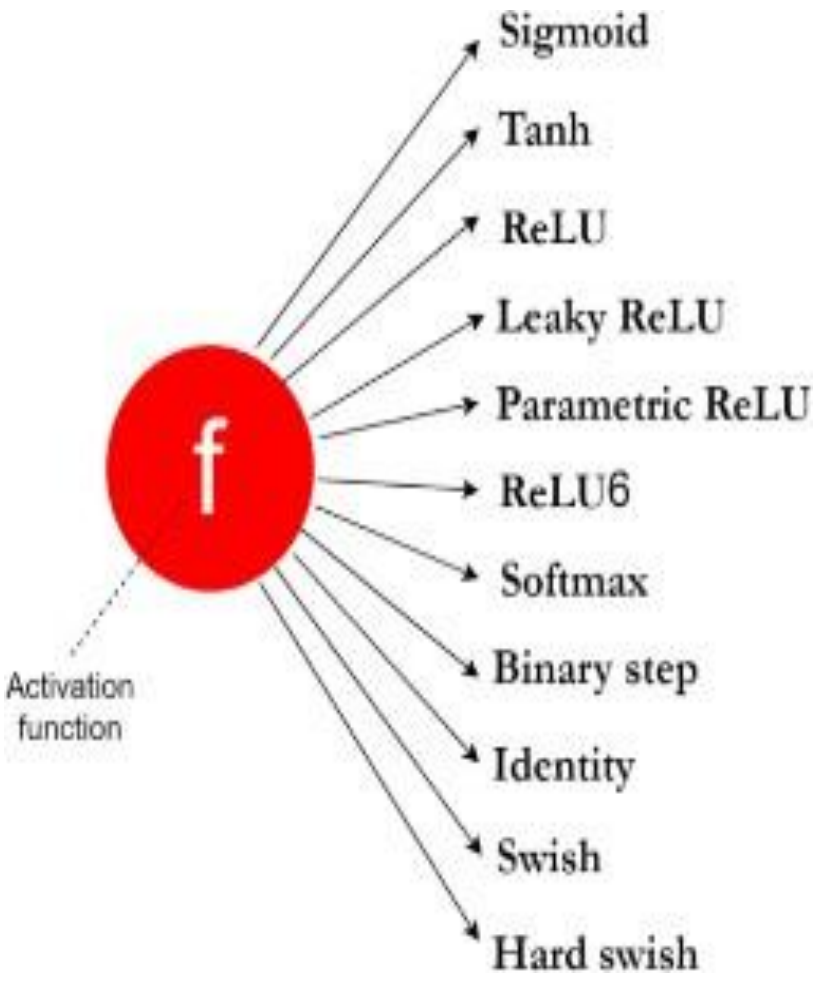
They can create a complex mapping between the input and output.

Since the non-linear function comes up with derivative functions, so the problems related to backpropagation has been successfully solved.

For the creation of deep neural networks, it permits the stacking up of several layers of the neurons.



A selection of commonly used activation functions for artificial neurons.

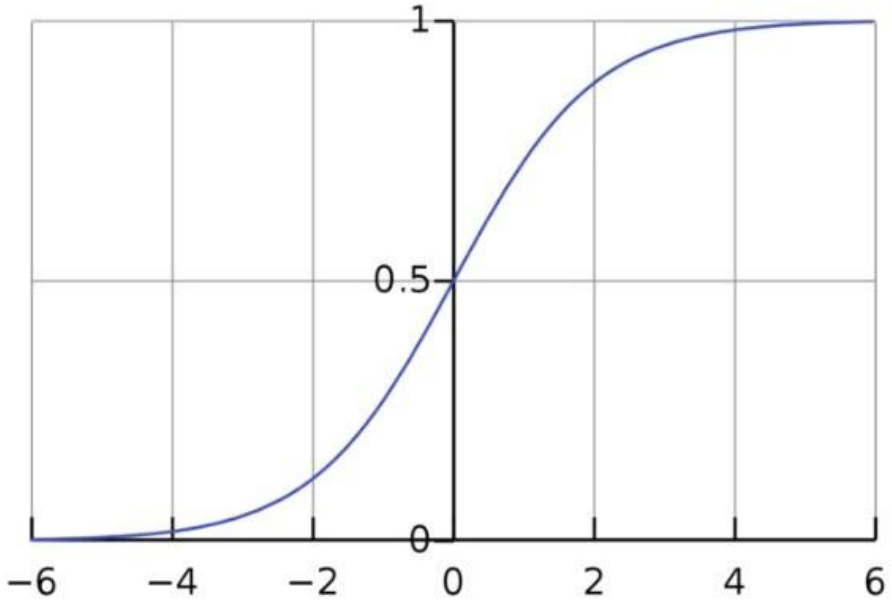


Activation Functions-

1. Sigmoid Function-

Sigmoid is an 'S' shaped mathematical function whose formula is-

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + 1}$$



Cont..

Pros-

- Sigmoid Function is continuous and differentiable.
- It will limit output between 0 and 1
- Very clear predictions for **binary classification.**

Cons-

- It can cause the vanishing gradient problem.
- It’s not centered around zero.
- Computationally Expensive

Hyperbolic Tangent (tanh) -

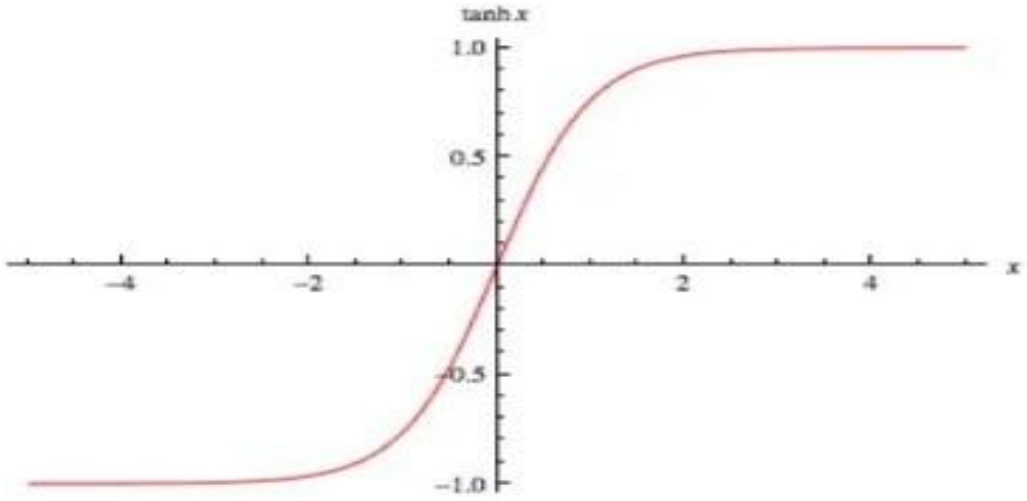
Hyperbolic Tangent or in short 'tanh' is represented by-

$$\tanh x = \frac{\sinh x}{\cosh x}$$

Image by Author

$$\tanh x = \frac{e^x - e^{-x}}{2} \div \frac{e^x + e^{-x}}{2} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

It is very similar to the sigmoid function. It is centered at zero and has a range between -1 and +1.



Source: Wikipedia

Pros-

It is continuous and differentiable everywhere.

It is centered around zero.

It will limit output in a range of -1 to +1.

Cons-

It can cause the vanishing gradient problem.

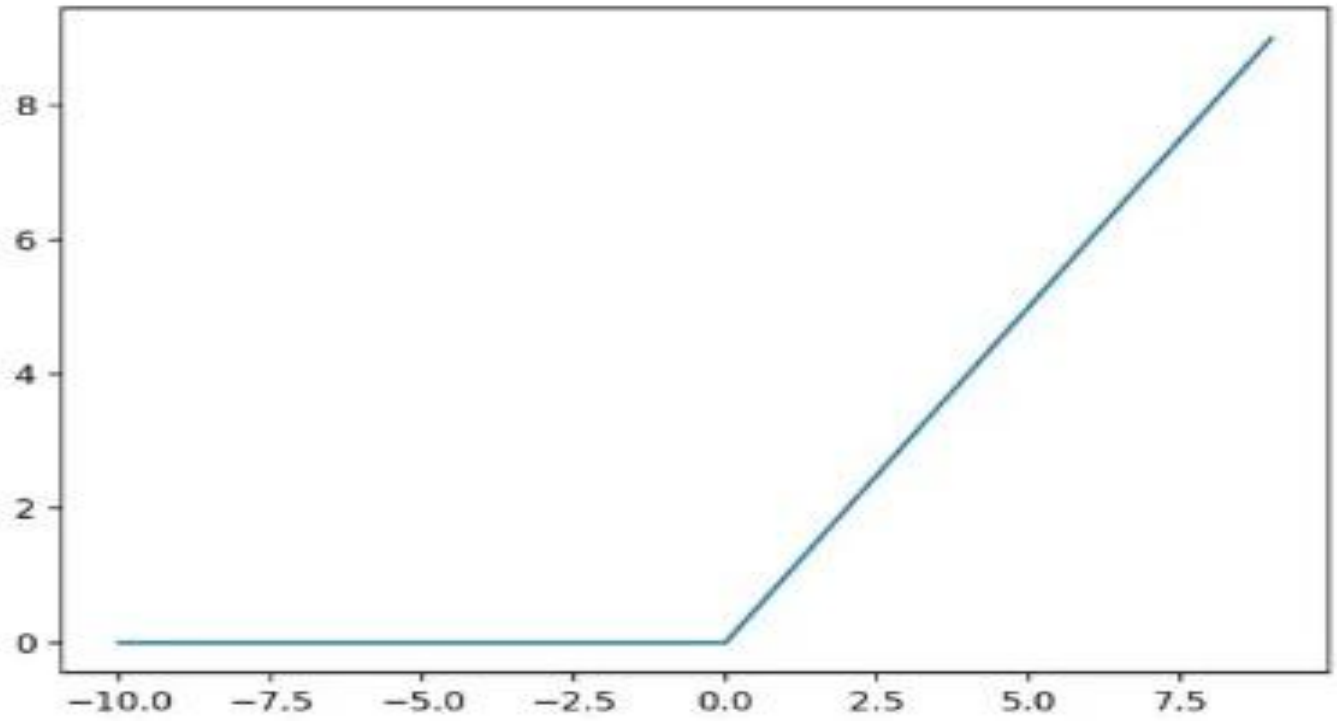
Computationally expensive.

Relu

Rectified linear Unit often called as just a rectifier or relu is-

$$f(x) = x^+ = \max(0, x),$$

Image by Author



Pros-

Easy to compute.

Does not cause vanishing gradient problem

As all neurons are not activated, this creates sparsity in the network and hence it will be fast and efficient.

Outperforms where the output should be non-negative

Cons-

Cause Exploding Gradient problem.

Not Zero Centered.

Can kill some neurons forever as it always gives 0 for negative values.

Leaky Relu-

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases}$$

Source: Wikipedia

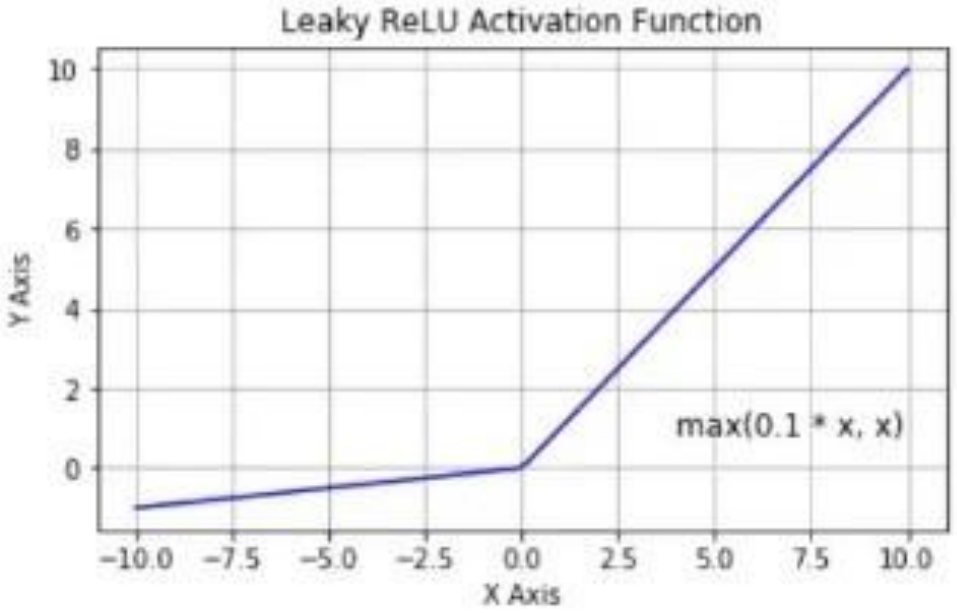


Image Source

Cont..

Pros-

- Easy to compute.
- Does not cause vanishing gradient problem
- Does not cause the dying relu problem.

Cons-

- Cause Exploding Gradient problem.
- Not Zero Centered

Parameterized ReLU

In parameterized relu, instead of fixing a rate for the negative axis, it is passed as a new trainable parameter which network learns on its own to achieve faster convergence.

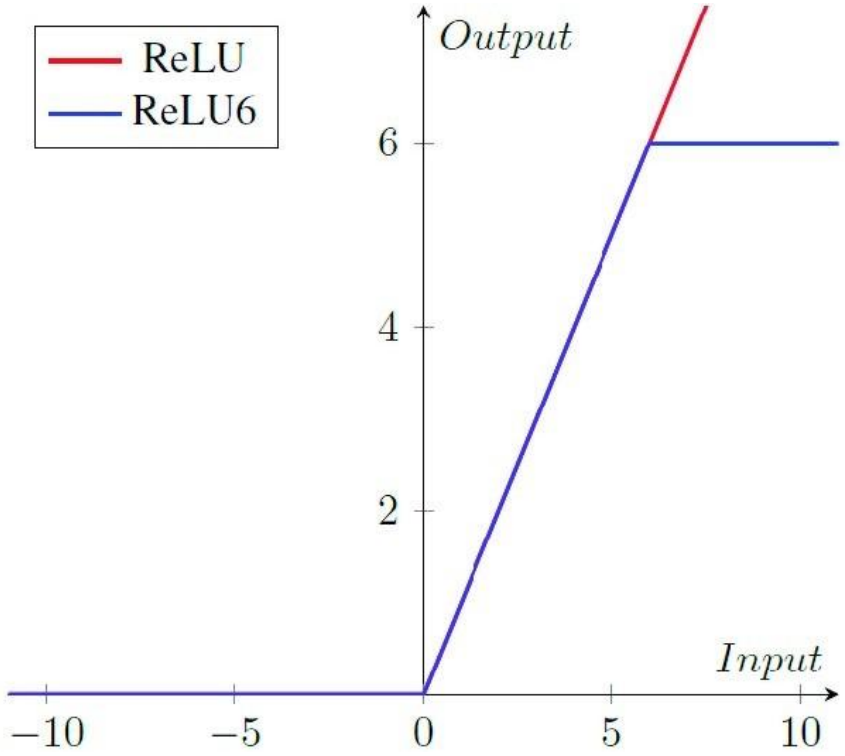
$$f(x) = \begin{cases} x & \text{if } x > 0, \\ ax & \text{otherwise.} \end{cases}$$

Source: Wikipedia

ReLU6

A Modified Version of Rectified Linear Unit

The ReLU6 function is defined as f(x) = min(max(0,x),6). The primary reason to use ReLU6 instead of the original ReLU function is **to improve the robustness of low-precision computation.** Neural networks are typically designed to use high-precision floating point arithmetic. However, in applications where computational resources are limited, the use of low-precision arithmetic (e.g. 8-bit or 16-bit) can lead to faster computation with minimal degradation in accuracy.



Softmax Function-

- The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1.
- The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into **values between 0 and 1,** so that they can be interpreted as probabilities.
- If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1.

The standard (unit) softmax function $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is defined by the formula

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

Source: Wikipedia

Cont..

Pros-

It can be used for **multiclass classification** and hence used in the output layer of neural networks.

Cons-

It is computationally expensive as we have to calculate a lot of exponent terms.

*Characteristics of a **Sigmoid Activation Function***

- Used for Binary Classification in the Logistic Regression model
- The probabilities sum does not need to be 1
- Used as an Activation Function while building a Neural Network

*Characteristics of a **Softmax Activation Function***

- Used for Multi-classification in the Logistics Regression model
- The probabilities sum will be 1
- Used in the different layers of Neural Networks

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

Sigmoid input values: -0.5, 1.2, -0.1, 2.4

Sigmoid output values: 0.37, 0.77, 0.48, 0.91

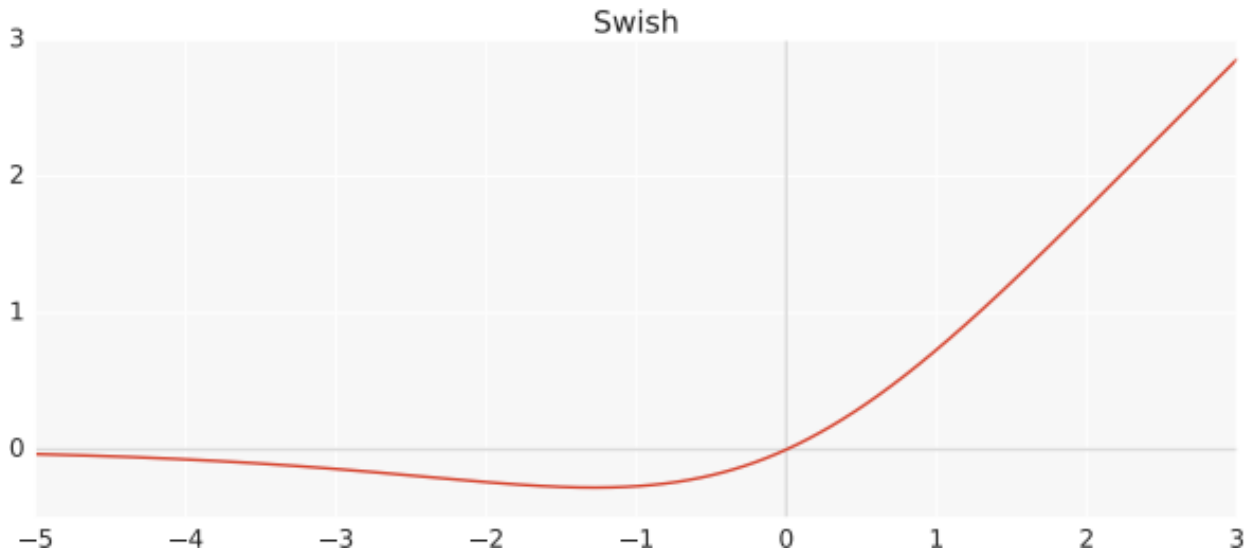
$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for j} = 1,\ldots,\text{K}$$

SoftMax input values: -0.5, 1.2, -0.1, 2.4

SoftMaxoutput values: 0.04, 0.21, 0.05, 0.70

Swish

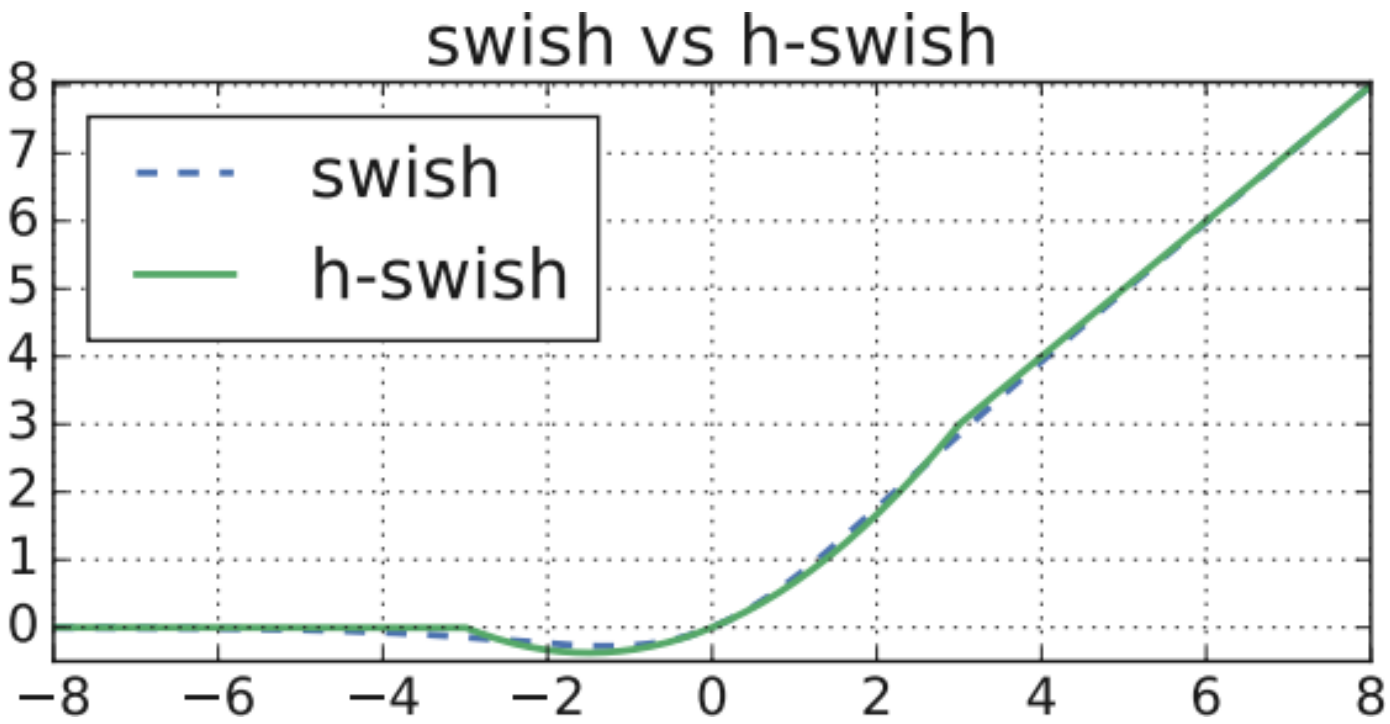
Google Brain Team proposed a new activation function, named Swish, which is simply $f(x) = x \cdot \text{sigmoid}(x)$. Their experiments show that Swish tends to work better than ReLU on deeper models across a number of challenging data sets. In very deep networks, Swish achieves higher test accuracy than ReLU.



Hard Swish

Hard Swish is a type of activation function based on Swish, but replaces the computationally expensive sigmoid with a piecewise linear analogue:

$$\text{h-swish}(x) = x \frac{\text{ReLU6}(x+3)}{6}$$



Hard Swish

The hard swish has no discernible difference in accuracy but multiple advantages from a deployment perspective.

First, optimized implementations of ReLU6 are available on virtually all software and hardware frameworks.

Second, in quantized mode, it eliminates potential numerical precision loss caused by different implementations of the approximate sigmoid.

Finally, in practice, h-swish can be implemented as a piece-wise function to reduce the number of memory accesses driving the latency cost down substantially.