Activation Functions

Introduction

- **An Activation Function** decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.

- The primary role of the Activation Function is to transform the summed weighted input from the node into an output value to be fed to the next hidden layer or as output.

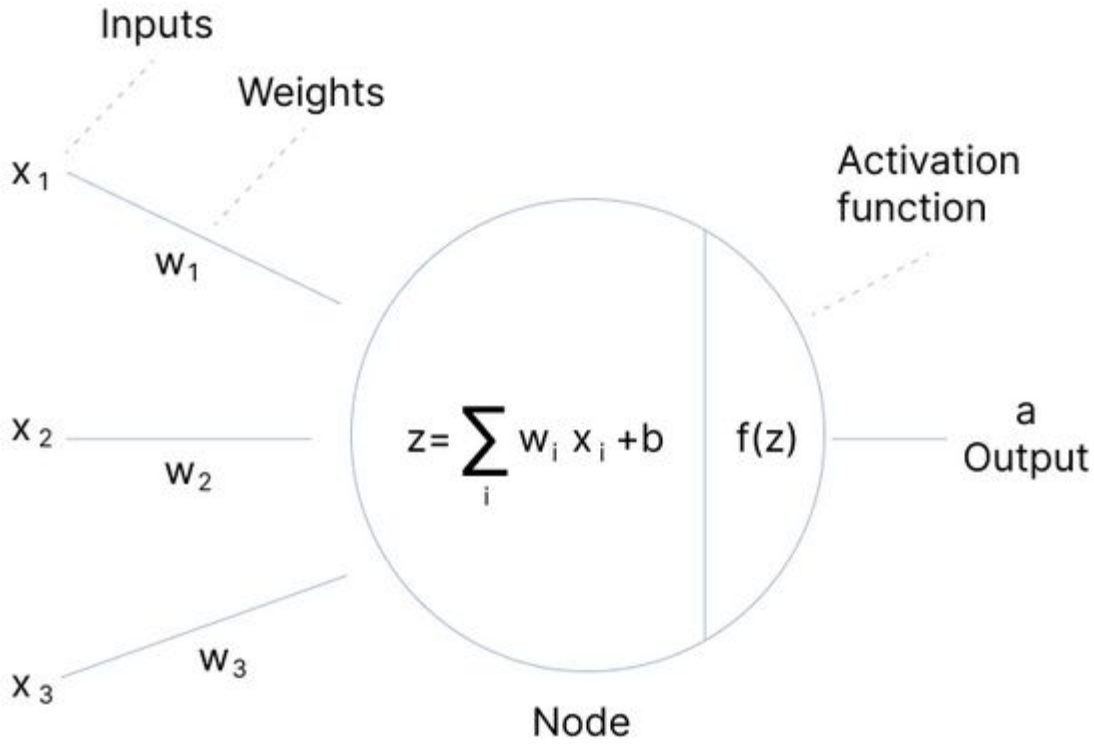


Figure: Single Neural Network Architecture

Need of Activation Functions

- The purpose of an activation function is to add non-linearity to the neural network.

- Every neuron will only be performing a linear transformation on the inputs using the weights and biases. It's because it doesn't matter how many hidden layers we attach in the neural network; all layers will behave in the same way because the composition of two linear functions is a linear function itself.

- Although the neural network becomes simpler, learning any complex task is impossible, and our model would be just a linear regression model.

- Sometimes the activation function is called a “transfer function.”

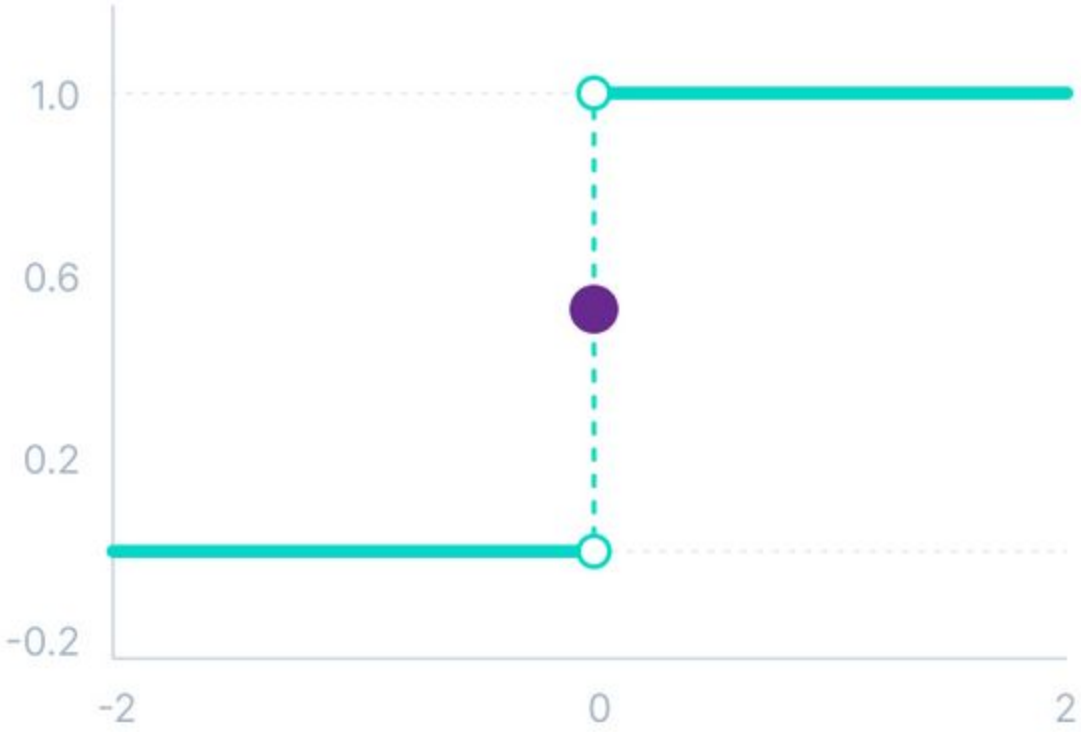
- The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.

Binary Step Function

- Binary step function depends on a threshold value that decides whether a neuron should be activated or not.

- The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.

- The gradient of the step function is zero, which causes a hindrance in the back propagation process.

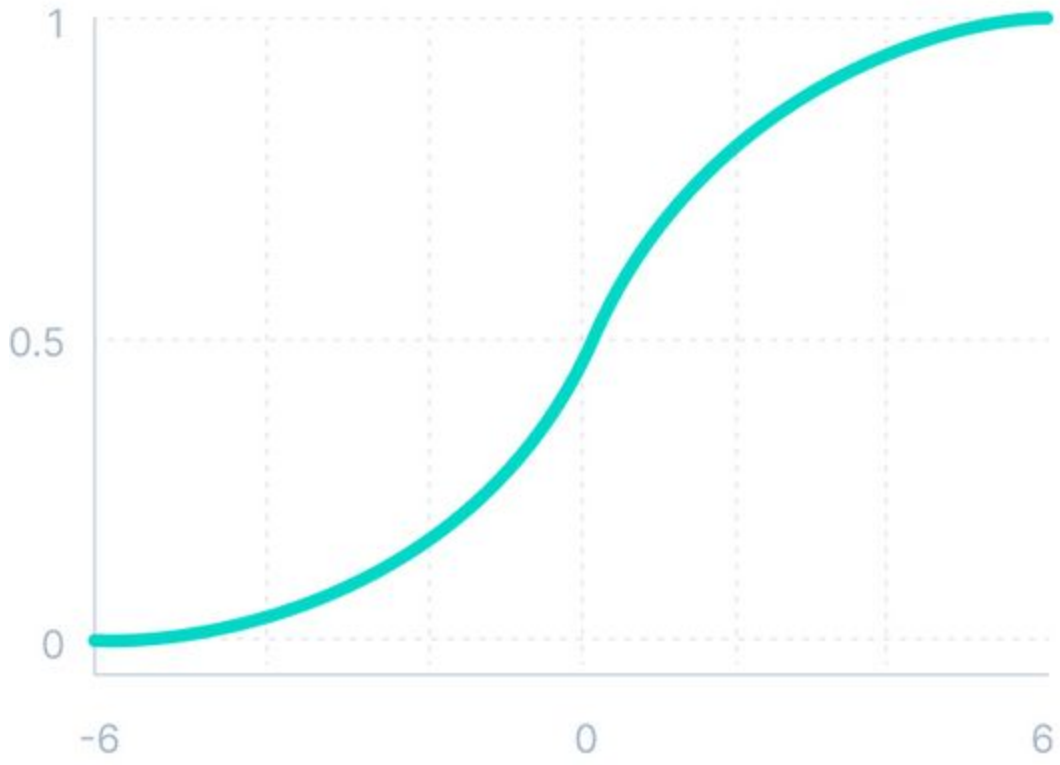


$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

Sigmoid Function

- The function ranges from 0 to 1 having an S shape. Also known by the name of the logistic or squashing function in some literature. The sigmoid function is used in output layers of the DNN and is used for probability-based output.

- Its major drawbacks are sharp damp gradients during back propagation, gradient saturation, slow convergence, and non-zero centered output thereby causing the gradient updates to propagate in different directions.



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

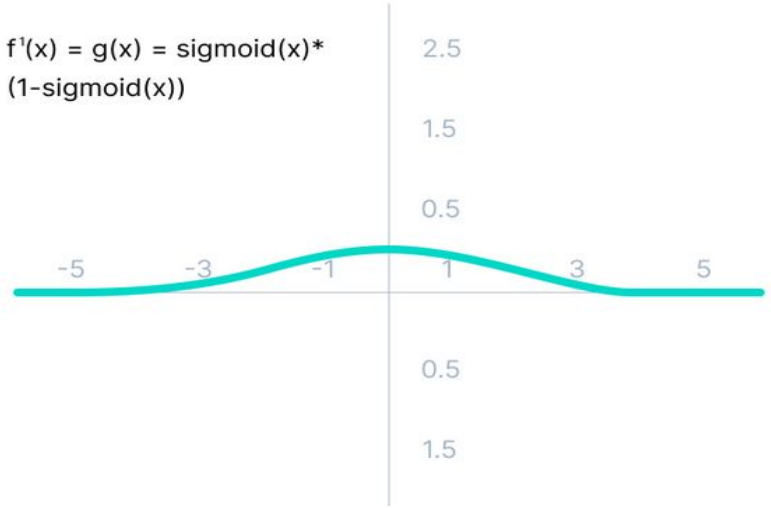
- It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.

- The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

Derivative of Sigmoid Function

- The derivative of the function is

$$f'(x) = sigmoid(x)*(1-sigmoid(x))$$



- with logistic output unit
  - with tanh output unit

$$y_j = \frac{1}{1+\exp(-z_j)} \qquad \frac{\partial y_j}{\partial z_j} = y_j(1-y_j)$$

$$y_j = \tanh(z_j)$$

$$= \frac{2}{1+\exp(-z_j)} - 1 \qquad \frac{\partial y_j}{\partial z_j} = (1+y_j)(1-y_j)$$

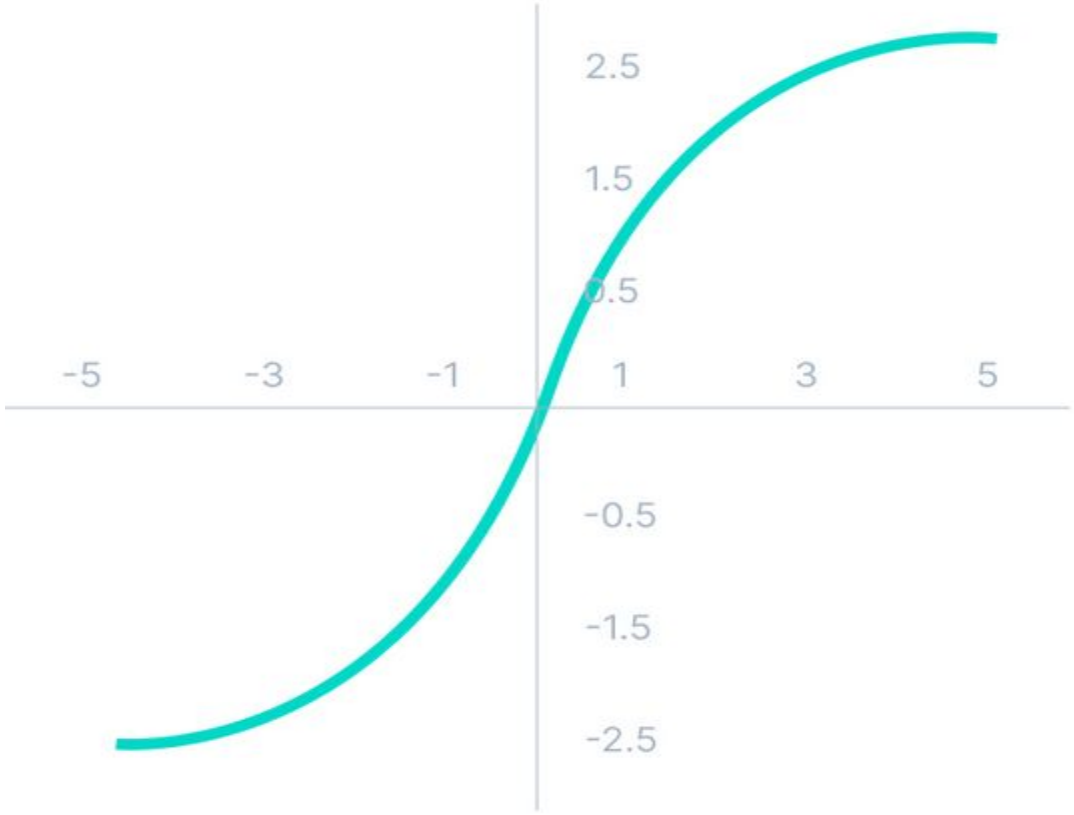
- As we can see from the above Figure, the gradient values are only significant for range -3 to 3, and the graph gets much flatter in other regions.

- It implies that for values greater than 3 or less than -3, the function will have very small gradients. As the gradient value approaches zero, the network ceases to learn and suffers from the *Vanishing gradient* problem.

Tanh Function (Hyperbolic Tangent)

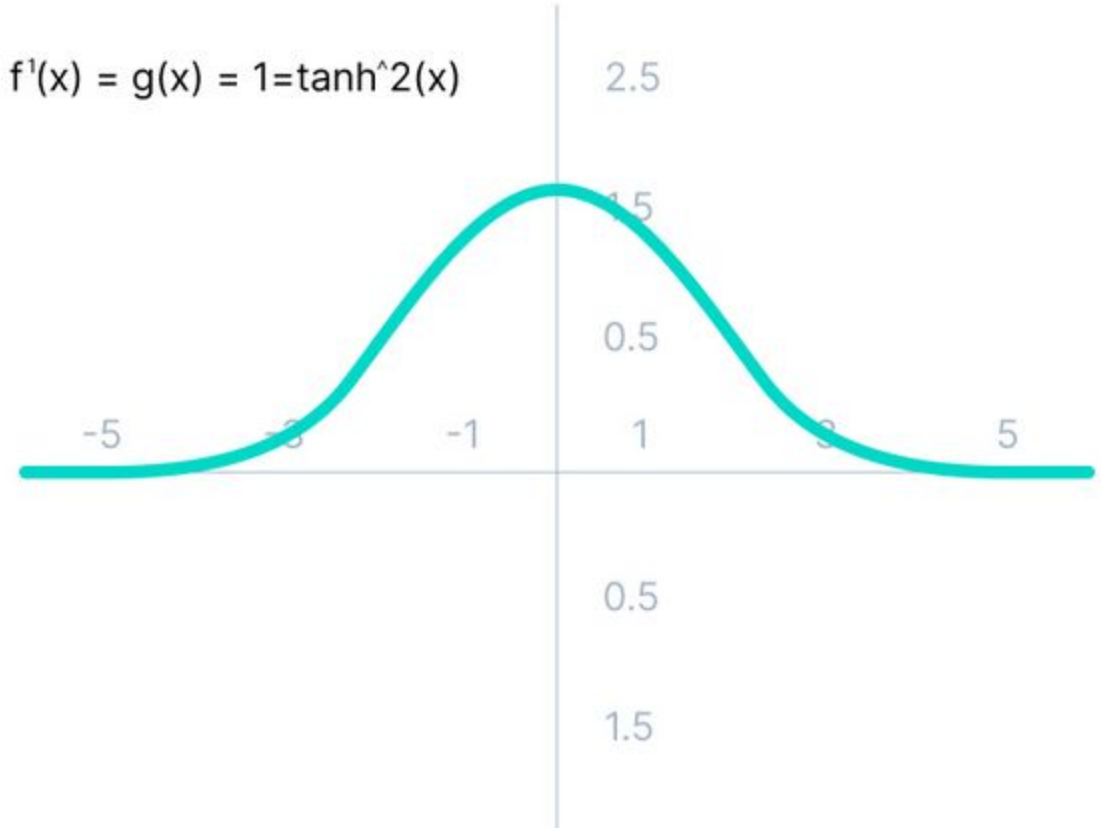
- Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

- As this function is zero centered, this makes it easier to model inputs that have strongly negative, neutral, and strongly positive values.



$$f(x) = \frac{\left(e^{x} - e^{-x}\right)}{\left(e^{x} + e^{-x}\right)}$$

Derivative of tanh Function

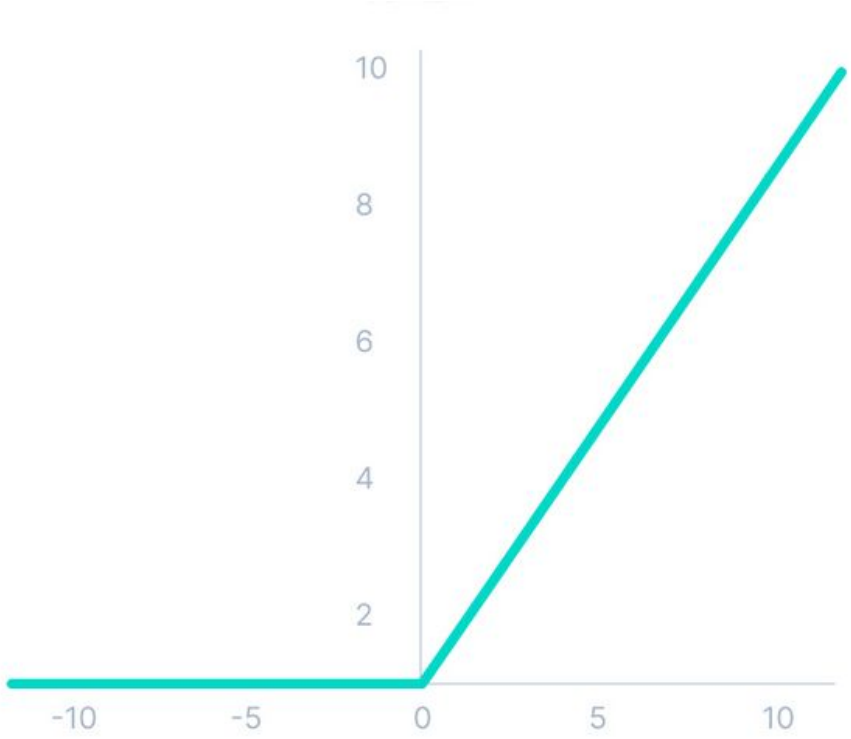


Relu Activation Function

- ReLU has been the most widely used activation function for DL applications with state-of-the-art results.

- It provides the upper hand in performance and generalization compared to the Sigmoid and Tanh activation functions.

- Along with the overall speed of computation enhanced, ReLU provides faster computation since it does not compute exponentials and divisions.

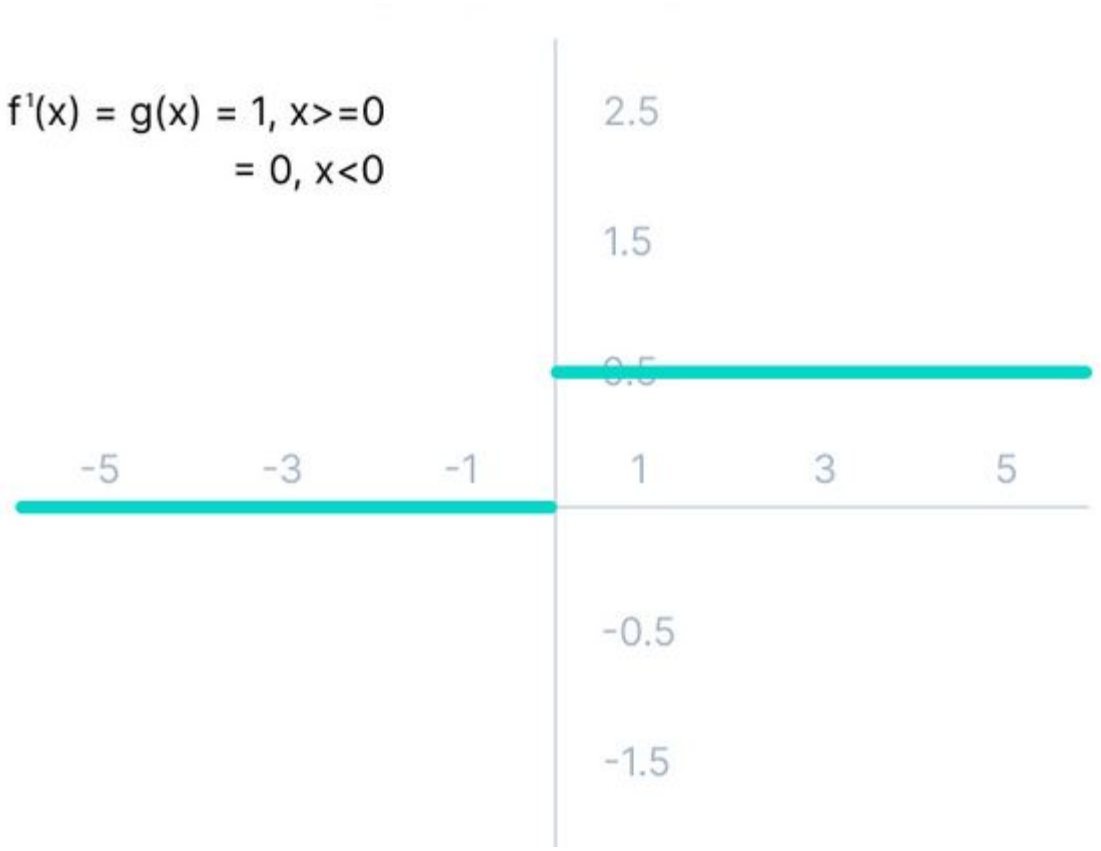


$$f(x) = max\,(0, x)$$

- Hence optimization is *easier* in this method hence in practice it is always preferred over Sigmoid function .

- It also faces the problem of vanishing gradients similar to the sigmoid activation function. Plus the gradient of the tanh function is much steeper as compared to the sigmoid function.

Dying Relu Problem



- The negative side of the graph makes the gradient value zero. Due to this reason, during the back propagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated.

- All the negative input values become zero immediately, which decreases the model's ability to fit or train from the data properly.

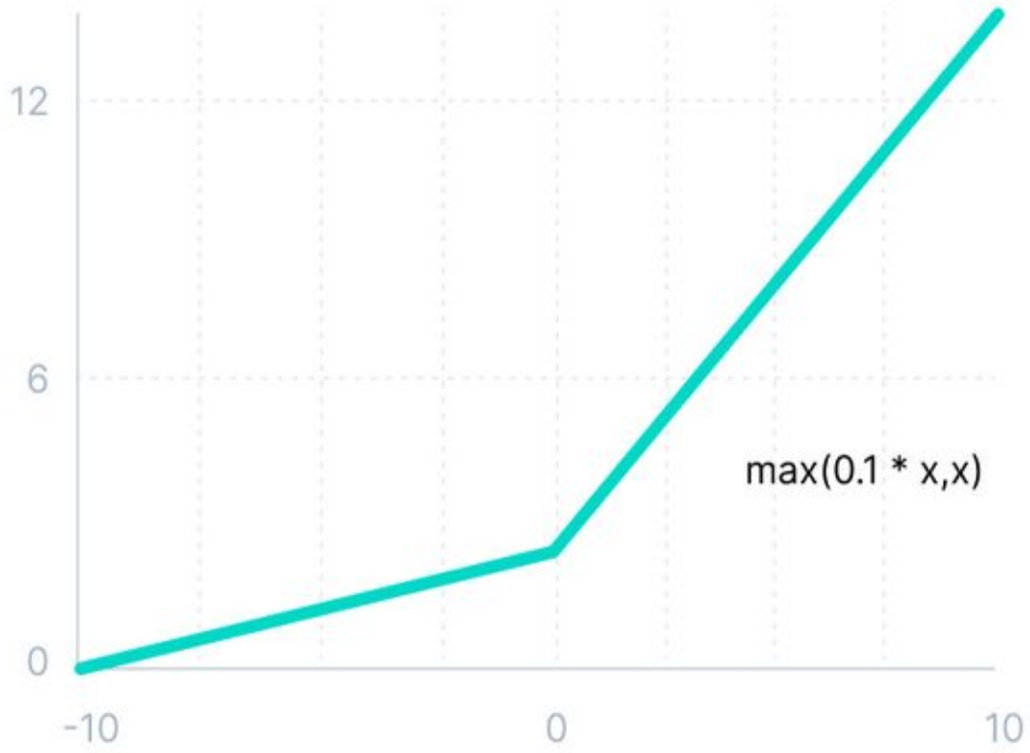
- To fix this problem another modification was introduced called ***Leaky ReLu*** to fix the problem of dying neurons. It introduces a small slope to keep the updates alive.

Leaky ReLU Function

- Leaky ReLU is an improved version of ReLU function to solve the Dying ReLU problem as it has a small positive slope in the negative area.

- The advantages of Leaky ReLU are same as that of ReLU, in addition to the fact that it does enable back propagation, even for negative input values.

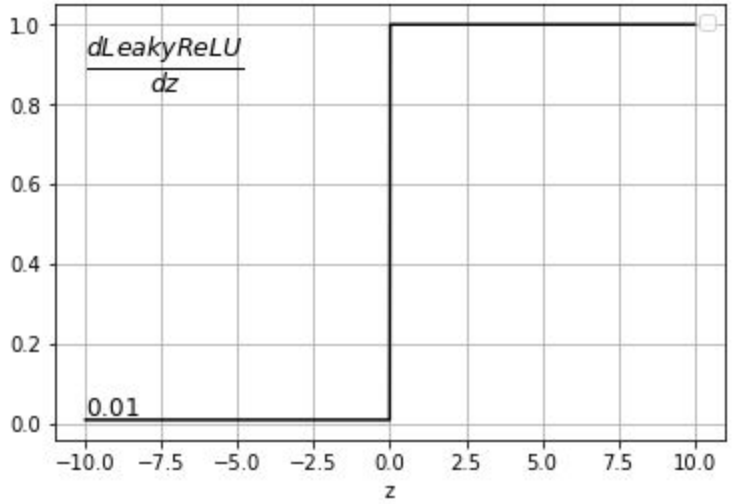
- By making this minor modification for negative input values, the gradient of the left side of the graph comes out to be a non-zero value. Therefore, we would no longer encounter dead neurons in that region.



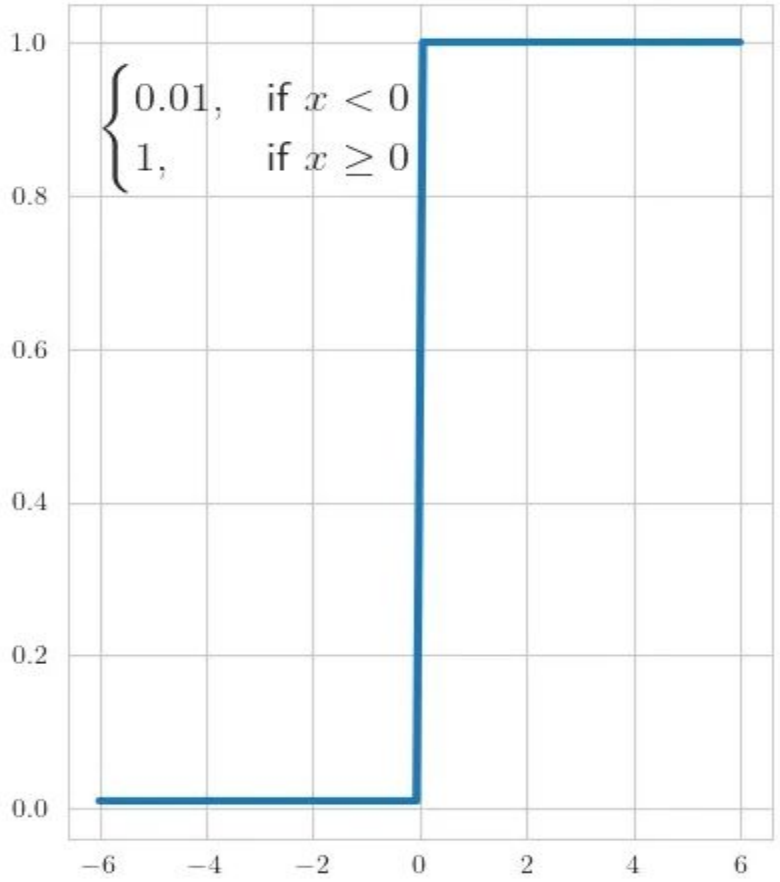
$$f(x) = max\ (0.1x, x)$$

- The advantages of Leaky ReLU are same as that of ReLU, in addition to the fact that it does enable backpropagation, even for negative input values.

- By making this minor modification for negative input values, the gradient of the left side of the graph comes out to be a non-zero value. Therefore, we would no longer encounter dead neurons in that region.



Derivative of Leaky ReLU Function



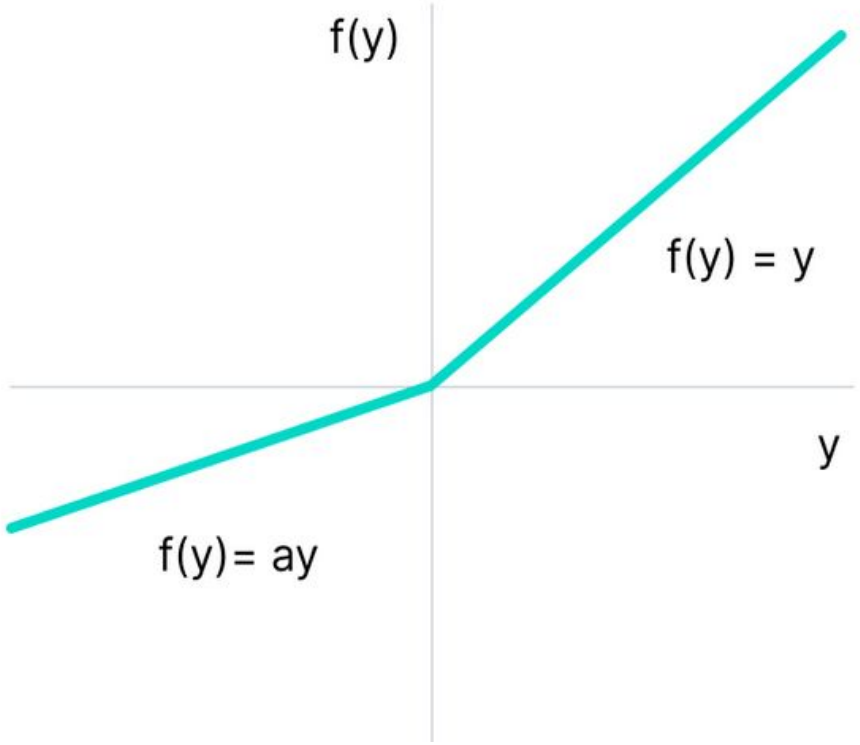
Parametric ReLU Function

- Parametric ReLU is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis.

- This function provides the slope of the negative part of the function as an argument *a*. By performing backpropagation, the most appropriate value of *a* is learnt.

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

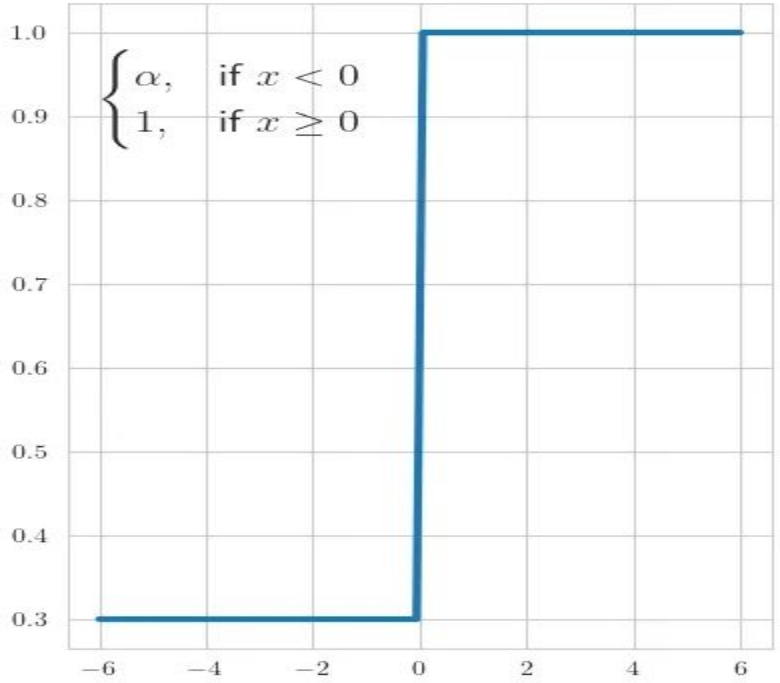
Where "*a*" is the slope parameter for negative values.



The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.

This function's limitation is that it may perform differently for different problems depending upon the value of slope parameter ***a.***

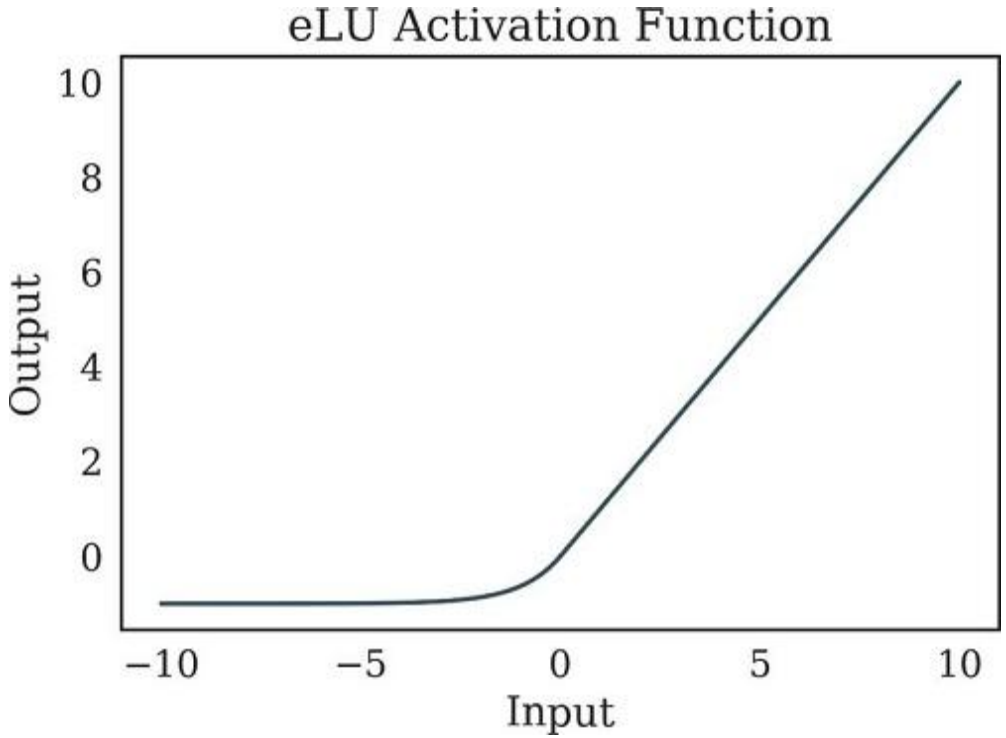
Derivative



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

Exponential Linear Units(ELUs)

- Exponential Linear Unit was proposed by Clevert in 2015.
- As it decreases bias shifts by pushing mean activation towards zero during training, ELU represents a good alternative to the ReLU.
- A limitation of the ELU is that the ELU does not center the values at zero.

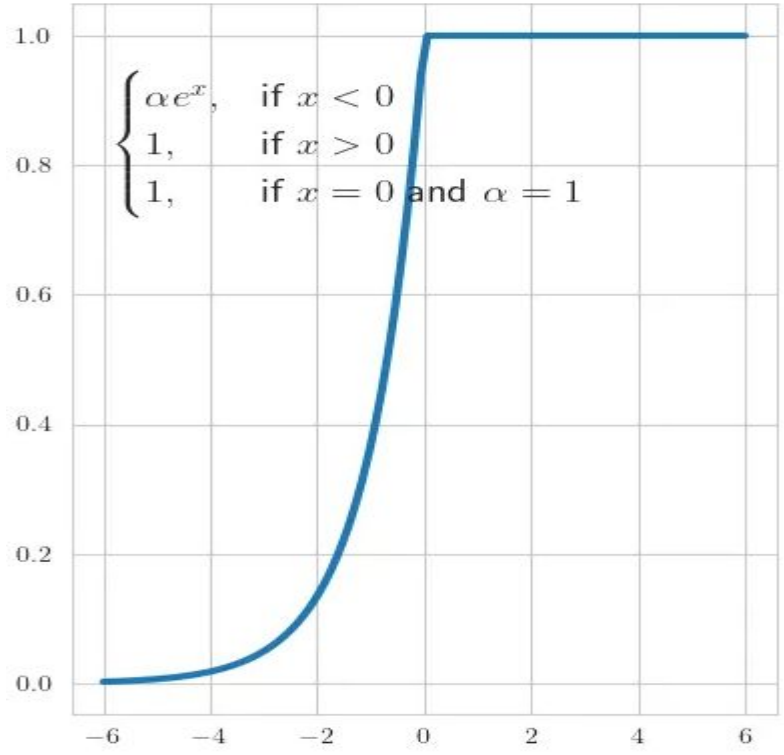


$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha\left(e^{x}-1\right) & for\ x < 0 \end{cases}$$

ELU is a strong alternative for f ReLU because of the following advantages:

- ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.
- Avoids dead ReLU problem by introducing log curve for negative values of input. It helps the network nudge weights and biases in the right direction.

Derivative



$$f'(x) = \begin{cases} \alpha e^x, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \\ 1, & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$$

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.

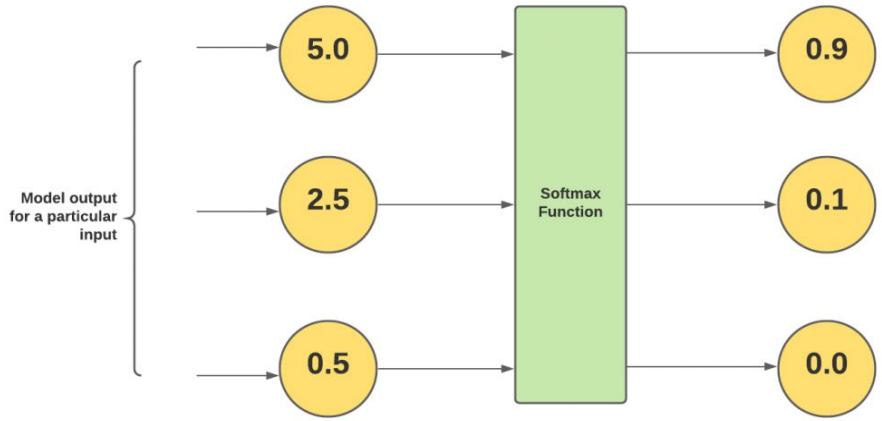
- It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.

- the Softmax function is described as a combination of multiple sigmoids.

- It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class.

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

Let's understand this with an example. Let's say the models (such as those trained using algorithms such as multi-class LDA, and multinomial logistic regression) output three different values such as 5.0, 2.5, and 0.5 for a particular input. In order to convert these numbers into probabilities, these numbers are fed into the ure.softmax function as shown in fig.



- Notice that the softmax outputs are less than 1. And, the outputs of the softmax function sum up to 1. Owing to this property, the Softmax function is considered an activation function in neural networks and algorithms such as multinomial logistic regression. Note that for binary logistic regression, the activation function used is the sigmoid function.

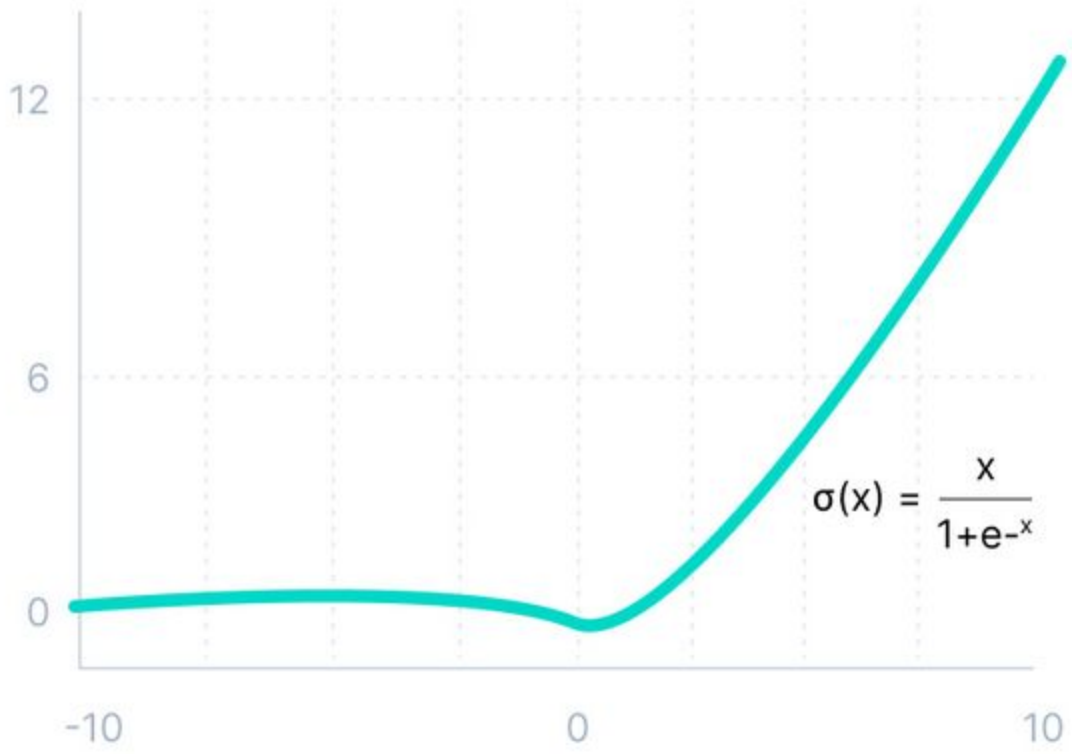
- Based on the above, it could be understood that the output of the softmax function maps to a [0, 1] range. And, it maps outputs in a way that the total sum of all the output values is 1. Thus, it could be said that the output of the softmax function is a **probability distribution**.

Swish Activation Function

- The *Swish activation function* is one of the first compound *activation function* proposed by the combination of the *sigmoid activation function* and the input function, to achieve a *hybrid AF*. The properties of the Swish function include *smoothness, non-monotonic, bounded* below, and unbounded in the upper limits.

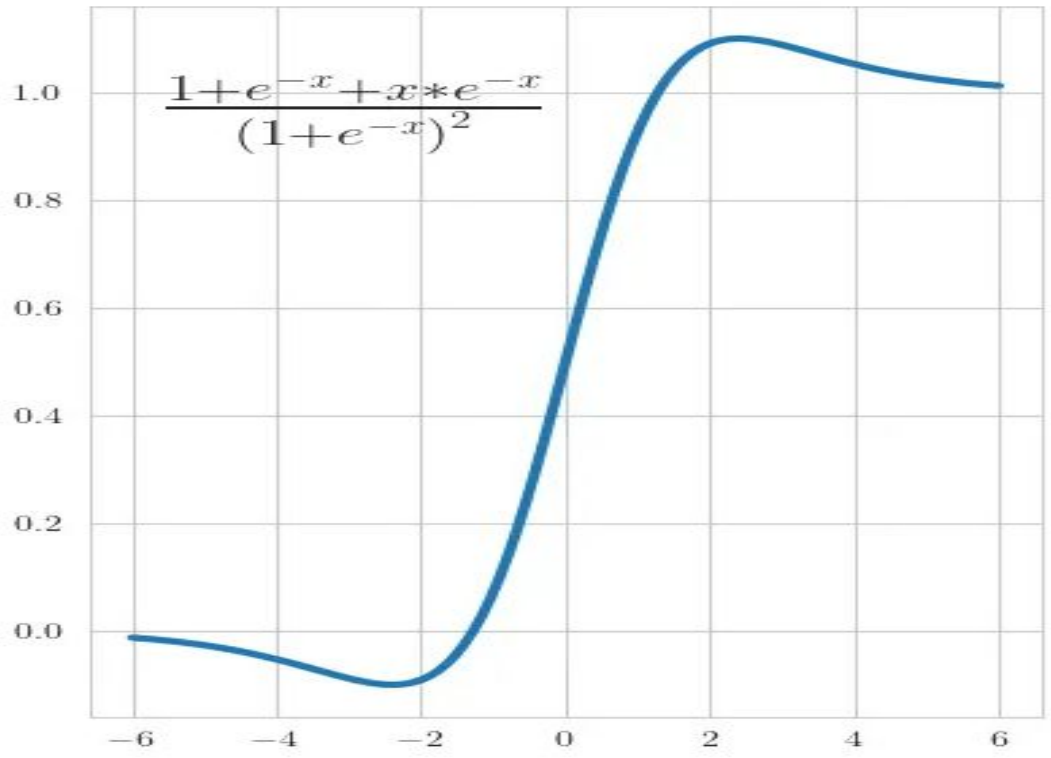
- developed by researchers at Google.

- Swish consistently matches or outperforms ReLU activation function on deep networks applied to various challenging domains such as image classification, machine translation etc.



$$f(x) = x * sigmoid(x)$$

Derivative

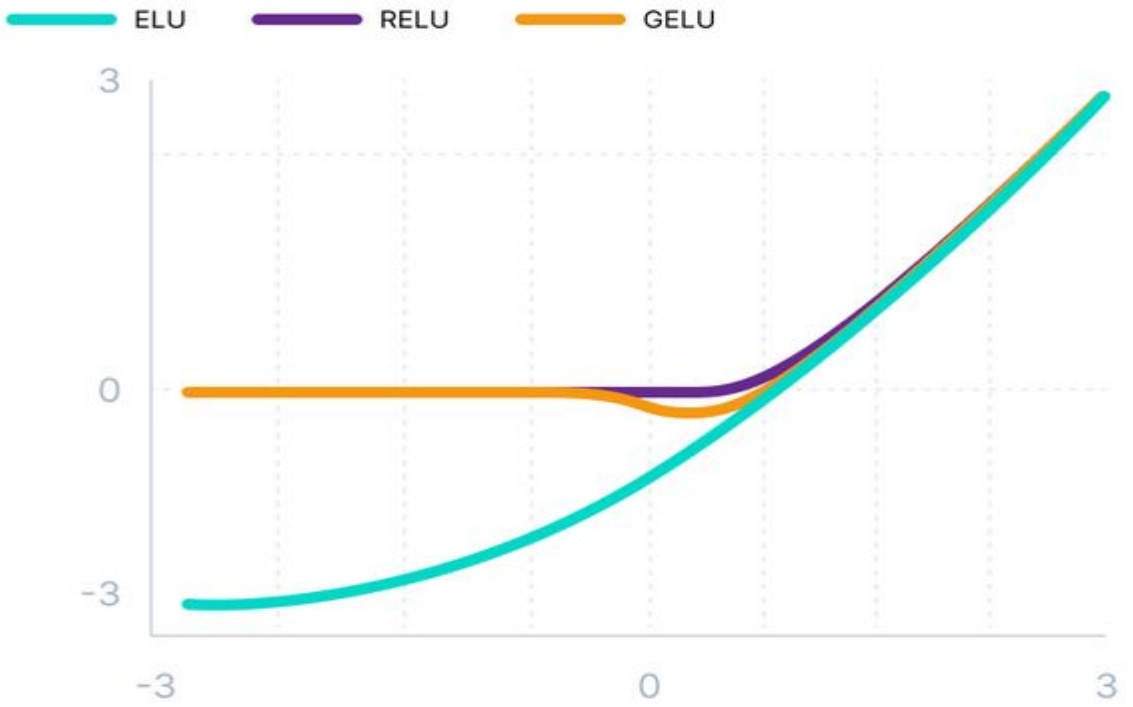


Gaussian Error Linear Unit (GELU)

- The Gaussian Error Linear Unit (GELU) activation function is compatible with BERT, ROBERTa, ALBERT, and other top NLP models. This activation function is motivated by combining properties from dropout, zoneout, and ReLUs.

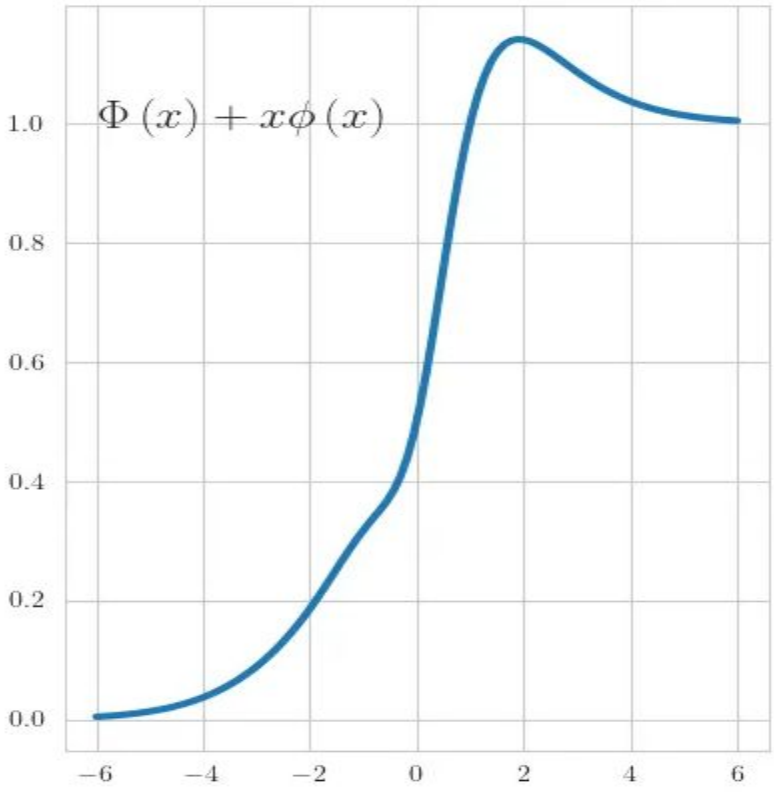
- ReLU and dropout together yield a neuron's output. ReLU does it deterministically by multiplying the input by zero or one (depending upon the input value being positive or negative) and dropout stochastically multiplying by zero.

- RNN regularizer called zoneout stochastically multiplies inputs by one.



$$f(x) = xP(X \leq x) = x\Phi(x)$$
$$= 0.5x\left(1 + tanh\left[\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right]\right)$$

Derivative



Scaled Exponential Linear Unit (SELU)

- SELU was defined in self-normalizing networks and takes care of internal normalization which means each layer preserves the mean and variance from the previous layers. SELU enables this normalization by adjusting the mean and variance.

- SELU has both positive and negative values to shift the mean, which was impossible for ReLU activation function as it cannot output negative values.

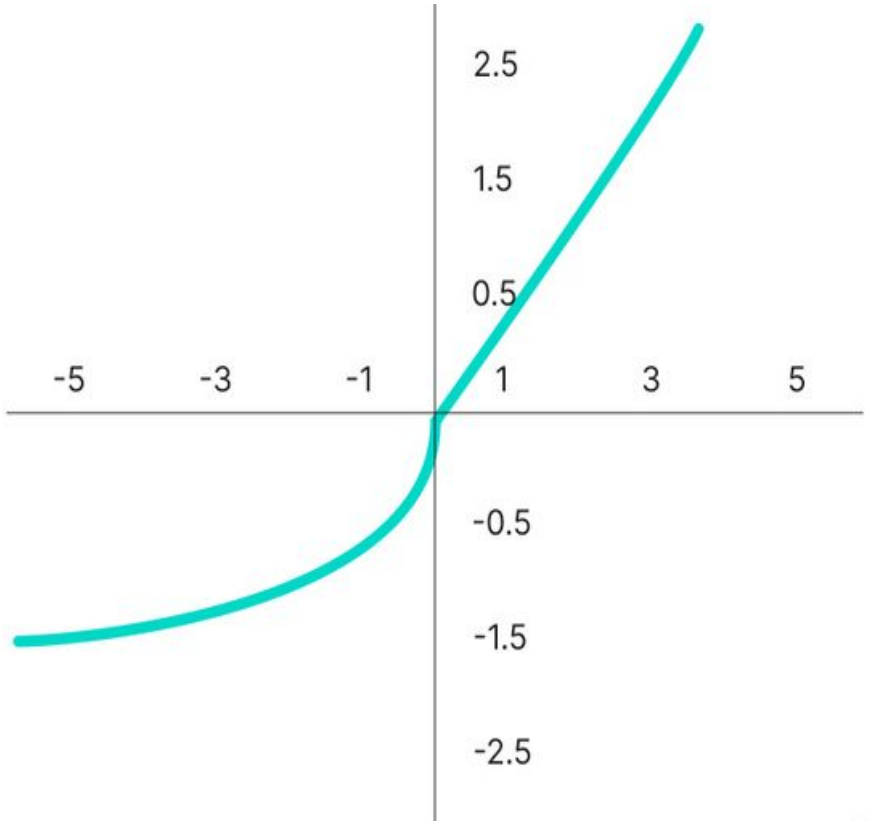
- Gradients can be used to adjust the variance. The activation function needs a region with a gradient larger than one to increase it.

- SELU has values of alpha α and lambda λ predefined.

Here's the main advantage of SELU over ReLU:

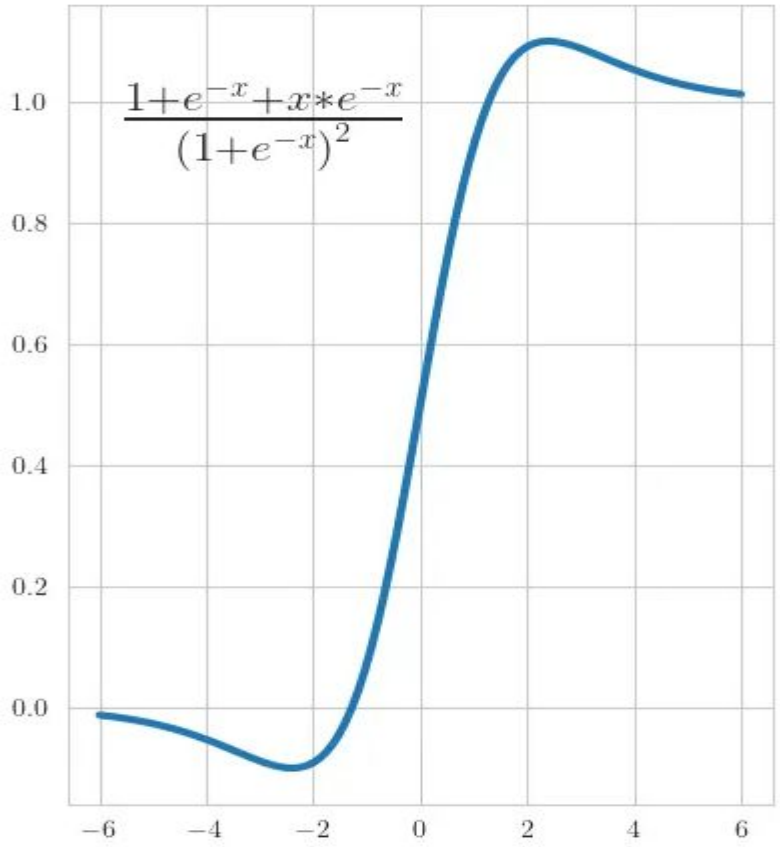
- Internal normalization is faster than external normalization, which means the network converges faster.

- SELU is a relatively newer activation function and needs more papers on architectures such as CNNs and RNNs, where it is comparatively explored.



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

Derivative



Summary

- Activation Functions are used to introduce non-linearity in the network.
- A neural network will almost always have the same activation function in all hidden layers. This activation function should be differentiable so that the parameters of the network are learned in backpropagation.
- ReLU is the most commonly used activation function for hidden layers.

- While selecting an activation function, you must consider the problems it might face: vanishing and exploding gradients.

- Regarding the output layer, we must always consider the expected value range of the predictions. If it can be any numeric value (as in case of the regression problem) you can use the linear activation function or ReLU.

- Use Softmax or Sigmoid function for the classification problems.