# Activation Function

## What is a neural network activation function and how does it work?

- The segregation plays a key role in helping a neural network properly function, ensuring that it learns from the useful information rather than get stuck analyzing the not-useful part.
- And this is also where activation functions come into the picture.
- Activation Function helps the neural network to use important information while suppressing irrelevant data points.

## What is a Neural Network Activation Function?

**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 role of the Activation Function is to derive output from a set of input values fed to a node (or a layer).

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.

## Elements of a Neural Networks Architecture

neural network made of interconnected neurons. Each of them is characterized by its **weight**, **bias**, and **activation function**.

Highlight
All hidden layers usually use the same activation function. However, the output layer will typically use a different activation function from the hidden layers. The choice depends on the goal or type of prediction made by the model.

Highlight
**Feedforward Propagation** - the flow of information occurs in the forward direction. The input is used to calculate some intermediate function in the hidden layer, which is then used to calculate an output.

In the feedforward propagation, the Activation Function is a mathematical "gate" in between the input feeding the current neuron and its output going to the next layer.

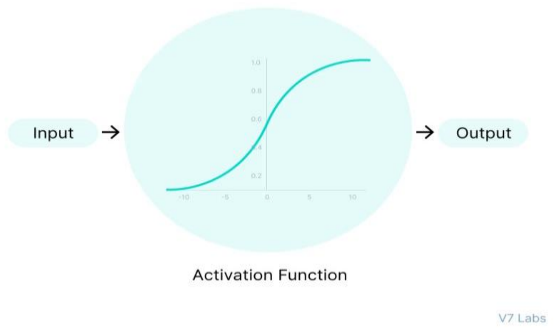
Highlight

**Backpropagation** - the weights of the network connections are repeatedly adjusted to minimize the difference between the actual output vector of the net and the desired output vector.

To put it simply—backpropagation aims to minimize the cost function by adjusting the network's weights and biases. The cost function gradients determine the level of adjustment with respect to parameters like activation function, weights, bias, etc.

## Why do Neural Networks Need an Activation Function?

Well, the purpose of an activation function is to **add non-linearity** to the neural network.



- Activation functions introduce an additional step at each layer during the forward propagation, but its computation is worth it. Here is why

—

Let's suppose we have a neural network working **without** the activation functions.

In that case, 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.

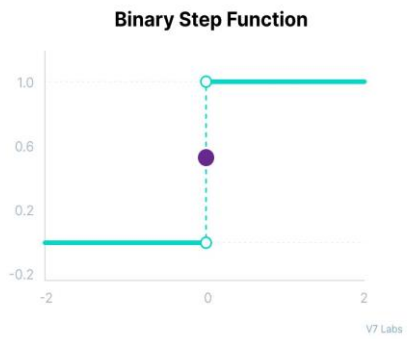
Although the neural network becomes simpler, learning any complex task is impossible, and our model would be just a linear regression model.

## 3 Types of Neural Networks Activation Functions :

1. Binary Step Function.
2. Linear Activation Function.
3. Non-Linear Activation Function (10 types)

### ➔ <u>Binary Step Function –</u>

- 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.



Mathematically it can be represented as:

$$\textit{Binary step}$$

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

Here are some of the limitations of binary step function:
- It cannot provide multi-value outputs—for example, it cannot be used for multi-class classification problems.
- The gradient of the step function is zero, which causes a hindrance in the backpropagation process.

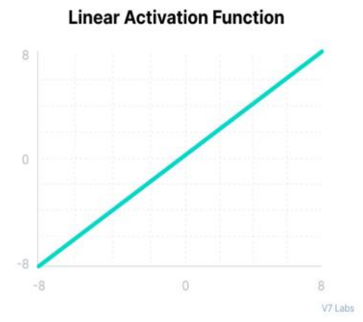
## ➔ <u>Linear Activation Function –</u>

- The linear activation function, also known as "no activation," or "identity function" (multiplied x1.0), is where the activation is proportional to the input.
- The function doesn't do anything to the weighted sum of the input, it simply spits out the value it was given.

Mathematically it can be represented as:

*Linear*

$$f(x) = x$$



However, a linear activation function has two major problems :
- It's not possible to use backpropagation as the derivative of the function is a constant and has no relation to the input x.
- All layers of the neural network will collapse into one if a linear activation function is used. No matter the number of layers in the neural network, the last layer will still be a linear function of the first layer. So, essentially, a linear activation function turns the neural network into just one layer.

## ➔ <u>Non-Linear Activation Functions : -</u>

The linear activation function shown above is simply a linear regression model.
Because of its limited power, this does not allow the model to create complex mappings between the network's inputs and outputs.

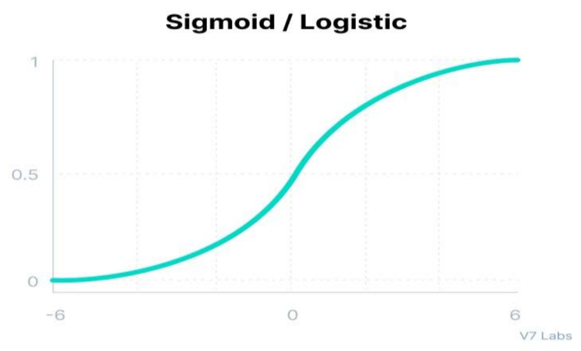
Non-linear activation functions solve the following limitations of linear activation functions:
- They allow backpropagation because now the derivative function would be related to the input, and it's possible to go back and understand which weights in the input neurons can provide a better prediction.
- They allow the stacking of multiple layers of neurons as the output would now be a non-linear combination of input passed through multiple layers. Any output can be represented as a functional computation in a neural network.

Now, let's have a look at ten different non-linear neural networks activation functions and their characteristics.

## 10 Non-Linear Neural Networks Activation Functions

## ➔ 1. Sigmoid / Logistic Activation Function :

- This function takes any real value as input and outputs values in the range of 0 to 1.
- 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 0.0, as shown below.



*Sigmoid / Logistic*

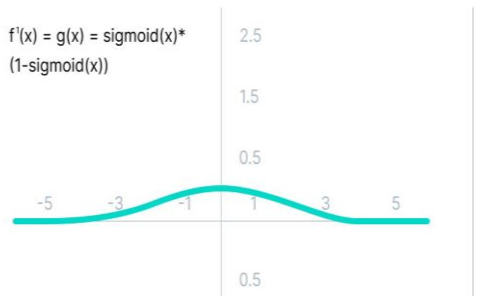
$$f(x) = \frac{1}{1 + e^{-x}}$$

Here’s why sigmoid/logistic activation function is one of the most widely used functions:

- 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.

The limitations of sigmoid function are discussed below:

- The derivative of the function is f'(x) = sigmoid(x)*(1-sigmoid(x)).

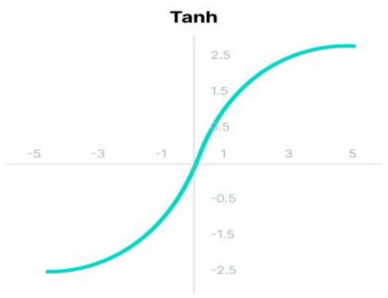


As we can see from the above Figure, the gradient values are only significant for range -3 to 3, and the graph gets much flatter in other regions.
It implies that for values greater than 3 or less than -3, the function will have very small gradients. As the gradient value approaches zero, the network ceases to learn and suffers from the *Vanishing gradient* problem.

- The output of the logistic function is not symmetric around zero. So the output of all the neurons will be of the same sign. This makes the training of the neural network more difficult and unstable.

## ➔ 2. 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.



*Tanh*

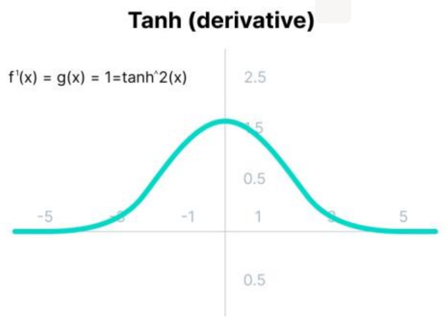
$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

Mathematically it can be represented as:

Advantages of using this activation function are:

- The output of the tanh activation function is Zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers of a neural network as its values lie between -1 to; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.

Have a look at the gradient of the tanh activation function to understand its limitations.

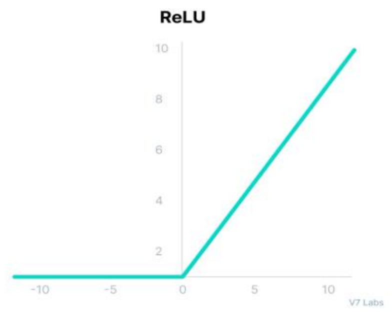


As you can see— it also faces the problem of vanishing gradients similar to the sigmoid activation function.
Plus the gradient of the tanh function is much steeper as compared to the sigmoid function.

○ Highlight
**Note:** Although both sigmoid and tanh face vanishing gradient issue, tanh is zero centered, and the gradients are not restricted to move in a certain direction. Therefore, in practice, tanh nonlinearity is always preferred to sigmoid nonlinearity.

## ➔ 3. ReLU Function

- ReLU stands for Rectified Linear Unit.
- Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.
- The main catch here is that the ReLU function does not activate all the neurons at the same time.
- The neurons will only be deactivated if the output of the linear transformation is less than 0.



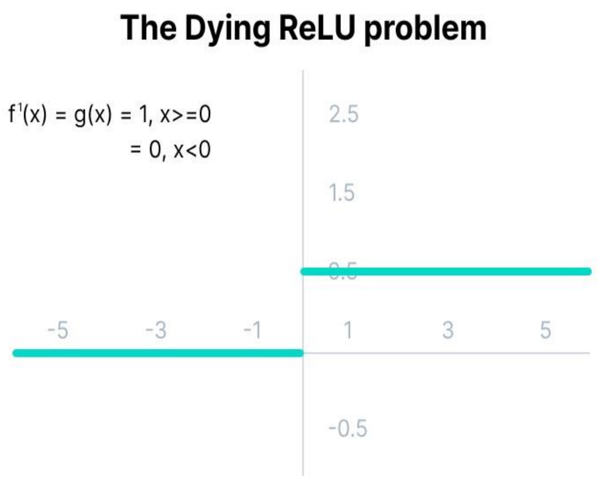
*ReLU*

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

Mathematically it can be represented as:

The advantages of using ReLU as an activation function are as follows:
- 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.

The limitations faced by this function are:
- The Dying ReLU problem, which I explained below.

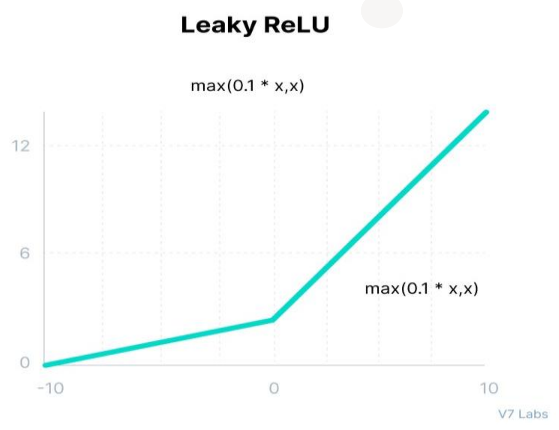


The negative side of the graph makes the gradient value zero. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated.
- All the negative input values become zero immediately, which decreases the model's ability to fit or train from the data properly.

**Note:** For building the most reliable ML models, split your data into train, validation, and test sets.

## ➔ 4. <u>Leaky ReLU Function:</u>

- 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.



*Leaky ReLU*

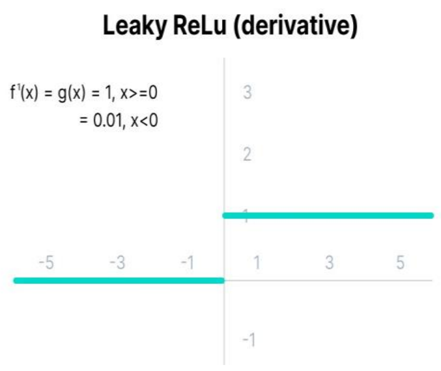
$$f(x) = max\ (0.1x, x)$$

Mathematically it can be represented as:

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.

Here is the derivative of the Leaky ReLU function.

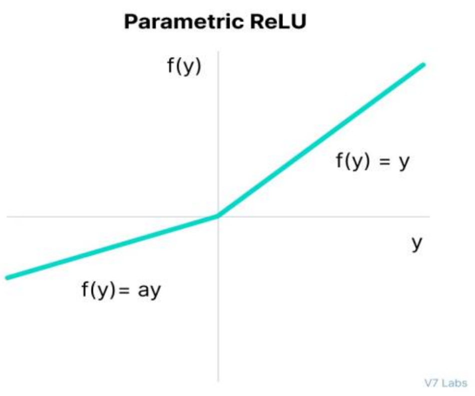


The limitations that this function faces include:

- The predictions may not be consistent for negative input values.
- The gradient for negative values is a small value that makes the learning of model parameters time-consuming.

## ➔ 5. 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.



Mathematically it can be represented as:

*Parametric ReLU*

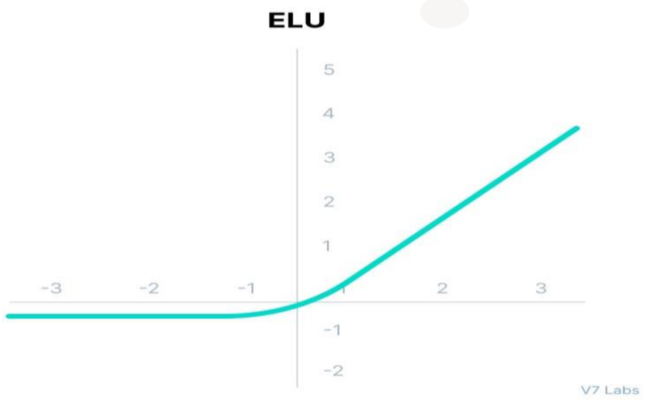
$$f(x) = max\ (ax, x)$$

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.**

## ➔ 6. Exponential Linear Units (ELUs) Function :

- Exponential Linear Unit, or ELU for short, is also a variant of ReLU that modifies the slope of the negative part of the function.
- ELU uses a log curve to define the negative values unlike the leaky ReLU and Parametric ReLU functions with a straight line.



ELU

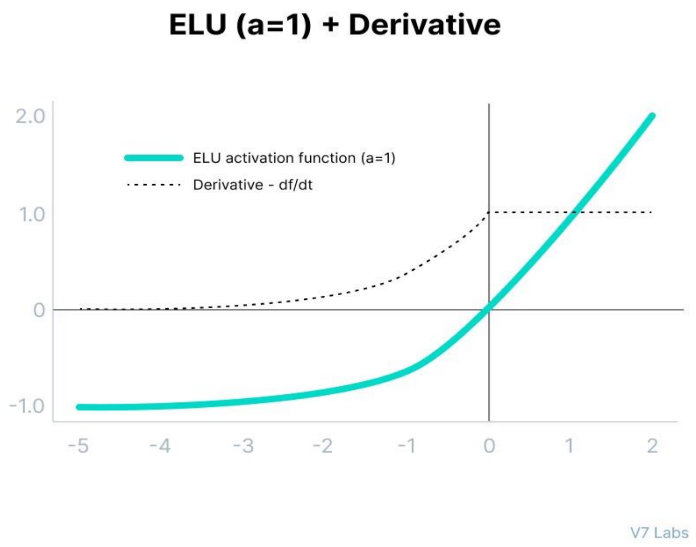
$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha(e^x - 1) & for\ x < 0 \end{cases}$$

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.

The limitations of the ELU function are as follow:

- It increases the computational time because of the exponential operation included
- No learning of the 'a' value takes place
- Exploding gradient problem

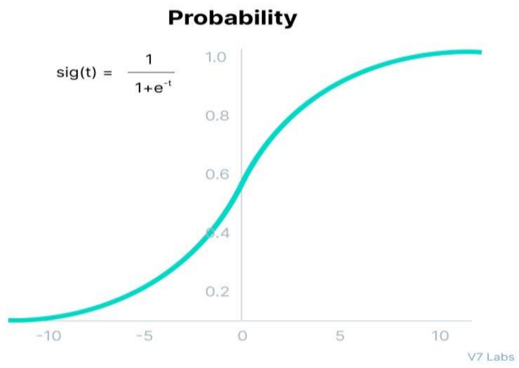


ELU derivative

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

## ➔ 7. Softmax Function :

- The **softmax activation function** is a mathematical function commonly used in the final layer of neural networks for classification tasks. It converts raw output scores (also called logits) into probabilities, where each output corresponds to a probability distribution over multiple classes. The values produced by the softmax function are between 0 and 1 and sum to 1, which makes them interpretable as probabilities.
- The output of the sigmoid function was in the range of 0 to 1, which can be thought of as probability.



**Softmax**

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

Where:
- eee is the base of the natural logarithm.
- ziz_izi is the iii-th element of the input vector.
- The denominator sums the exponentials of all input elements, ensuring the output is a valid probability distribution.

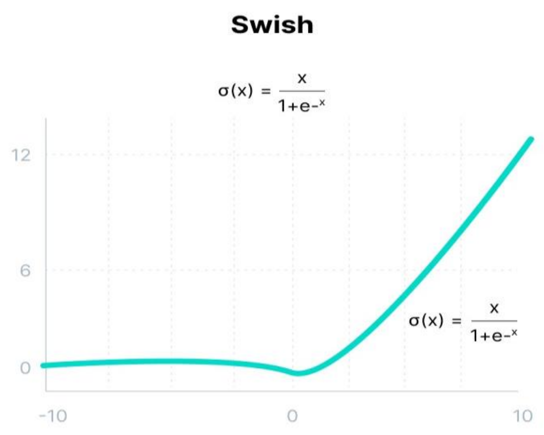
**Key Properties:**
- **Output Range**: The output values lie between 0 and 1.
- **Probabilities**: The sum of all output values equals 1, meaning they can be interpreted as probabilities for each class.
- **Differentiable**: This is crucial for backpropagation, allowing gradient-based optimization techniques (like gradient descent) to be applied.

**Usage in Neural Networks:**

Softmax is primarily used in:
- **Multiclass classification problems**, where the goal is to classify an input into one of several categories.
- **Probabilistic interpretation**, helping to make predictions and decisions based on the output probabilities.

## ➔ 8. Swish :

- It is a self-gated activation function 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.



This function is bounded below but unbounded above i.e. Y approaches to a constant value as X approaches negative infinity but Y approaches to infinity as X approaches infinity.

Mathematically it can be represented as:

$$\mathit{Swish}$$

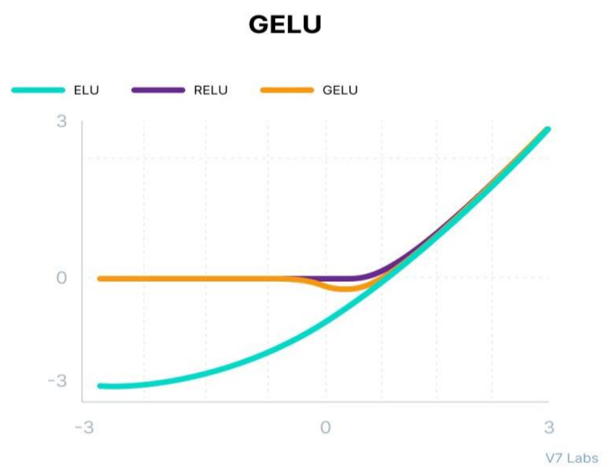
$$f(x) = x*sigmoid(x)$$

Here are a few advantages of the Swish activation function over ReLU:

- Swish is a smooth function that means that it does not abruptly change direction like ReLU does near x = 0. Rather, it smoothly bends from 0 towards values < 0 and then upwards again.
- Small negative values were zeroed out in ReLU activation function. However, those negative values may still be relevant for capturing patterns underlying the data. Large negative values are zeroed out for reasons of sparsity making it a win-win situation.
- The swish function being non-monotonous enhances the expression of input data and weight to be learnt.

## ➔ 9. Gaussian Error Linear Unit (GELU) :

The **GELU** (Gaussian Error Linear Unit) activation function is a smooth, differentiable activation function that combines the benefits of ReLU (Rectified Linear Unit) and the properties of Gaussian distributions. It was introduced to provide better performance in some deep learning models, especially for transformers like BERT and GPT, which rely on smooth activation functions.



GELU(x)=x·Φ(x)

Where:

- x is the input value.
- Φ(x)\Phi(x)Φ(x) is the cumulative distribution function (CDF) of the standard normal distribution. In simpler terms, it is the probability that a random variable from a standard normal distribution (mean 0, variance 1) is less than or equal to xxx.

**Key Characteristics of GELU:**

- **Smoothness**: Unlike ReLU, which has a sharp non-linearity, GELU is smooth and differentiable, helping with optimization.
- **Non-zero Negative Outputs**: Unlike ReLU (which outputs 0 for negative values), GELU can produce small negative values, which allows for better flow of gradients during backpropagation and might help with convergence.

**Behavior:**

- For **positive values** of xxx, GELU behaves similarly to xxx, allowing the gradient to pass through.
- For **negative values** of xxx, the function produces small negative values, unlike ReLU which simply zeros out the negative inputs.

**Visual Comparison (GELU vs ReLU):**

- **ReLU**: The output is max⁡(0,x)\max(0, x)max(0,x). For negative inputs, ReLU outputs 0.
- **GELU**: Outputs smoother values, even for negative inputs, and is less prone to "dead neurons" compared to ReLU.

**Applications:**

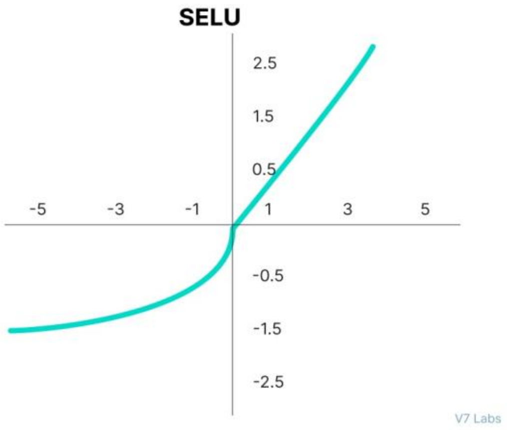
- **Transformers**: GELU is widely used in modern transformer models (like GPT, BERT, and others) because it allows for better learning and smoother gradients.
- **Deep Learning**: GELU can help reduce the likelihood of issues like vanishing gradients and dead neurons, making it useful in deep networks.

GELU is considered one of the state-of-the-art activation functions for deep learning, especially for models where smooth, non-linear activations are crucial for capturing complex patterns.

> GELU nonlinearity is better than ReLU and ELU activations and finds performance improvements across all tasks in domains of computer vision, natural language processing, and speech recognition.

## ➔ 10. 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.



Mathematically it can be represented as:

$$SELU$$

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

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.

## Why are deep neural networks hard to train?

There are two challenges you might encounter when training your deep neural networks.

**Vanishing Gradients**
Like the sigmoid function, certain activation functions squish an ample input space into a small output space between 0 and 1.

Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small. For shallow networks with only a few layers that use these activations, this isn’t a big problem.

However, when more layers are used, it can cause the gradient to be too small for training to work effectively.

**Exploding Gradients**
Exploding gradients are problems where significant error gradients accumulate and result in very large updates to neural network model weights during training.
An unstable network can result when there are exploding gradients, and the learning cannot be completed.
The values of the weights can also become so large as to overflow and result in something called NaN values.

## How to choose the right Activation Function?

You need to match your activation function for your output layer based on the type of prediction problem that you are solving—specifically, the type of predicted variable.
Here’s what you should keep in mind.

As a rule of thumb, you can begin with using the ReLU activation function and then move over to other activation functions if ReLU doesn’t provide optimum results.

And here are a few other guidelines to help you out.

1. ReLU activation function should only be used in the hidden layers.
2. Sigmoid/Logistic and Tanh functions should not be used in hidden layers as they make the model more susceptible to problems during training (due to vanishing gradients).
3. Swish function is used in neural networks having a depth greater than 40 layers.

Finally, a few rules for choosing the activation function for your output layer based on the type of prediction problem that you are solving:

1. **Regression** - Linear Activation Function
2. **Binary Classification**—Sigmoid/Logistic Activation Function
3. **Multiclass Classification**—Softmax
4. **Multilabel Classification**—Sigmoid
   The activation function used in hidden layers is typically chosen based on the type of neural network architecture.
5. **Convolutional Neural Network (CNN)**: ReLU activation function.
6. **Recurrent Neural Network**: Tanh and/or Sigmoid activation function.
   And hey—use this cheatsheet to consolidate all the knowledge on the Neural Network Activation Functions that you've just acquired :)