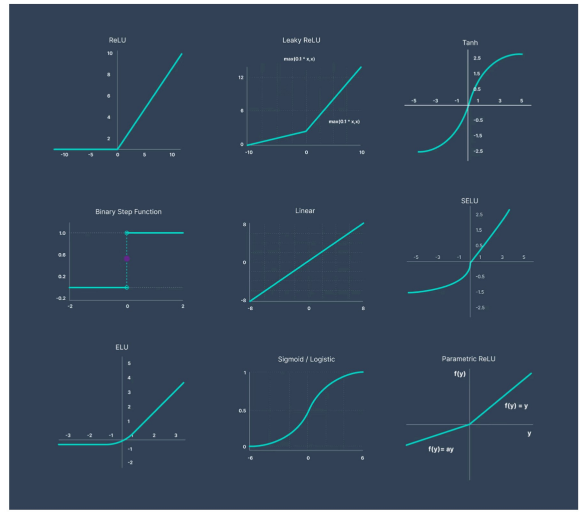
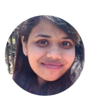
Computer vision

Activation Functions in Neural Networks [12 Types & Use Cases]

19 min read — May 27, 2021



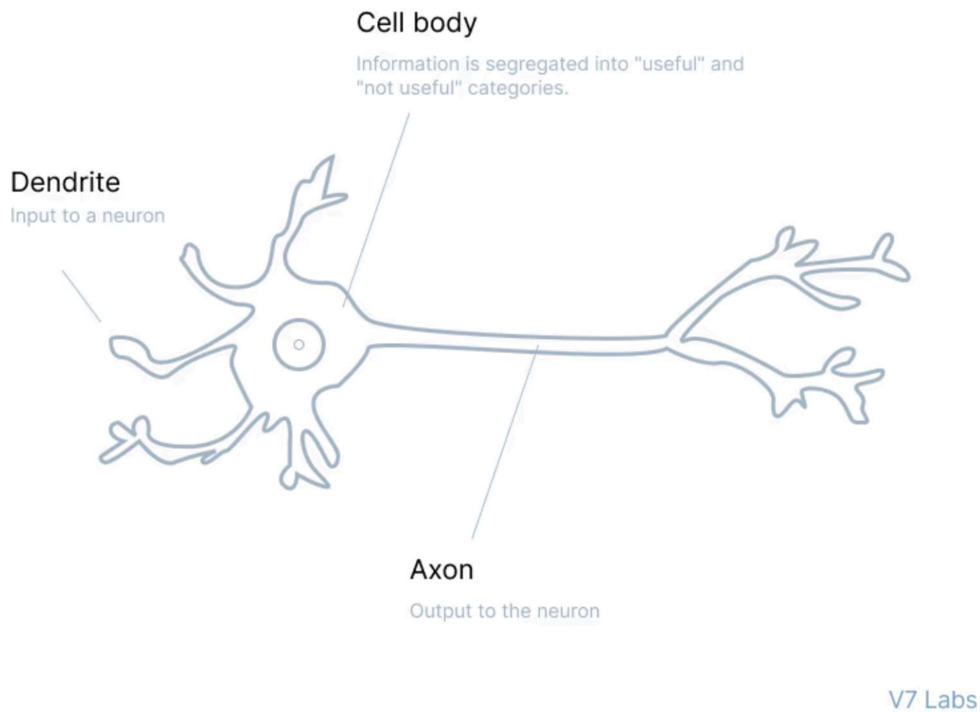
What is a neural network activation function and how does it work? Explore twelve different types of activation functions and learn how to pick the right one.



Pragati Baheti

“The world is one big data problem.”

As it turns out—



A similar process occurs in artificial neural network architectures in deep learning.

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.

Sounds a little confusi

- Types of Neural Networks Activation Functions
- 10 Non-Linear Neural Networks Activation Functions
- Why are Deep Neural Networks hard to train?
- How to choose the right Activation Function

Ready? Let's get started :)

What is a Neural Network Activation Function?

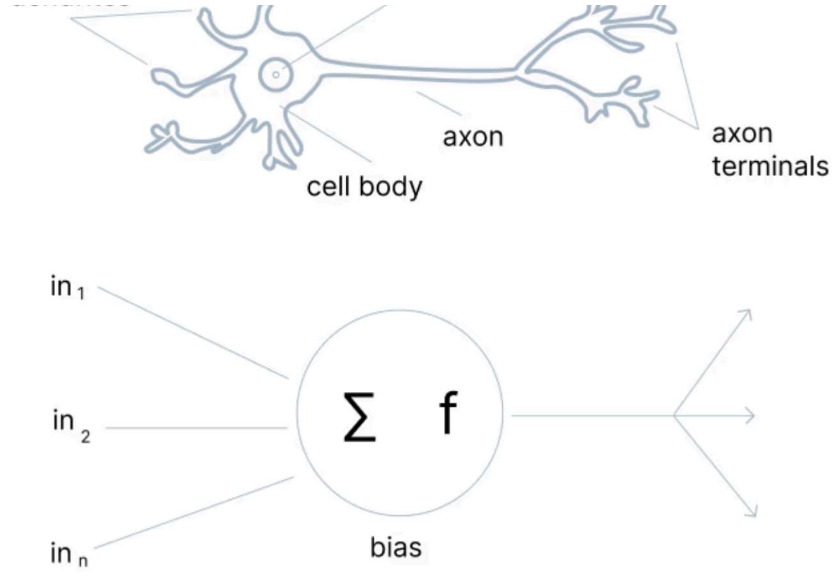
An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.

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

But—

Let's take a step back and clarify: What exactly is a **node**?

Well, if we compare the neural network to our brain, a node is a replica of a neuron that receiv

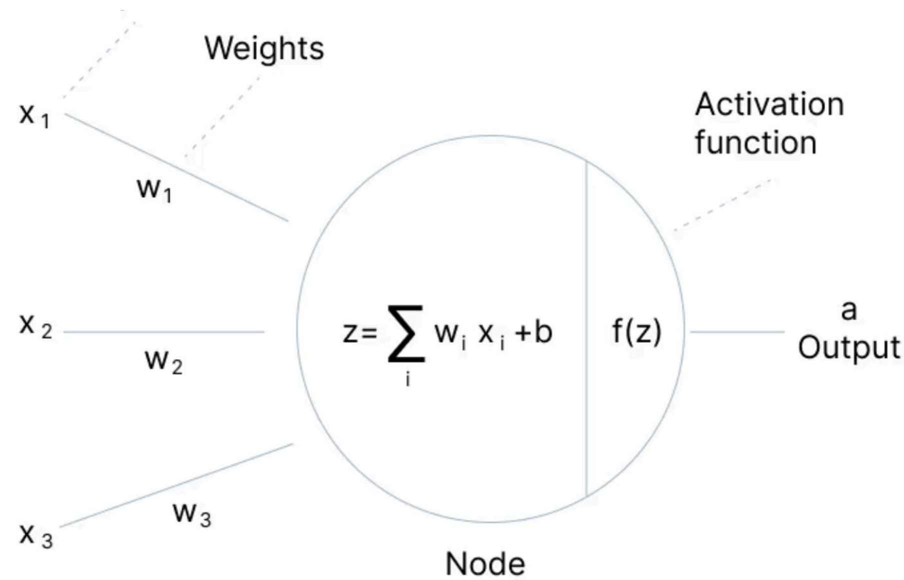


V7 Labs

Depending on the nature and intensity of these input signals, the brain processes them and decides whether the neuron should be activated ("fired") or not.

In deep learning, this is also the role of the Activation Function—that's why it's often referred to as a **Transfer Function** in Artificial Neural Network.

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.



V7 Labs

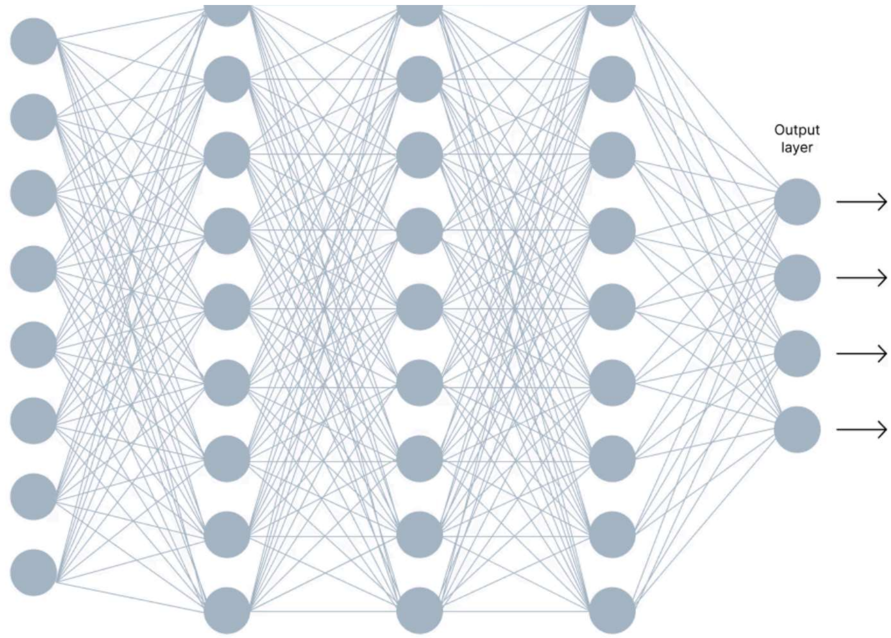
Now, let's have a look at the Neural Networks Architecture.

Elements of a Neural Networks Architecture

Here's the thing—

If you don't understand the concept of neural networks and how they work, diving deeper into the topic of activation functions might be challenging.

That's why it's a good idea to refresh your knowledge and take a quick look at the structure of the Neural Networks Architecture and its components. Here it is.



V7 Labs

In the image above, you can see a neural network made of interconnected neurons. Each of them is characterized by its **weight**, **bias**, and **activation function**.

Here are other elements of this network.

Input Layer

The input layer takes raw input from the domain. No computation is performed at this layer. Nodes here just pass on the information (features) to the hidden layer.

Hidden Layer

As the name suggests, the nodes of this layer are not exposed. They provide an abstraction

Output Layer

It's the final layer of the network that brings the information learned through the hidden layer and delivers the final value as a result.

> 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.

Feedforward vs. Backpropagation

When learning about neural networks, you will come across two essential terms describing the movement of information—feedforward and backpropagation.

Let's explore them.

> **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

V7 Labs

> **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.

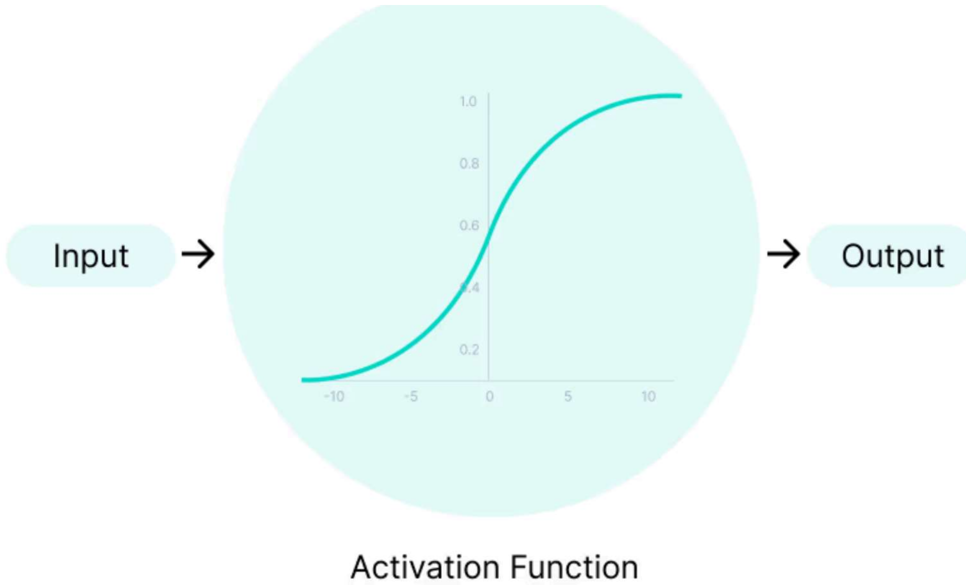
To put it simply—backpropagation aims to minimize the cost function by adjusting the network's weights and biases. The cost function gradients determine the level of adjustment with respect to parameters like activation function, weights, bias, etc.

Why do Neural Networks Need an Activation Function?

So we know what Activation Function is and what it does, but—

Why do Neural Networks need it?

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



V7 Labs

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.

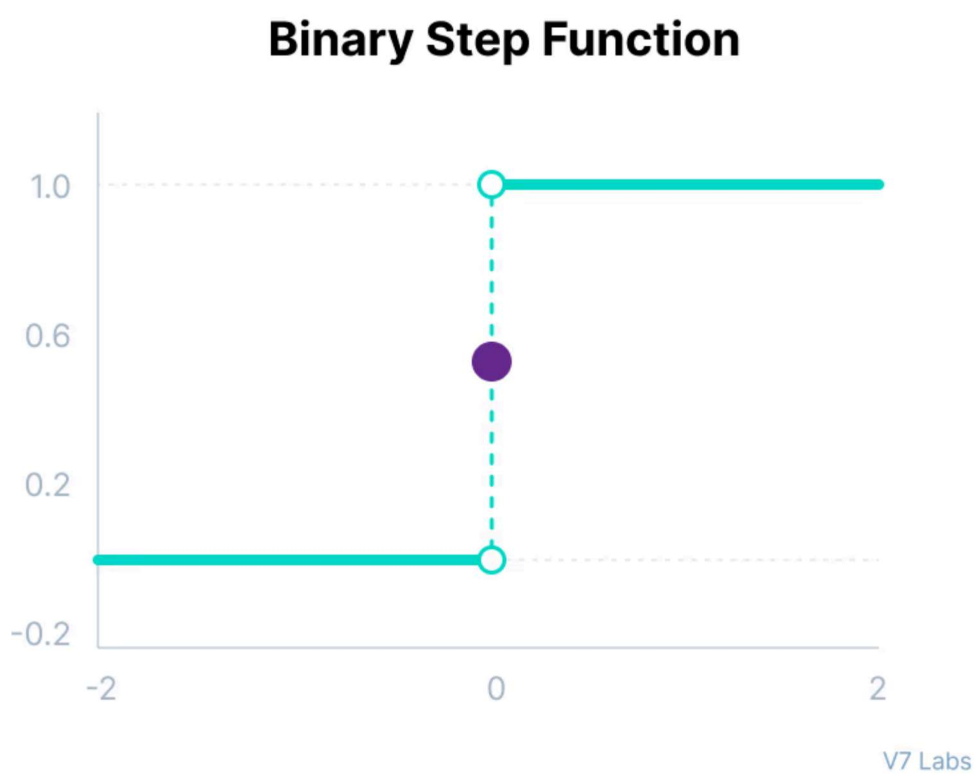
In that case, every neuron will only be performing a linear transformation on the inputs using the weights and biases. It's because it doesn't matter how many hidden layers we attach in the neural network; all layers will behave in the same way because the composition of two linear functions is a linear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

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

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.



Mathematically it can l

Binary step

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

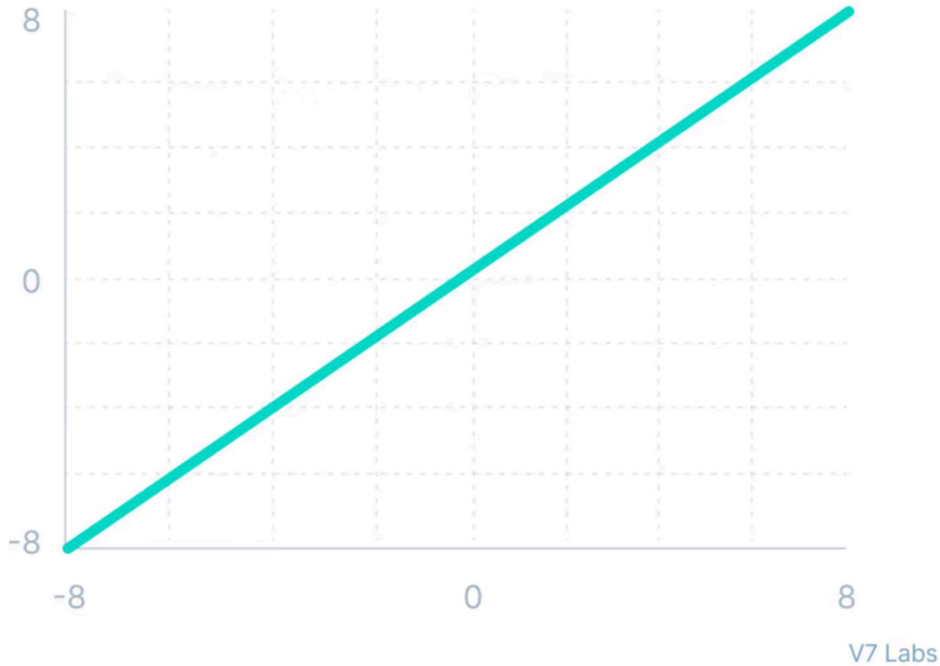
Here are some of the limitations of binary step function:

- 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.

Linear Activation Function

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 or

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.

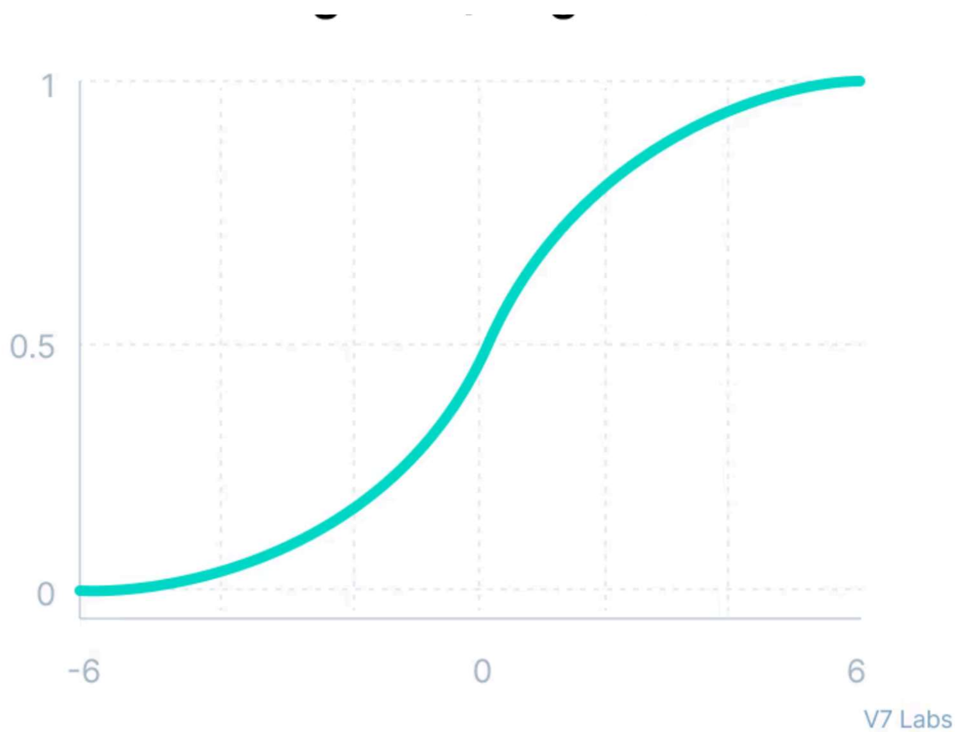
Now, let's have a look at ten different non-linear neural networks activation functions and their characteristics.

10 Non-Linear Neural Networks Activation Functions

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, a



Mathematically it can be represented as:

Sigmoid / Logistic

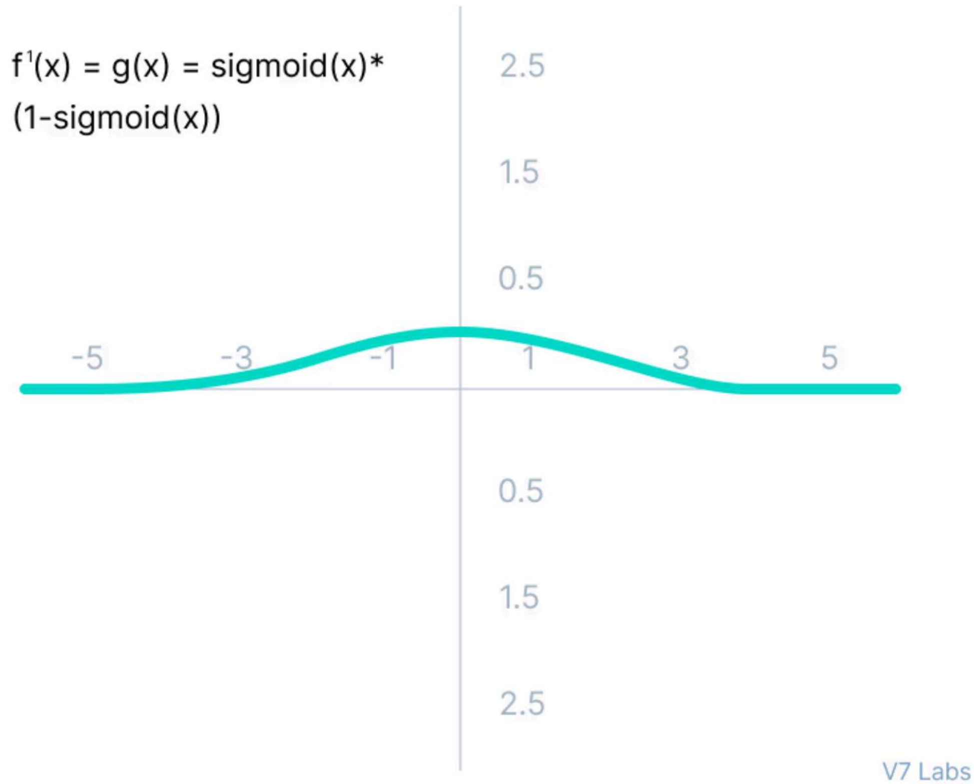
$$f(x) = \frac{1}{1 + e^{-x}}$$

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))



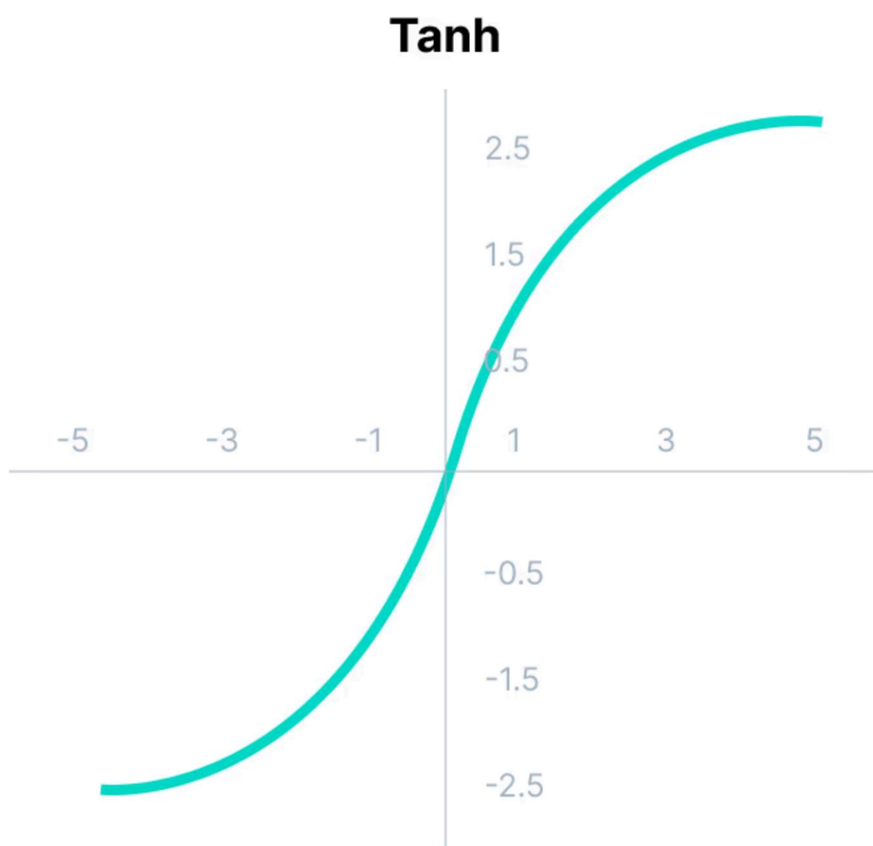
As we can see from the above Figure, the gradient values are only significant for range -3 to 3, and the graph gets much flatter in other regions.

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.

Tanh Function (Hy

-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.



Mathematically it can be represented as:

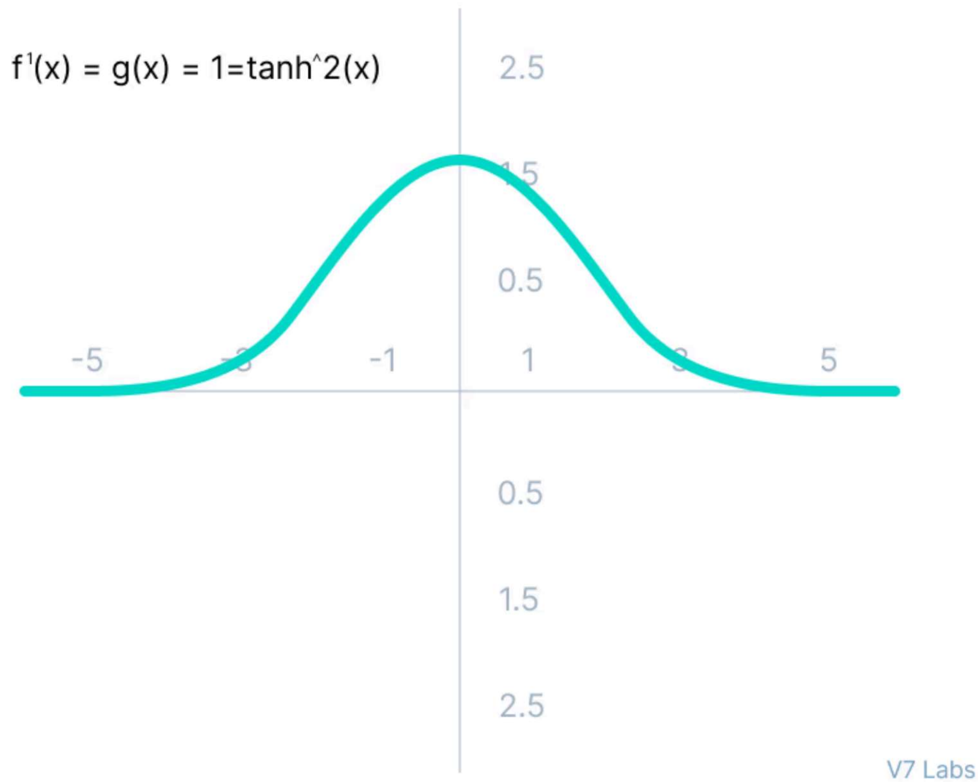
$$Tanh$$

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

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.

> **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.

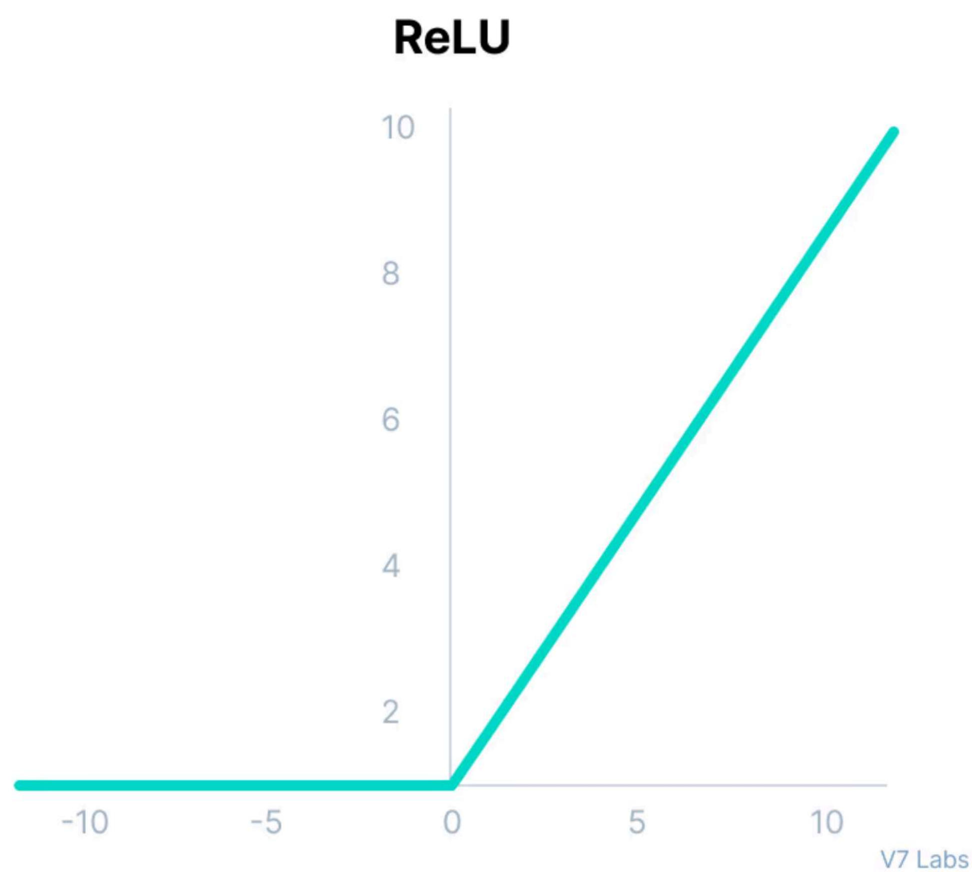
ReLU Function

ReLU stands for Rectif

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.



Mathematically it can be represented as:

$$ReLU$$

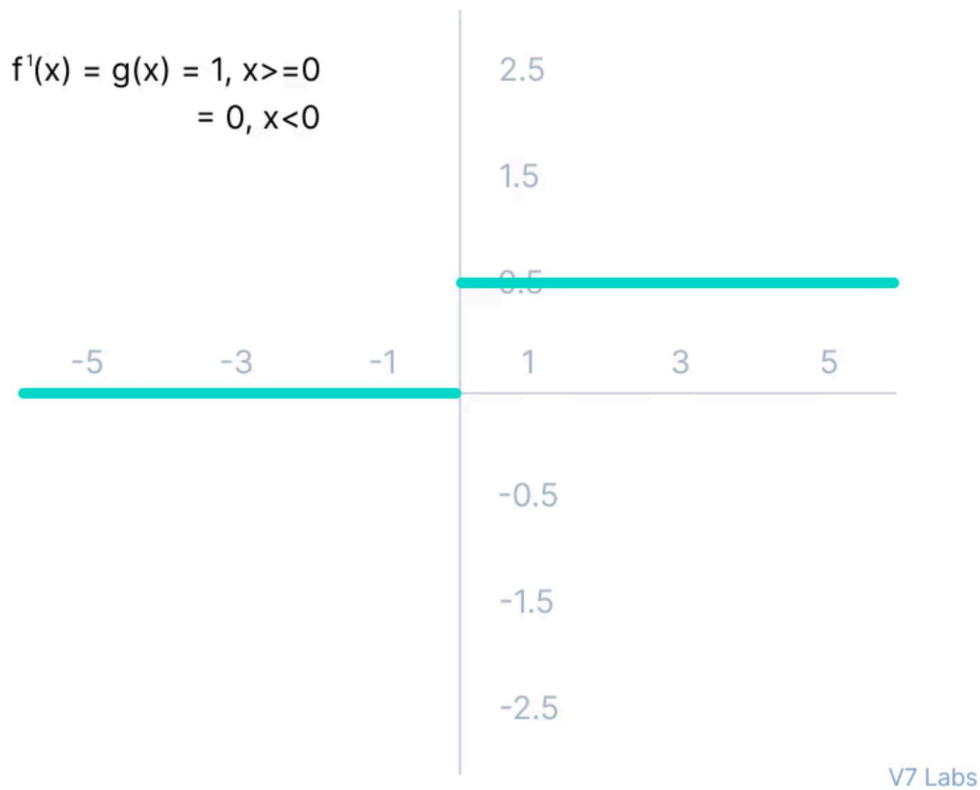
$$f(x) = max\ (0, x)$$

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.



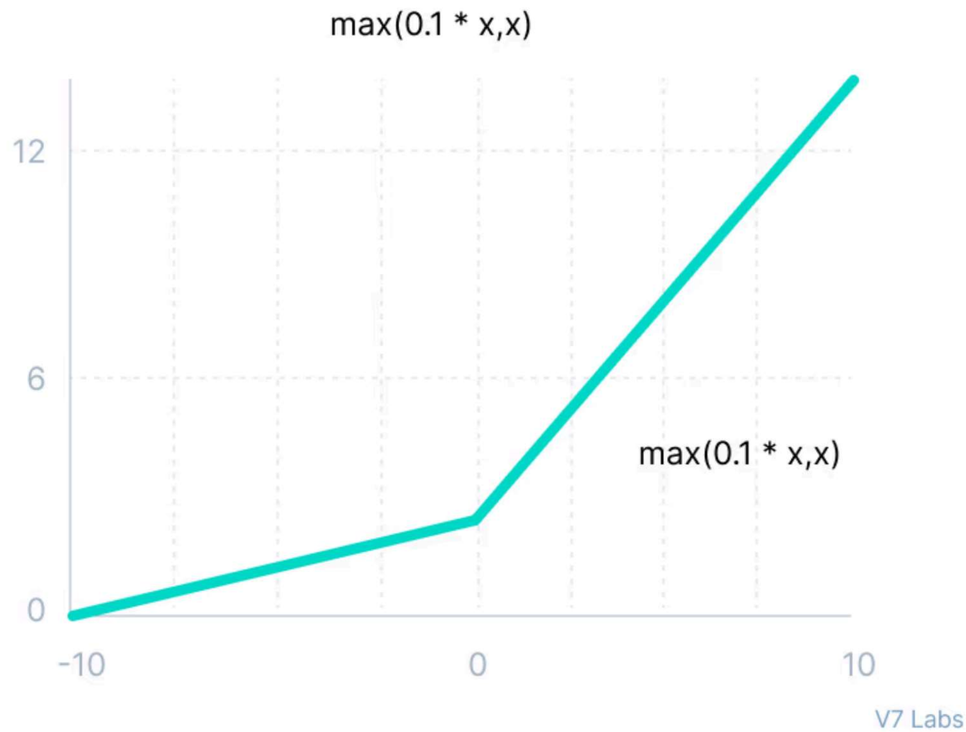
The negative side of the graph makes the gradient value zero. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create dead neurons which never get activated.

- 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.

Leaky ReLU Function

Leaky ReLU is an improved version of ReLU function to solve the Dying ReLU problem as it has



Mathematically it can be represented as:

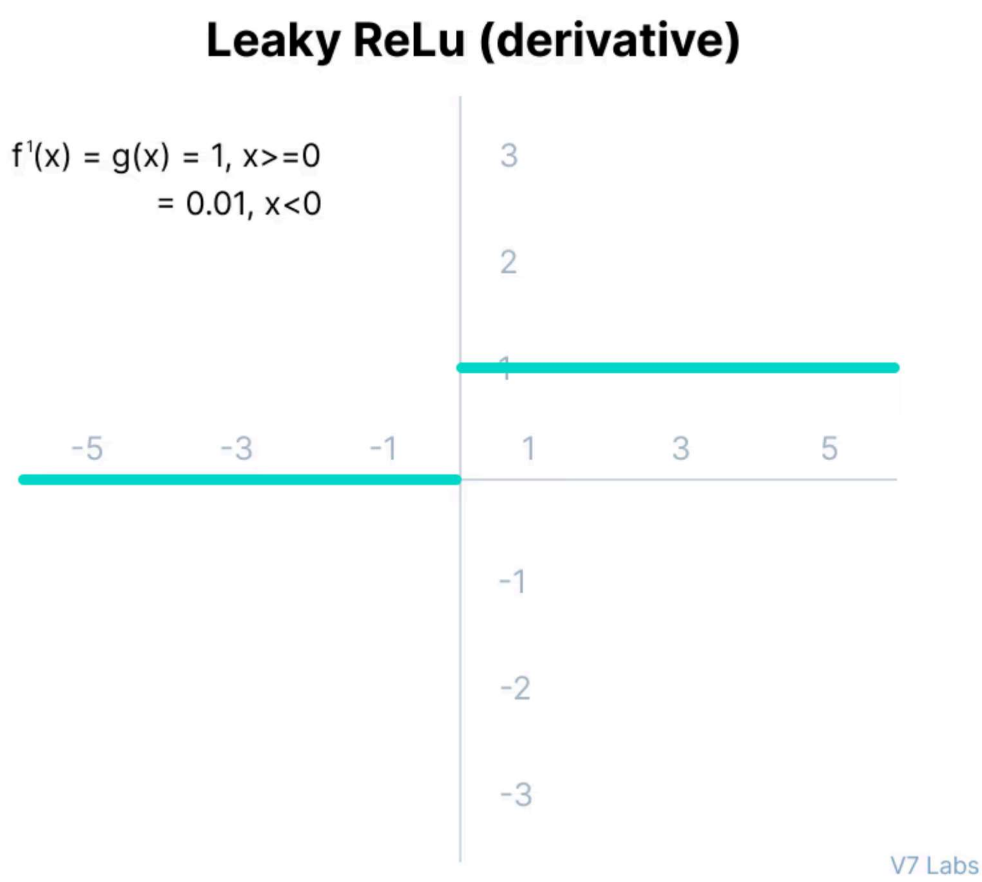
$$\textit{Leaky ReLU}$$

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

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.

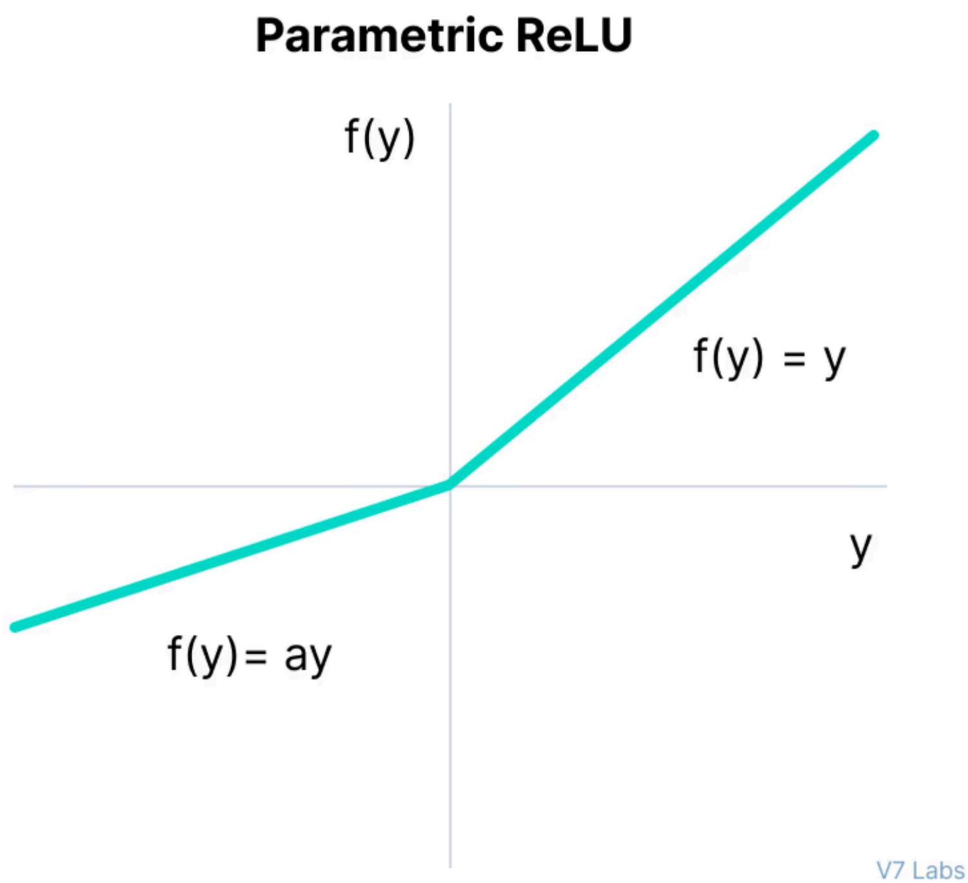


The limitations that this function faces include:

- The predictions ma
- The gradient for ne

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.

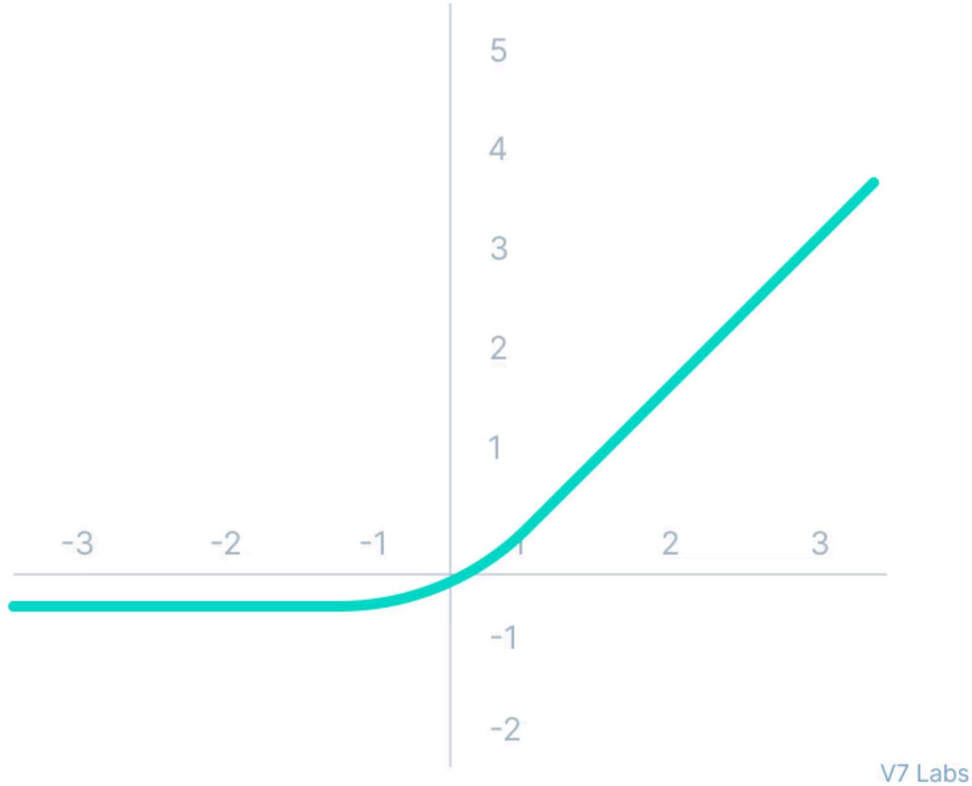
The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.

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

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 negativ values unlike the leaky ReLU and Parametric ReLU functions with a straight line.



Mathematically it can be represented as:

$$ELU$$

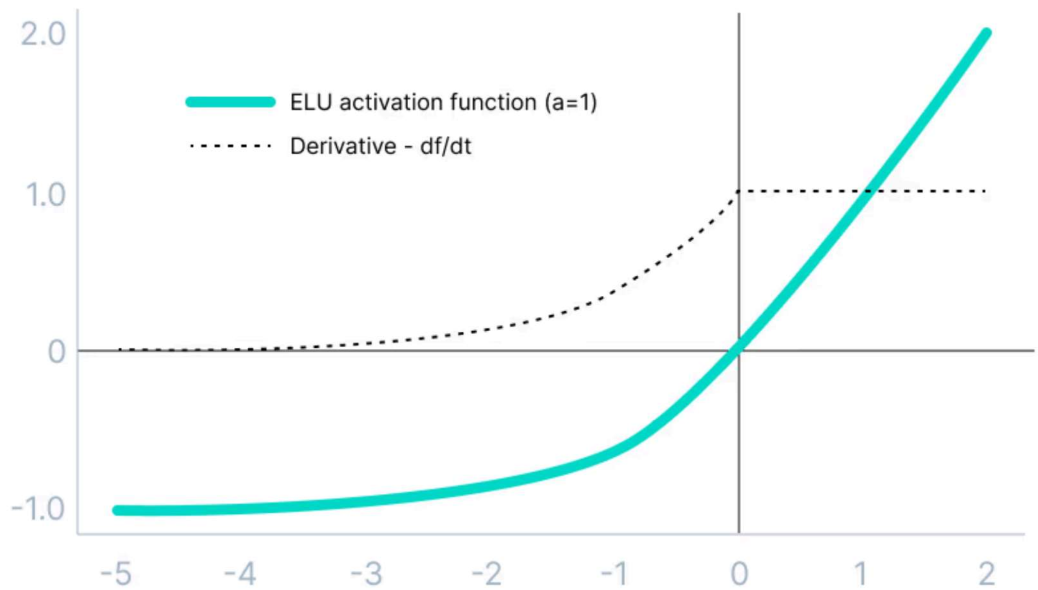
$$\begin{cases} x & for\ x \geqslant 0 \\ \alpha\left(e^{x} - 1\right) & for\ x < 0 \end{cases}$$

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.

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



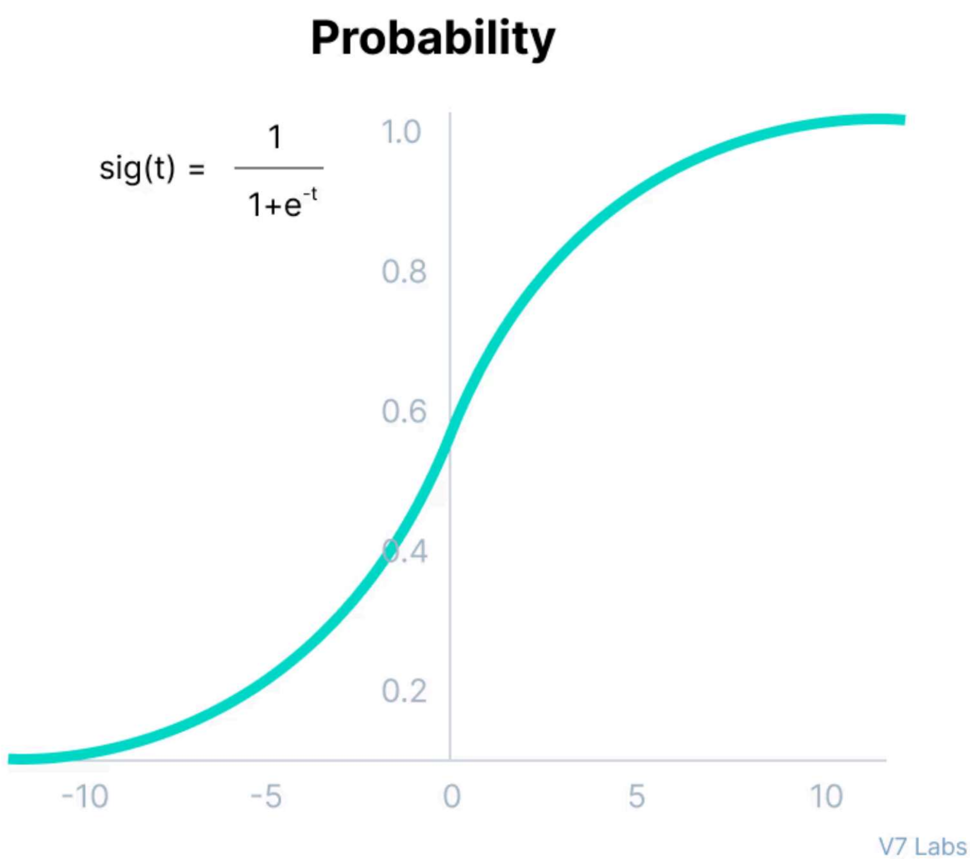
V7 Labs

Mathematically it can be represented as:

ELU derivative

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

Before exploring the ins and outs of the Softmax activation function, we should focus on its building block—the sigmoid/logistic activation function that works on calculating probability values.



The output of the sigmoid function was in the range of 0 to 1, which can be thought of as probability.

But—

This function faces certain problems.

Let's suppose we have

The above values don't make sense as the sum of all the classes/output probabilities should be equal to 1.

You see, 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.

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

Mathematically it can be represented as:

Softmax

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

V7 Labs

Let's go over a simple

the neurons is [1.8, 0.9, 0.68].

Applying the softmax function over these values to give a probabilistic view will result in the following outcome: [0.58, 0.23, 0.19].

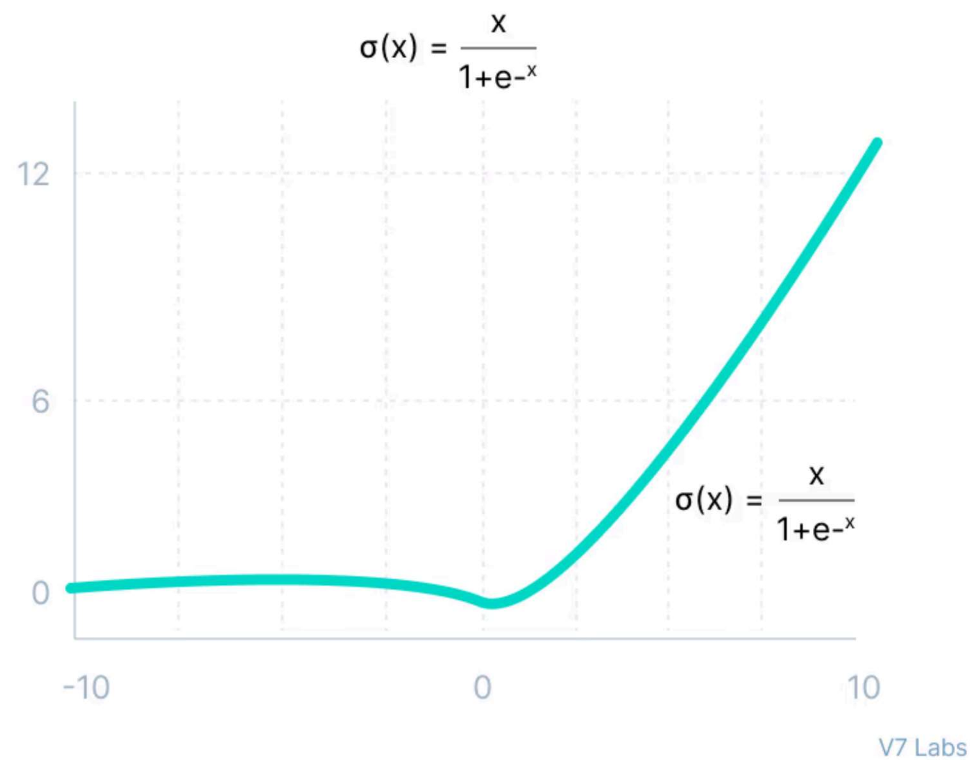
The function returns 1 for the largest probability index while it returns 0 for the other two array indexes. Here, giving full weight to index 0 and no weight to index 1 and index 2. So the output would be the class corresponding to the 1st neuron(index 0) out of three.

You can see now how softmax activation function make things easy for multi-class classification problems.

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:

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.

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.

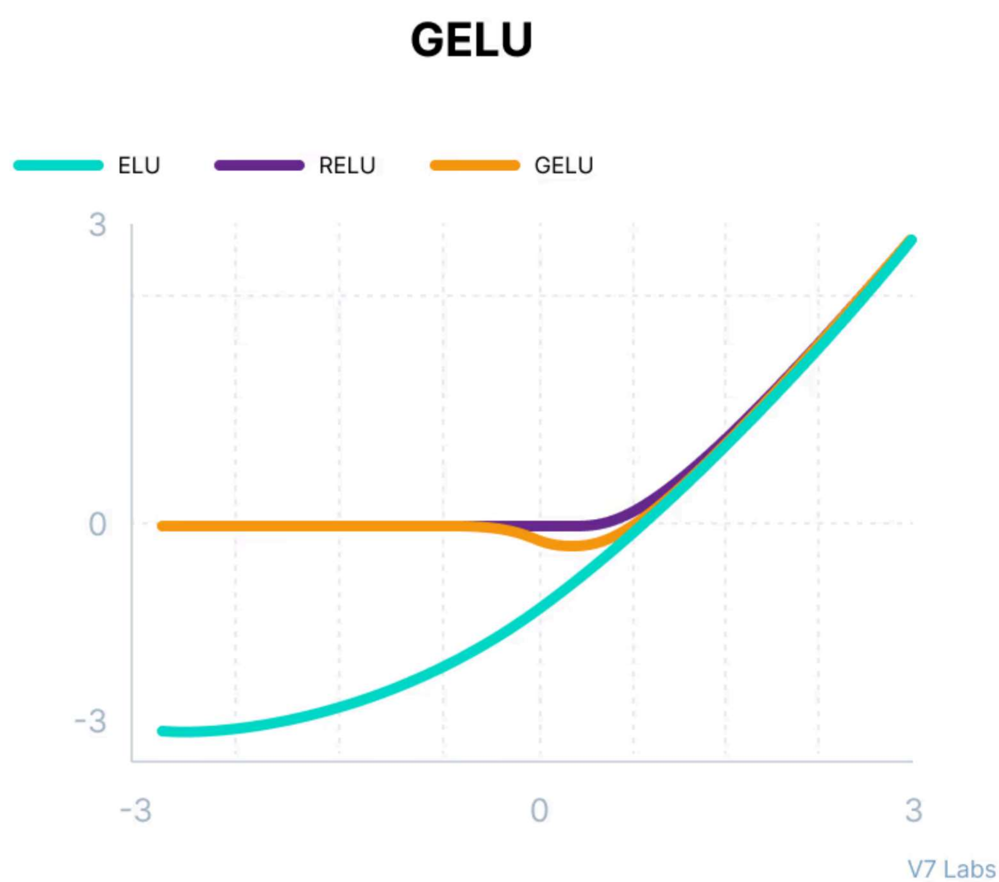
ReLU and dropout tog

RNN regularizer called zoneout stochastically multiplies inputs by one.

We merge this functionality by multiplying the input by either zero or one which is stochastically determined and is dependent upon the input. We multiply the neuron input x by

m ~ Bernoulli(Φ(x)), where Φ(x) = P(X ≤x), X ~ N (0, 1) is the cumulative distribution function of the standard normal distribution.

This distribution is chosen since neuron inputs tend to follow a normal distribution, especially with Batch Normalization.



GELU

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

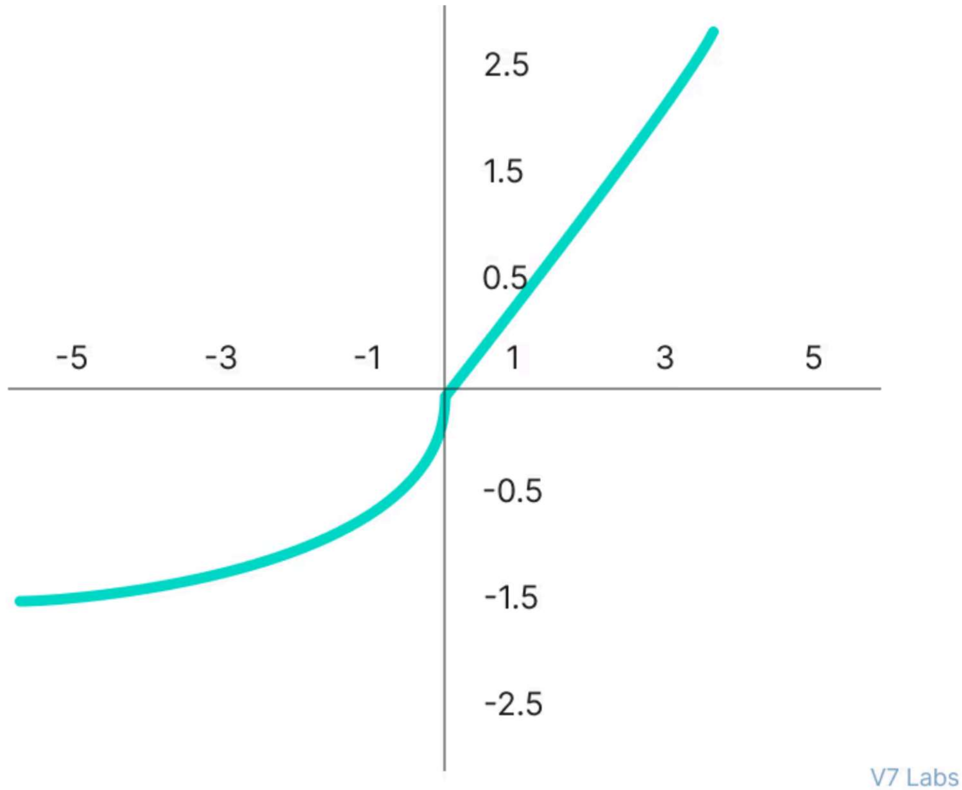
GELU nonlinearity is better than ReLU and ELU activations and finds performance improvements across all tasks in domains of computer vision, natural language processing, and speech recognition.

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:

$$\mathit{SELU}$$

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

- 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.

Let's discuss them in more detail.

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.

accumulate and result in very large updates to neural network model weights during training.

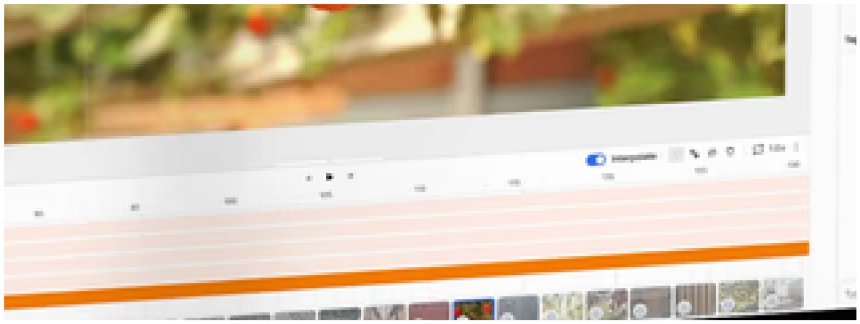
An unstable network can result when there are exploding gradients, and the learning cannot be completed.

The values of the weights can also become so large as to overflow and result in something called NaN values.

Up next

Data labeling

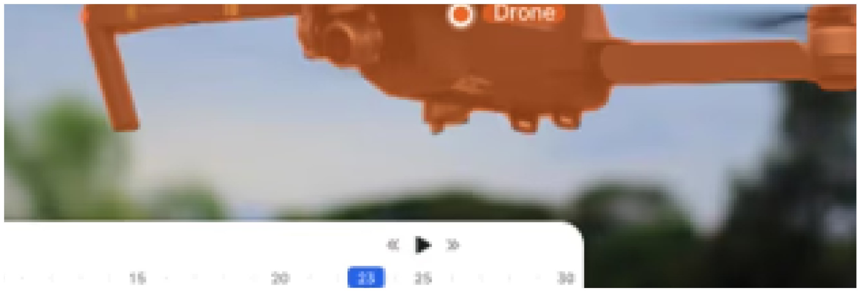
Mar 18, 2024



Data labeling tools guide: how to choose + 6 top companies reviewed

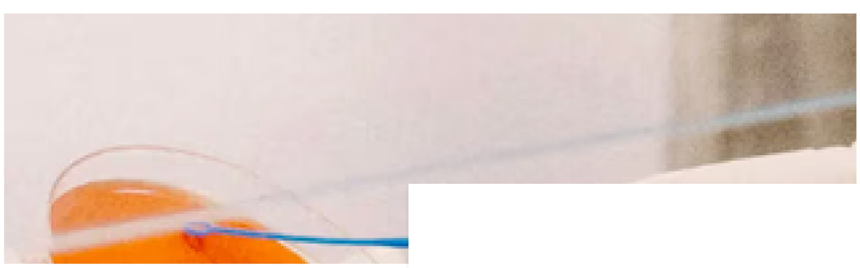
Playbooks

Feb 22, 2024



How to Segment Objects in Videos with V7 Auto-Track

AI implementation Jan 23, 2024



7 Life-Saving AI Use Cases in Healthcare

Next steps

Label videos
with V7.
Rewind less,
achieve more.

Book a demo

Try our free tier or talk to one of our experts.

Video annotation

AI video annotation

Get started today

Explore V7 Darwin

Subscribe to the V7 Newsletter