Comparison of Different Activation Functions in Neural Network

Activation functions are an essential component of neural networks, as they determine the output of the neural network. Non-linear activation functions are the most commonly used and are necessary for introducing complexity into the network. Activation functions help map input values to a known range, which helps stabilize training and helps map values to a desired output in the last layer. These functions are used in neural networks for several crucial reasons:

- *Introducing Non-Linearity*: Activation functions introduce non-linearity into the network. Without them, the entire neural network would behave like a linear model, and it would not be able to learn and represent complex, non-linear relationships in data. Real-world data is often non-linear, so the ability to capture these non-linearities is crucial for the effectiveness of neural networks.

- *Enabling Complex Representations*: Activation functions allow neural networks to learn and represent complex features and patterns in data. As information passes through the layers of neurons, the non-linear transformations introduced by activation functions enable the network to capture and understand intricate relationships in the input data.

- *Facilitating Training (Backpropagation)*:During the training process, neural networks adjust their weights to minimize the difference between predicted and actual outputs. Activation functions are essential for this process, particularly in backpropagation, where the network learns by iteratively adjusting its weights. The non-linearities introduced by activation functions enable the network to learn and adapt effectively.

- *Avoiding Vanishing or Exploding Gradients*:In deep neural networks, gradients can become very small (vanish) or very large (explode) as they are propagated backward through the layers during training. Activation functions like ReLU help mitigate the vanishing gradient problem by maintaining a more consistent gradient flow, which facilitates better training.

- *Controlling Neuron Output*: Activation functions determine the output of a neuron based on its input. This helps in controlling the information flow through the network and allows each neuron to contribute selectively to the overall computation.

In general, 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 doesnot 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.

1 Three Types of Neural Networks Activation Functions

Now, as we've covered the essential concepts, let's go over the most popular neural networks activation functions.

1.1 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.

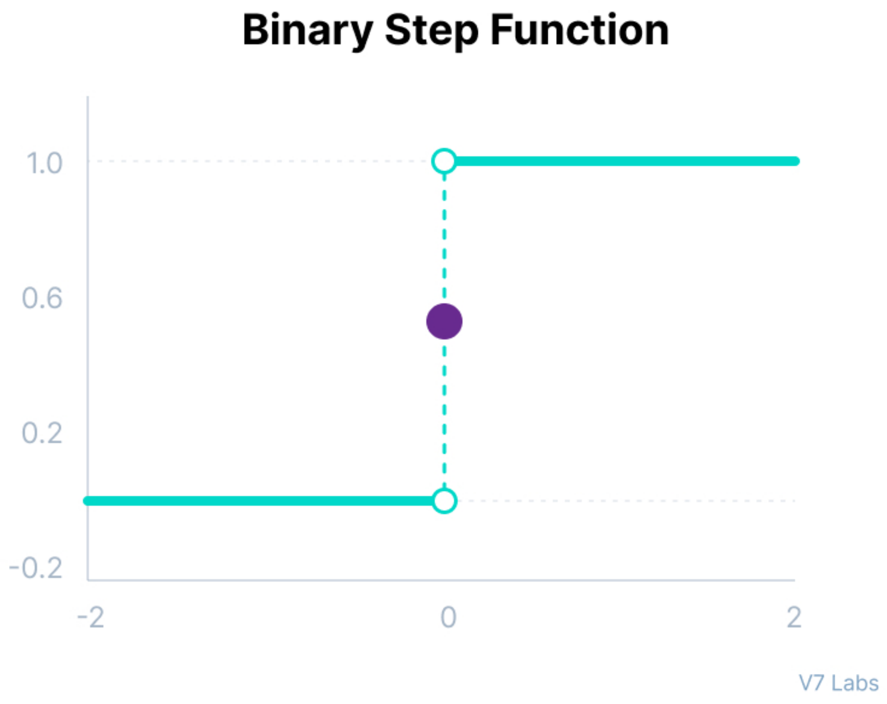

Figure 1: Binary Step Function

Mathematically it can be represented as:

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

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 back-propagation process.

1.2 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.

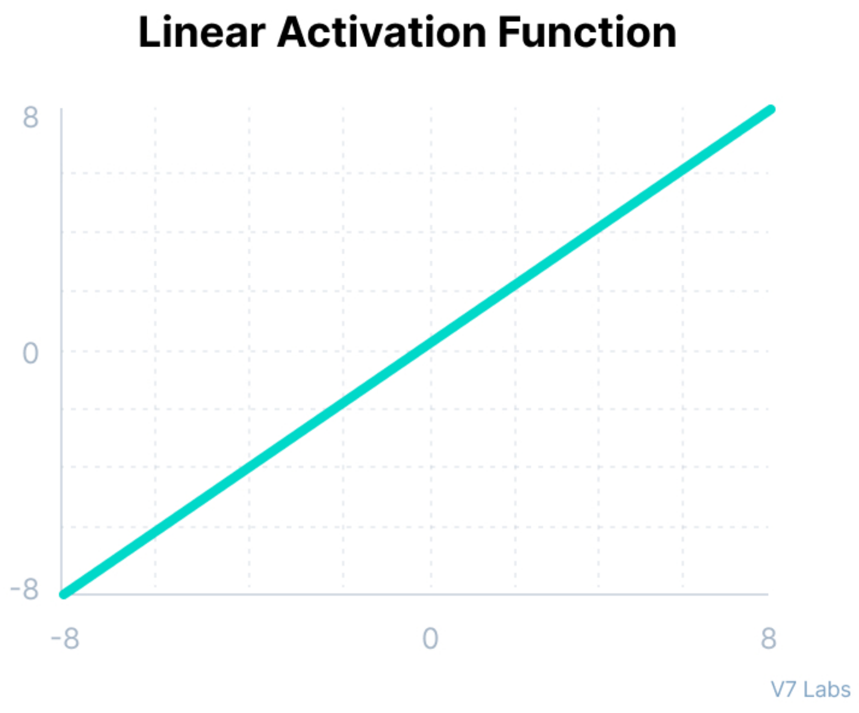

Figure 2: Linear Activation Function

Mathematically it can be represented as:

$$f(x) = x$$

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.

1.3 Non-Linear Activation Functions

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.

1.4 10 Non-Linear Neural Networks Activation Functions

1.4.1 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.

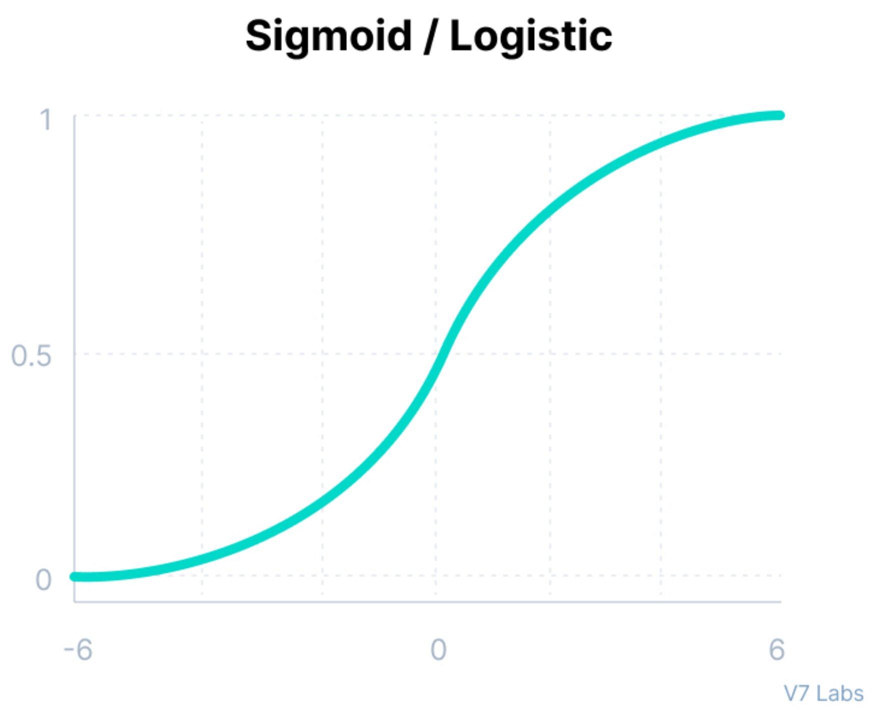

Figure 3: Sigmoid/Logistic Activation Function

Mathematically it can be represented as:

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

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 $f'(x) = \text{sigmoid}(x) \times (1 - \text{sigmoid}(x))$.

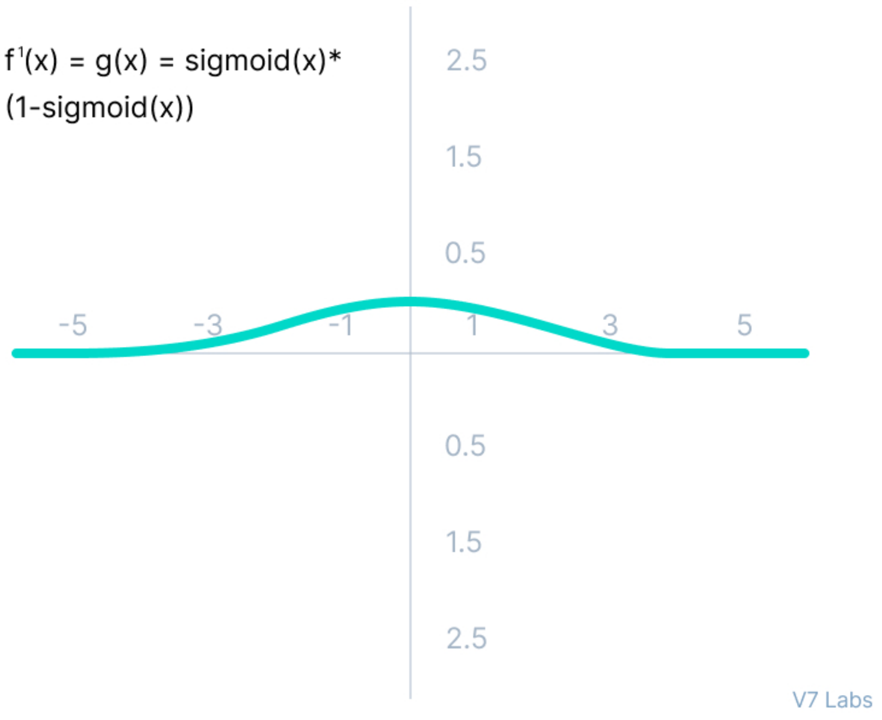

Figure 4: The derivative of the Sigmoid Activation Function

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.

1.5 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.

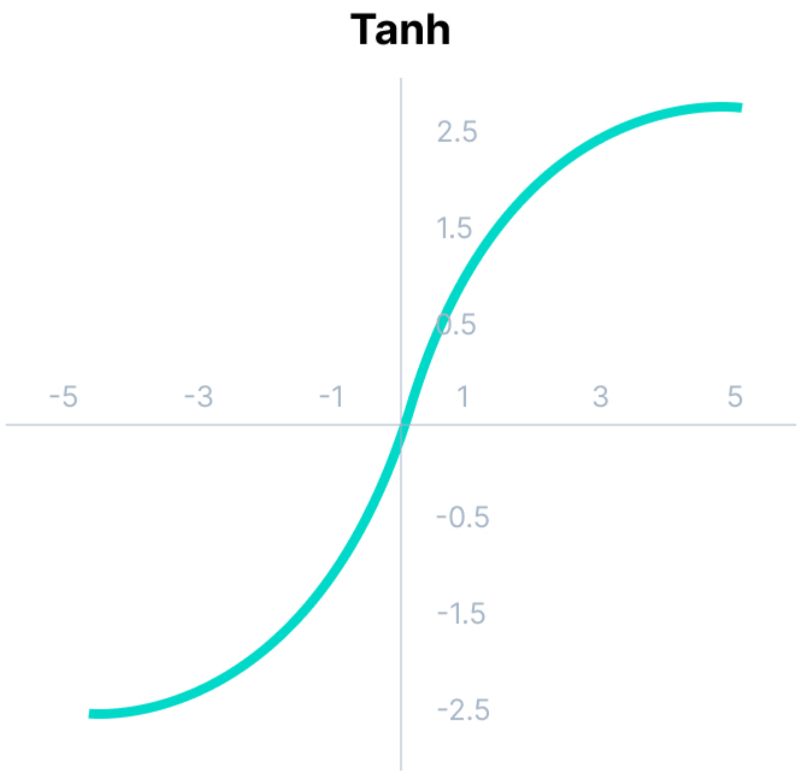

Figure 5: Tanh Function (Hyperbolic Tangent)

Mathematically it can be represented as:

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

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.

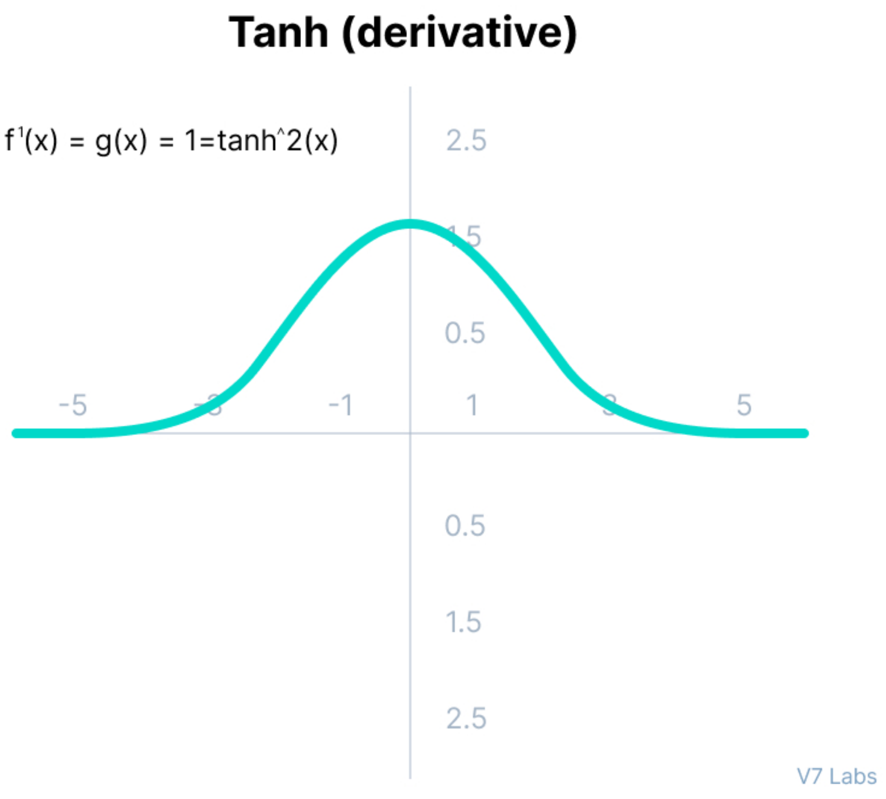

Figure 6: Gradient of the Tanh Activation Function

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.

1.6 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.

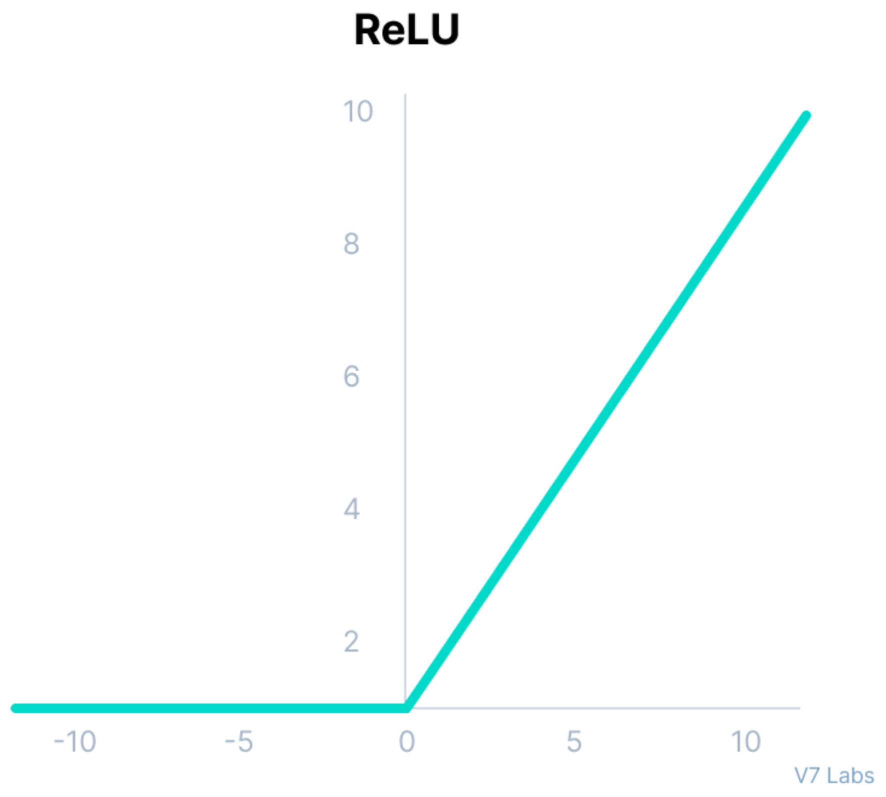

Figure 7: ReLU Activation Function

Mathematically it can be represented as:

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

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.

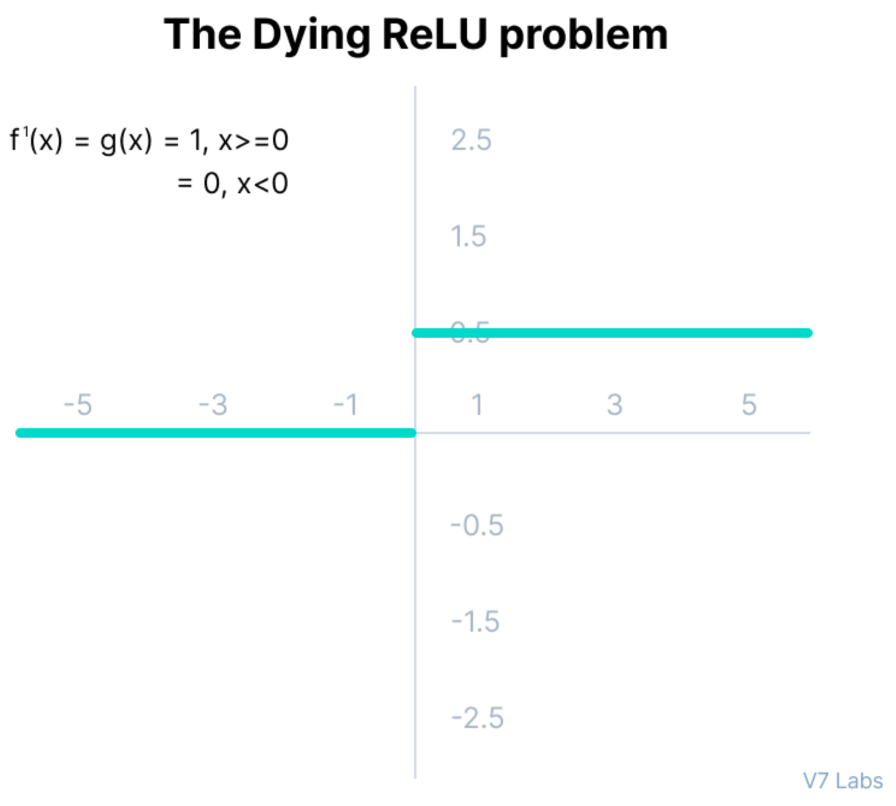

Figure 8: The Dying ReLU problem

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.

1.7 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.

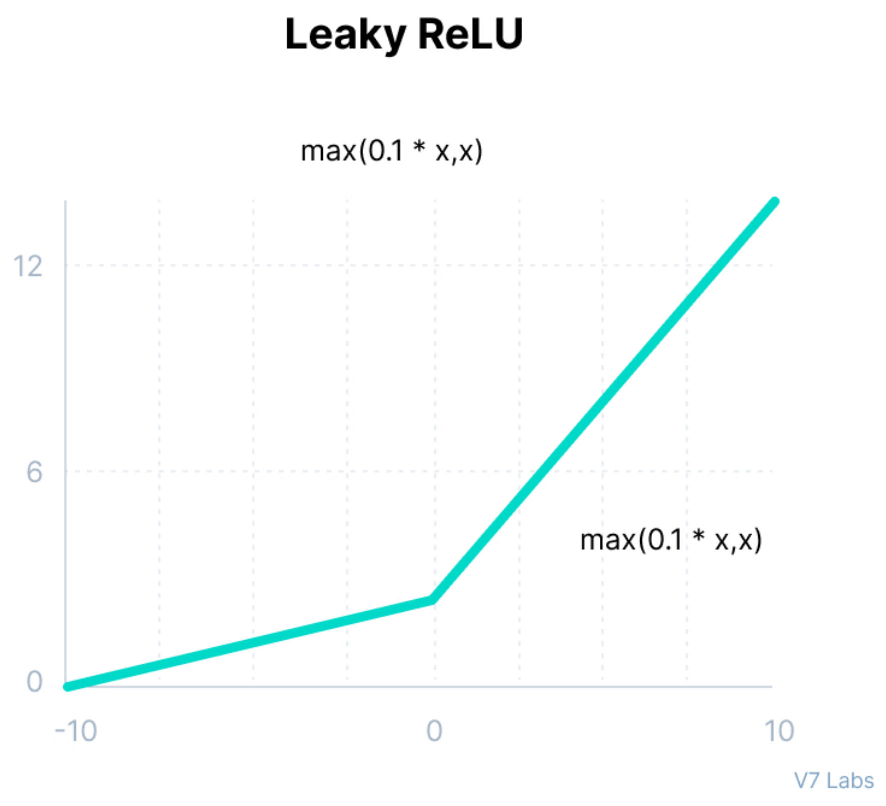

Figure 9: Leaky ReLU

Mathematically it can be represented as:

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

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.

Here is the derivative of the Leaky ReLU function.

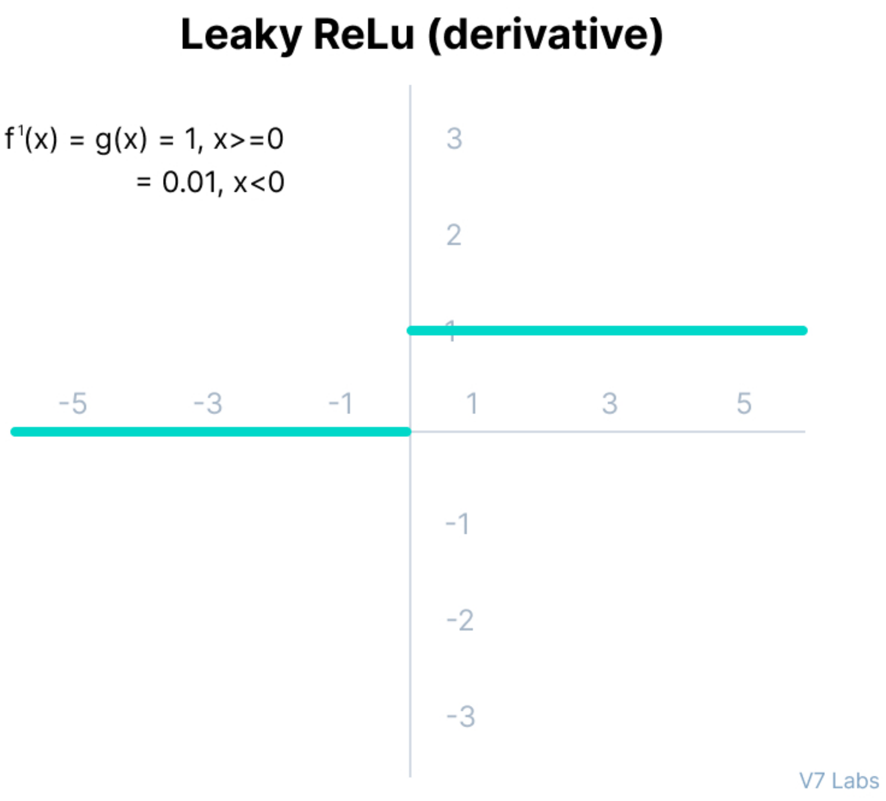

Figure 10: The derivative of the Leaky ReLU function

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.

1.8 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.

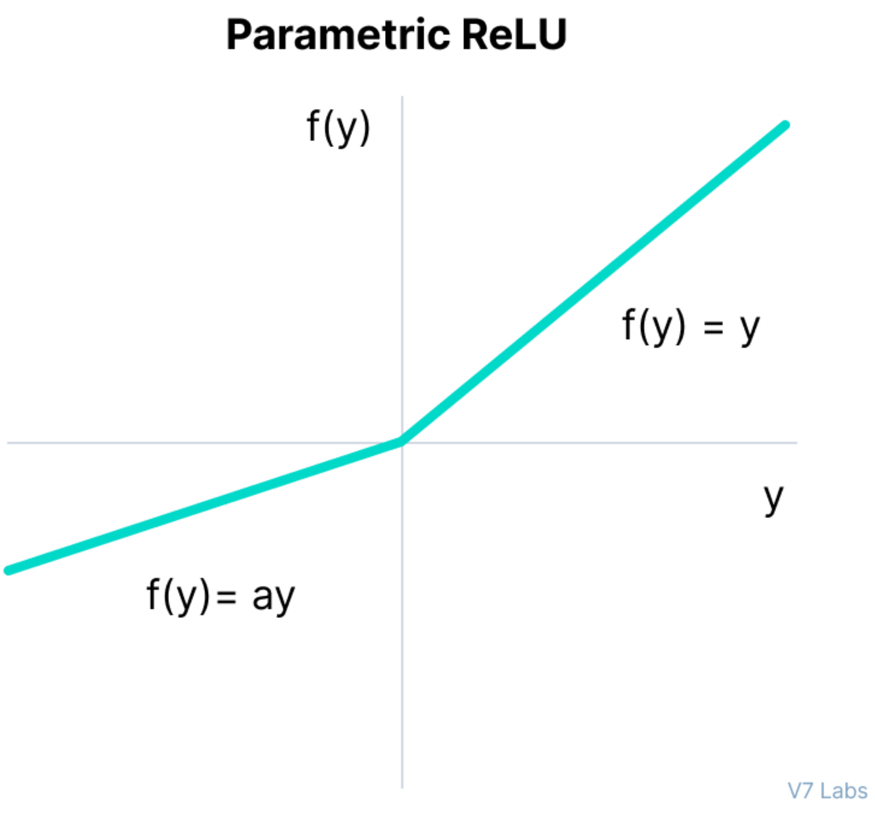

Figure 11: Parametric ReLU

Mathematically it can be represented as:

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

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.

1.9 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.

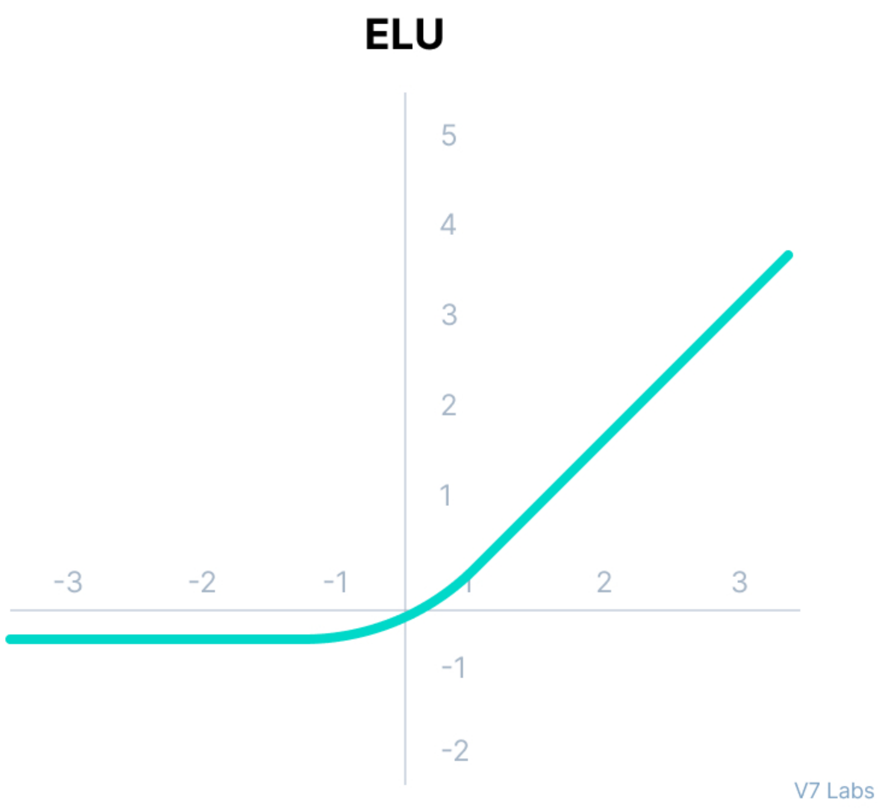

Figure 12: ELU Activation Function

Mathematically it can be represented as:

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

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

- ELU becomes smooth slowly until its output equal to $-\alpha$ 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

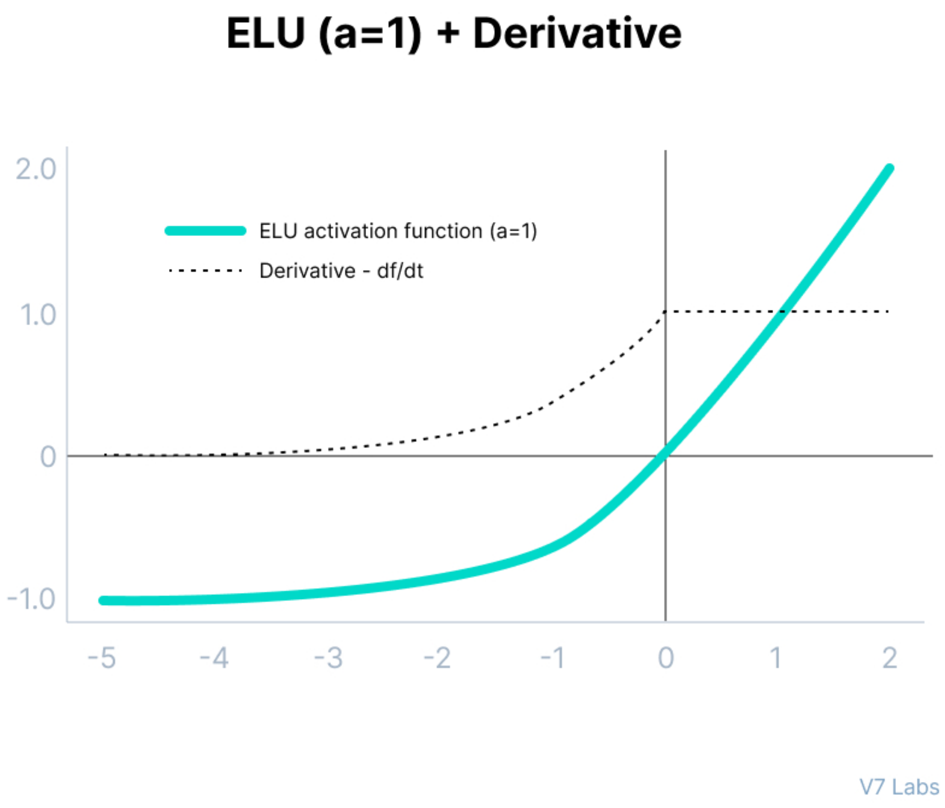

Figure 13: ELU Activation Function and its derivative

Mathematically it can be represented as:

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

1.10 Softmax Function

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.

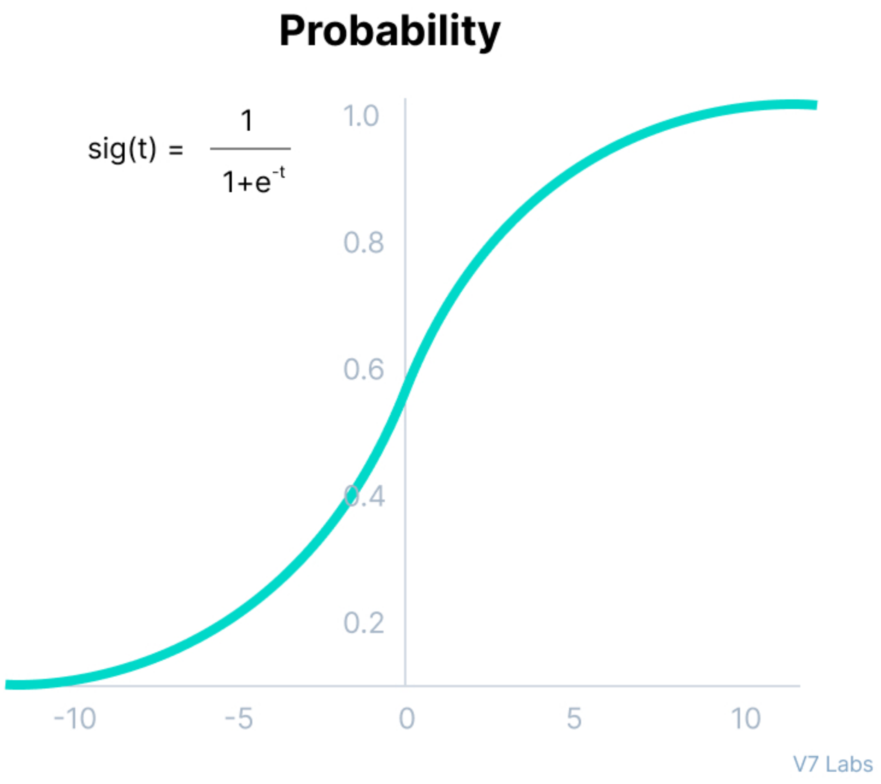

Figure 14: Probability

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 five output values of 0.8, 0.9, 0.7, 0.8, and 0.6, respectively. How can we move forward with it?

The answer is: We can't.

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:

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

Let's go over a simple example together.

Assume that you have three classes, meaning that there would be three neurons in the output layer. Now, suppose that your output from 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.

1.11 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.

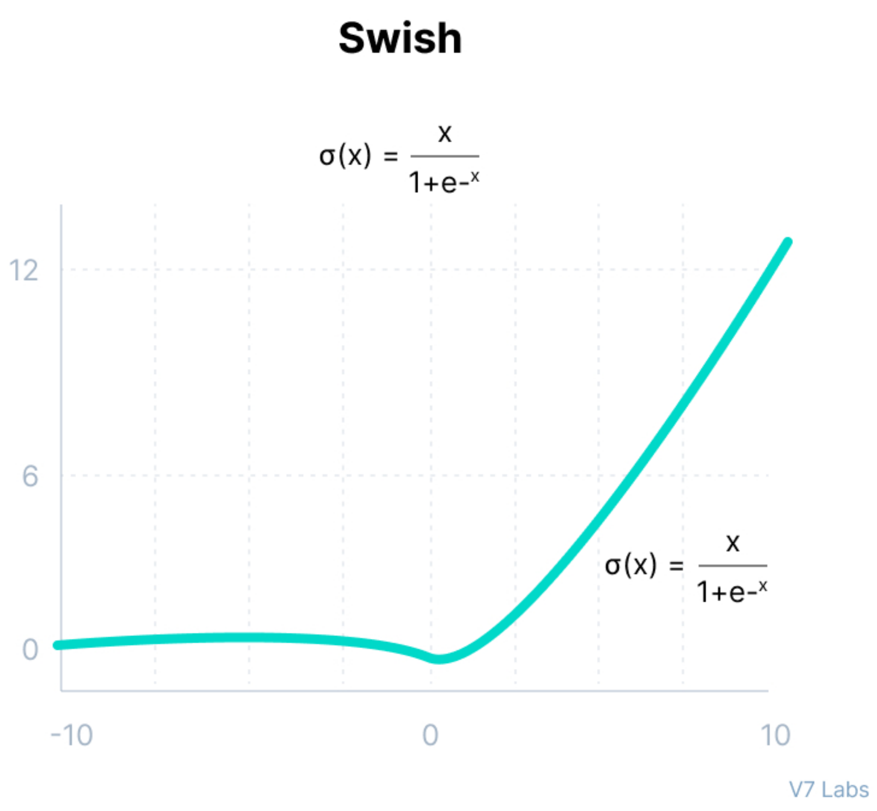

Figure 15: Swish Activation Function

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:

$$f(x) = x \times \text{sigmoid}(x)$$

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.

1.12 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.

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

$m \sim \text{Bernoulli}(\phi(x))$, where $\phi(x) = P(X \geq x)$, $X \sim N(0, 1)$ 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.

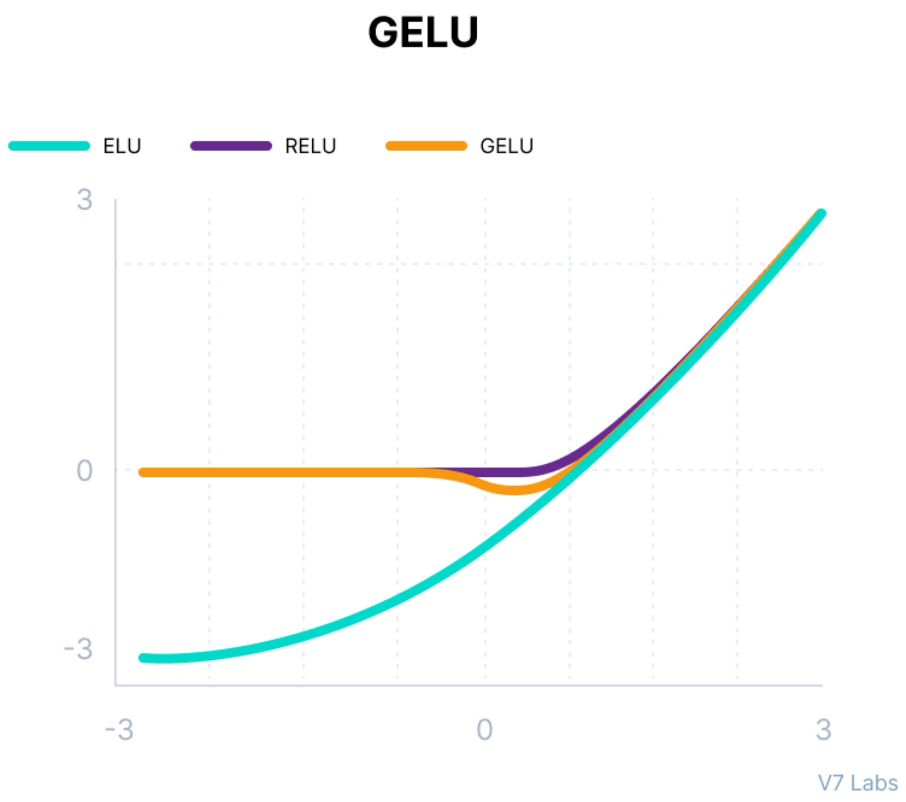

Figure 16: Gaussian Error Linear Unit (GELU) Activation Function

Mathematically it can be represented as:

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

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.

1.13 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.

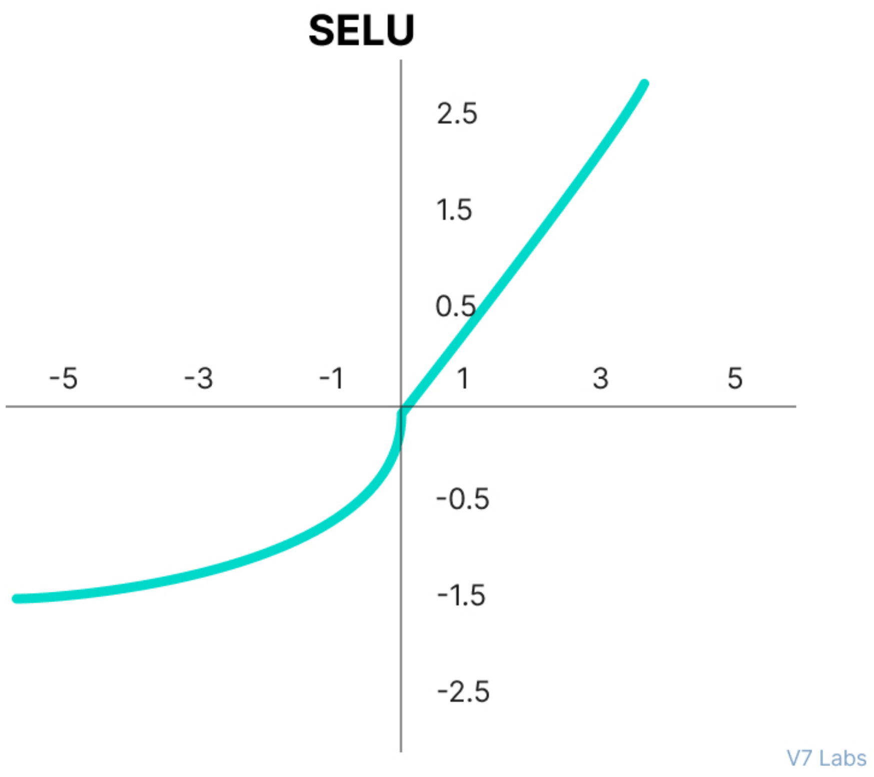

Figure 17: SELU Activation Function

Mathematically it can be represented as:

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

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.

2 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.

2.1 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.

2.2 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.

3 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 :)

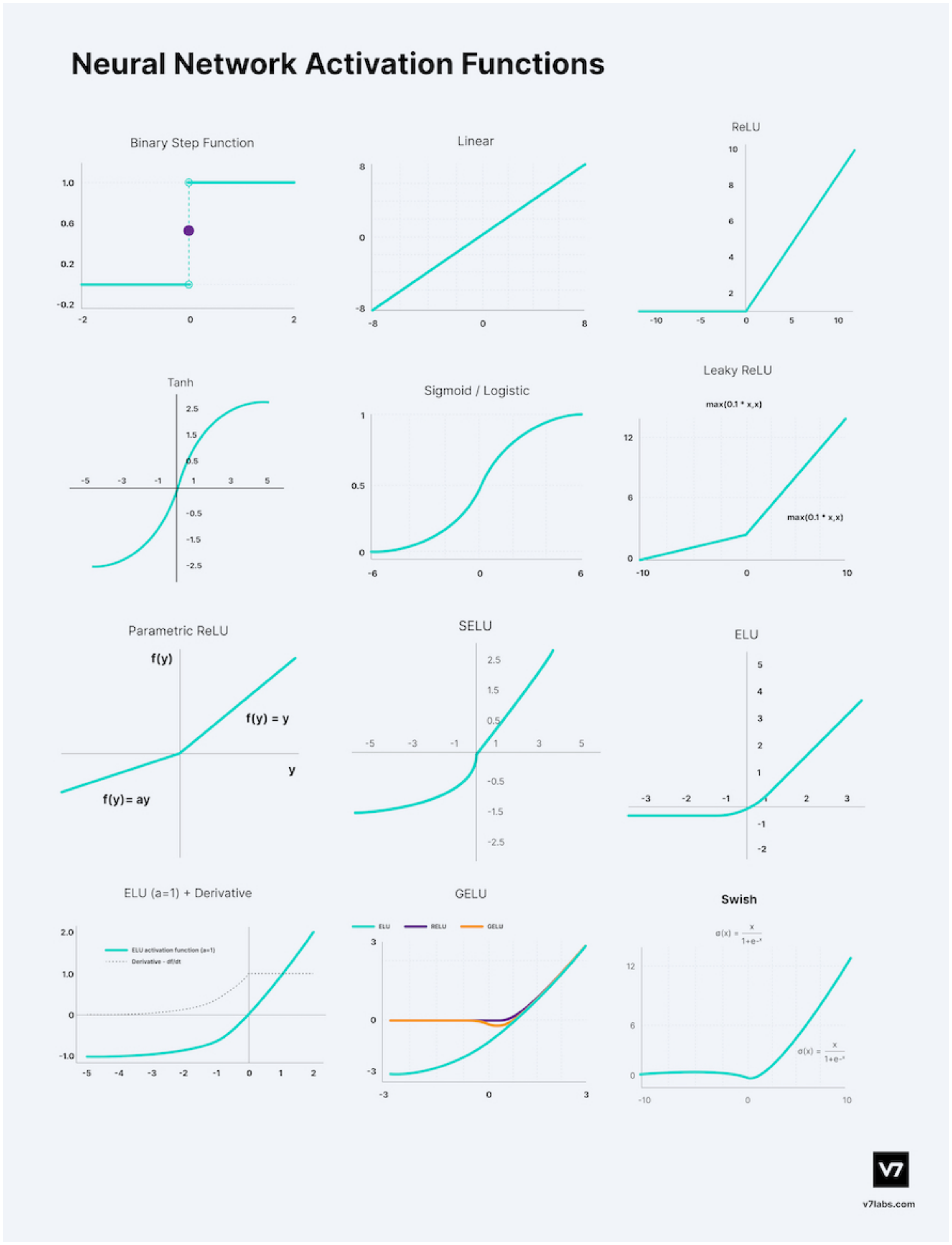

Figure 18: Neural Network Activation Functions: Cheat Sheet

4 Neural Networks Activation Functions in a Nutshell

Now, let's have a quick recap of everything you've learnt in this topic:

- 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.