# How to Choose an Activation Function for Deep Learning

**Activation functions** are a critical part of the design of a neural network.

The choice of activation function in the hidden layer will control how well the network model learns the training dataset. The choice of activation function in the output layer will define the type of predictions the model can make.

As such, a careful choice of activation function must be made for each deep learning neural network project.

In this tutorial, you will discover how to choose activation functions for neural network models.

After completing this tutorial, you will know:

- Activation functions are a key part of neural network design.
- The modern default activation function for hidden layers is the ReLU function.
- The activation function for output layers depends on the type of prediction problem.

## Tutorial Overview

This tutorial is divided into three parts; they are:

1. Activation Functions
2. Activation for Hidden Layers
3. Activation for Output Layers

## Activation Functions

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.

Sometimes the activation function is called a “*transfer function*.” If the output range of the activation function is limited, then it may be called a “*squashing function*.” Many activation functions are nonlinear and may be referred to as the “*nonlinearity*” in the layer or the network design.
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.

Technically, the activation function is used within or after the internal processing of each node in the network, although networks are designed to use the same activation function for all nodes in a layer.

A network may have three types of layers: input layers that take raw input from the domain, **hidden layers** that take input from another layer and pass output to another layer, and **output layers** that make a prediction.
All hidden layers typically use the same activation function. The output layer will typically use a different activation function from the hidden layers and is dependent upon the type of prediction required by the model.

Activation functions are also typically differentiable, meaning the first-order derivative can be calculated for a given input value. This is required given that neural networks are typically trained using the backpropagation of error algorithm that requires the derivative of prediction error in order to update the weights of the model.

There are many different types of activation functions used in neural networks, although perhaps only a small number of functions used in practice for hidden and output layers.

Let’s take a look at the activation functions used for each type of layer in turn.

## Activation for Hidden Layers

A hidden layer in a neural network is a layer that receives input from another layer (such as another hidden layer or an input layer) and provides output to another layer (such as another hidden layer or an output layer).

A hidden layer does not directly contact input data or produce outputs for a model, at least in general.

A neural network may have zero or more hidden layers.

Typically, a differentiable nonlinear activation function is used in the hidden layers of a neural network. This allows the model to learn more complex functions than a network trained using a linear activation function.

*In order to get access to a much richer hypothesis space that would benefit from deep representations, you need a non-linearity, or activation function.*

There are perhaps three activation functions you may want to consider for use in hidden layers; they are :

- Rectified Linear Activation (**ReLU**)
- Logistic (**Sigmoid**)
- Hyperbolic Tangent (**Tanh**)

This is not an exhaustive list of activation functions used for hidden layers, but they are the most commonly used.

Let’s take a closer look at each in turn.

## ReLU Hidden Layer Activation Function

The rectified linear activation function, or ReLU activation function, is perhaps the most common function used for hidden layers.

It is common because it is both simple to implement and effective at overcoming the limitations of other previously popular activation functions, such as Sigmoid and Tanh. Specifically, it is less susceptible to vanishing gradients that prevent deep models from being trained, although it can suffer from other problems like saturated or "*dead*" units.

The ReLU function is calculated as follows:

- max(0.0, x)

This means that if the input value (x) is negative, then a value 0.0 is returned, otherwise, the value is returned.

You can learn more about the details of the ReLU activation function in this tutorial:

- A Gentle Introduction to the Rectified Linear Unit (ReLU)

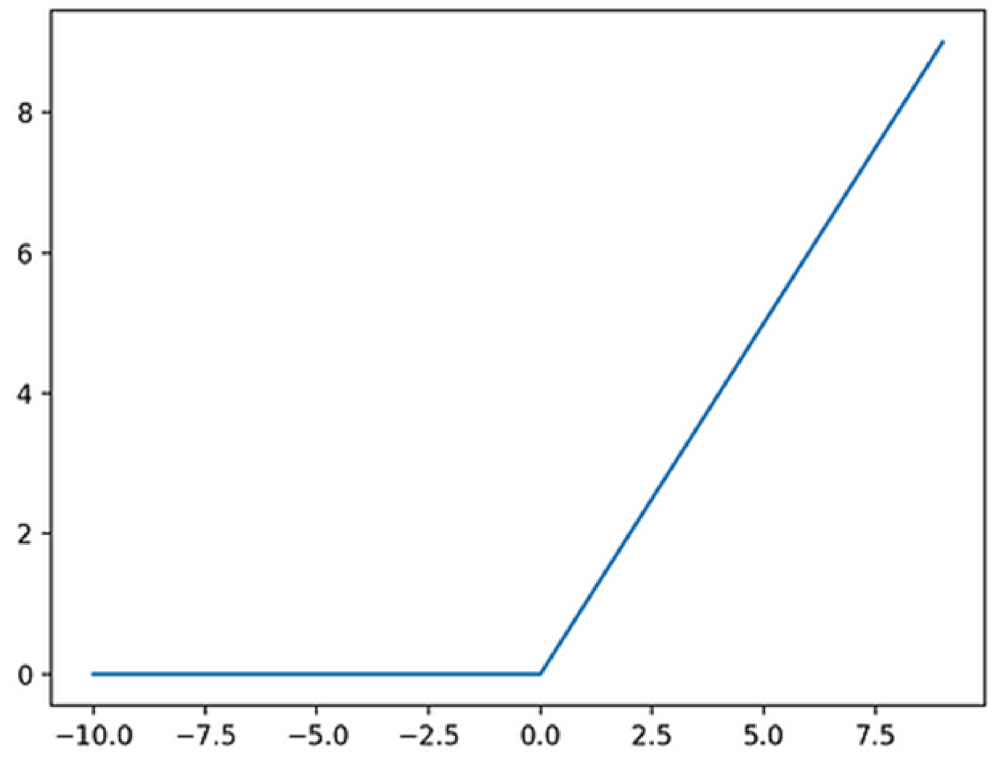
We can get an intuition for the shape of this function with the worked example below.

```
# example plot for the relu activation function
from matplotlib import pyplot


# rectified linear function
def rectified(x):
 return max(0.0, x)


# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
```

```
outputs = [rectified(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Running the example calculates the outputs for a range of values and creates a plot of inputs versus outputs.

We can see the familiar kink shape of the ReLU activation function.



When using the ReLU function for hidden layers, it is a good practice to use a "*He Normal*" or "*He Uniform*" weight initialization and scale input data to the range 0-1 .(normalize) prior to training

## Sigmoid Hidden Layer Activation Function

The sigmoid activation function is also called the logistic function.

It is the same function used in the logistic regression classification algorithm.

The function takes any real value as input and outputs values in the range 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.

The sigmoid activation function is calculated as follows:

· 1.0 / (1.0 + e^-x)

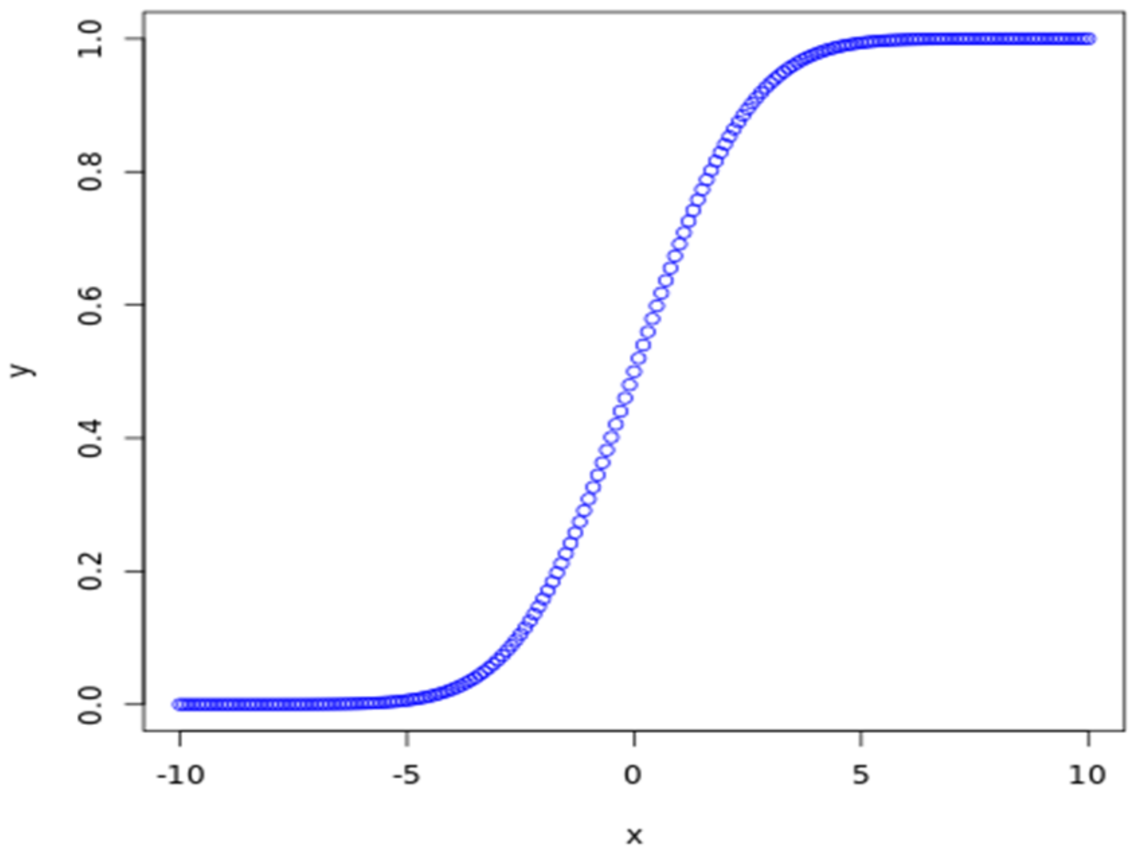
Where e is a mathematical constant , which is the base of the natural logarithm.
We can get an intuition for the shape of this function with the worked example below.

```
# example plot for the sigmoid activation function
from math import exp
from matplotlib import pyplot

# sigmoid activation function
def sigmoid(x):
 return 1.0 / (1.0 + exp(-x))

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [sigmoid(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
```

```
15 pyplot.show()
```

Running the example calculates the outputs for a range of values and creates a plot of inputs versus outputs.

We can see the familiar S-shape of the sigmoid activation function



When using the Sigmoid function for hidden layers, it is a good practice to use a "*Xavier Normal*" or "*Xavier Uniform*" weight initialization (also referred to Glorot initialization, named for Xavier Glorot) and scale input data to the range 0-1 (e.g. the range of the .activation function) prior to training

## Tanh Hidden Layer Activation Function

The hyperbolic tangent activation function is also referred to simply as the Tanh (also "*tanh*" and "*TanH*") function.
It is very similar to the sigmoid activation function and even has the same S-shape.

The function takes any real value as input and outputs values in the range -1 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 -1.0.

The Tanh activation function is calculated as follows:

· (e^x – e^-x) / (e^x + e^-x)

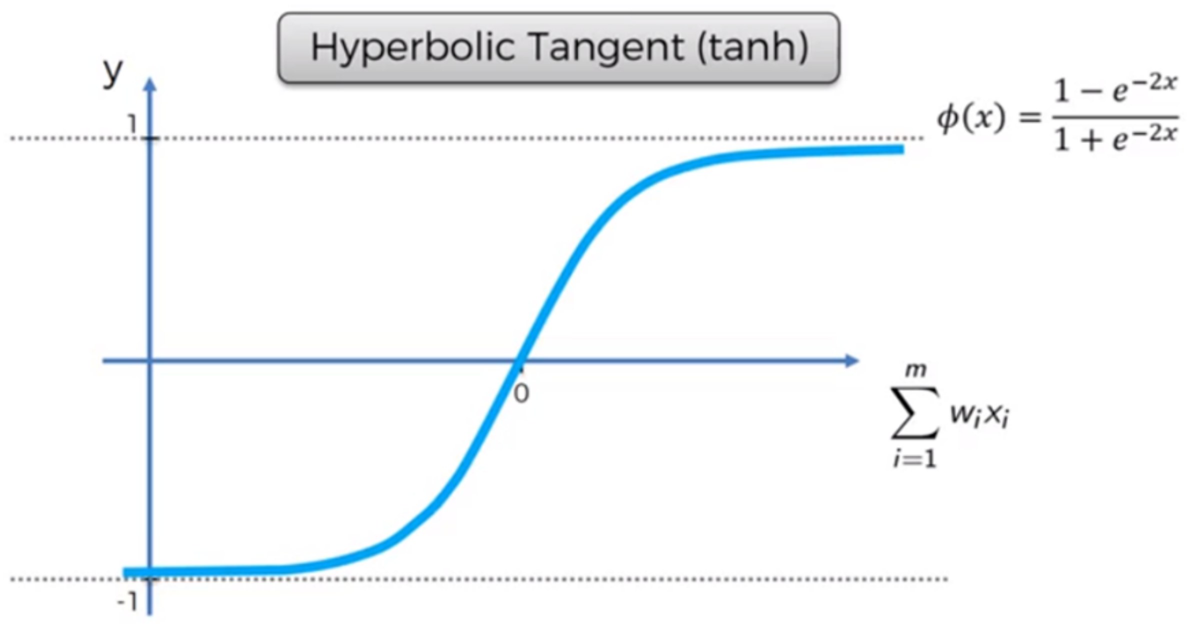
Where e is a mathematical constant that is the base of the natural logarithm.
We can get an intuition for the shape of this function with the worked example below.

```
# example plot for the tanh activation function

from math import exp

from matplotlib import pyplot


# tanh activation function

def tanh(x):

 return (exp(x) - exp(-x)) / (exp(x) + exp(-x))


# define input data

inputs = [x for x in range(-10, 10)]

# calculate outputs

outputs = [tanh(x) for x in inputs]

# plot inputs vs outputs
```

```
pyplot.plot(inputs, outputs)

pyplot.show()
```

Running the example calculates the outputs for a range of values and creates a plot of inputs versus outputs.

We can see the familiar S-shape of the Tanh activation function.



When using the TanH function for hidden layers, it is a good practice to use a "*Xavier Normal*" or "*Xavier Uniform*" weight initialization (also referred to Glorot initialization, named for Xavier Glorot) and scale input data to the range -1 to 1 (e.g. the range of .the activation function) prior to training

## How to Choose a Hidden Layer Activation Function

A neural network will almost always have the same activation function in all hidden layers.

It is most unusual to vary the activation function through a network model.

Traditionally, the sigmoid activation function was the default activation function in the 1990s. Perhaps through the mid to late 1990s to 2010s, the Tanh function was the default activation function for hidden layers.

*… the hyperbolic tangent activation function typically performs better than the logistic sigmoid.*

Both the sigmoid and Tanh functions can make the model more susceptible to problems during training, via the so-called vanishing gradients problem.

You can learn more about this problem in this tutorial:

• A Gentle Introduction to the Rectified Linear Unit (ReLU)The activation function used in hidden layers is typically chosen based on the type of neural network architecture.

Modern neural network models with common architectures, such as MLP and CNN, will make use of the ReLU activation function, or extensions.

*In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU …*

Recurrent networks still commonly use Tanh or sigmoid activation functions, or even both. For example, the LSTM commonly uses the Sigmoid activation for recurrent connections and the Tanh activation for output.

- **Multilayer Perceptron (MLP)**: ReLU activation function.
- **Convolutional Neural Network (CNN)**: ReLU activation function.
- **Recurrent Neural Network**: Tanh and/or Sigmoid activation function.

If you’re unsure which activation function to use for your network, try a few and compare the results.

The figure below summarizes how to choose an activation function for the hidden layers of your neural network model.

## Activation for Output Layers

The output layer is the layer in a neural network model that directly outputs a prediction.

All feed-forward neural network models have an output layer.

There are perhaps three activation functions you may want to consider for use in the output layer; they are:

- Linear
- Logistic (Sigmoid)
- Softmax

This is not an exhaustive list of activation functions used for output layers, but they are the most commonly used.

Let’s take a closer look at each in turn.

### Linear Output Activation Function

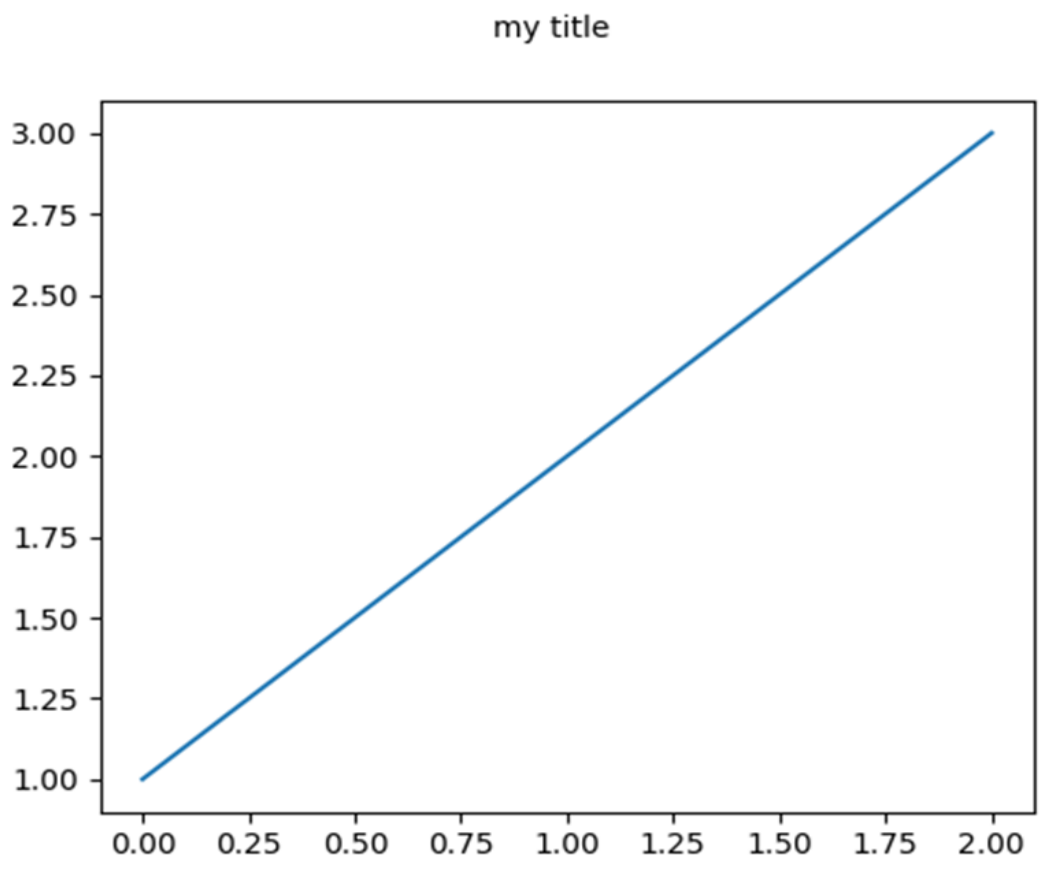
The linear activation function is also called “*identity*” (multiplied by 1.0) or “*no activation*.” This is because the linear activation function does not change the weighted sum of the input in any way and instead returns the value directly.

We can get an intuition for the shape of this function with the worked example below.

```
# example plot for the linear activation function
from matplotlib import pyplot

# linear activation function
def linear(x):
 return x

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [linear(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Running the example calculates the outputs for a range of values and creates a plot of inputs versus outputs.

We can see a diagonal line shape where inputs are plotted against identical outputs.



Target values used to train a model with a linear activation function in the output layer are typically scaled prior to modeling using normalization or standardization transforms.

## Sigmoid Output Activation Function

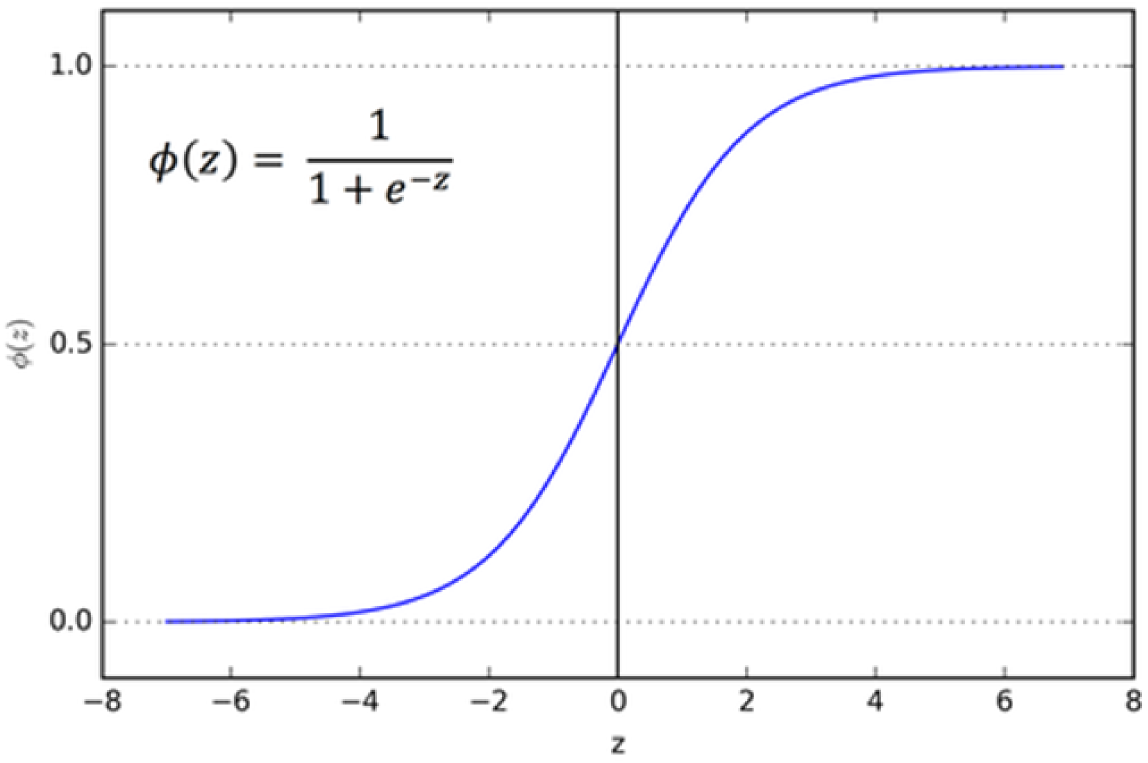
The sigmoid of logistic activation function was described in the previous section.

Nevertheless, to add some symmetry, we can review for the shape of this function with the worked example below.

```
# example plot for the sigmoid activation function
from math import exp
from matplotlib import pyplot

# sigmoid activation function
def sigmoid(x):
 return 1.0 / (1.0 + exp(-x))

# define input data
inputs = [x for x in range(-10, 10)]
# calculate outputs
outputs = [sigmoid(x) for x in inputs]
# plot inputs vs outputs
pyplot.plot(inputs, outputs)
pyplot.show()
```

Running the example calculates the outputs for a range of values and creates a plot of inputs versus outputs.

We can see the familiar S-shape of the sigmoid activation function.



Target labels used to train a model with a sigmoid activation function in the output layer .will have the values 0 or 1

## Softmax Output Activation Function

The softmax function outputs a vector of values that sum to 1.0 that can be interpreted as probabilities of class membership.

It is related to the argmax function that outputs a 0 for all options and 1 for the chosen option. Softmax is a "*softer*" version of argmax that allows a probability-like output of a winner-take-all function.

As such, the input to the function is a vector of real values and the output is a vector of the same length with values that sum to 1.0 like probabilities.

The softmax function is calculated as follows:

· e^x / sum(e^x)

Where *x* is a vector of outputs and e is a mathematical constant that is the base of the natural logarithm.

You can learn more about the details of the Softmax function in this tutorial:

- Softmax Activation Function with PythonWe cannot plot the softmax function, but we can give an example of calculating it in Python.

```
from numpy import exp

# softmax activation function
def softmax(x):
 return exp(x) / exp(x).sum()

# define input data
inputs = [1.0, 3.0, 2.0]
# calculate outputs
outputs = softmax(inputs)
# report the probabilities
print(outputs)
# report the sum of the probabilities
print(outputs.sum())
```

Running the example calculates the softmax output for the input vector.

We then confirm that the sum of the outputs of the softmax indeed sums to the value 1.0.

```
[0.09003057 0.66524096 0.24472847]
1.0
```

Target labels used to train a model with the softmax activation function in the output layer will be vectors with 1 for the target class and 0 for all other classes

## How to Choose an Output Activation Function

You must choose the activation function for your output layer based on the type of prediction problem that you are solving.

Specifically, the type of variable that is being predicted.

For example, you may divide prediction problems into two main groups, predicting a categorical variable (*classification*) and predicting a numerical variable (*regression*).
If your problem is a regression problem, you should use a linear activation function.

· **Regression**: One node, linear activation.

If your problem is a classification problem, then there are three main types of classification problems and each may use a different activation function.

Predicting a probability is not a regression problem; it is classification. In all cases of classification, your model will predict the probability of class membership (e.g. probability that an example belongs to each class) that you can convert to a crisp class label by rounding (for sigmoid) or argmax (for softmax).

If there are two mutually exclusive classes (binary classification), then your output layer will have one node and a sigmoid activation function should be used. If there are more than two mutually exclusive classes (multiclass classification), then your output layer will have one node per class and a softmax activation should be used. If there are two or more mutually inclusive classes (multilabel classification), then your output layer will have one node for each class and a sigmoid activation function is used.

· **Binary Classification**: One node, sigmoid activation.
· **Multiclass Classification**: One node per class, softmax activation.
· **Multilabel Classification**: One node per class, sigmoid activation.

The figure below summarizes how to choose an activation function for the output layer of your neural network model.

**Github url :** https://github.com/Amirhossein1410/Activation_Function