**ANN**

**Elements of a Neural Network:**

**Input Layer:** This layer accepts input features. It provides information from the outside world to the network. The information is passed to Hidden Layers.

**Hidden Layer:**Nodes of this layer are not exposed to the outer world, they are the part of the abstraction provided by any neural network. Hidden layer performs all sort of computation on the features entered through the input layer and transfer the result to the output layer.

**Output Layer:** This layer brings up the information learned by the network to the outer world.

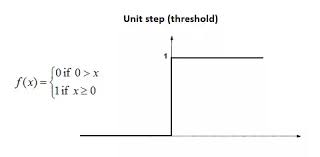
**Activation Functions**

**Definition:**Activation function decides, whether a neuron should be activated or not.

(A neural network without an activation function is essentially just a linear regression model).

1. Threshold:

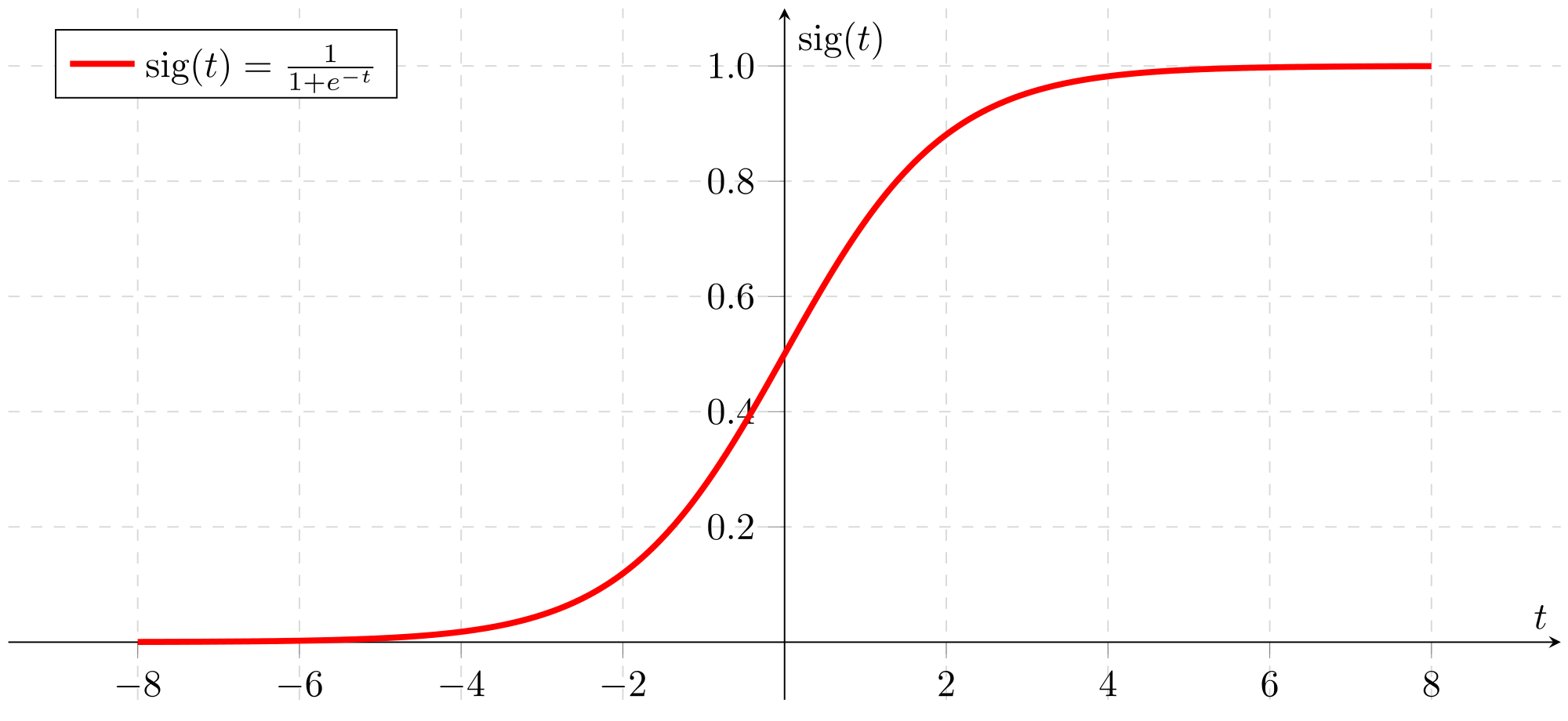
Activation function that gives 0 or 1 as output. Usually there is a *threshold* that decides the output value. For instance, the output value will be 0 if input is negative and for every non-negative (it includes 0 as well) input output will be 1.



1. Sigmoid:

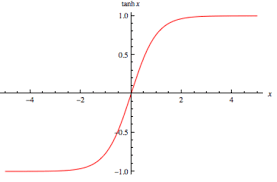
Activation Function that is like Threshold but has continuous values from 0 to 1. As it deals with real numbered values between 0 and 1, we can say that is gives probabilistic values in the output. Therefore, it is especially used for models where we have to **predict the probability** as an output. Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice. But they tend to *vanish* the gradient and are less convergent.

The function is **monotonic (i.e. entirely non-decreasing or non-increasing curve)**but function’s derivative is not.



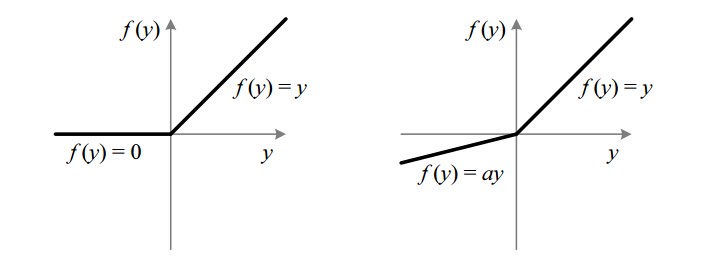
1. Hyperbolic Tangent:

It is the extensive form of sigmoid with activation values extending from +1 to -1. The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. This function is heavily prescribed in Optimization Problems but face the same problems as sigmoid, so they are less referred. *Both tanh and logistic sigmoid activation functions are used in feed-forward NNs.* The function is **monotonic (i.e. entirely non-decreasing or non-increasing curve)**but function’s derivative is not.



1. Rectifier (ReLU):

Rather practical in favorable environments, this activation function is very simplistic and provides the capability of efficient computation and solve the problems that sigmoid and hyperbolic tangent can’t. But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately. As you can see, the ReLU is half rectified (from bottom), f(z) is zero when z is less than zero (<0) and f(z) is equal to z when z is above or equal to zero (>=0). The function is **monotonic (i.e. entirely non-decreasing or non-increasing curve).** Its derivative is monotonic too.



1. Leaky ReLU:

The ReLU (or dying ReLU) doesn’t map negative values properly so there comes a modified version of ReLU i.e. Leaky ReLU. For negative values ‘z’ the function is f(z)=az. The leak helps to increase the range of the ReLU function (New Range = -infinity to +infinity). Usually, the value of **a**is 0.0d1 or so. The function is **monotonic** like ReLU**.** Its derivative is monotonic too.

Note:

The basic rule of thumb is if you really don’t know what activation function to use, then simply use *RELU* as it is a general activation function and is used in most cases these days.

Important Link:

<https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

**Cost Functions**

A cost function is a measure of "how good" a neural network did with respect to its given training sample and the expected output. We tend to minimize the cost function’s value, hence minimizing the error rate. There are many strategies to calculate (or minimize) the cost functions, one of the very important is Gradient Descent. E.g. C = ½ (y2-y1)

* Gradient Descent:

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (going downhill).

* Stochastic Gradient Descent:

GD can trap in a local minimum so we use Stochastic gradient descent instead of primitive GD. In this variation, the gradient descent procedure is run but the update to the coefficients is performed for each training instance, rather than at the end of the batch of instances. As it’s an iterative approach, so at the end we get global minima easily.

**Back Propagation**

The learning approach where we update the weights backwards after performing learning on the whole Neural Network initially. We complete traversing and learning (forward propagate) the neural network and in the end, we insert the end weights to the NN itself. The new updated weights minimize the cost, hence lowering the cost function (error rate) effectively.