[Activation Function](" https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044)   
We need a function that progressively changes from 0 to 1 with no discontinuity.

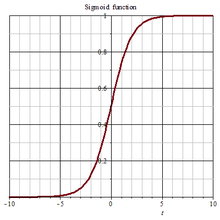
Mathematically, this means that we need a continuous function that allows us to compute the derivative.

Hence, we decided to add “activation functions” for this purpose. To check the resultant value produced by a neuron and decide whether outside connections should consider this neuron as “fired” or not. Or rather let’s say “activated” or not.

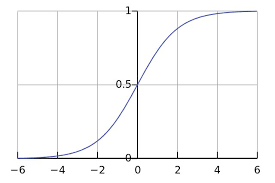
**Sigmod()**

**Sigmoid函數**得名因其形狀像**S**字母。

* [Logistic function](#)
* It mimics the one-hot encoded labels better than the absolute values.
* If we use the absolute (modulus) values we would lose information, while the exponential intrinsically takes care of this.
* [Hyperbolic tangent](#) (shifted and scaled version of the logistic function, above)



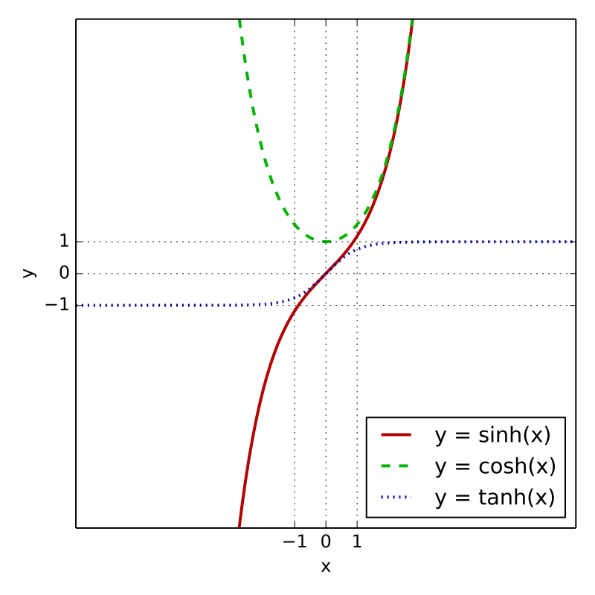
Softmax()



A more generalized logistic activation function that is used for multiclass classification called softmax function

Softmax turn logits (numeric output of the last linear layer of a multi-class classification neural network) into probabilities by take the exponents of each output and then normalize each number by the sum of those exponents so the entire output vector adds up to one — all probabilities should add up to one.

Tanh()



* It is nonlinear in nature, so we can stack layers
* It is bound to the range (-1, 1)
* The gradient is stronger for tanh than sigmoid ( derivatives are steeper)
* Like sigmoid, tanh also has a vanishing gradient problem.

vanishing gradients

The problem of vanishing gradients arises due to the nature of the backpropagation optimization

Gradients tend to get smaller and smaller as we keep on moving backward

Implies that neurons in earlier layers learn very slowly compared to neurons in the last layers

Vanishing Gradient Problem results in a decrease in the prediction accuracy of the model and take a long time to train a model.

Relu

ReLU is linear (identity) for all positive values, and zero for all negative values

Benefits of ReLU

* Cheap to compute as there is no complicated math and hence easier to optimize
* It converges faster. It accelerates the convergence of SGD compared to sigmoid and tanh (around 6 times).
* Not have vanishing gradient problem like tanh or Sigmoid function
* It is capable of outputting a true zero value allowing the activation of hidden layers in neural networks to contain one or more true zero values called Representational Sparsity

**Problems with ReLU**

* The downside for being zero for all negative values called dying ReLU. So if once neuron gets negative it is unlikely for it to recover. This is called “dying ReLU” problem
* If the learning rate is too high the weights may change to a value that causes the neuron to not get updated at any data point again.
* ReLU generally not used in RNN because they can have very large outputs so they might be expected to be far more likely to explode than units that have bounded values.