Activation functions

That function which takes weighted sum as a input and calculates the outputs for neurons like 1 or 0, -1 to 1 etc.

* Mathematically It is used to estimate non-linearity of an input.
* **Example** – Neuron which is more activated(features) than it will learn it more by giving output as 1 else 0.
* Used to normalizing the data.

1). **Sigmoid** - It always gives an output between range 0- 1.

* So, if we want probability in an output, we can use Sigmoid in the model like O/P layer of NN.
* For every -ve value of x it will give output 0 and for +ve values it give 1 as output. Simply means positive values are more activated.
* zero centric function hence help in normalizing the data

Disadvantages -

* We can get the slope at any two point hence it is differentiable. But at certain point of time it won’t converge and hence vanishing gradient problem.
* It is exponential function hence slow



**2). Tanh or hyperbolic tangent –** - It always gives an output between range -1 - 1.

* F(x) = e^(x) – e^(-x) / e^(x) –+e^(-x)
* So, advantage of using this function is it is strongly gives the output og negative values as -1 and for positive +1
* Used for classification between two classes. Used in hidden layers
* zero centric function hence help in normalizing the data

Disadvantages -

* We can get the slope at any two point hence it is differentiable. But at certain point of time it won’t converge and hence vanishing gradient problem.



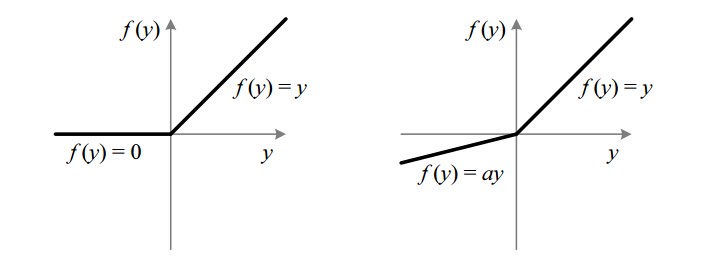
**3). ReLU or Rectified Linear Unit -** - It always gives an output between range 0 -- Infinite (max(0,value))

* F(x) = max(0,x)
* As It is linear so much faster than Tanh and Sigmoid
* No Vanishing Gradient problem for positive values.
* It turns negative values as 0 so for negative values It doesn’t fit properly.
* Not a zero centric function hence don not help in normalizing the data

****

**4). Leaky ReLU –** when there is a leak a = 0.01 in the ReLU.

* Max(0,x) for positive values
* For -ve values its 0.01 , hence it will adjust negative values too.

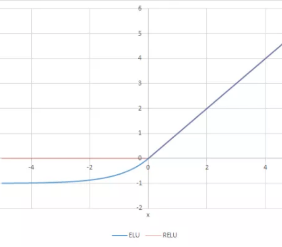


**5). PReLU –** For every -ve value there is alpha(or “a” ) learnable paramtere with ReLU.

F(x) = x , x>0 and ax , x<=0 ; here a is a learnable paramter, not a constant just like above

**6). ELU or Exponential Linear Unit -** Similar to ReLU but it has some learnable parameter alpha with e^(x) -1 output for negative values.

* F(x) = x , x>0
* Alpha \* (e^(x)-1) , x<=0
* Zero centric function
* bit exponential for negative values

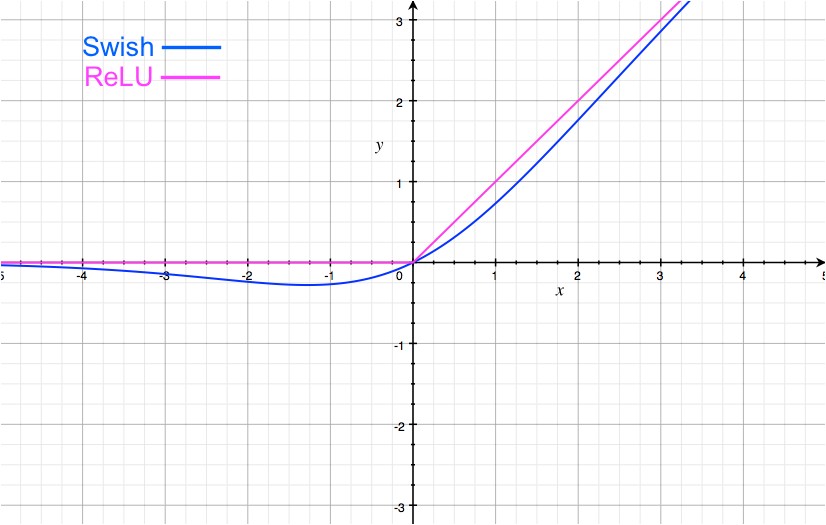


**7). SoftMax** – Used for Muticlass classification in the output layer.

* Softmax(x(i)) = e^(x(i)) / sum(e^(xi))
* It gives the probablity of occurenace of each target classs over possible target class.

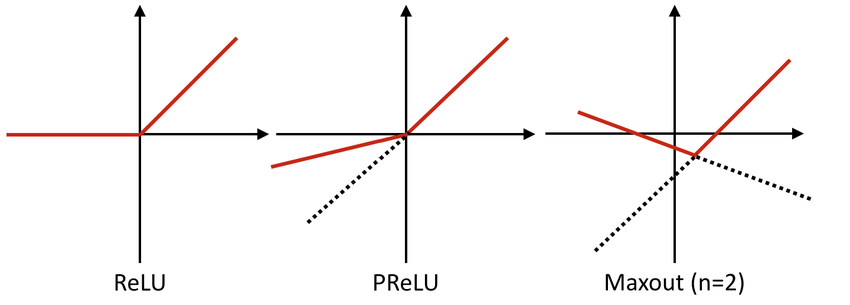
**8). Swish or A self Gated function –**

* F(x) = x \* sigmoid(x)



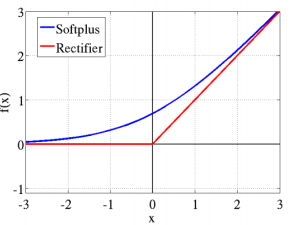
**9). Maxout - max(w1x1 +b1 , w2x2 +b2)**

* It first learn then find the value
* Complexity is linear
* Just like a combination of reLU+Leaky ReLU



**10).Soft Plus - log(1+exp(x))**

* Providing more smoothness toh ReLU



**LOSS FUNCTIONS**

In General, to calculate the cost/error of the model we use Loss functions.

Let assume y’ is the predicted value and y is the original value for some random variable x then **error or cost = y’-y .**

**Why we are doing so?**

Because our main goal is to reduce the error / converge to global minima then only we can say we have predicted the right value which is approximately equal to original y.

**1). L1 loss functions –** Least absolute deviation

* L1 loss = sum 1->N |y-y’|
* Advantage in case of a smaller number of outliers. As we are calculating absolute.

**2). L2 Loss functions - Least square error**

* L2 loss = sum 1-> N (y-y’) ^2
* In case of huge number of outliers, we are calculating squares hence it will increases error.

**3). Huber Loss** – It will control the disadvantage of L2 Loss , delta =1 or

* **L(x,f(y)) = { ½ (y-y’) ^ 2 if |y-y’| < delta}** or **{delta . |y-y’| - ½ delta ^2 otherwise}**

**4) Psudeo Huber Loss –** It is used to provide smoothing Huber loss function

* L = delta ^(sqr (1+(a/delta)^2) -1)

**5). Hinge Loss –** It is used inside a classifier

* Loss(y) = max (0,1-t.y) where t = number of classes

**6). Cross Entropy** - Used for binary classification problems

**Loss = - sum 1->n ( tn (log(yn))+ (1-tn)log(1-yn))** where t is the binary class 0/1 or -1 /1

**7). Sigmoid cross entropy** - or Binary cross entropy

It is a sigmoid activation + cross entropy

**L = -log(f(Sigmoid)) if t =1** or

**L = -log(1-f(sigmoid)) if t=0**

**8). Softmax cross entropy –**

L = -t + log sum(e^f) = - log (e^f / sum(e^f))

**9). Mean Square Error –**

It is the mean of square difference of prediction y’ and original y

MSE = 1/n sum 1->n(y-y’)2

**10). Root mean square error** – to avoid big square term

RMSE = sqrt(MSE)