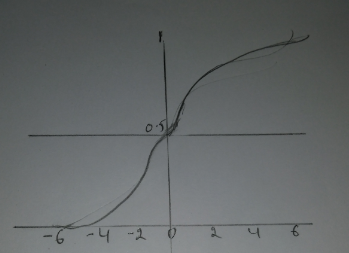
**Activation Functions:**

Activation function are something which maps a particular output to a particular set of inputs. So they are used for containing the output in between zero to one or any given values.

They are also used to impart a non-linearity and they are one of the important factors which affect your results and accuracy of your model.

Sigmoid:

Sigmoid function is a type of logistic function, it is having a 'S' shaped curve, it is also known as Squashing function.



**Formula:**



**Usage:**

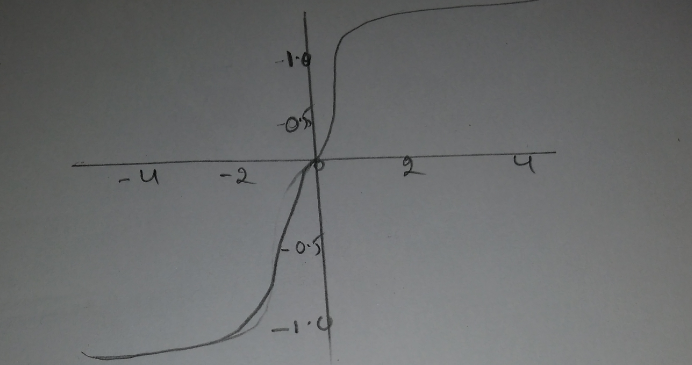
Sigmoid function takes the input between 0 and 1 , making it simpler to infer the output

This function is not often used in hidden layer as it **vanishes the gradient decent** and it uses the **exponential operation which takes time for calculation**

The vanishing gradient decent problem is when the magnitude of the input to the activation function becomes sufficiently large such that the derivative of the function approaches to zero. This causes of the network to stop learning during back propagation, in this situation we adjust the weights.

**TanH:**

tanh:(Hyperbolic tangent function): It used as a nonlinear activation function between the layers of neural network, it takes the values between -1 and 1



**Formula:**



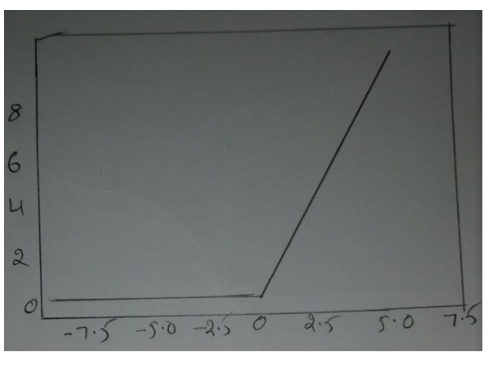
* It’s output is zero centered because its range in between -1 to 1 i.e.. -1 < output <1
* Optimization can be done easily using tanh function, hence is always preferred over sigmoid function, still it has vanishing gradient decent problem
* It usually used in hidden layer of neural network as it takes the values between -1 to 1 , hence the mean of hidden layer comes out be 0 or very close to it, hence it helps to bring the mean to 0 i.e.. centering the data, this helps to next layer in the network to learn quickly

**ReLu:**

ReLu (Rectified Linear Unit): it is a nonlinear activation function, it takes the values between [0,inf].it is a Nonlinear function, which means that it can easily back propagate the errors and it activates multiple layers of neuron. ReLu is a less computational function compare with Sigmoid and TanH function

**Formula:**





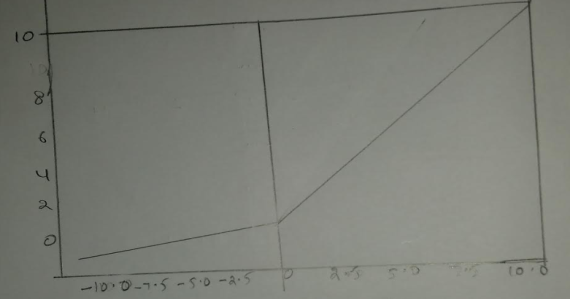
* It involves a simple mathematical operation
* At a time it activates few neurons and makes the network efficient for computation
* It avoids vanishing gradient problem, most of the deep learning models use ReLu function
* Problem with ReLu function is that some gradients can be breakable during training and can die.
* It can cause a weight update which will make a never update at any data point again. ReLu could result in dead neuron

**Leaky ReLu:**

To fix Dying Relu problem, we have to use LeakyRelu function, it introduces small slope to keep updates live, instead of function being 0 when x<0, leaky ReLu instead have small negative slope ( of 0.10, or so)

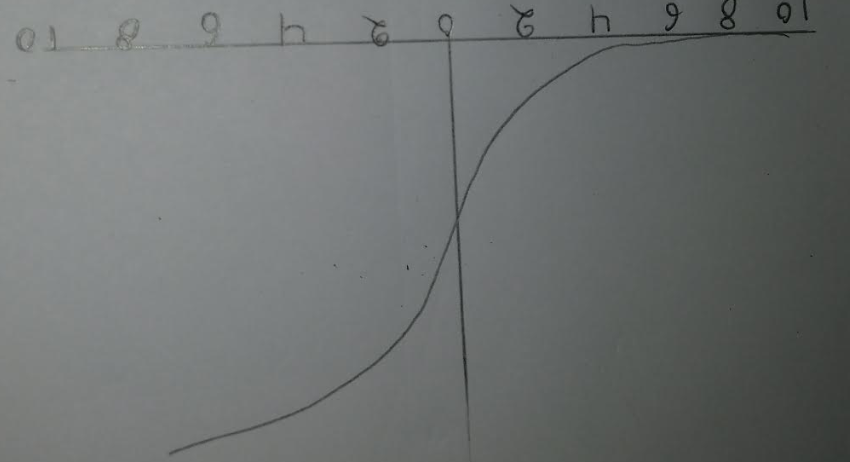


Where alpha is small constant

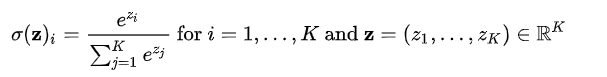


**Softmax:**

Softmax function also known as **softargmax** function, this function takes input a vector of K real numbers and normalize it into a probability distribution consisting of K probabilities of proportion to the exponentials of the input numbers, it does normalization on vectors, it maps a vector of unconstrained real number into a vector of small values (0..1) that sum to one.

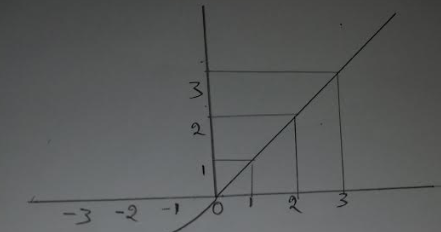


Formula:



* If element i is greater than element j, that property is preserved in the output
* Adding a constant input\*c doesn’t change the output

**ELU (Exponential Linear Unit) :**



**Formula:**

Y=a(ex-1)

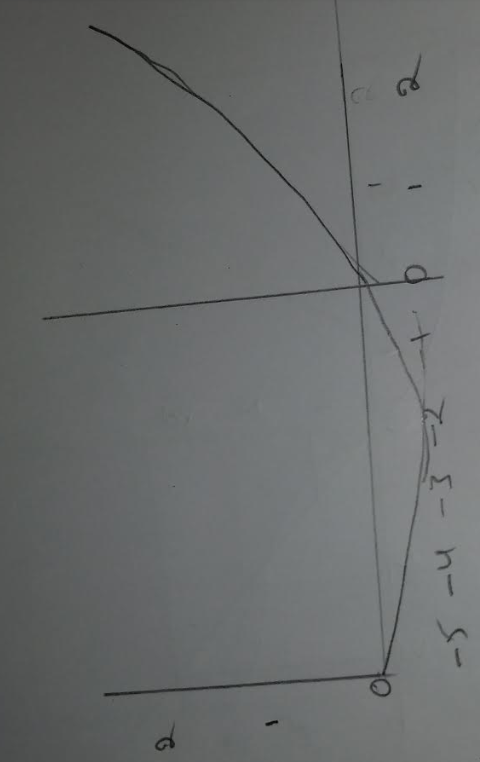
* ELU Function that tend to converge cost to zero faster and produce more accurate results
* Doesn’t have the dying ReLu problem
* More of a merge between good features of ReLu and Leaky ReLu
* It saturates for long negative values

**Swish:**

Swish function is non-monotonic function, that consistently match or outperforms ReLU.

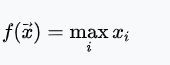
Formula:



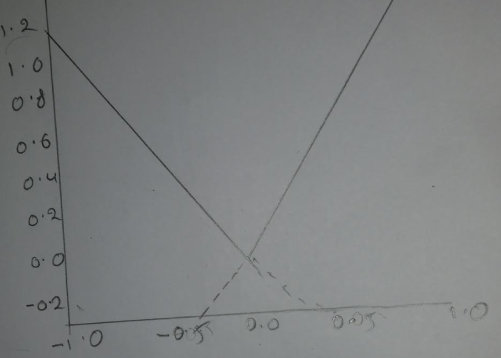


**Maxout:**

Maxout function is a generalization of the ReLU and the Leaky Relu functions. It is a learnable activation function. It is piecewise linear function that returns the maximum inputs, designed to be used in conjunction with the dropout of regurgitation technique



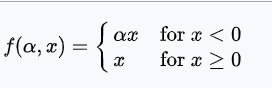
Maxout function have all benefits of a ReLU neuron without having to be a dying ReLU. Its only downside is that would require double for the number of parameters, so it would have high total number of parameters

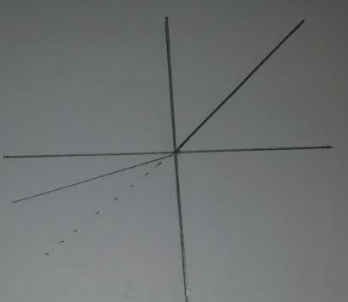


**PReLU: (Parametric ReLU)**

PreLU has the goal to increase the learning speed by not deactivating some neurons. If we can learn small value during training so that our activation function can better adapt to the other parameters. This is where PReLU comes in . we can learn the slope parameter using back propagation at a negligible increase in the cost of training

**Formula:**





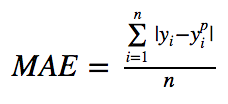
**Loss Functions:**

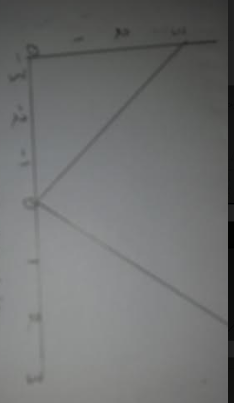
A Loss function or cost function is a function that maps an event or values of one or more variables onto a real number represents some "Cost" associated with the event

**L1 Loss function: (Mean Absolute Error) :**

This function used for regression models. MAE is the sum of absolute difference between our target and predicted variables. It measures the average magnitude of errors in a set of predictions without considering their directions, The range is also 0 to ∞

**formula:**

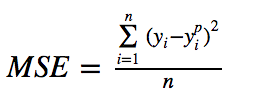


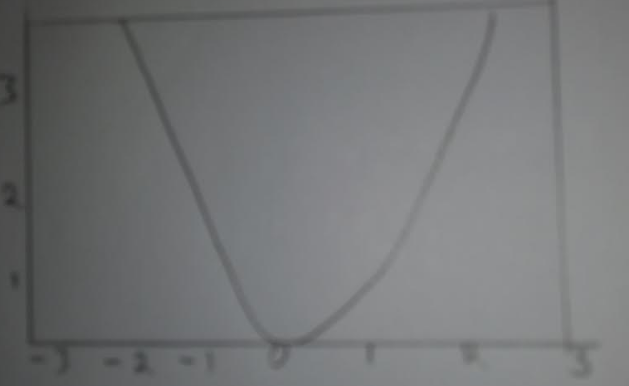


**L2-Loss Function (Mean squared Error)**

Mean squared error is the most commonly used regression loss function. MSE is the sum of squared distances between our target variable and predicated values

**Formula:**



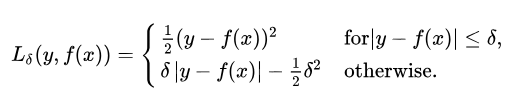


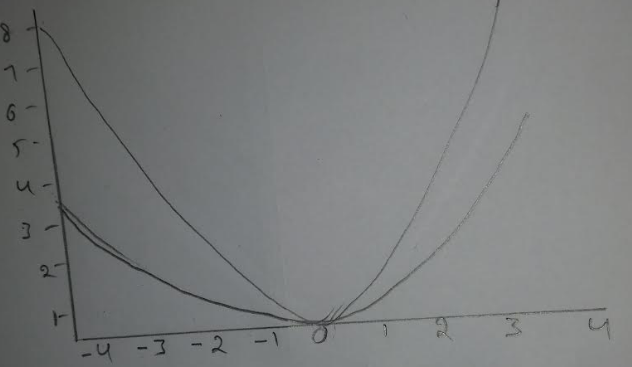
**Huber Loss:**

Huber loss function used in robust regression, that is less sensitive to outlier in data than the squared error loss. It's also differentiable at 0. It's basically absolute error, which comes quadratic when error is small.

* Using MAE( L1 Loss) for training of neural networks is its constantly large gradient, which can lead missing minima at the end of training using gradient descent. for MSE (L2 loss) gradient decreases as the loss get close to its minima, making it more precise
* in such cases Huber loss can really helpful, as it curves around the minima which decreases the gradient and it's more robust to outlier to MSE. Therefore it combines good properties from both MSE and MAE.

**Formula:**





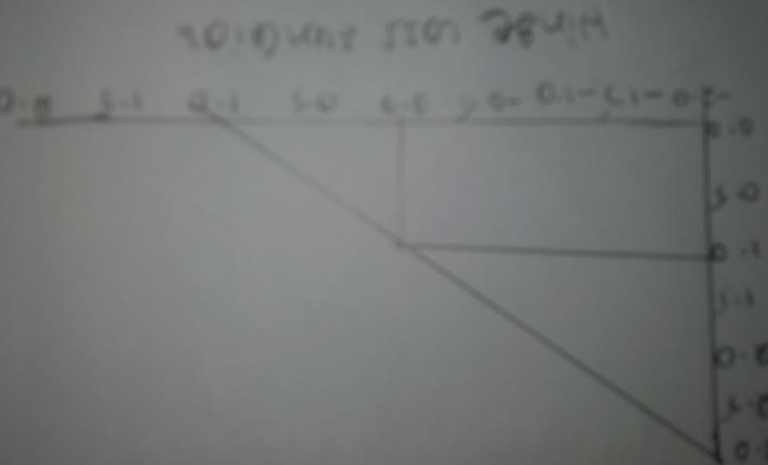
**Hinge loss:**

Hinge loss function is used for training classifiers. The hinge loss is used for "maximum-margin" classification most notably for support vector machines (SVMs). for an intended output t=+/-1 and a classifier score y, the hinge loss of the prediction y is defined as



here y is a raw output of the classifiers decision not the predicted class label

Hinge loss function is easy to compute, Hinge loss can also be faster to train via SGD. since much of the time the gradient is 0 so you won't have to update weights



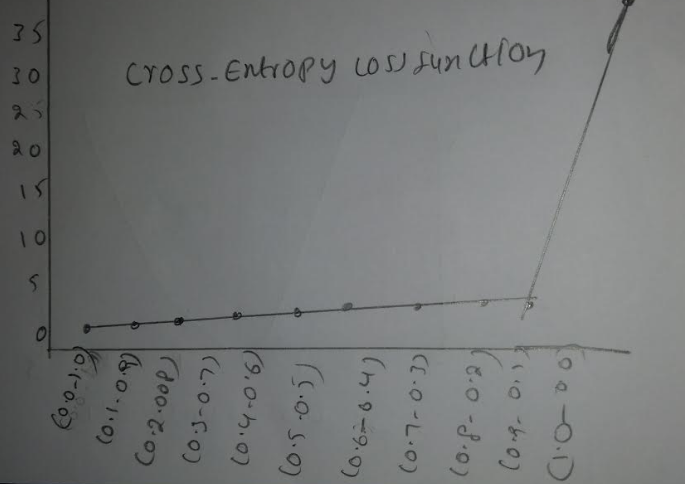
**Cross Entropy:**

Cross entropy function between two probability distribution p with respect to q measures how many bits you need on average to encode data from with the code that is optimal for q.

it helps to predict the class probabilities and MSE to predict values.

**Formula:**





**Sigmoid cross entropy loss:**

It is a sigmoid activation plus a cross entropy loss. Unlike softmax loss it is independent for each vector component meaning that the loss computed for every CNN output vector component is not effected by other component values.

**Softmax cross entropy loss:**

The softmax classifier is a liner classifier that uses the cross-entropy loss function. In other words the gradient of the above function tell a softmax classifier how exactly to update it weights using some optimization like gradient descent

It normalise the network predictions so that they can be interpreted as probabilities

cross entropy indicates the distance between what the model believes the output distribution should be, and what the original distribution is. Cross entropy is widely used alternative of squared error.

