

Activation Function:

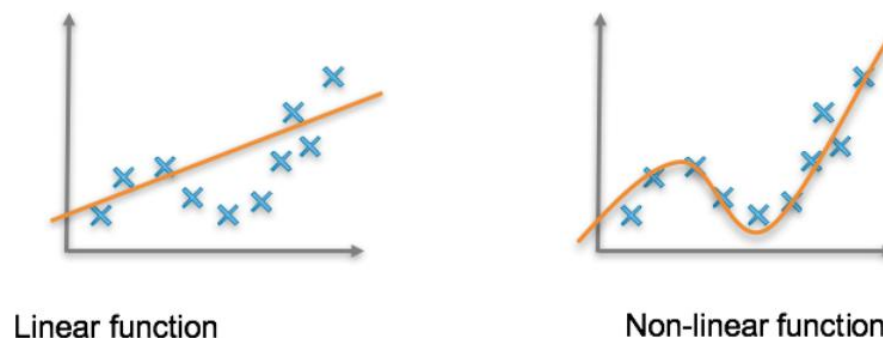
An activation function is a non-linear function applied by a neuron to introduce non-linear properties in the network.

Activation Function behaves like gate and normalizes the data.

Activation function understands the complex data because of non-linearity.

**Linear:** A relationship is linear if a change in the first variable corresponds to a constant change in the second variable. Linear Activation function does not capture complex pattern. The input and output values are very large. Equation is:  $f(z) = \alpha z$ . where  $\alpha$  is constant.

**Non Linear:** A relationship is nonlinear if first variable doesn't necessarily correspond with a constant change in the second. It may impact each other but it appears to be unpredictable.



Why Zero Centric function is preferred in Neural Network?

Zero Centric Functions will always ensure that the mean activation value is centred around zero.

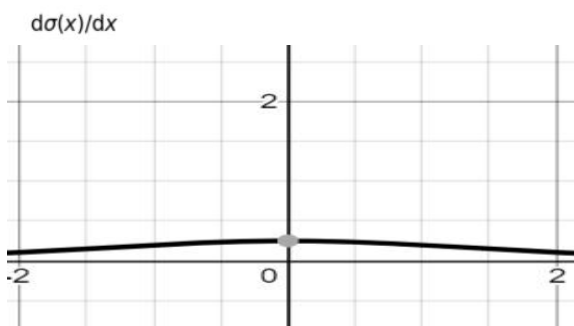
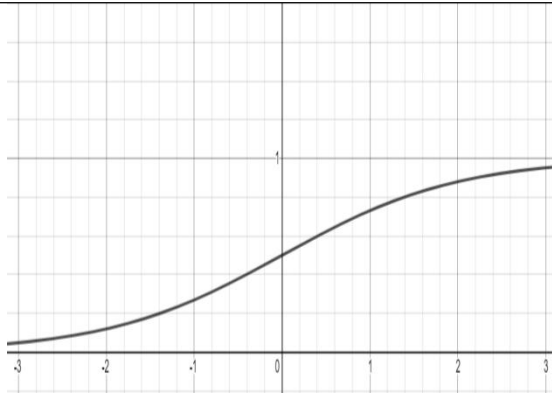
This is important in deep learning because models operating on normalized data (Input or latest activations) converges faster.

There is an exception in this category like tanh where it will saturate at their asymptotes and gradient will get vanish smaller over the time leading to the weak training signal.

Relu, although is not zero centric, avoids this problem.

There are several techniques like batch normalization, weight normalization to mitigate this type of issue.

<b>Non Linear Activation Functions:</b>	
<b>Sigmoid:</b>	
$\sigma(x)$	<ul style="list-style-type: none"><li>• Sigmoid is non-linear function therefore it can capture more complex patterns.</li><li>• Sigmoid function can be defined as</li></ul>



firing rate of neuron. In middle of slope it is large and sensitive to neuron. Whereas on the side of slope is gentle and is inhibitory area.

- Sigmoid is smoothing function and easy to derive.

Equation:

$$\frac{1}{(1 + e^{-x})}$$

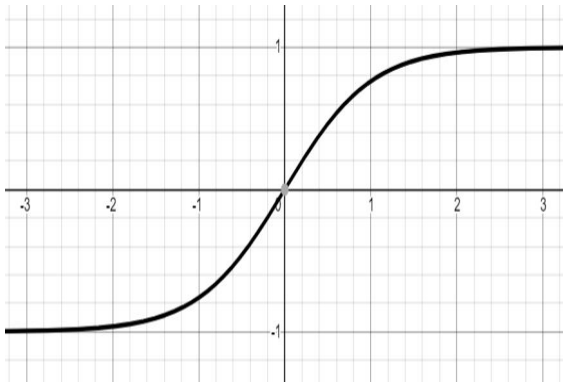
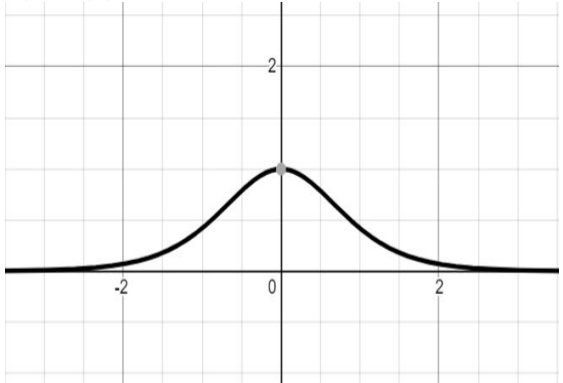
Output Range: 0 to 1

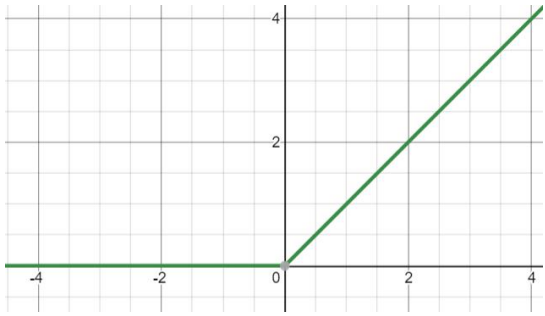
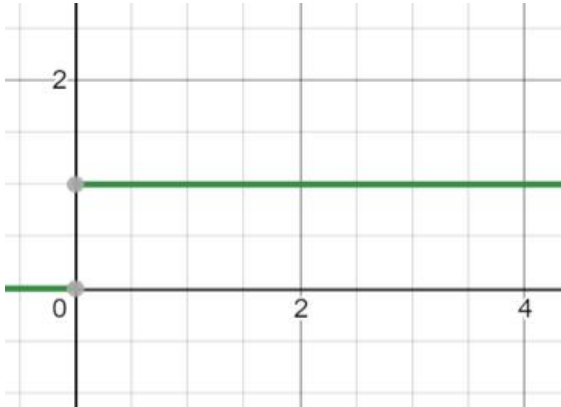
#### Advantages:

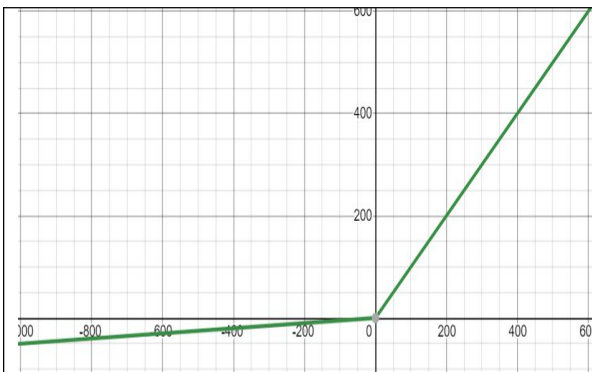
- Sigmoid is smooth function and thereby preventing fluctuations in output values
- The output of sigmoid is stable and between 0 and 1. Sigmoid ensures the normalization of output of coming out from each neuron.
- By looking at the Sigmoid output in left, we can say that got the x range between -2 and 2, y values are close to 0 and 1 respectively. Therefore Sigmoid provides clear prediction.

#### Disadvantages:

- The function output is not zero centred, which reduces weight updation efficiency.
- As we see from the mathematical equation, sigmoid have exponential function, which takes larger time for computation, which makes the operation slow and increasing power consumption
- **Vanishing Gradient:** There is no or negligible change in prediction for very small values of input. This will lead to non-convergence of output to global minima and network stops learning thereby giving incorrect prediction.
- **Gradient saturation or Gradient dispersion:** When there is slight shift from coordinate origin, the gradient becomes very negligible near to zero. In back propagation, differential of chain rule is applied to calculate derivative of weight and biases. During the calculation, we pass the derivative to sigmoid

	<p>function, the output coming becomes very small, thereby making differential chain becomes very negligible. When it passes through multiple layer, it passes through many sigmoid functions, which will make little or no effect on loss function. Therefore weight optimization becomes difficult.</p>
<b>Tanh:</b>	
<p>Tanh(x)</p>  <p>d(tanh(x))/dx</p> 	<ul style="list-style-type: none"> <li>Tanh (hyperbolic tangent) is non-linear function therefore it can capture more complex patterns. This is better than sigmoid and most of features are similar with Sigmoid.</li> </ul> <p>Equation:</p> $\frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ <p>Output Range: -1 to 1</p>
<p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>Better than Sigmoid</li> <li>Unlike Sigmoid, Tanh is zero centric function.</li> <li>The output is bounded which prevents getting large values</li> <li>This is zero centered which makes model input to have strong positive, strong negative and neutral values.</li> <li>Used in binary Classification with Tanh as hidden layer and sigmoid as output layer.</li> </ul>	<p><b>Disadvantages:</b></p> <ul style="list-style-type: none"> <li>All disadvantages of Sigmoid are applicable.</li> <li>Tanh is not good for DNN because of vanishing gradient issue.</li> <li>Although Tanh is Zero centred, but for large or small input, the output becomes smooth and gradient is small which is unfavourable for weight update during back propagation.</li> </ul>

<b>RELU(Rectified Linear Unit)</b>	
<p>Relu</p>  <p>dRelu/dx</p> 	<ul style="list-style-type: none"> <li>• Relu is one of most popular function which filled all the shortcomings of its predecessor like sigmoid and Tanh.</li> <li>• Major problem of gradient saturation in deep neural network is solved by relu. Now DNN with more than 100 layers can easily be trained.</li> <li>• Mathematically it takes maximum value and is not fully interval derivable, but can take sub-gradient.</li> </ul> <p>Equation: <math>\text{Max}(0, x)</math></p> <p>Output Range: 0 to positive infinity</p>
<p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>• No gradient saturation problem because the value is always positive.</li> <li>• This is computationally efficient in both directions because it allows network to converge very fast. The derivate of Relu is Step out which is used in back propagation</li> </ul>	<p><b>Disadvantages:</b></p> <ul style="list-style-type: none"> <li>• Dead Neuron/Dyeing ReLu problem: When the input is negative, the gradient becomes zero. The network can perform back propagation, but cannot learn.</li> <li>• The neuron is completely useless- it outputs 0 regardless of which training example comes in, and no matter how much training, it will always output 0</li> <li>• This is non centric function because the input is always 0 or positive number.</li> </ul>
<b>Leaky RELU(Leaky Rectified Linear Unit)</b>	
<p>Leaky RELU</p>	<ul style="list-style-type: none"> <li>• To Overcome the issue with Dead Neuron/Dead ReLU problem, researchers proposed that values from 0 to negative infinity should have 0.01x in place of 0.</li> <li>• It inherits all the features from ReLu</li> </ul>



$d(\text{Leaky Relu})/dx$



- Theoretically, it solved the issues with Dead Neuron/Dead ReLU issue but practically it was not fully proved.

Equation:  $\text{Max}(0, x)$

Output Range: negative infinity to positive infinity

#### Advantages:

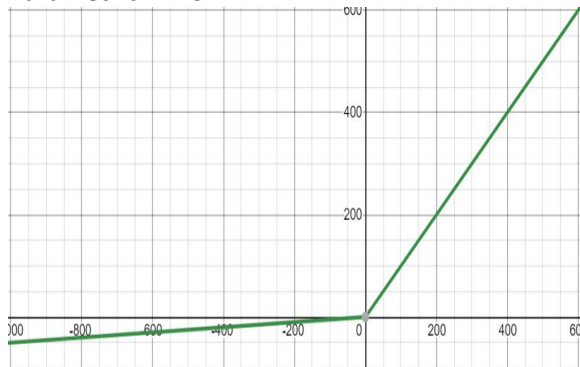
- Prevents the dying ReLU problem with the introduction of small positive slope in negative area, thereby enabling back propagation even for negative values.

#### Disadvantages:

- Although it enabled the back propagation for negative values but failed to provide consistent predictions for negative input values.

#### Parametric ReLU(Parametric Linear Unit):

##### Parametric ReLU

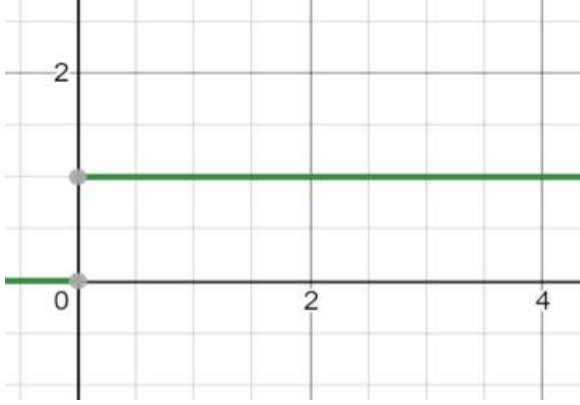
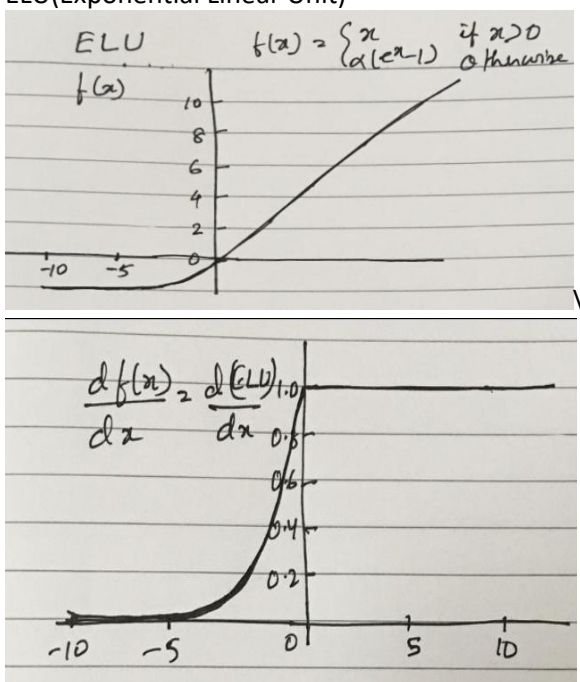


$d(\text{Parametric Relu})/dx$

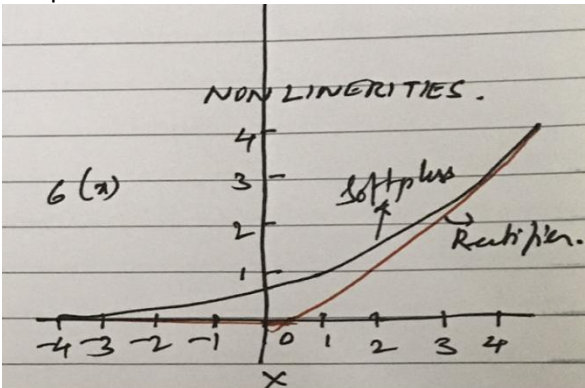
- Parametric ReLU is one step ahead of Leaky where it introduced learnable parameter  $\alpha$ , which is learnt in back propagation.

Equation:  $\text{Max}(\alpha(x), x)$

Output Range: negative infinity to positive infinity

	
<b>Advantages:</b> <ul style="list-style-type: none"> <li>Parametric Relu provided learnable parameter <math>\alpha</math> for the negative part of the slope. Therefore in back propogation it learnt most of the values of <math>\alpha</math></li> <li>Inherits all the features of ReLu</li> </ul>	<b>Disadvantages:</b> <ul style="list-style-type: none"> <li>The result varies from problem to problem and is not consistent.</li> </ul>
<b>ELU(Exponential Linear Unit)</b>	<ul style="list-style-type: none"> <li></li> </ul>
<p>ELU(Exponential Linear Unit)</p> 	<ul style="list-style-type: none"> <li>ELu inherits all features of ReLu and is designed to solve the problem of Dead Relu</li> </ul> <p>Equation: refer to diagram</p> <p>Output Range: negative infinity to positive infinity</p>
<b>Advantages:</b> <ul style="list-style-type: none"> <li>No Dead Relu/Dead Neuron issues.</li> <li>Gradient Clipping (output close to zero) is avoided.</li> <li>Zero centred Mean</li> </ul>	<b>Disadvantages:</b> <ul style="list-style-type: none"> <li>Computationally intensive</li> <li>The result varies from problem to problem and is not consistent.</li> </ul>
<b>Softmax/Logit</b>	

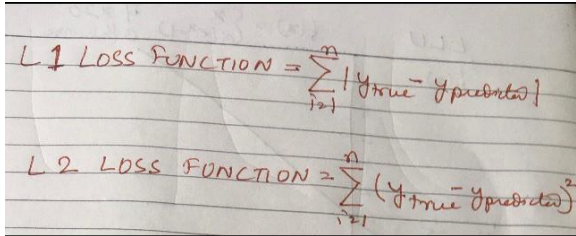
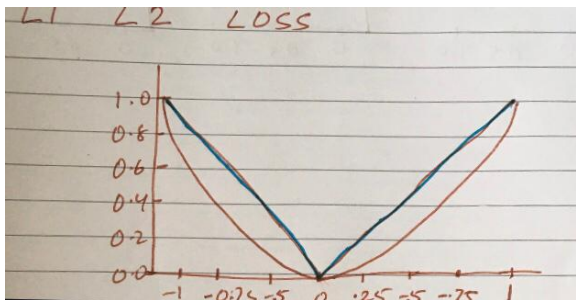
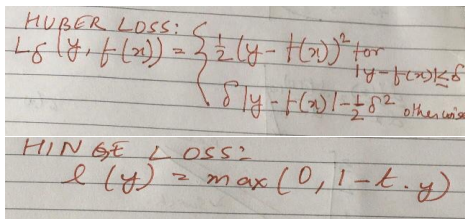
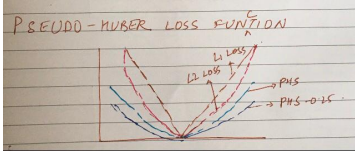
<p>Softmax/Logit</p> <div data-bbox="229 259 834 546"> <p> <math display="block">\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}</math> </p> <p>Range: 0 to 1</p> <p>Divides output so that the total sum of the output is equal to 1</p> </div>	<ul style="list-style-type: none"> <li>Softmax is probabilistic function which takes considers smallest and largest value as an output. Softmax consider smaller value should have smaller probability and is different from normal max function.(refer to mage in left)</li> <li>Softmax is mainly used in Multiclass Classification.</li> <li>Softmax behaves like Sigmoid in binary classification</li> </ul> <p>Equation: refer to diagram</p> <p>Output Range: 0 to 1</p>
<p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>Softmax is able to handle multiple class and only one class in other activation functions, normalizes the output between 0 and 1, divides the sum, thereby giving probability of input value to specific class.</li> <li>Softmax is used in output layer for multiple categories.</li> </ul>	<p><b>Disadvantages:</b></p> <ul style="list-style-type: none"> <li>Softmax in outer layer should become risky for medical and mission critical prediction. Because probability of 0.03 can be fatal.</li> </ul>
<p><b>Swish</b></p>	
<p>Swish</p> <div data-bbox="189 1196 791 1514"> </div>	<ul style="list-style-type: none"> <li>Swish is self-gated activation function used in LSTM and higher network.</li> <li>Seft gating is mechanism where same value is used</li> <li>Swish was discovered by researchers at google. Swish performs better than ReLu with similar.</li> <li>In ImageNet competition, it performed better in achieving accuracy of 0.6 to 0.9 higher.</li> </ul> <p>Equation: <math>x * \text{sigmoid}(x)</math></p> <p>Output Range: negative infinity to positive infinity</p>
<p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>The derivate of function will always give changing some value which makes back propagation smooth and better than Relu.</li> <li>Single scalar input is used in self-gating rather than normal gating requires multiple inputs, which makes Swish function to easily replaceable like</li> </ul>	<p><b>Disadvantages:</b></p> <ul style="list-style-type: none"> <li>The output is not bounded which prevents in attaining strong regularization and resolution of larger negative inputs</li> </ul>

<p>Relu, without disturbing the function internal working</p> <ul style="list-style-type: none"> <li>• No vanishing Gradient and always give smooth curve which helps in optimization and generalization</li> <li>• The output is not bounded which prevents the gradient approaching to zero in slow training causing saturation.</li> </ul>	
<p><b>Maxout</b></p>	
	<ul style="list-style-type: none"> <li>• Maxout was introduced by GoodFellow and it generalizes Relu and Leaky ReLu.</li> <li>• The Maxout neuron has all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks</li> <li>• This is learnable activation function.</li> <li>• Maxout can be seen as adding a layer of activation function to the deep learning network, which contains a parameter k.</li> <li>• Compared with ReLU, sigmoid, etc., this layer is special in that it adds k neurons and then outputs the largest activation value.</li> </ul> <p>Equation:</p> $\max(w_1^T x + b_1, w_2^T x + b_2)$
<p><b>Advantages:</b></p> <ul style="list-style-type: none"> <li>• The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks</li> <li>• This is learnable activation function.</li> </ul>	<p><b>Disadvantages:</b></p>
<p><b>Softplus</b></p>	
<p>Softplus</p> 	<ul style="list-style-type: none"> <li>• The Soft plus is similar to ReLu and relatively smooth.</li> <li>• Soft plus supresses negative value like Relu.</li> </ul> <p>Equation: <math>f(x) = \ln(1+\exp x)</math></p> <p>Output Range: 0 to positive infinity</p>



<b>Advantages:</b> <ul style="list-style-type: none"> <li>Inherits all features of Relu.</li> <li>Relatively smooth curve.</li> <li>Output is always positive</li> </ul>	<b>Disadvantages:</b>
<b>Best Practices:</b> <ul style="list-style-type: none"> <li>Sigmoid functions and their combinations generally work better in the case of classifiers</li> <li>Sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem</li> <li>ReLU function is a general activation function and is used in most cases these days</li> <li>In case of dead neurons in our networks the leaky ReLU function is the best choice</li> <li>ReLU function should only be used in the hidden layers</li> <li>As a rule of thumb, you can begin with using ReLU function and then move over to other activation functions in case ReLU doesn't provide with optimum results.</li> </ul>	

**Loss Function:** Loss function helps in optimizing the parameters of neural network. Loss Function minimizes the loss of neural network by optimizing the weights in back propagation. The loss is difference of target and predicted value.

<b>Loss Functions</b>	
<p><b>L1 and L2 loss:</b></p>  	<ul style="list-style-type: none"> <li>L1 and L2 are mainly used in machine learning.</li> <li>L1 loss function is known as Least Absolute Deviations (LAD) and L2 loss function is known as Least Square Error (LS)</li> <li>L1 Loss function is used to minimize the error which is sum of all the absolute differences between true value and predicted value.</li> <li>L2 Loss function is used to minimize the error which is sum of all the squared differences between true value and predicted value.</li> <li>Drawback: In case of outliers the L2 norm will account for main component of the loss.</li> <li>If the residual is small, the loss function is L2 norm, and when the residual is large, it is a linear function of L1 norm</li> </ul>
<p><b>Huber Loss:</b></p> 	<ul style="list-style-type: none"> <li>Huber loss is used in regression problem.</li> <li>Huber loss is less sensitive to outlier</li> <li>The pseudo Huber loss provides smooth approximation of Huber loss.</li> </ul> 

**Hinge Loss:**

HINGE LOSS:

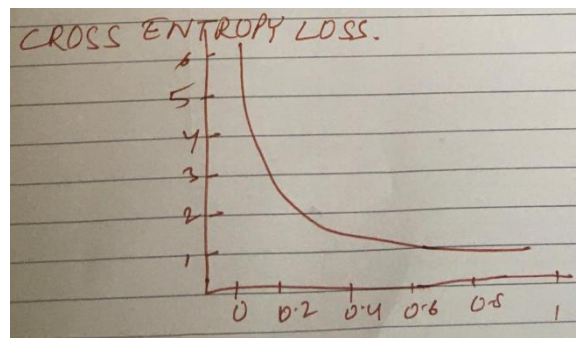
$$l(y) = \max(0, 1 - t \cdot y)$$

- Hinge loss is used in binary classification
- For ground true:  $t = 1$  or  $-1$ ,  $t = 1$  or  $-1$ , predicted value  $y = mx + c$
- Used for maximum-margin classification for SVM

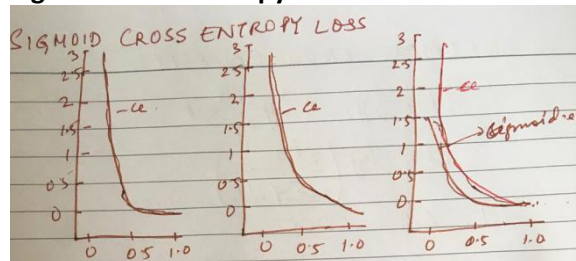
**Cross-entropy loss:**

CROSS ENTROPY LOSS

$$J(w) = -\frac{1}{N} \sum_{n=1}^N [y_n \log \hat{y}_n + (1 - y_n) \log (1 - \hat{y}_n)]$$



- Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1 where  $t_n$  lies between 1 and 0 respectively.
- Cross-entropy loss increases as the predicted probability diverges from the actual label.

**Sigmoid-Cross-entropy loss:**

- It is a Sigmoid activation plus a Cross-Entropy loss.
- It is independent for each vector component (class),
- The loss computed for every CNN output vector component is not affected by other component values.

**Softmax cross-entropy loss:**

SOFTMAX CROSS ENTROPY:

$$L_i = -f y_i + \log \sum_j e^{f_j}$$

$$L_i = -\log \left( \frac{e^{f_i}}{\sum_j e^{f_j}} \right)$$

- Softmax also implements a vector of 'squashes'  $k$ -dimensional real value to the  $[0, 1]$  range of  $k$ -dimensional, while ensuring that the cumulative sum is 1.
- Softmax-cross-entropy-loss uses a Softmax function to convert the score vector into a probability vector.

**Binary Cross-entropy:**

- Only one output node to classify the data into two classes.
- The output value should be passed through a *sigmoid* activation function and the range of output is  $(0 - 1)$ .

<b>Categorical Cross-entropy:</b>	<ul style="list-style-type: none"> <li>• This is used for multi-class classification task.</li> <li>• There must be the same number of output nodes as the classes.</li> <li>• In final layer output should be passed through a <i>Softmax</i> activation so that each node output a probability value between (0–1).</li> </ul>
<b>Sparse Categorical Cross-entropy:</b>	<ul style="list-style-type: none"> <li>• This is similar to categorical cross entropy.</li> <li>• There is no need to one hot encode the target vector. If the target image is of a dog, you simply pass 1.</li> </ul>

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

Single Label and Multiple Classes: multiple classes (e.g. objects in an image, topics in emails, suitable products to advertise) and they are exclusive — each item only has one label