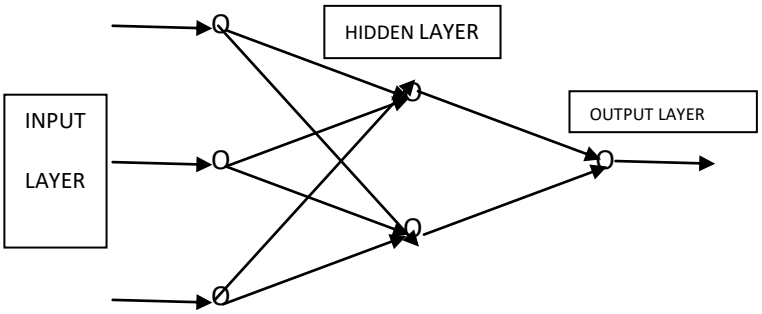
1.0 Activation Function

In neural network, whenever we want to send a data set as input to the hidden layer, it must require an activation function. An activation function behaves like a GET function that restricts the kind of output generated at a particular time. This makes it want to normalise a data set when a high input data set is sent into the network. It will try to give you a data set range between 0 and 1 and other activation functions may give between -1 to 1 depending on your model and choice of function.



The activation function plays a major role in the whole model training process in the sense that it gives non-linear data set or helps to understand complex data set which is key in determining the output of any particular model to be trained.

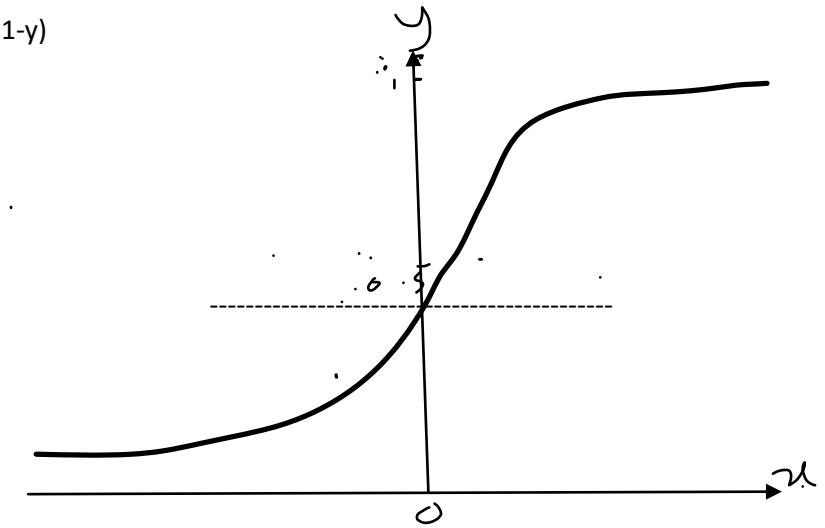
1.1 Sigmoid Activation Function

The sigmoid function is not a probability function as people confused it to be and it does not give data in form of probability. Being one of the most widely used activation function, only produces positive numbers between 0 and 1. In neural network data training, sigmoid function is most suitable when data between 0 and 1 is to be trained. Its activation function is non-symmetric. This can be mathematically represented as;

span: 0 < y < 1;

y = 1/ (1+ exp (- 2 *s* x)),

d = 2 * x * y * (1-y)

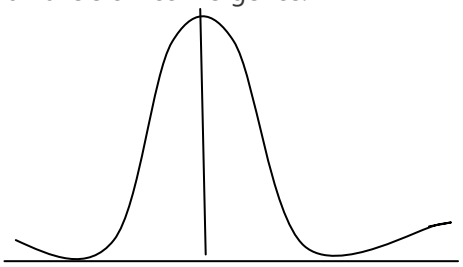


Advantages of sigmoid function

- its easier to train
- finds general solution better i.e does well with untrained data

However, as popular as this function is, there are some draw back attached to it that makes it fell out of popularity. These are stated below;

- It is well known for its vanishing gradient problem
- Its output isn't zero centered. It makes the gradient updates go too far in different directions i.e 0 < output < 1, and it makes optimization harder.
- In back propagation, the weight is constantly changed
- Sigmoid saturate and kill gradients.
- Sigmoid have slow convergence.



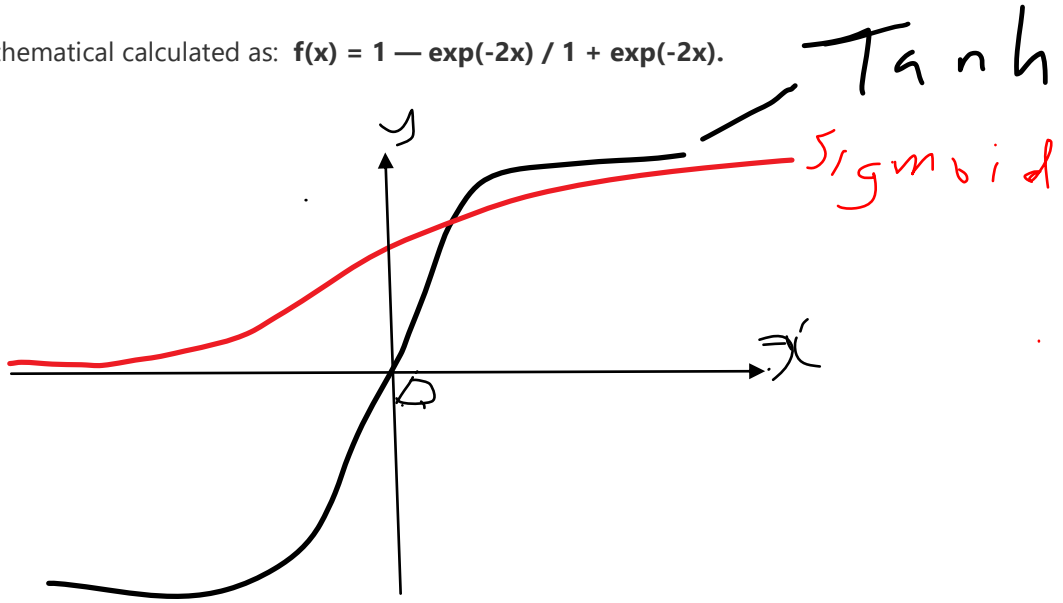
Sigmoid Derivative graph

1.2 Hyperbolic Tangent function- Tanh *:*

As the draw back in Sigmoid function continues to gain momentum, so are people getting concern with how to solve the problems attached to it like not being zero centered and the likes hence the emergence of Hyperbolic Tangent Function (Tanh).

Tanh Function is designed as an alternative to the sigmoid function. It takes into consideration the training of negative data neurons because its range is between -1 to 1. It's quite common in neural networks because its derivative form is as simple as the sigmoid function. As we know, derivative of activation function is used to calculate the error during back-propagation, hence the need to ensure function effectiveness.

It's mathematical calculated as: **f(x) = 1 — exp(-2x) / 1 + exp(-2x).**



Advantages of Tanh Function

- It's an antisymmetric activation function
- Back-propagation learning with antisymmetric activation functions can yield faster convergence.
- it's output is zero centered because its range in between -1 to 1 i.e -1 < output < 1
- Optimization is *easier* in this method

Disadvantages of Tanh Function

- Vanishing gradient problem

1.3 Relu (Rectified Linear Unit) Activation Function

It is safe to say the Relu is the default activation function in deep learning of convolutional neural network as it is the most used activation function. It has an improvement in convergence of up to 5 or 6 times compared to the Tanh function. It's popular in deep learning because of its simplicity and efficiency. It trains data neurons between 0 and infinity and its function and derivatives are monotonic (never decrease or increase).

It computes the function f(x)=max(0,x) - so, once your input below zero , your output is zero, once the input is positive, the output becomes equal to input... All the negative values automatically become zero, hence lower the inability to train date properly. This means that all negative inputs passed to ReLu function turns zero in the graph and inturn affects the resulting graph by not mapping the negative value correctly.

One of its accomplishments is solving the problem of Vanishing Convergent which is prominent in sigmoid and tanh function.

Given below is ReLu in its mathematical form;

R(x) = max(0,x) i.e if x < 0,

R(x) = 0 and if x >= 0,

R(x) = x

Advantages

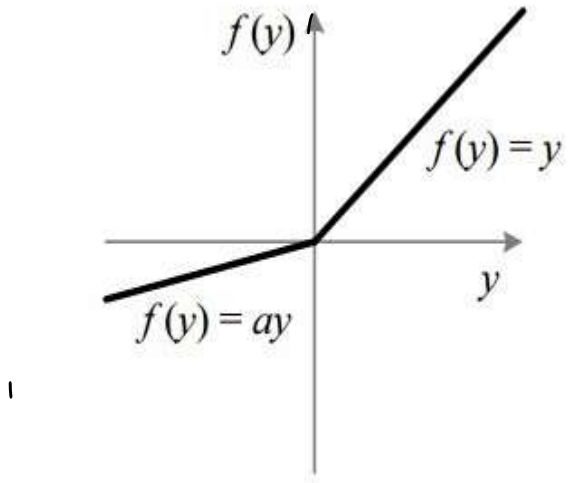
- It does not activate all the neurons all at once.
- Reduces possibility of vanishing gradient

Disadvantages

- It's most suitable only for the hidden layer of neural network.
- Fragile gradient during training i.e dead neuron

1.4 Leaky ReLu

The leaky Relu function is designed to plug the draw backs of the 'dying' ReLu function hence the name 'Leaky'. The leak increases the range of the ReLu function from 0 to 0.01. Instead of the function being zero when x < 0, a leaky ReLU just will return some small negative number instead. Meaning, there will be a small negative slope (of 0.01, or so) in the region of negative inputs. However, a scenario called the Randomized ReLu occurs when the minimum range value is not 0.01 and when this happened the range value changes from –infinity to infinity.



Leaky ReLu also solve the problem of gradient vanishing attributed to Sigmoid function. The mathematical form computation is given below;

$f(x)=1(x<0)(\alpha x)+1(x>=0)(x)$, where α is a small constant

Advantages of Leaky ReLu

- Achieve higher performance
- Fixes the problem of dying ReLu problem, that is fixes the problem of dead neuron

Disadvantages

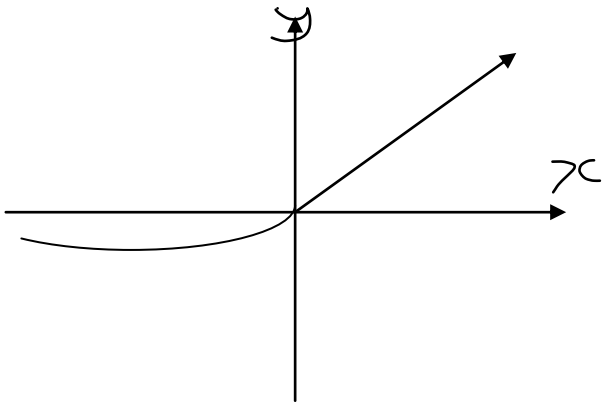
- The slope in the negative region can also be made into a parameter of each neuron, which makes the tuning more complicated.

1.5 ELU (Exponential Linear Unit) Function

The Exponential Linear Unit function is one of the commonly used activation function in model training both in Deep and Machine Learning. It is designed to solve the problem of Gradient Clipping as observed with Leaky ReLu. Also with ELU, the mean output is always close to zero. It's adjudged the best for Normalisation i.e zero-centric function. ELU can be represented with the mathematical derivative given below;

$$dx = \begin{cases} x & x>0 \\ d(ex-1) & x<0 \end{cases}$$

However, for the negative data set, it will be a little weight compute extensive because an exponential operation is performed in the negative axis. This is highlighted in the graphical illustration below;



1.6 Softmax Activation Function

Softmax function is designed to always or mostly give the probability of the occurrence of number in any given data set. It is used to find the probabilities of multiple class classification. It can be mathematically represented as follows;

$$S(x_1) = \frac{e^{x1}}{\sum_{1+n}^{x} {}_1 \; e^{x1}}$$

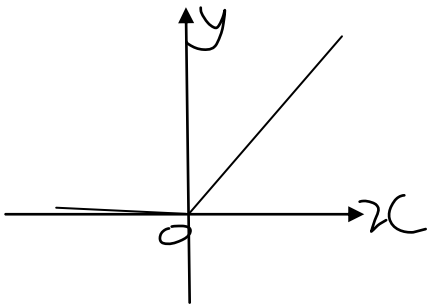
As the name suggests, softmax function is a "soft" version of max function. Instead of selecting one maximum value, it breaks the whole (1) with maximal element getting the largest portion of the distribution, but other smaller elements getting some of it as well.

1.7 P ReLu (Parametric ReLu)

Similar to ReLu, Parametric ReLu takes different types of parameters. Different data is trained with different parameter. In its mathematical derivative, P ReLu make alpher (α) to be a learnable parameter. This can be illustrated below;

$$F(x) = \begin{cases} x & x>0 \\ \alpha x & x \geq 0 \end{cases}$$, where α can be different parameter i.e 0 or 0.01

However, referencing the above calculations, if α tries to learn itself for a negative reason with respect to slope or gradient, then the process is called P ReLu function. During these process α will not be zero and this is class as one of the advantages of P ReLu. It is graphically represented below;



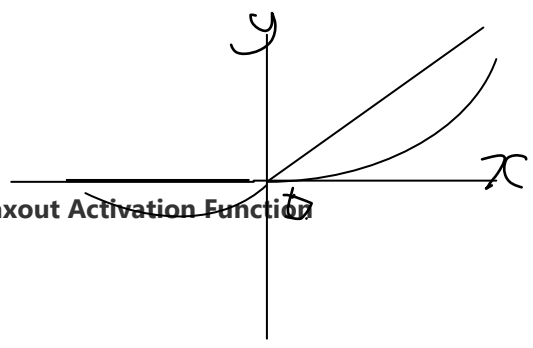
1.8 Swis Activation Function

The SWIS activation function avoids the vanishing gradient problem attributed to the Sigmoid function. The data set of all the derivatives is easily gotten as a result of the avoidance of the vanishing gradient. Irrespective of the data set input, SWIS activation function will give you an output which will change the weights into a back propagation because the continuous gradient will always be established. This can be represented with mathematical derivatives as given below;

Y = x.Sigmoid(x)

SWIS function is designed purposely for LSTM type of network, which involves training a weight. Graphical presentation of SWIS function is shown below;

.



1.9 Maxout Activation Function

Maxout activation function is the generalised version of ReLu and Leaky ReLu. It came into existence in 2014. It is designed to give the maximum value of a data set (y). It is mathematically represented below;

Maxout = ($\max(w_1x_1 + b_1, w_2x_2 + b_2)$

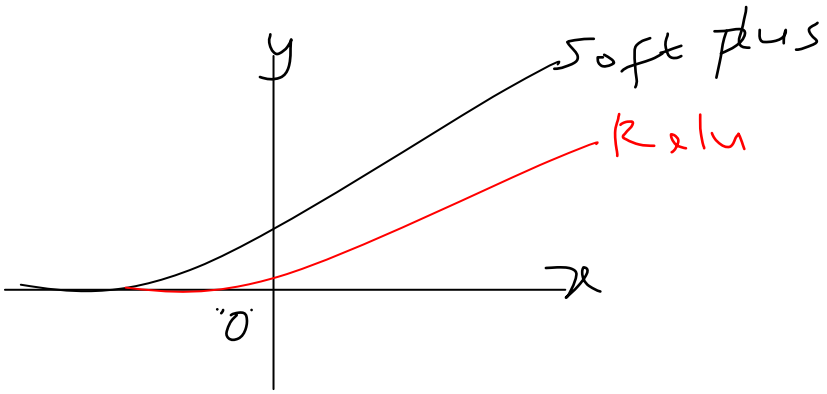
Maxout need to train the weight by finding the new value of n. The first value is learnt and from the learnt parameter, a final value to pass to the function shall be derived. Maxout function does not depend on a pre-defined gradient like the Sigmoid function or ReLu, rather it depends on the weight of the trained model and based on this, the maximum output is generated. This activation function do not indicate whether to generate negative or positive value but to produce the maximum value of whatever value declared.

1.10 Soft Plus Activation Function

Softplus activation function is equally similar to the ReLu function but it is a smoother function. It's mathematical derivatives is given below;

$$f(x) = \text{lon}\,(1 + \exp(x))$$

The function has a range of 0 to n. The divergence of its curve is smoother compared to ReLu which makes it quicker than the ReLu function. Graphically, it is shown below;



2.0 LOSS FUNCTIONS

In Neural network, whenever we provide data into a network, those data goes into the hidden layer where we apply the relevant activation function depending on the expected outcome. At the output layer, the output is learnt and predicted. The y^n is always compared with the y and a loss (error) is generated based on the comparison i.e the difference between the generated value and the expected value.

Types Of Loss Functions

2.1 L1 (Least Absolute Deviation) Loss Function

L1 loss function is used to minimise the error (loss) in the difference between the Absolute Value and the Predicted Value. It is used to check the level of deviation of predicted value to the absolute value. Its mathematical derivation is given below:

L1 loss function = $\sum^{N}_{1\text{-}N}\ (y - y^{n})$

Advantages of L1

- It allows for predefined weight decrease mechanism which is not as high as increasing the loss by the squared value at one point.

2.2 L2 (Least Squared Error) Loss Function

L2 (Least Squared Error) is the square of the error margins derived from the deviation in difference of absolute value and the predicted value, meaning that whatever difference in loss value generated as a result of the subtraction, the value is squared.

Advantages of L2 loss function.

- L2 comes handy if a model does not have a predefined outlier value to predict the loss hence the squared loss could be the key to get the job done.

Disadvantages of L2 loss function.

- The outlier (loss) is always squared, which means it is always decreased by the square of the error which makes it difficult to use a predefined gradient mechanism to help with the decrease in loss to get a new weight.

2.3 Huber Loss Function

Huber loss function is designed to control the draw backs in Least Squared Error (L2) especially in the areas of outlier where the loss is squared i.e $(y - y^{n})^{2}$. The mathematical derivative of Huber loss function shows a new term, Delta (ɗ), was introduced to address the loss (error). Delta (ɗ)is also known as the **threshold**, used to set a limit to the loss value that can be generated and a condition can be put in place to execute an action if the set condition is met, otherwise a different condition is executed.

The mathematical derivative is given below:

$$L(x, ɗ(y) = \begin{cases} \tfrac{1}{2}(y - y^{n})^{2} & \Rightarrow \text{ if } (y - y^{n}) < ɗ \\ ɗ\,(y - y^{n}) - \tfrac{1}{2}\,ɗ^{2} & \Rightarrow \text{ if } > ɗ \end{cases}$$

For example, if (y-yn) is less than the threshold (ɗ), the equation (½(y - y^n)^2) is executed and if the loss (error) is less than the threshold (ɗ), then the equation (ɗ (y – y^n) – $^1/_2$ $ɗ^2$) is executed.

2.4 Pseudo Huber Loss Function

The Pseudo Huber loss function is one of the not-too-popular loss functions in which operations performed by this function is basically for smoothening purpose. Here, approximation of the derivatives is carried out by finding the square root of the loss(error) while ensuring the decomposition of the loss.

Its mathematical derivative is shown below:

$$L(x) = ɗ^2 \left(\sqrt{1 + \left(\frac{\alpha}{ɗ}\right)^{2-1}}\right)$$

The Pseudo Huber loss function as stated earlier, is not commonly used in deep learning or machine learning network model because of its limitation to smoothening operations only.

2.5 Hinge Loss Function

Training a date set to minimise the loss as low as we can by reducing the difference between the absolute value and the actual value to a level where further reduction is not possible is the basis of effective data set modelling. However, Hinge loss function is classifier based function. A model that involves operational classifier is best suitable for the Hinge loss function.

Represented mathematically as;

$$L(y) = \text{Max}\ (\ 0,\ 1 – t\ .\ y\ )$$

Where t is the number of defined classes and y is the received value.
Loss = Max of 0 and (1 – t . y).

The closer to 1 the value of t . y is, the lesser the loss generated and vice versa. Hinge loss function come in handy when trying to train a model like Support Vendor Machine.

2.6 Squared Hinge Loss Function

The Squared Hinge Loss Function is an extension of the Hinge Loss Function which is sometimes a subject of investigation with support vector machines models and the likes. The function extension is used to simply calculate the square of the score hinge loss. It has the effect of smoothing the surface of the error function and making it numerically easier to work with.

For instance, If using a hinge loss does result in better performance on a given binary classification problem, is likely that a squared hinge loss may be appropriate. As with using the hinge loss function, the target variable must be modified to have values in the set {-1, 1}. This is mathematically derived as shown below:

$L(y)2 = (y1 - t'xi)2$ where xi is undefined at x = 1

2.7 Cross Entropy Loss Function

Also known as the Log Loss, the Cross Entropy Loss Function is mainly applicable to the Binary operation classification. This binary classification is made evident in the mathematical derivatives by the string character 't' where 't' can be (0 or 1) or (-1 or 1).

Mathematical derivative

$$J = -\sum^{n}_{m+1} t_m (\log (y_n)) + (1 - t_m) \log (1 - y_m)$$

The loss is defined in terms of the cross entropy. Hence the range of value as output is between 0 and 1 because loss is equal to t (class). Assuming there is a class (t) between 0 and 1. If t is equal to 0 in

$$J = -\sum^{n}_{m+1} t_m (\log (y_n))$$

$t_m \downarrow 0$

The entire calculation above will turn to 0 and the other part of the mathematical derivative;

$$(1 - t_m) \log (1 - y_m)$$

$\downarrow 0$ will give an output between 0 and 1.

This cross entropy function is the most widely used loss function in recent times because of its flexibility especially in the areas of logistic recreation.

2.8 Sigmoid Cross Entropy Loss Function

The sigmoid cross entropy loss function is designed to predict the value of the probability of a given data set in any network model. In predicting the probability of loss deviation, instead of using a y as a direct value, the y can be substituted with a sigmoid function directly. With this, you will be able to get a value for the loss. It can be mathematically derived as given below:

$$L = -\sum t_m (\log (y_m) + (1 - t_m) \log (1 - y_m)$$

2.9 Softmax Cross Entropy Loss Function

The term Softmax loss and Cross-Entropy loss are used interchangeably. The softmax classifier is a linear classifier that uses the cross-entropy loss function. In other words, the gradient of the softmax cross entropy function tells a softmax classifier how to update its weights using some optimization like the gradient descent.

The function normalises the network predictions so that they can be interpreted as probabilities. Once your network is predicting a probability distribution over labels for each input, the log loss is equivalent to the cross entropy between the true label distribution and the network predictions. This property of softmax function that it outputs a probability distribution makes it suitable for probabilistic interpretation in classification tasks. This is mathematically represented below;

$$L = -d + \log \sum efi$$

$$Li = (-\log (efi/\sum efi))$$

Cross entropy indicates the distance between what the model believes the output distribution should be, and what the original distribution is. Cross entropy measure is a widely used alternative of squared error. For example, it is used when the output is a probability distribution. Thus it is used as a loss function in neural networks which have softmax activations in the output layer.

2.10 Sparse Multiclass Cross-Entropy Loss

For any data scientist that is familiar with using cross entropy is training deep learning neural network, it is very uncommon to be frustrated when using this function (cross entropy) with classification problems with a large number of labels by the one hot encoding process involved.

Sparse cross-entropy addresses this by performing the same cross-entropy calculation of error, without requiring that the target variable be one hot encoded prior to training. This function can be used for multi class classification in keras.

Assuming we were to predict the words in a vocabulary which may have tens or hundreds of thousands of categories, one for each label. This can mean that the target element of each training example may require a one hot encoded vector with tens or hundreds of thousands of zero values, requiring significant memory space. Mathematically represented, assuming, that each y_i belongs to exactly one class, say: C_{y_i}, then we can write:

$$-{}^{1}/_{N} \sum^{N}_{i=1} \log P_{model} [y_i \in C_{yi}]$$