Mohammed Zeeshan Mulla
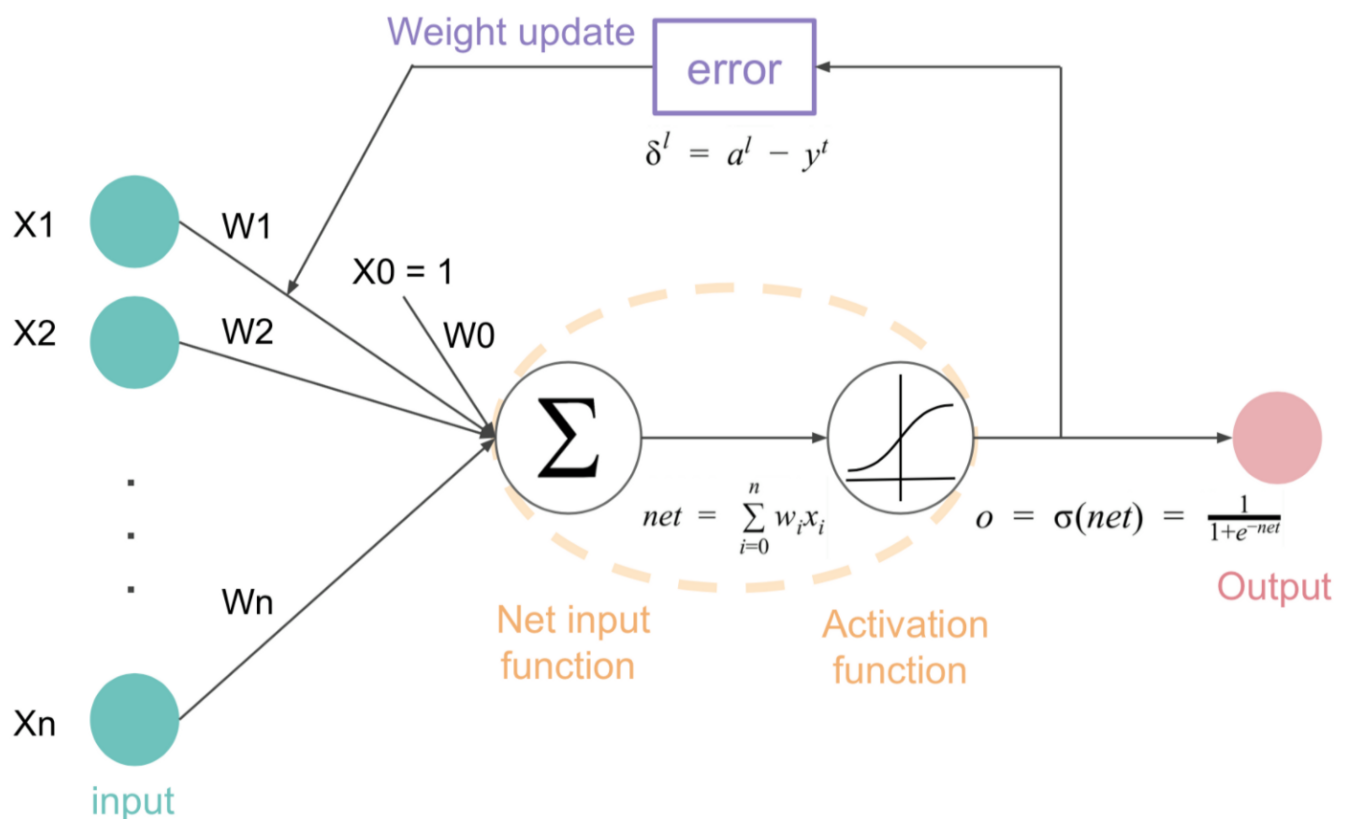
Follow   41 Followers   About

Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these?

Mohammed Zeeshan Mulla Apr 26, 2020 · 18 min read

What is the difference between cost function and activation function?

- A cost function is a measure of error between what value your model predicts and what the value actually is. For example, say we wish to predict the value yi for data

$$f_\theta(x_i)$$

- represent the prediction or output of some arbitrary model for the point xi with parameters θ. One of the many cost functions could be

$$\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

this function is known as the L2 loss. Training the hypothetical model we stated above would be the process of finding the θ that minimizes this sum.

-An activation function transforms the shape/representation of the data going into it. A simple example could be max(0, xi), a function which outputs 0 if the input xi is negative or xi if the input xi is positive. This function is known as the "ReLU" or "Rectified Linear Unit" activation function. The choice of which function(s) are best for a specific problem using a particular neural architecture is still under a lot of discussions. However, these representations are essential for making high-dimensional data linearly separable, which is one of the many uses of a neural network.

What is the difference between cost function and loss function?

As mentioned by others, cost and loss functions are synonymous (some people also call it error function). The more general scenario is to define an objective function first, which you want to optimize. This objective function could be to

- maximize the posterior probabilities (e.g., naive Bayes)
- maximize a fitness function (genetic programming)
- maximize the total reward/value function (reinforcement learning)
- maximize information gain/minimize child node impurities (CART decision tree classification)
- minimize a mean squared error cost (or loss) function (CART, decision tree regression, linear regression, adaptive linear neurons, …
- maximize log-likelihood or minimize cross-entropy loss (or cost) function
- minimize hinge loss (support vector machine)

$$\mathbf{w}^* \;=\; \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \boxed{\lambda \sum_{i=1}^{k} |w_i|}$$

Generally cost and loss functions are synonymous but cost function can contain regularization terms in addition to loss function. although it is not always necessary.

Which LOSS AND ACTIVATION function to use?

Check this Out: "https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8"

LOSS FUNCTION:

Loss function is a method of evaluating "how well your algorithm models your dataset". If your predictions are totally off, your loss function will output a higher number. If they're pretty good, it'll output a lower number. As you tune your algorithm to try and improve your model, your loss function will tell you if you're improving or not. 'Loss' helps us to understand how much the predicted value differ from actual value

Types of Loss function:

1. Regression Loss Function:

Regression models deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of the room. The loss function used in the regression problem is called "Regression Loss Function".

2. Binary Classification Loss Functions:

- Binary classification is a prediction algorithm where the output can be either one of two items, indicated by 0 or 1. The output of binary classification algorithms is a prediction score (mostly). So the classification happens based on the threshold the value (default value is 0.5). If the prediction score > threshold then 1 else 0.
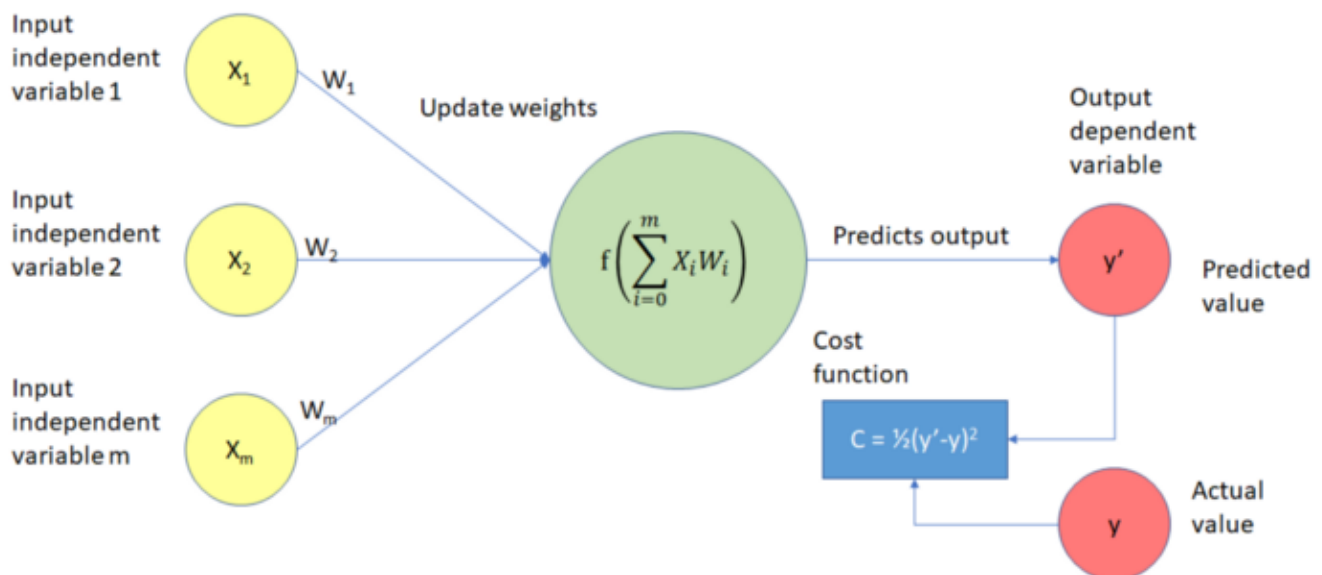
*3. Multi-class Classification Loss Functions:*

COST FUNCTION:

The loss error is computed for a single training example. If we have 'm' number of examples then the average of the loss function of the entire training set is called 'Cost function'.
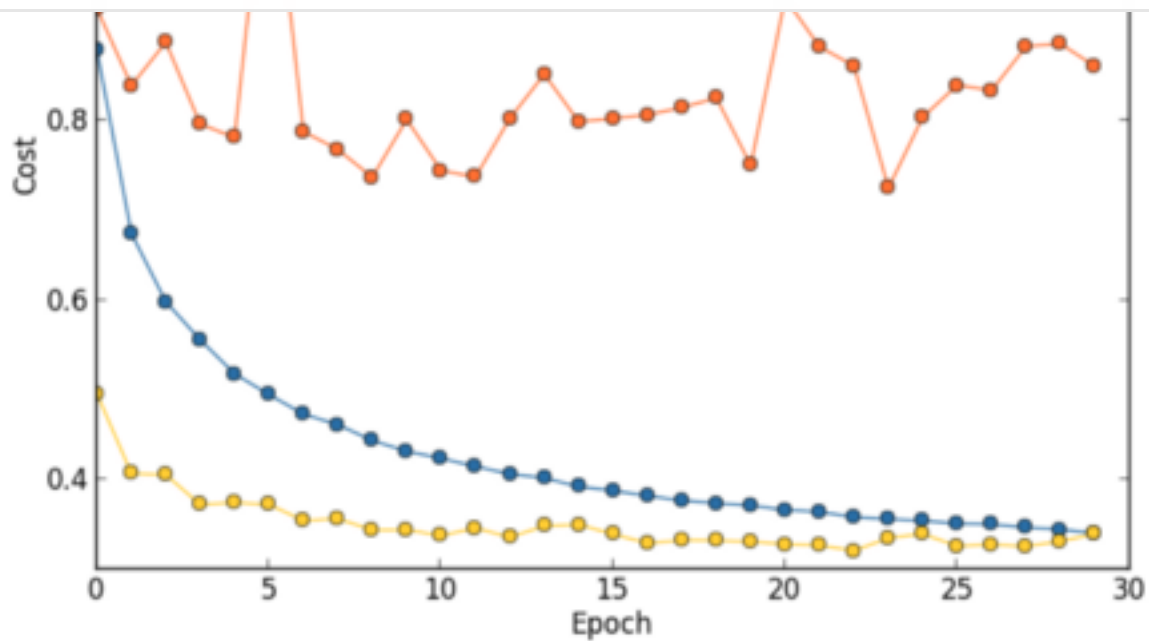
Cost function (J) = 1/m (Sum of Loss error for 'm' examples)

The shape of cost function graph against parameters (W and b) is a cup up parabola with a single minimum value called 'local optima'.
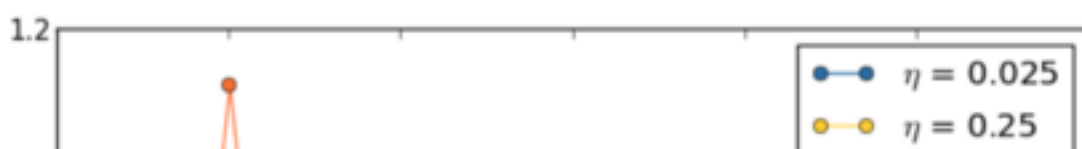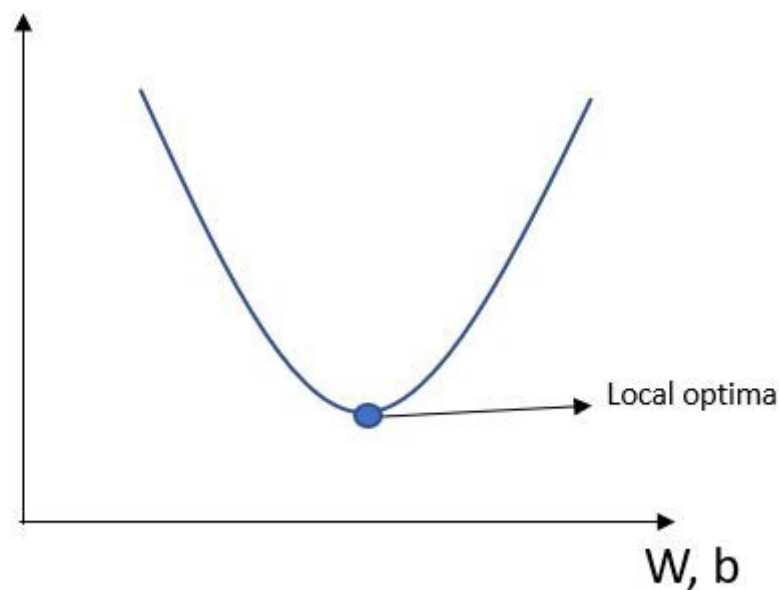
Evaluating model performance during the training process. (Source
http://neuralnetworksanddeeplearning.com/)

It is a function that **measures the performance of a Machine Learning model** for given data. Cost Function quantifies the error between predicted values and expected values and **presents it in the form of a single real number**. Depending on the problem Cost Function can be formed in many different ways. The purpose of Cost Function is to be either:

Get started   Open in app   ●◖

number as possible.

- **Maximized** — then the value it yields is named a **reward**. The goal is to find values of model parameters for which returned number is as large as possible.

- Given a model using the following formula:

$$\hat{y} = wx$$

where:

- $\hat{y}$ — predicted value,

- x — vector of data used for prediction or training,

- w — weight.

TYPES OF LOSS FUNCTION:

1. Regression Loss Functions…

2. Mean Squared Error Loss

3. Mean Squared Logarithmic Error Loss

4. Mean Absolute Error Loss

2. Binary Classification Loss Functions

1. Binary Cross-Entropy

2. Hinge Loss

3. Squared Hinge Loss

3. Multi-Class Classification Loss Functions

1. Multi-Class Cross-Entropy Loss

2. Sparse Multiclass Cross-Entropy Loss

3. Kullback Leibler Divergence Loss

- Mean Squared Error (MSE)

- Mean Absolute Error (MAE)

- Root Mean Squared Error (RMSE)

- Categorical Cross Entropy Cost Function.

- Binary Cross Entropy Cost Function.

"THERE ARE MANY BUT I HAVE COVERED A FEW (MOST COMMON)".

Mean Absolute Error

Regression metric which measures the **average magnitude of errors in a group of predictions**, without considering their directions. In other words, it's a **mean of absolute differences among predictions and expected results where all individual deviations have even importance**.

$$\text{MAE} = \underbrace{\frac{1}{n} \sum_{i=1}^{n}}_{\text{test set}} |\underbrace{y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i}_{\text{actual value}}|$$

MAE

where:

- i — index of sample,

- ŷ — predicted value,

- y — expected value,

- m — number of samples in dataset.

Sometimes it is possible to see the form of formula with swapped predicted value and expected value, but it works the same.

Mean Squared Error

alteration of MAE where instead of taking the absolute value of differences, they are squared.

In MAE, the partial error values were equal to the distances between points in the coordinate system. Regarding MSE, each partial error is equivalent to the area of the square created out of the geometrical distance between the measured points. All region areas are summed up and averaged.

The MSE formula can be written like this:

$$MSE = \underbrace{\frac{1}{n} \sum_{i=1}^{n}}_{\text{test set}} (\underbrace{y_i}_{\text{predicted vaue}} - \underbrace{\hat{y}_i}_{\text{actual value}})^2$$

MSE

- i — index of sample,

- $\hat{y}$ — predicted value,

- y — expected value,

- m — number of samples in dataset.

There are different forms of MSE formula, where there is no division by two in the denominator. Its presence makes MSE derivation calculus cleaner.

Calculating derivative of equations using absolute value is problematic. MSE uses exponentiation instead and consequently has good mathematical properties which make the computation of it's derivative easier in comparison to MAE. It is relevant when using a model that relies on the Gradient Descent algorithm.

Root Mean Square error (RMSE)

$$\sqrt{\sum_{i}^{N} (Predicted_i - Actual_i)^2}$$

Root Mean Square error is the extension of MSE — measured as the average of square root of sum of squared differences between predictions and actual observations.

Binary Cross Entropy Loss Function

This cost function originally stems from information theory with the transfer of bits and how much bits have been lost in the process.

Binary cross entropy measures how far away from the true value (which is either 0 or 1) the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss.

We can define cross entropy as the difference between two probability distributions **p** and **q**, where **p** is our true output and **q** is our estimate of this true output.

This difference is now applied to our neural networks, where it is extremely effective because of their strong usage of probability.

$$H(x) = \sum_{i=1}^{N} \underset{\text{true label}}{p}(x) \log \underset{\text{estimate}}{q}(x)$$

Cross entropy will work best when the data is normalized (forced between 0 and 1) as this will represent it as a probability. This normalization property is common in most cost functions.

Categorical cross-entropy

It is a loss function that is used for single label categorization. This is when only one category is applicable for each data point. In other words, an example can belong to one class only.

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

Use categorical crossentropy in classification problems where only one result can be correct.

Example: In the MNIST problem where you have images of the numbers 0,1, 2, 3, 4, 5, 6, 7, 8, and 9. Categorical crossentropy gives the probability that an image of a number is, for example, a 4 or a 9.

where $\hat{y}$ is the predicted value.

Categorical cross-entropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

Hinge Loss

Hinge loss
Exp loss
0–1 loss
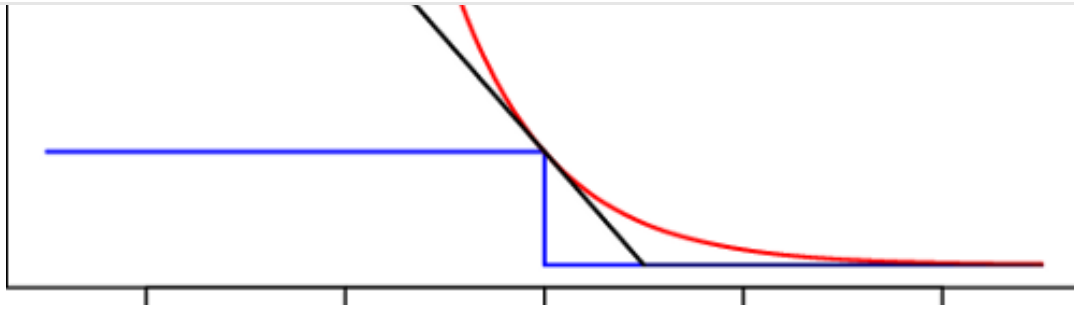
The function `max(0,1-t)` is called the hinge loss function. It is equal to 0 when `t≥1` . Its derivative is `-1` if `t<1` and 0 if `t>1` . It is not differentiable at `t=1` . but we can still use gradient descent using any subderivative at t=1.

- Hinge loss is most popular loss function during pre-deep learning era.

- Hinge loss is often used for binary classification problems.

- This type of loss is primarily used in SVM classifiers where the target values are in the set {-1, 1}.

- When using hinge loss it is important to change the class label to -1 and +1.

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

Hinge Loss not only penalizes the wrong predictions but also the right predictions that are not confident. Hinge loss output the value ranges from -inf to 1. Hinge loss is used for maximum-margin classification. Though hinge loss is not differentiable, it's convex function which makes it easy to work with usual convex optimizers used in machine learning domain.

Multi-class Cross Entropy Loss:
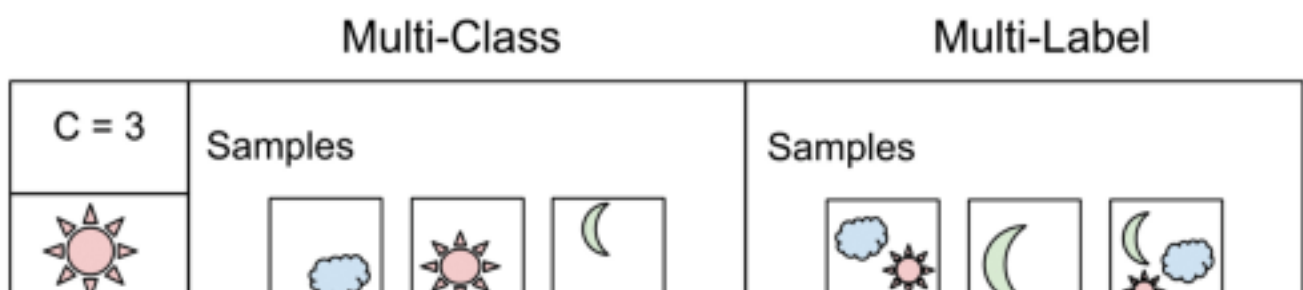
[0 0 1] [1 0 0] [0 1 0] | [1 0 1] [0 1 0] [1 1 1]

- Most time,Multi-class Cross Entropy Loss is used as a loss function. The generalised form of cross entropy loss is the multi-class cross entropy loss.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

M — No of classes

- M — No of classes

- y — binary indicator (0 or 1) if class label c is the correct classification for input o

- p — predicted probability score of input o belonging to class c.

Kullback-Liebler Divergence LOSS (KL-Divergence):

KL Divergence is a measure of how a probability one distribution differs from another distribution.

Dkl(P||Q) is interpreted as the information gain when distribution Q is used instead of distribution P .

Dkl(Q||P) is interpreted as the information gain when distribution P is used instead of distribution Q.

Information gain (IG) measures how much "information" a feature gives us about the class.

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

$$D_{KL}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx$$

$$D_{KL}(P \| Q) = \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right)$$

$$= 0.36 \ln\left(\frac{0.36}{0.333}\right) + 0.48 \ln\left(\frac{0.48}{0.333}\right) + 0.16 \ln\left(\frac{0.16}{0.333}\right)$$

$$= 0.0852996$$

Get started   Open in app

from Q to P.

1. Dkl(P||Q) !=Dkl(Q||P). Divergence from Q to P is not symmetric to Divergence from P to Q.

Eg: Dkl(P||Q) =0.086 ;Dkl(Q||P)=0.096

It clear from the example that Dkl(P||Q) not symmetric to Dkl(Q||P).

2. Dkl(P||P) =0 meaning both have identical distribution and not information is gained from other distribution.

3.The value of Dkl(P||Q) will be greater than or equal to zero.

So, the goal of the **KL divergence loss** is to approximate the **true probability distribution P** of our target variables with respect to the input features, given some **approximate distribution Q.** This Can be achieved by minimizing the Dkl(P||Q) then it is called **forward KL.** If we are minimizing Dkl(Q||P) then it is called **backward KL.**

Forward KL → applied in Supervised Learning

Backward KL → applied in Reinforcement learning

Q.Why KL divergence is not considered as distance metric?

Reason is simple KL divergence is not symmetric.

KL- Divergence is functionally similar to multi-class Cross entropy .

Huber loss :

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

The Huber loss function describes the penalty incurred by an estimation procedure f. Huber (1964) defines the loss function piecewise by [^]

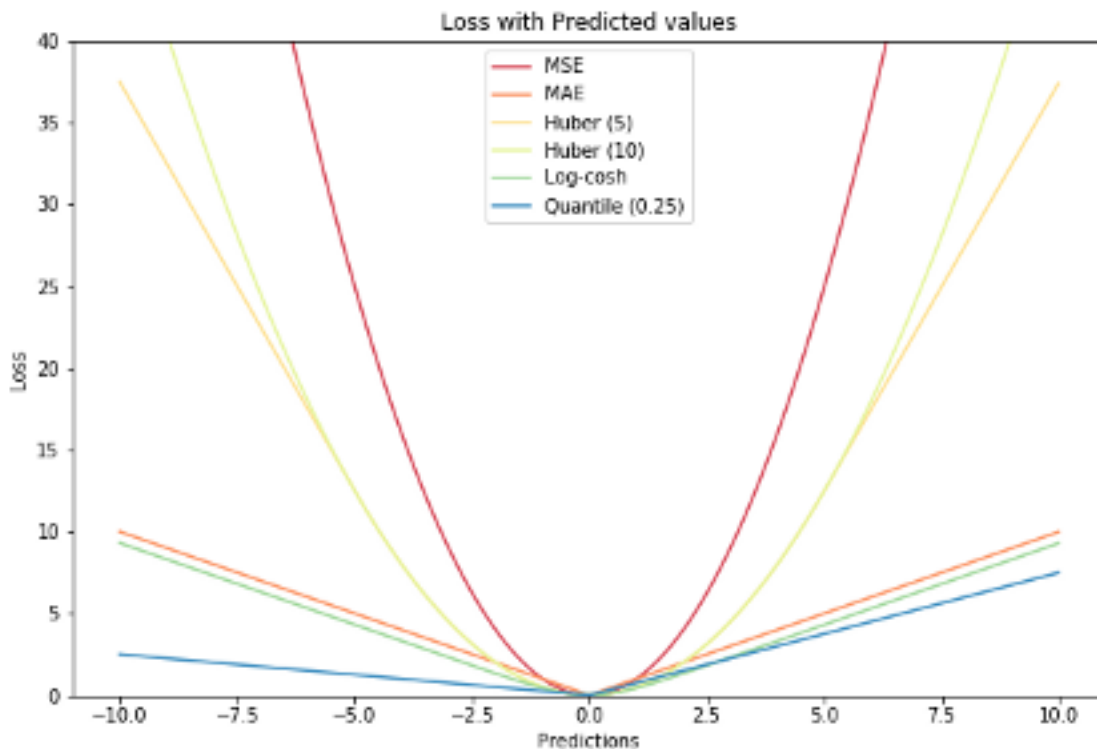. The variable a often refers to the residuals, that is to the difference between the observed and predicted values

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

so the former can be expanded to<-

- Huber loss is less sensitive to outliers in data than the squared error loss. This function is quadratic for small values of error /residual, and linear for large values. What is that small value for the error to make the function quadratic depends on the "*hyperparameter*", δ (delta), which can be tuned.

- Huber loss approaches **MAE when $\delta \sim 0$ and MSE when $\delta \sim \infty$ (large numbers).** Choice of delta is important because it decide what value you are willing set as an

ACTIVATION FUNCTION:
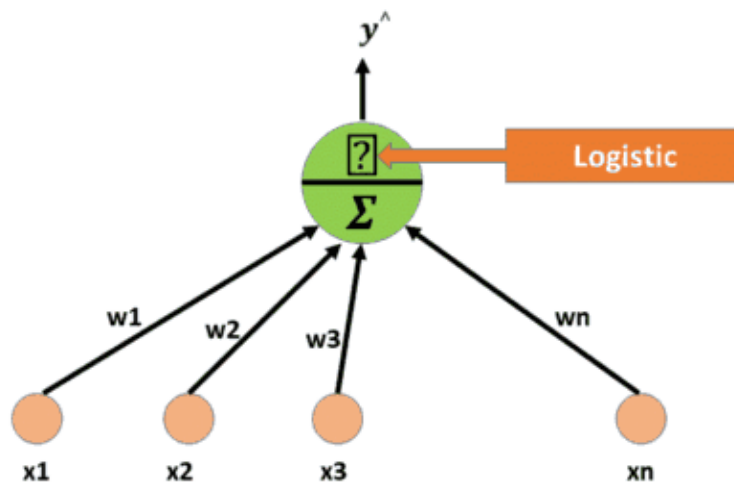
An activation function is a very important feature of an artificial neural network , they basically decide whether the neuron should be activated or not.
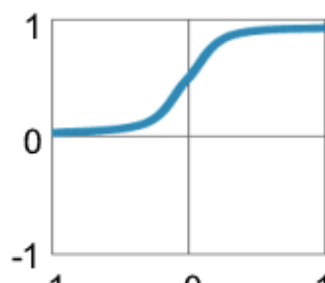
In artificial neural networks, the **activation function** defines the output of that node given an input or set of inputs.
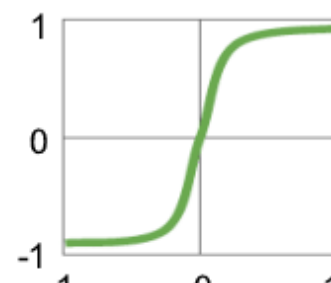
Activation function Types :

- Linear function

- Binary Step function

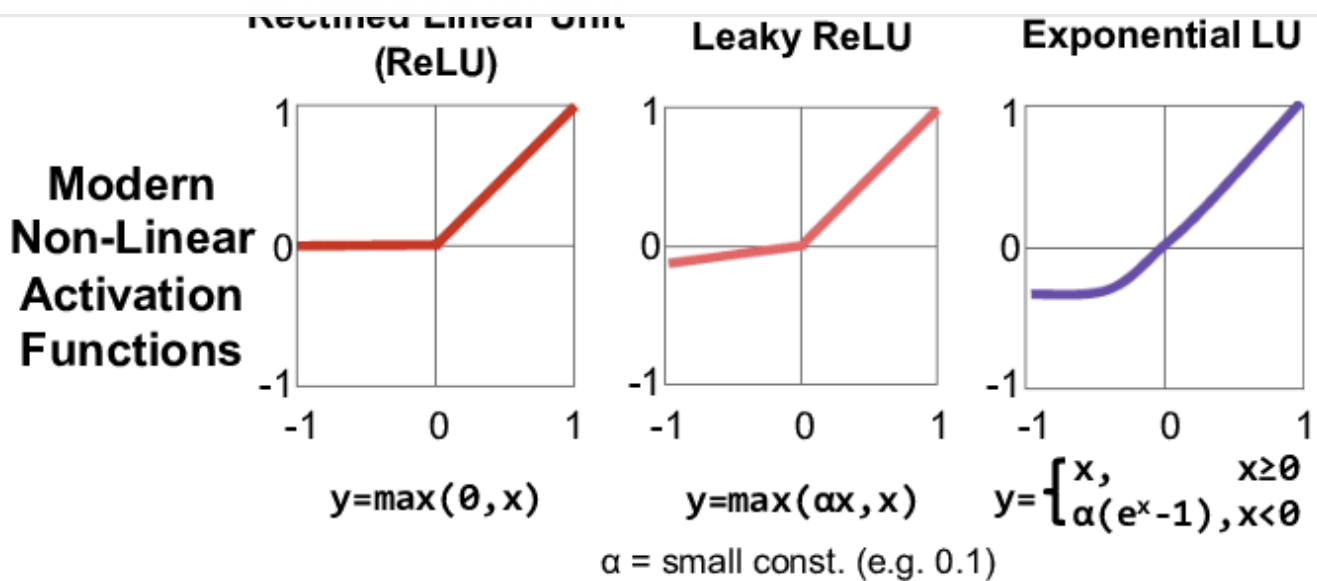- Non-Linear function

Modern Non-Linear Activation Functions

Rectified Linear Unit (ReLU)

$y = \max(0, x)$

Leaky ReLU

$y = \max(\alpha x, x)$

Exponential LU

$y = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$

α = small const. (e.g. 0.1)

Properties that Activation function should hold?

Derivative or Differential: Change in y-axis w.r.t. change in x-axis. It is also known as slope. (Back prop)

Monotonic function: A function which is either entirely non-increasing or non-decreasing.

Advantage of Non-linear function over the Linear function :

- Differential are possible in all the non -linear function.

- Stacking of network is possible , which helps us in creating the deep neural nets.

Linear Types:

A linear activation function takes the form:

y=mx+c (m is line equation represents W and c is represented as b in neural nets so equation can be modified as $y=Wx+b$)

It takes the inputs ($Xi's$), multiplied by the weights($Wi's$) for each neuron, and creates an output proportional to the input. In simple term, weighted sum input is proportional to output.

Binary Step Function:

Binary step function are popular known as " Threshold function". It is very simple function.

Another problem with a step function is that it can handle binary class problem alone. (Though with some tweak we can use it for multi-class problem)

Non-linear Types:

The deep learning rocketing to the sky because of the non-linear functions.Most modern neural network use the non-linear function as their activation function to fire the neuron. Reason being they allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality. Differential are possible in all the non -linear function.

Stacking of network is possible , which helps us in creating the deep neural nets.

The Nonlinear Activation Functions are mainly divided on the basis of their **range or curves.**

1. Sigmoid / Logistic
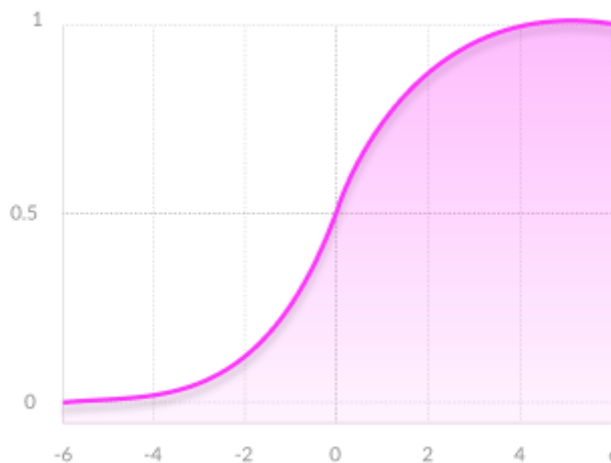
Advantages

- **Smooth gradient**, preventing "jumps" in output values.

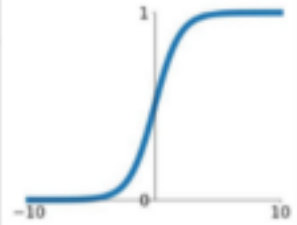- **Output values bound** between 0 and 1, normalizing the output of each neuron.

predictions.

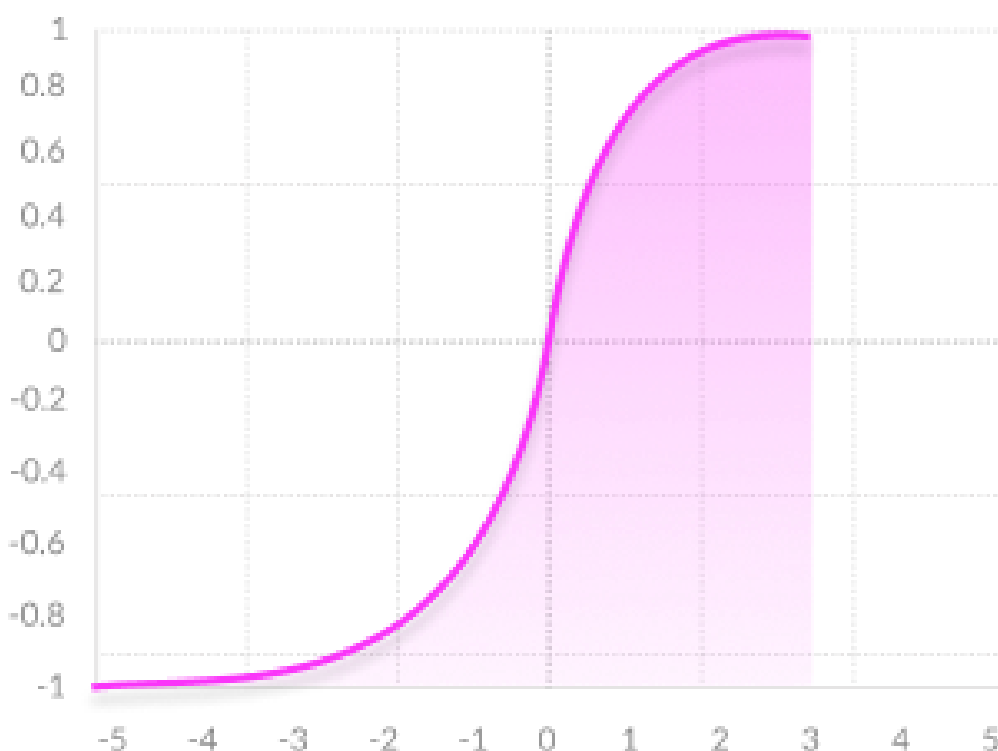| Function	Equation	Range	Derivative
Sigmoid (Logistic)	$f(x) = \dfrac{1}{1+e^{-x}}$	0,1	$f'(x) = f(x)(1 - f(x))$

Disadvantages

- **Vanishing gradient** — for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.

- **Outputs not zero centered**.

- **Computationally expensive**

2. TanH / Hyperbolic Tangent

- **Zero centered** — making it easier to model inputs that have strongly negative, neutral, and strongly positive values.

- Otherwise like the Sigmoid function.

Function	Equation	Range	Derivative
Tanh (Hyperbolic tangent)	$f(x) = \dfrac{2}{1+e^{-2x}} - 1$	-1,1	$f'(x) = 1 - f(x)^2$
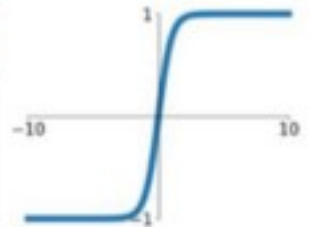
Disadvantages

- Like the Sigmoid function

3. ReLU (Rectified Linear Unit)
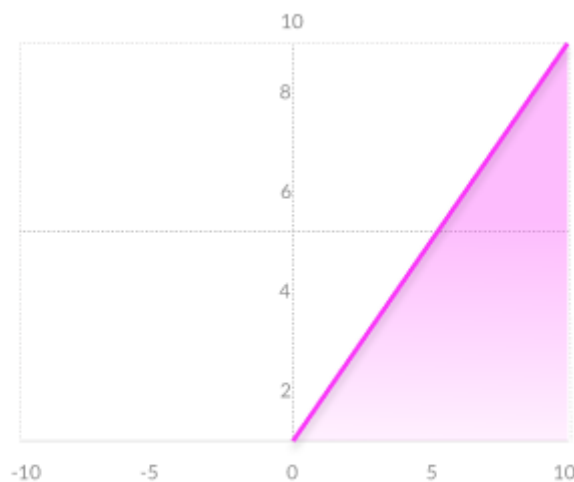
Advantages

- **Computationally efficient** — allows the network to converge very quickly

- **Non-linear** — although it looks like a linear function, ReLU has a derivative function and allows for backpropagation

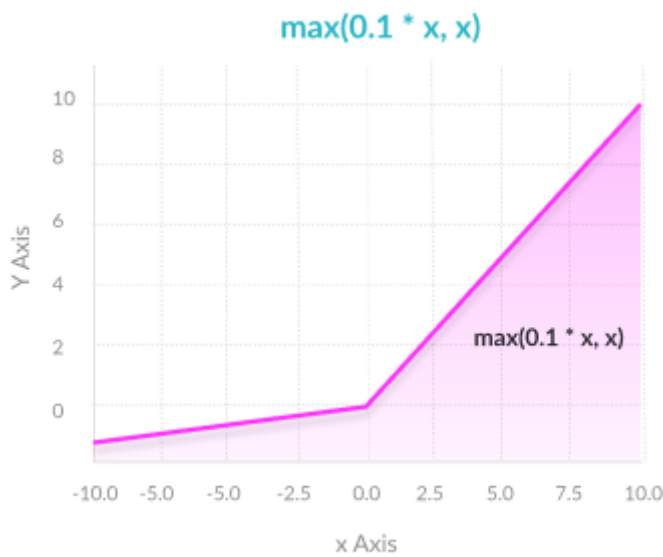| ReLu (Rectified Linear Unit) | $f(x) = \begin{cases} 0 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$ | $0, +\infty$ | $f'(x) = \begin{cases} 0 \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$ | |

Disadvantages

- **The Dying ReLU problem** — when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn
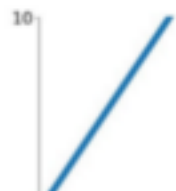
4. Leaky ReLU

max(0.1 * x, x)

Advantages

- **Prevents dying ReLU problem** — this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values

- Otherwise like ReLU

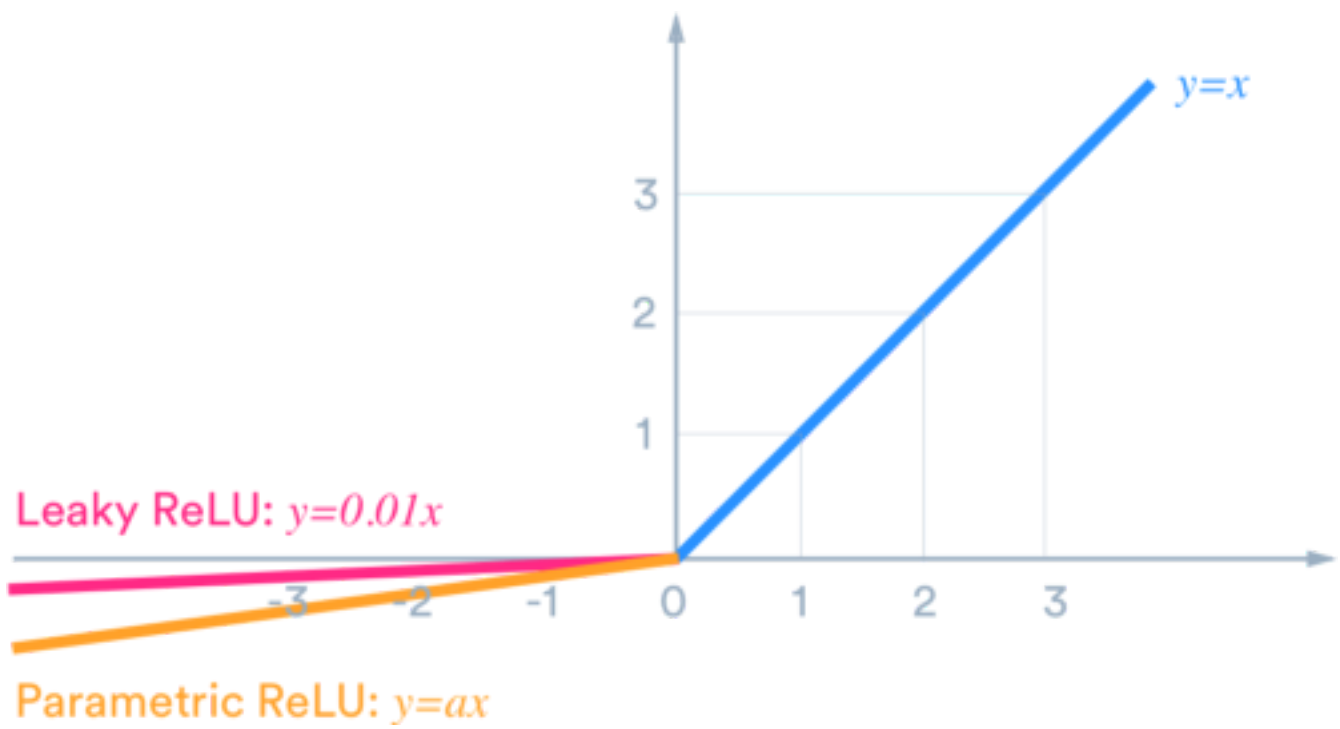| Function	Equation	Range	Derivative	
Leaky ReLu	$f(x) = \begin{cases} \alpha x \ for \ x < 0 \\ x \ \ for \ x \geq 0 \end{cases}$	$-\infty, +\infty$	$f'(x) = \begin{cases} \alpha \ for \ x < 0 \\ 1 \ for \ x \geq 0 \end{cases}$	

Disadvantages

- **Results not consistent** — leaky ReLU does not provide consistent predictions for negative input values.

5. Parametric ReLU

Leaky ReLU: $y=0.01x$

Parametric ReLU: $y=ax$

$y=x$

Advantages

- **Allows the negative slope to be learned** — unlike leaky ReLU, this function provides the slope of the negative part of the function as an argument. It is, therefore, possible to perform backpropagation and learn the most appropriate value of α.

- Otherwise like ReLU

Parameteric rectified linear unit (PReLU)[20]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)^{[2]}$

Disadvantages

6. Softmax

$$\sigma(z)j= \frac{ezj}{\sum_{k=1ezk}^{k}} \quad for\ j=\ 1....k.$$

Advantages

- **Able to handle multiple classes** only one class in other activation functions — normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class.

- **Useful for output neurons** — typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories.

Name ◆	Equation ◆	Derivatives ◆	Range ◆
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, ..., J$	$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$[5]	$(0,1)$

## 7. Swish (A Self-Gated) Activation Function

$$\sigma(x) = \frac{x}{1+e^{-x}}$$
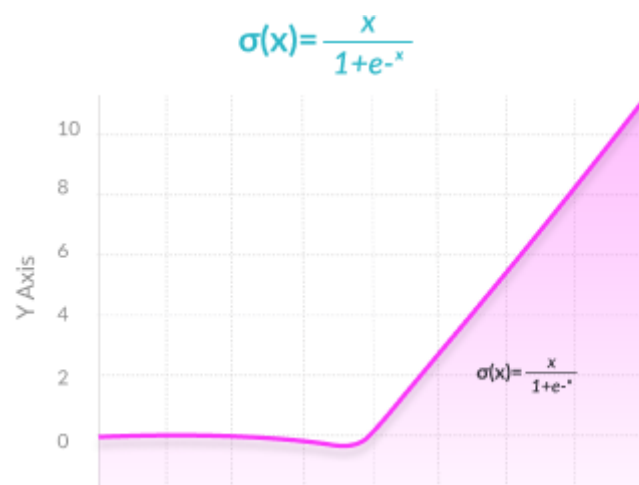
Swish is a new, self-gated activation function discovered by researchers at Google. According to their paper, it performs better than ReLU with a similar level of

higher.

1. **Unboundedness:** Unlike sigmoid and tanh functions, Swish is unbounded above which makes it useful near the gradients with values near to 0. This feature avoids *Saturation as training becomes slow near 0 gradient values.*

2. **Smoothness of the curve:** Smoothness plays an important role in generalization and optimization. Unlike ReLU, Swish is a smooth function which makes it less sensitive to initializing weights and learning rate.

3. **Bounded Below:** Like most of the activation functions out there, Swish is also bounded below which helps in strong regularization effects. Like ReLU and softplus, Swish produces produces negative outputs for small negative inputs due to its non-monotonicity. The non-monotonicity of Swish increases expressivity and improves gradient flow, which is important considering that many preactivations fall into this range.

8. ELU (Exponential Linear Unit)
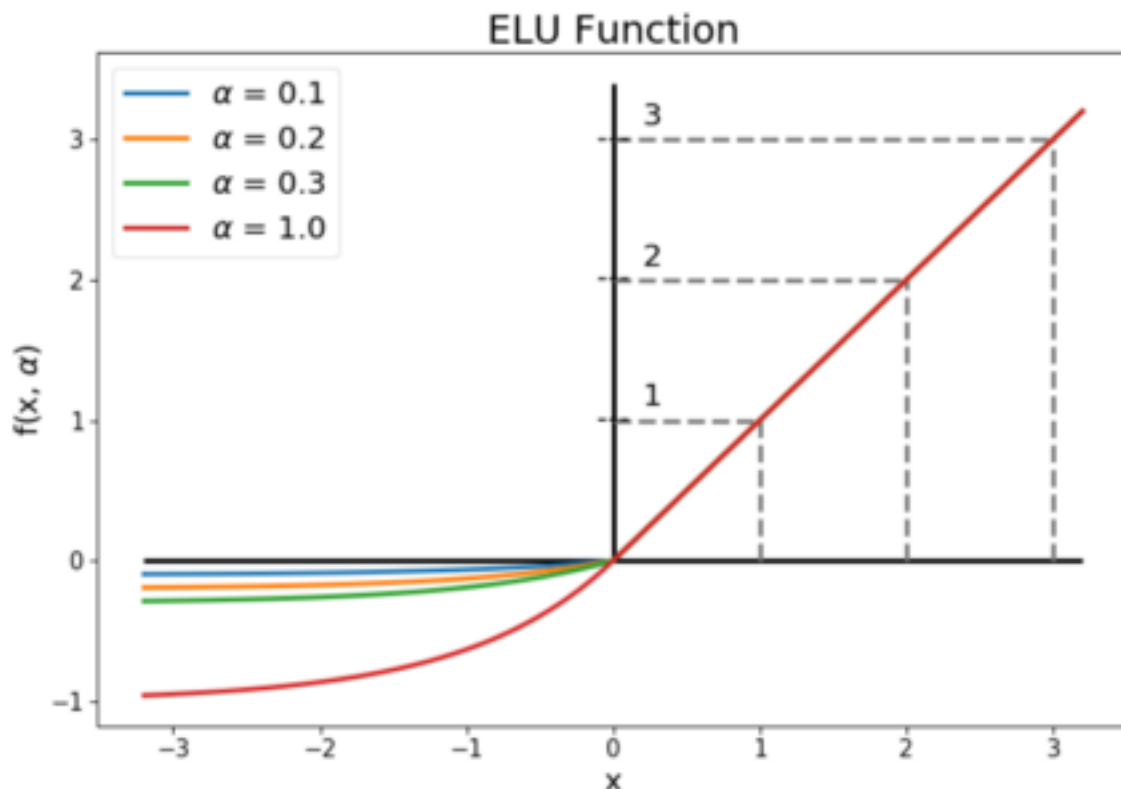
- ELU is also proposed to solve the **problem of dying neuron.**

- No Dead ReLU issues

- Zero-centric

Exponential linear unit (ELU)[23]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	$(-\alpha, \infty)$

Disadvantages:

- Computationally intensive.

- Similar to Leaky ReLU, although theoretically better than ReLU, there is currently no good evidence in practice that ELU is always better than ReLU.

- f(x) is monotonic only if alpha is greater than or equal to 0.

- f'(x) derivative of ELU is monotonic only if alpha lies between 0 and 1.

- Slow convergence due to exponential function.

9. Softmax
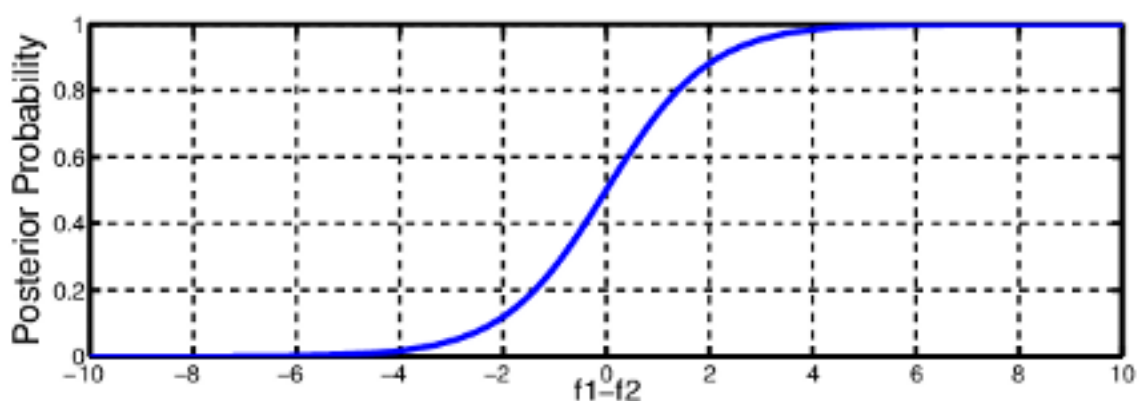
The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. The output values are between the range [0,1] which is nice because we are able to avoid binary classification and accommodate as many classes or dimensions in our
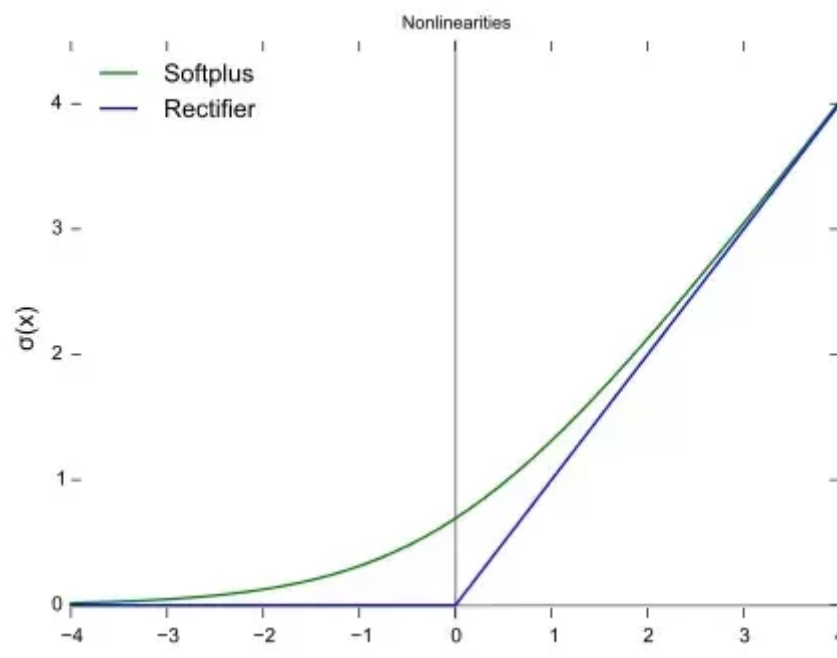
SoftMax Function

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- It is generalization of Logistic Function
- It squashes a K- dimension$^{\sigma(\mathbf{z})}$ bitrary reak value to a K-dimensional vector   of real values
- Range : (0,1)
- In probability theory, the output of it is used for categorical distribution

10. Softplus

Softplus function: $f(x) = \ln(1+e^x)$. The softplus function is similar to the ReLU function, but it is relatively smoother.

Derivative of the Softplus function is f'(x) is <u>logistic function</u> $(1/(1+exp\ x))$.
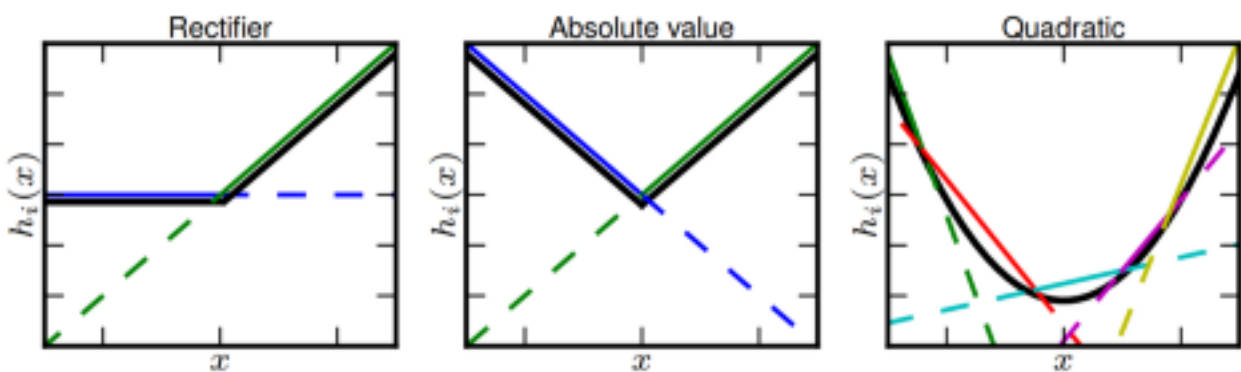
Get started   Open in app

and its derivative than for the softplus function which has log(.) and exp(.) in its formulation.Interestingly, the derivative of the softplus function is the logistic function

11. Maxout function

$$max(w_1^T x + b_1, w_2^T x + b_2)$$

The Maxout activation is a generalization of the ReLU and the leaky ReLU functions. **It is a learnable activation function**.

Depiction of how the Maxout activation function can implement the ReLU, Absolute function, and approximate the quadratic function. A Maxout unit can approximate arbitrary convex functions

A maxout unit can **learn** a piecewise linear, convex function with up to k pieces.

So when k is 2, you can implement the ReLU, absolute ReLU, leaky ReLU, etc., or it can learn to implement a new function. If k is let's say 10, you can even approximately learn the convex function.

When k is 2: the Maxout neuron computes the function $max(wT1x+b1,wT2x+b2)max(w1Tx+b1,w2Tx+b2)$. Both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have $w1,b1=0w1,b1=0$). The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU).

- It is a piecewise linear function that returns the maximum of the inputs, designed to be used in conjunction with the dropout regularization technique.

- Both ReLU and leaky ReLU are special cases of Maxout. The Maxout neuron, therefore, enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU).

- However, it doubles the total number of parameters for each neuron and hence, a higher total number of parameters need to be trained.
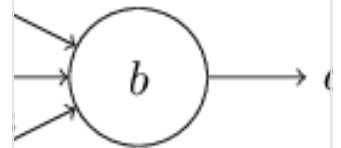
REFERENCES:

For Deep Study and code.

Neural networks and deep learning

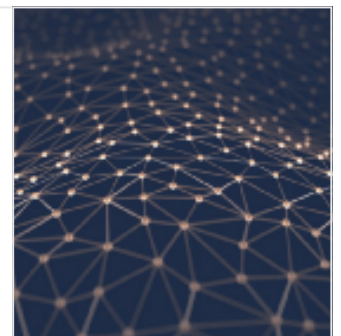When a golf player is first learning to play golf, they usually spend most of their time developing a basic swing. Only...

neuralnetworksanddeeplearning.com

Neural Networks: Cost Function and Backpropagation

Index \({(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots , (x^{(m)}, y^{(m)})}\) are the \(m\) training examples L is...

machinelearningmedium.com

Fundamentals of Deep Learning - Activation Functions and When to Use Them?

Overview Activation function is one of the building blocks on Neural Network Learn about the different activation...

www.analyticsvidhya.com

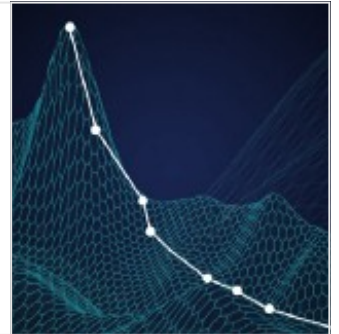**7 Types of Activation Functions in Neural Networks: How to Choose?**

missinglink.ai

A Detailed Guide to 7 Loss Functions for Machine Learning Algorithms with Python Code

Overview What are loss functions? And how do they work in machine learning algorithms? Find out in this article Loss...

www.analyticsvidhya.com

Wikipedia

Wikipedia is a free online encyclopedia, created and edited by volunteers around the world and hosted by the Wikimedia...

www.wikipedia.org

If There Is Any Mistake Do Not Hesitate Mentioning It.

I Hope You Got What You Were Looking For.

If You Appreciate My Effort,

Do Not Forget To Give A *CLAP*. It Cheers Me Up. Thank YOU!

Machine Learning   Deep Learning   Artificial Intelligence   Cost   Activation Functions

Medium

About   Write   Help   Legal