ACTIVATION FUNCTIONS IN NEURAL NETWORKS

Anshul Dhingra
Email - anshuljdhingra@gmail.com

ABSTRACT

The Backpropagation algorithm allows multilayer feed forward neural networks to learn input/output mappings from training observations. Backpropagation in neural networks adapt themselves to learn the relationship between the set of example patterns that could be leveraged to apply the same relationship to new input patterns. The network is able to focus on the features of an arbitrary input. The activation function is used to transform the activation level of a unit (neuron) into an output signal. There are a number of common activation functions in use with artificial neural networks (ANN). This paper aims to perform analysis of the different activation functions and provide a benchmark of it. The purpose is to figure out the optimal activation function for the problem of predicting flight delays or no delyas for Hawaiin Airflights. This further intends to help in better planning and control of disruption management.

1. INTRODUCTION

Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. A neural network is called a mapping network if it is able to compute some functional relationship between its input and output. For example, if the input to a network is the value of an angle, and the output is the cosine of the angle, the network performs the mapping $\theta \rightarrow \cos(\theta)$.

Suppose we have a set of P vector pairs $(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)$ which are examples of a functional mapping $\boldsymbol{y = \varphi(x)}$**:** $\boldsymbol{x \in R^N}$**,** $\boldsymbol{y \in R^M}$**.** We have to train the network so that it will learn an approximation $o = y = \varphi(x)$.

It should be noted that learning in a neural network means finding an approximate set of weights. Function approximation from a set of input- output pairs has numerous scientifi and engineering applications. Multilayer feed forward neural networks have been proposed as a tool for nonlinear function approximation [1], [2], [3]. Parametric models represented by such networks are highly nonlinear. The back propagation (BP) algorithm is a widely used learning algorithm for training multilayer networks by means of error propagation via variational calculus [4], [5]. It iteratively adjusts the network parameters to minimize the sum of squared approximation errors using a gradient descent technique. Due to the highly nonlinear modeling power of such networks, the learned function may interpolate all the training points. When noisy training data are present, the earned function can oscillate abruptly between data points. This is clearly undesirable for function approximation from noisy data.

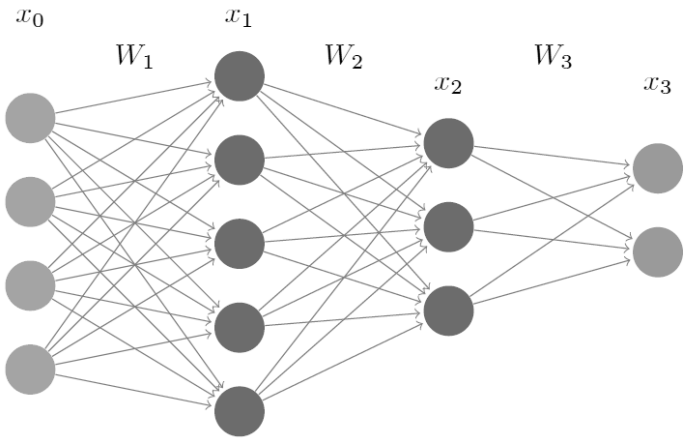
2. FUNDAMENTALS OF BACKPROPAGATION

Backpropagation is an algorithm used to train neural networks, used along with an optimization routine such as gradient descent. Gradient descent requires access to the gradient of the loss function with respect to all the weights in the network to perform a weight update, in order to minimize the loss function. Backpropagation computes these gradients in a systematic way. Backpropagation along with Gradient descent is arguably the singleB most important algorithm for training Deep Neural Networks and could be said to be the driving force behind the recent emergence of Deep Learning.

Any layer of a neural network can be considered as an Affine Transformation followed by application of a non linear function. A vector is received as input and is multiplied with a matrix to produce an output , to which a bias vector may be added before passing the result through an activation function such as sigmoid.

Input=x
Output=f(Wx+b)

Consider a neural network with a single hidden layer like this one. It has no bias units. We derive forward and backward pass equations in their matrix form.



The forward propagation equations are as follows:

Input=x0
Hidden Layer1 output=x1=f1(W1x0)
Hidden Layer2 output=x2=f2(W2x1)
Output=x3=f3(W3x2)

To train this neural network, we could either use Batch gradient descent or Stochastic gradient descent. Stochastic gradient descent uses a single instance of data to perform weight updates, whereas the Batch gradient descent uses a a complete batch of data.

For simplicity, lets assume this is a multi-regression problem.
Stochastic update loss function:

$$E = \frac{1}{2}\|z - t\|_2^2$$

Batch update loss function: $$E = \frac{1}{2}\sum_{i \in Batch} \|z_i - t_i\|_2^2$$

Here t is the ground truth for that instance.

We will only consider the stochastic update loss function. All the results hold for the batch version as well.
Let us look at the loss function from a different perspective. Given an input x0, output x3 is determined by W1,W2 and W3. So the only tuneable parameters in E are W1,W2 and W3. To reduce the value of the error function, we have to change these weights in the negative direction of the gradient of the loss function with respect to these weights.

$w = w - \alpha w * \partial E / \partial w$ for all the weights w

Here αw is a scalar for this particular weight, called the learning rate. Its value is decided by the optimization technique used.
Backpropagation equations can be derived by repeatedly applying the chain rule. First we derive these for the weights in W3:

$$\begin{aligned}
\frac{\partial E}{\partial W_3} &= (x_3 - t)\frac{\partial x_3}{\partial W_3} \\
&= [(x_3 - t) \circ f_3'(W_3 x_2)]\frac{\partial W_3 x_2}{\partial W_3} \\
&= [(x_3 - t) \circ f_3'(W_3 x_2)]x_2^T \\
\text{Let } \delta_3 &= (x_3 - t) \circ f_3'(W_3 x_2) \\
\frac{\partial E}{\partial W_3} &= \delta_3 x_2^T
\end{aligned}$$

Here ◦ is the Hadamard product.
Lets sanity check this by looking at the dimensionalities. ∂E/∂W3 must have the same dimensions as W3. W3's dimensions are 2×3. Dimensions of (x3−t) is 2×1 and f′3(W3x2) is also 2×1, so δ3 is also 2×1. x2 is 3×1, so dimensions of δ3xT2 is 2×3, which is the same as W3.
Now for the weights in W2:

$$\begin{aligned}
\frac{\partial E}{\partial W_2} &= (x_3 - t)\frac{\partial x_3}{\partial W_2} \\
&= [(x_3 - t) \circ f_3'(W_3 x_2)]\frac{\partial (W_3 x_2)}{\partial W_2} \\
&= \delta_3 \frac{\partial (W_3 x_2)}{\partial W_2} \\
&= W_3^T \delta_3 \frac{\partial x_2}{\partial W_2} \\
&= [W_3^T \delta_3 \circ f_2'(W_2 x_1)]\frac{\partial W_2 x_1}{\partial W_2} \\
&= \delta_2 x_1^T
\end{aligned}$$

Lets sanity check this too. W2's dimensions are 3×5. δ3 is 2×1 and W3 is 2×3, so WT3δ3 is 3×1. f′2(W2x1) is 3×1, so δ2 is also 3×1. x1 is 5×1, so δ2xT1 is 3×5. So this checks out to be the same.
Similarly for W1:

$$\begin{aligned}
\frac{\partial E}{\partial W_1} &= [W_2^T \delta_2 \circ f_1'(W_1 x_0)]x_0^T \\
&= \delta_1 x_0^T
\end{aligned}$$

We can observe a recursive pattern emerging in the backpropagation equations. The Forward and Backward passes can be summarized as below:
The neural network has L layers. X0 is the input vector, xL is the output vector and t is the truth vector. The weight matrices are W1,W2,..,WL and activation functions are f1,f2,..,fL.

Forward Pass:

$$\begin{aligned}
x_i &= f_i(W_i x_{i-1}) \\
E &= \|x_L - t\|_2^2
\end{aligned}$$

Backward Pass:

$$\begin{aligned}
\delta_L &= (x_L - t) \circ f_L'(W_L x_{L-1}) \\
\delta_i &= W_{i+1}^T \delta_{i+1} \circ f_i'(W_i x_{i-1})
\end{aligned}$$

Weight Update:

$$\frac{\partial E}{\partial W_i} = \delta_i x_{i-1}^T$$

$$W_i = W_i - \alpha_{W_i} \circ \frac{\partial E}{\partial W_i}$$

Equations for Backpropagation, represented using matrices have two advantages.

One could easily convert these equations to code using either Numpy in Python or Matlab. It is much closer to the way neural networks are implemented in libraries. Using matrix operations speeds up the implementation as one could use high performance matrix primitives from BLAS. GPUs are also suitable for matrix computations as they are suitable for parallelization.

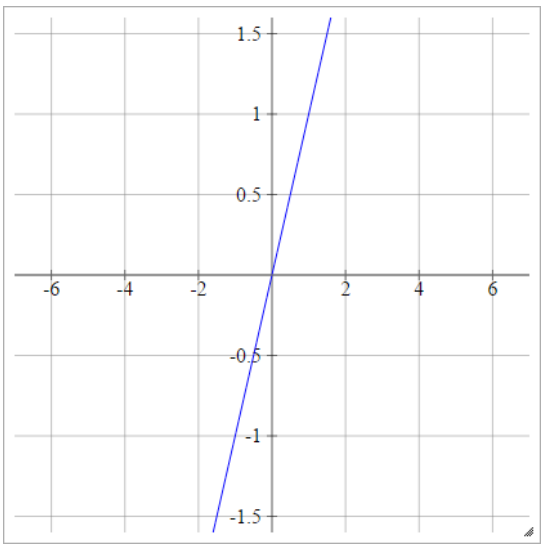
The matrix version of Backpropagation is intuitive to derive and easy to remember as it avoids the confusing and cluttering derivations involving summations and multiple subscripts.

3. TYPES OF ACTIVATION FUNCTIONS

3.1 Identity

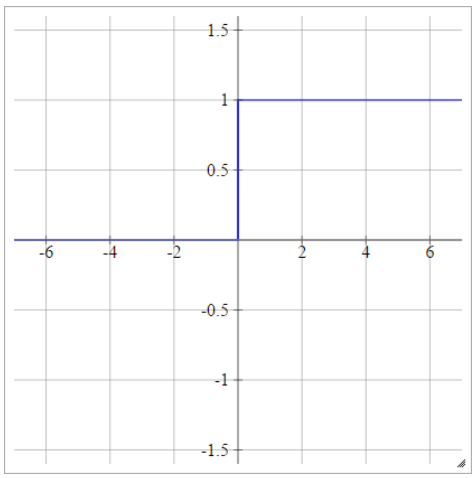
Also known as a linear activation function.

$$a_j^i = \sigma(z_j^i) = z_j^i$$



3.2 Step

$$a_j^i = \sigma(z_j^i) = \begin{cases} 0 & \text{if } z_j^i < 0 \\ 1 & \text{if } z_j^i > 0 \end{cases}$$



3.3 Piecewise Linear

Choose some xmin and xmax, which is our "range". Everything less than than this range will be 0, and everything greater than this range will be 1. Anything else is linearly-interpolated between. Formally:

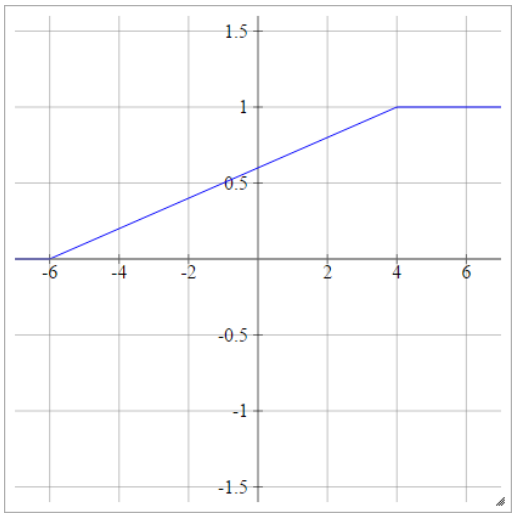
$$a_j^i = \sigma(z_j^i) = \begin{cases} 0 & \text{if } z_j^i < x_{\min} \\ mz_j^i + b & \text{if } x_{\min} \leq z_j^i \leq x_{\max} \\ 1 & \text{if } z_j^i > x_{\max} \end{cases}$$

Where

$$m=1/xmax-xmin$$

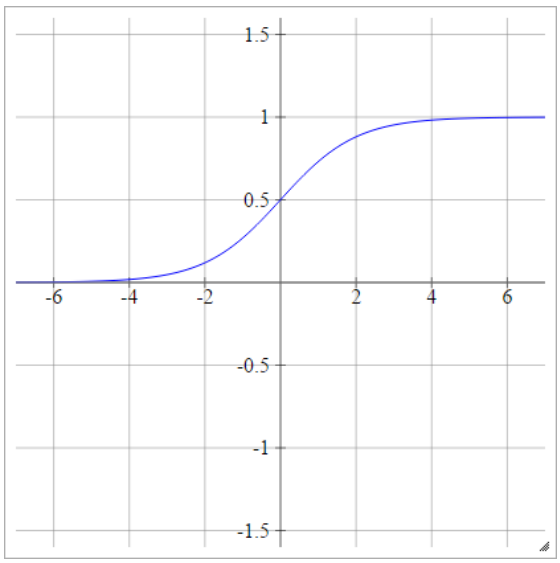
and

$$b=-mxmin=1-mxmax$$



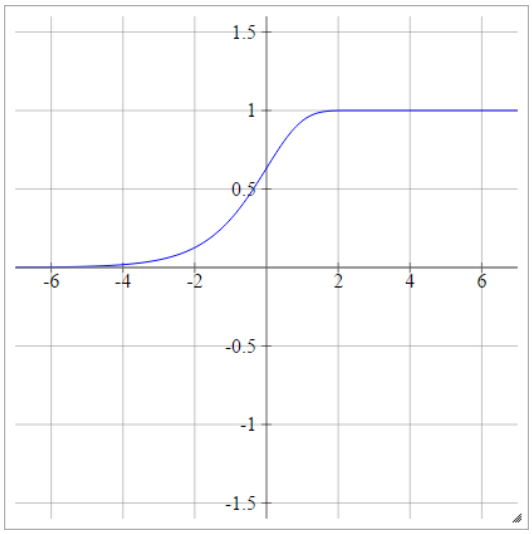
3.4 Sigmoid

$$a_j^i = \sigma(z_j^i) = \frac{1}{1 + \exp(-z_j^i)}$$



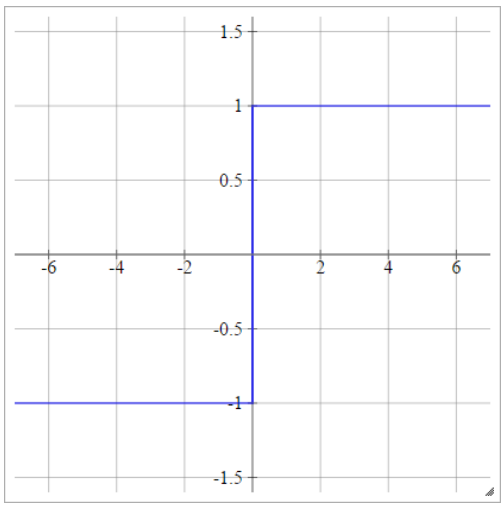
3.5 Complementary log-log

$$a_j^i = \sigma(z_j^i) = 1 - \exp\big(-\exp(z_j^i)\big)$$



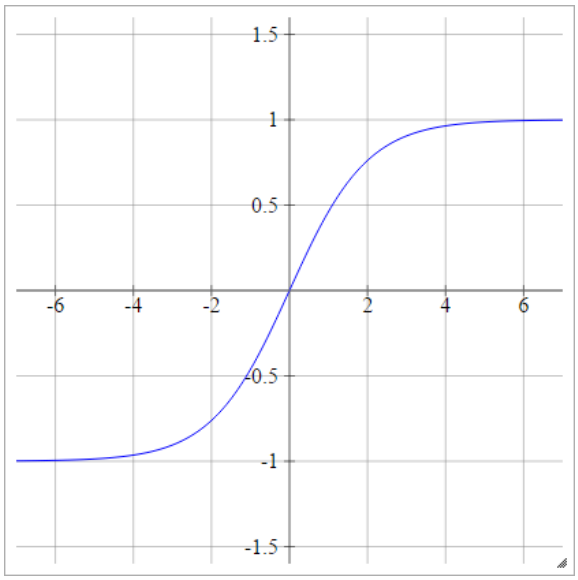
3.6 Bipolar

$$a_j^i = \sigma(z_j^i) = \begin{cases} -1 & \text{if } z_j^i < 0 \\ 1 & \text{if } z_j^i > 0 \end{cases}$$



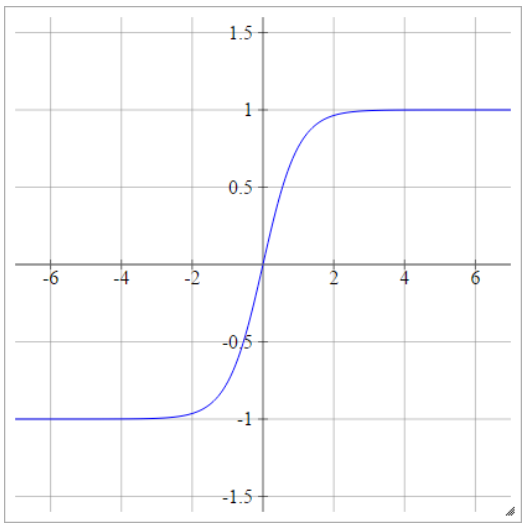
3.7 Bipolar Sigmoid

$$a_j^i = \sigma(z_j^i) = \frac{1 - \exp(-z_j^i)}{1 + \exp(-z_j^i)}$$



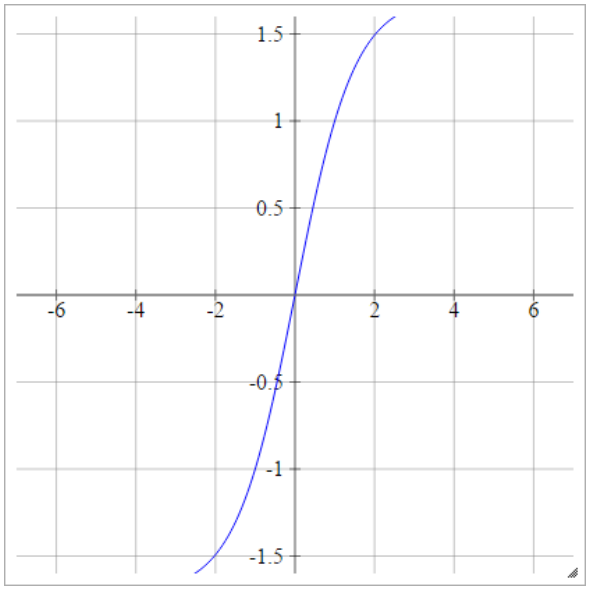
3.8 Tanh

$$a_j^i = \sigma(z_j^i) = \tanh(z_j^i)$$

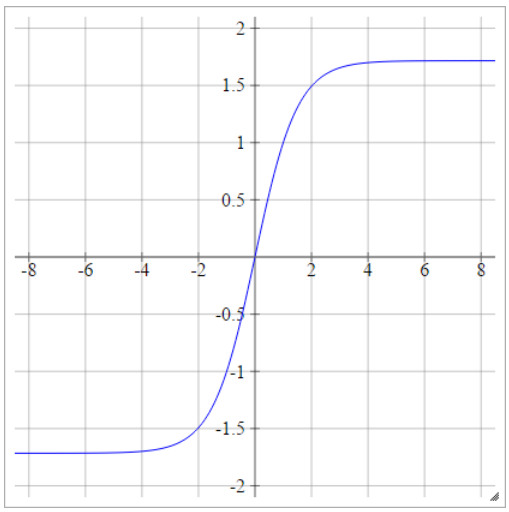


3.9 LeCun's Tanh

$$a_j^i = \sigma(z_j^i) = 1.7159 \tanh\left(\frac{2}{3} z_j^i\right)$$

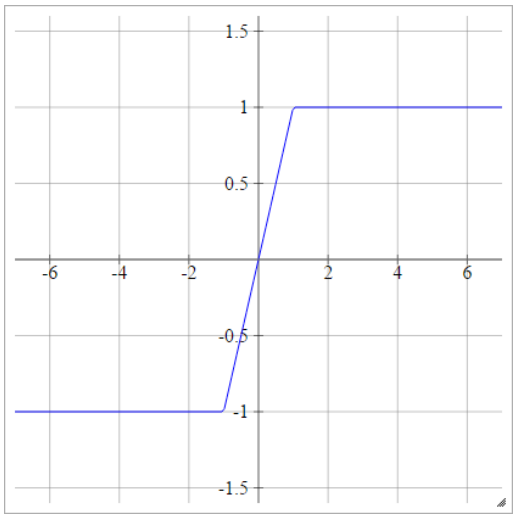


Scaled:



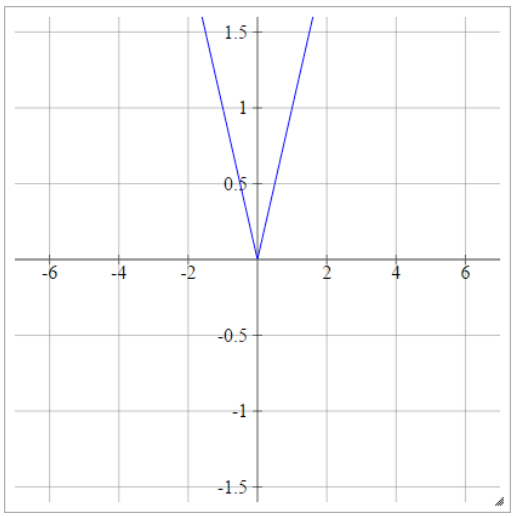
3.10 Hard Tanh

$$a_j^i = \sigma(z_j^i) = \max\big(-1, \min(1, z_j^i)\big)$$



3.11 Absolute

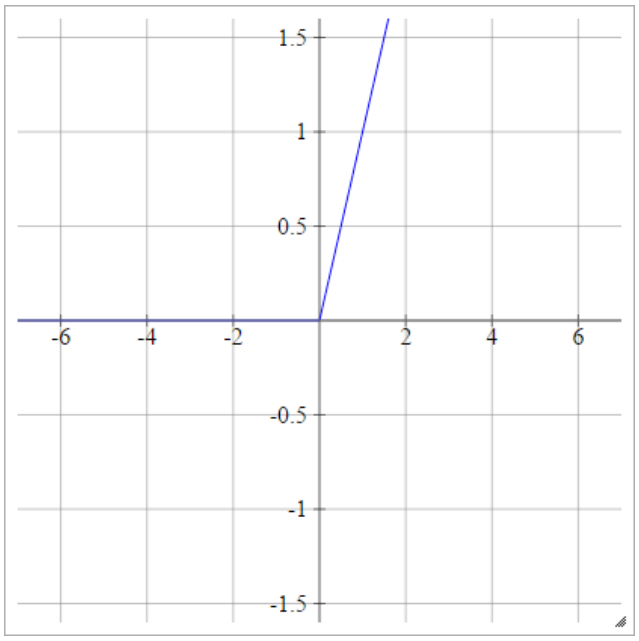
$$a_j^i = \sigma(z_j^i) = \mid z_j^i \mid$$



3.12 Rectifier

Also known as Rectified Linear Unit (ReLU), Max, or the Ramp Function.

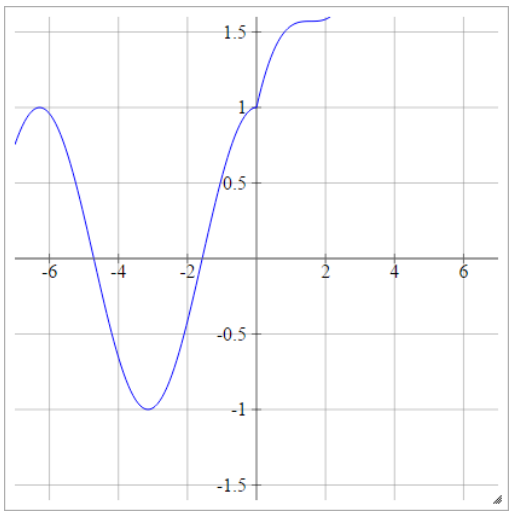
$$a_j^i = \sigma(z_j^i) = \max(0, z_j^i)$$



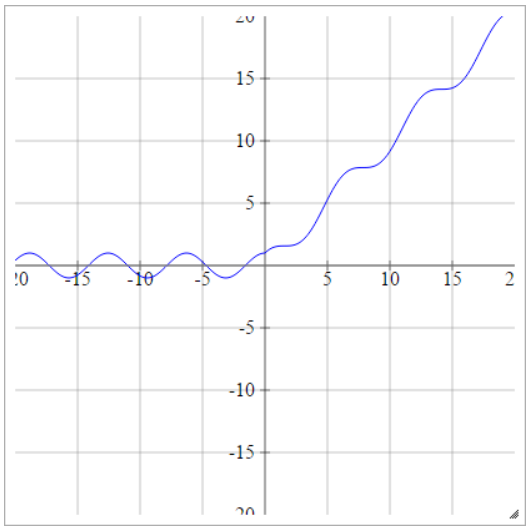
Modifications of ReLU

These are some activation functions that have been discovered over time :

$$a_j^i = \sigma(z_j^i) = \max(0, z_j^i) + \cos(z_j^i)$$

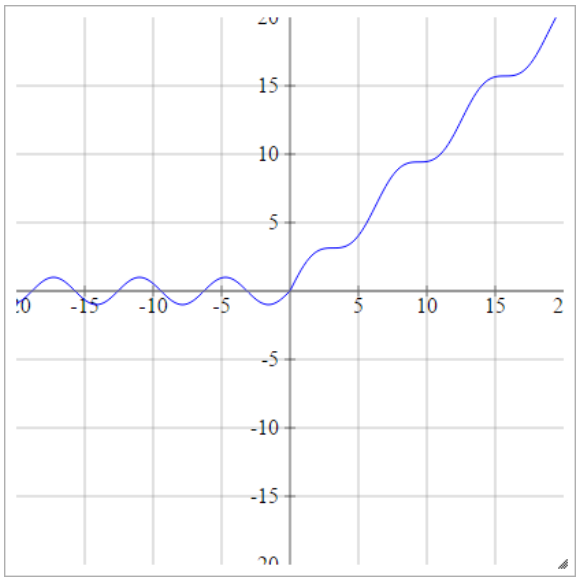


Scaled



$$a_j^i = \sigma(z_j^i) = \max(0, z_j^i) + \sin(z_j^i)$$

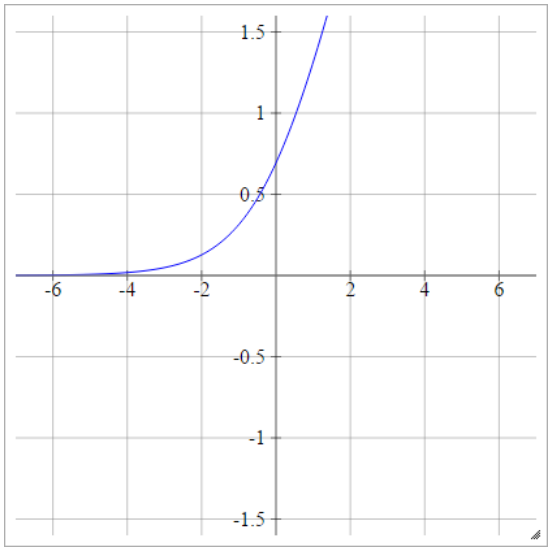
Scaled:



3.13 Smooth Rectifier

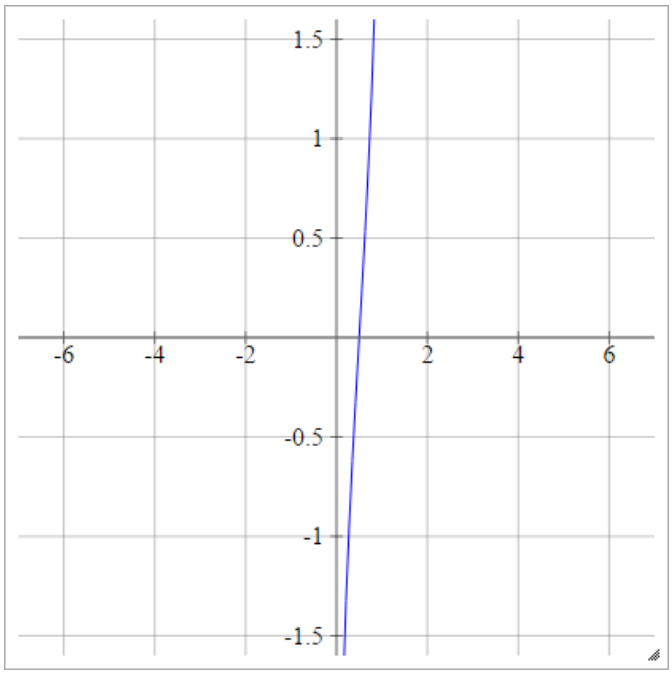
Also known as Smooth Rectified Linear Unit, Smooth Max, or Soft plus

$$a_j^i = \sigma(z_j^i) = \log\left(1 + \exp(z_j^i)\right)$$



3.14 Logit

$$a_j^i = \sigma(z_j^i) = \log\left(\frac{z_j^i}{(1 - z_j^i)}\right)$$



3.15 Probit

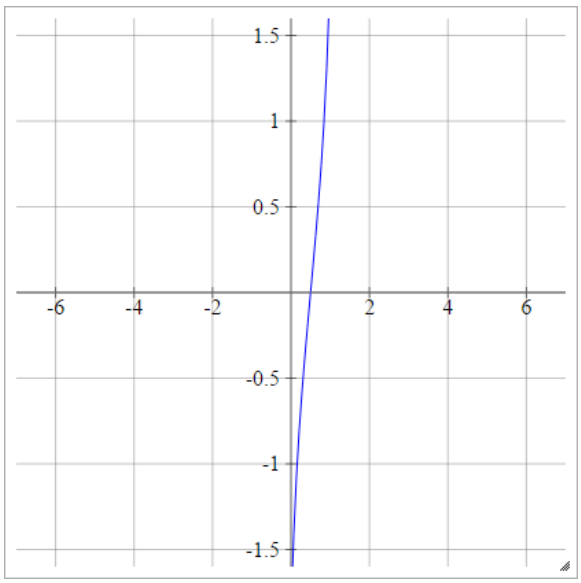
.

$$a_j^i = \sigma(z_j^i) = \sqrt{2}\,\mathrm{erf}^{-1}(2z_j^i - 1)$$

Where erf is the Error Function. It
can't be described via elementary functions,
Alternatively, it can be expressed as

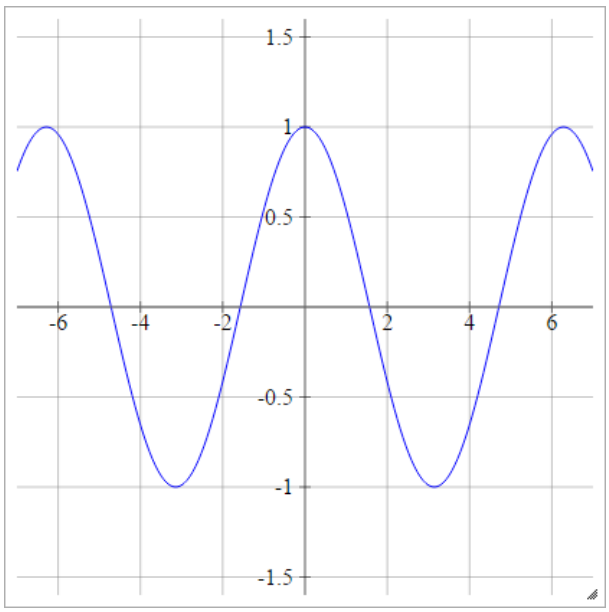
$$a_j^i = \sigma(z_j^i) = \phi(z_j^i)$$

Where ϕ is the Cumulative distribution function (CDF).



3.16 Cosine

$$a_j^i = \sigma(z_j^i) = \cos(z_j^i)$$



3.17 Softmax

Also known as the Normalized Exponential.

$$a_j^i = \frac{\exp(z_j^i)}{\sum_k \exp(z_k^i)}$$

3.18 Maxout

Essentially the idea is that we break up each neuron in our maxout layer into lots of sub-neurons, each of which have their own weights and biases. Then the input to a neuron goes to each of it's sub-neurons instead, and each sub-neuron simply outputs their z's (without applying any activation function). The aij of that neuron is then the max of all its sub-neuron's outputs.

Formally, in a single neuron, say we have n sub-neurons. Then

$$a_j^i = \max_{k \in [1,n]} s_{jk}^i$$

where

$$s_{jk}^i = a^{i-1} \bullet w_{jk}^i + b_{jk}^i$$

4. EXPERIMENT SETUP & RESULTS

A dataset was chosen for evaluation of the activation network. A simulator was specially developed for testing the activation function using an open source library FANN (Fast Artifical Neural Network). The simulator was developed in Python and language bindings for FANN was used which itself was created using SWIG (Simplified Wrapper Interface Generator).
The business case that this paper tries to solve is of Disruption Management for Hawaiin Airlines. The dataset has been created after merging of two different types of raw data related to flights and weather obtained from the following sources for flights operating from Honolulu airport for Hawaiin Airlines -

a) Flights Statistics & operations data obtained from Bureau of Transporation Statistics (https://www.bts.dot.gov/)
b) Weather information obtained from Wunderground (https://www.wunderground.com/)

The problem presented here is of binary classification that predicts whether a Hawaiin Airflight operating from Honolulu would be delayed or not based on the dataset fed into FANN. 36 different features like Flight Time, Aircraft Type, Time of flight, Ground Operations, Temperature, Humidity, Air pressure among others were fed into the input layer of FANN. These 36 features were converted into 122 different neurons on the basis of implict feature engineering carried out by 4 seperate hidden layers. At the output layer, sigmoid function is used to derive the probabilities of prediction for binary classification.
The following table describes the impact of different activationn functions on predition -

Activation Function	Total Number of Epochs	Error at Last Epoch	Bit Fail at Last Epoch
Identity	44	0.3106114744	23
Step	32	0.3110372145	5
Piecewise Linear	43	0.1207051133	5
Sigmoid	28	0.1191550087	46
Complementary log-log	55	0.0077741441	21
Bipolar	24	0.2257441336	7
Bipolar Sigmoid	21	0.2291534554	5
Tanh	47	0.1091559774	119
LeCun's Tanh	73	0.1093388141	93
Hard Tanh	23	0.2085678089	114
Absolute	35	0.4089784233	61
Relu	47	0.0160981354	43
Leaky Relu	46	0.0111534554	41
Smooth Rectifier	55	0.3091559774	51
Logit	47	0.5093388141	129
Probit	39	0.7085678089	79
Cosine	56	0.9089784233	72
Softmax	31	0.1060981354	81
Maxout	41	0.7060981354	112

5. CONCLUSION:

Activation function is one of the essential parameter in a Neural Network. The performance evaluation of different activation functions shows up that the selection of an activation function plays an important role in faster convergence and minimalization of the error, thereby, enhancing the network performance and increasing prediction accuracy. As shown in the table above,Leaky Relu and Relu outperforms other activation functions by good error margin (though Leaky Relu performs better than Relu only marginally).

This paper emphasizes that although selection of an activation function for a neural network or it's node is an important task, other factors like training algorithm, network sizing and learning parameters are also vital for proper training of the network.

6. REFERENCES:

[1] K. Homik, M. Stinchcombe, and H. White, “Multilayer feed forward networks are universal approximators”,ᔓ Neural Networks, vol. 2, pp.359-366, 1989.

[2] C. Ji, R. R. Snapp, and D. Psaltis, “Generalizing smoothness constraints from discrete samples”, Neural Computation, vol. 2, pp. 188-197, 1990.

[3] T. Poggio, and F. Girosi, “Networks for approximation and learning.” Proc. IEEE. vol. 78, no. 9, pp. 1481-1497. 1990.

[4] Y. Le Cun, “A theoretical framework for back propagation”, in Proc.1988 Connectionist Models Summer School, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. June 17-26, 1988. San Mateo, CA: Morgan Kaufmann, pp. 21-28.

[5] D. E. Rummelhart, G. E. Hinton. and R. J . Williams. “Parallel Distributed Processing: Explorations in the Microstructure of Cognition”, vol. I. MIT Press, ch. 8.

[6] James A. Freeman and David M. Skapura. “Neural Networks Algorithms, Applications and Programming Techniques”, pp 115-116, 1991.

[7] Jacek M. Zurada, "Introduction to Artificial Neural Systems", pp 32-36, 2006.

[8] Sudeep Raja , “A Derivation of Backpropagation in Matrix Form”, blog.