Performance Analysis of Various Activation Functions in Artificial Neural Networks

Performance Analysis of Various Activation Functions in Artificial Neural Networks

Jianli Feng, Shengnan Lu

School of Computer Science, Xi'an Shiyou University, Xi'an 710065, China

fjlnwpu@xsyu.edu.cn, lushengnan@xsyu.edu.cn

Abstract. The development of Artificial Neural Networks (ANNs) has achieved a lot of fruitful results so far, and we know that activation function is one of the principal factors which will affect the performance of the networks. In this work, the role of many different types of activation functions, as well as their respective advantages and disadvantages and applicable fields are discussed, so people can choose the appropriate activation functions to get the superior performance of ANNs.

1. Introduction

For an artificial neural network (ANNs), there are always many neurons that work in correspondence of weights and bias, and through these neurons people can get the outputs from the inputs[1], as shown in Fig. 1. Without activation functions, the outputs can be anything on the ranging $\left[-Inf,+Inf\right]$, so the neurons really don't know the bounds of the value. And, people always models ANNs as Input Layer, Hidden Layer and Output Layer no matter, and in fact, there is always more than one hidden layer. So, if activation functions absent, the output of each layer is a linear function of the upper layer. No matter how many layers the ANN has, the outputs are linear combination of the inputs. Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it, which will introduce non-linearity into the output of a neuron, and let the outputs on the ranging $\left[0,1\right]$, or $\left[-1,1\right]$. Activation Function is also known as Transfer Function. It can also be attached in between two neural networks. Meanwhile, in order to get the accurate outputs in ANNs, people need to update the weights and biases of the neurons on the basis of the error at the output. This process is known as the Back Propagation (BP) [2]. Activation functions make it possible since the gradients are supplied along with the error to update the weights and biases. Among them, Gradient Descent Optimization is the most commonly used optimization algorithm for the ANNs.

In all of the non-saturated activation functions, Rectified Linear Unit (ReLu) is the most famous, which was proposed by V.Nair(et al.) in 2010[3]. G.E.Dahl improved 4.2% by using ReLu in a Deep Neural Networks (DNNs), and 14.4% in a strong Gaussian Mixture Model/Hidden Markov Model (GMM/HMM) system[4]. Kazuyuki Hara analyzed the ReLu by soft-committee machine and explained that ReLu improves the learning process in theory[5]. Glorot Xavier (et al.) found rectifier activation functions not only are good at image classification tasks, but also have good performance in text mining[6]. Leaky ReLu was first presented by Xiaojie Jin(et al.). They used it in the DNNs acoustic models, and they got the performance improvement due to the sparsity and dispersion of ReLu[7]. Also, many people utilized Leaky ReLu on acoustic scene classification task[8-9]. Bing Xu(et al.) compared the performance of ReLu, Leaky Rectified Linear Unit (Leaky ReLu), Parametric

Rectified Linear Unit (PReLu) and Randomized Leaky Rectified Linear Unit (RReLu), then found the modified functions all outperformed the ReLu[10]. K.M.He (et al.) reported that their multi-model PReLu network achieved 4.94% top-5 on the 1000-class ImageNet 2012 dataset[11], exceeded O. Russakovsky's record 5.1%[12]. In addition, there are some other new activation functions, such as Adaptive Piecewise activation function[13], Maxout[14], and Network-in-Network[15]. Moreover, people are convinced that with the deepening of research, there will be more superior performance of the activation function appears.

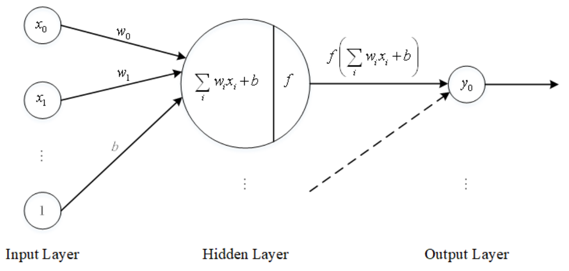


Fig. 1 The model of Artificial Neural Networks

2. Performance of Different Types of Activation Functions

Activation Functions can be basically divided into 2 types, linear activation functions and non-linear activation functions. Where linear activation functions maintain a constant, non-linear activation functions create more variation which utilizes the build of the neural network.

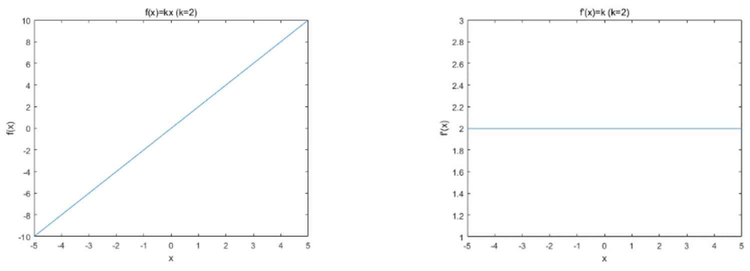
2.1. Linear activation function

Linear activation function has the equation similar to as of a straight line, the activation is proportional to input, it is given by,

$$f(x_i) = kx_i \tag{1}$$

Where x_i is the input of the activation function f on the i th channel, and k is a fixed constant, $f(x_i)$ is the corresponding output of the input. So, the derivative with respect to x_i is

$$f'(x_i) = k \tag{2}$$



(a) Curve of function (b) Curve of derivative

Fig. 2 Linear activation function and its derivative

According to Eq. 1, the Linear activation function is a linear function, the output is k times of input. If all layers' activation functions are linear, then no matter how many layers the net has, the final activation function of last layer is nothing but just a linear function of the input of first layer, that is to say the whole network is equivalent to a single layer with linear activation function, of course, the outputs can be anything on the ranging $[-Inf, +Inf]$. According to Eq. 2, the derivative is k , note k is a constant, so the gradient is a constant, and has no relationship with input, so people cannot decrease the error by gradient, you can also find it in Fig. 2.

2.2. Non-linear activation functions

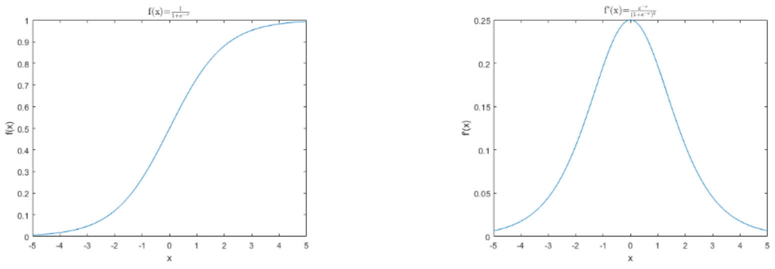
Compared to the linear activation functions, non-linear activation functions are more widely used. They make it easy for the model to generalize or adapt with a variety of data and to differentiate between the outputs. There are many typical non-linear activation functions, and they are always classified by their range or curves.

2.2.1. Sigmoid Function

The Sigmoid Function is given by,

$$f(x_i)=\frac{e^{x_i}}{1+e^{x_i}}=\frac{1}{1+e^{-x_i}}\ , \qquad f'(x_i)=\frac{e^{-x_i}}{\left(1+e^{x_i}\right)^2} \tag{3}$$

From Fig. 3, the Sigmoid Function is nonlinear in nature, it's on the ranging $[0,1]$, and its curve looks like a S-shape. Although it is monotonic, and its output is very steep, small changes of input would bring the large changes of output when the input is near 0, but towards either end of it, the output tend to 0 or 1, respond very less to the input, that is to say, the gradient at this region is going to be very small, even close to 0, this is called "vanishing gradients". In this case, the network will refuse to learn further, almost no signal will flow through the neuron to its weights and recursively to its data. So, people should pay extra caution when initializing the weights of sigmoid neurons to prevent saturation. Fortunately, there are ways to work around this problem and sigmoid is still very popular in classification problems, especially in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be **1** if value is greater than **0.5** and **0** otherwise.



(a) Curve of function (b) Curve of derivative

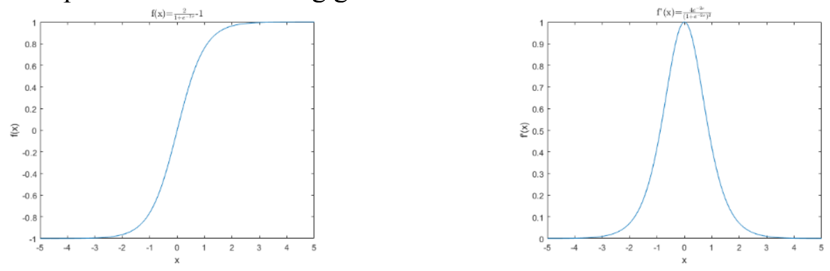
Fig.3 Sigmoid function and its derivative

2.2.2. Tanh Function

Tanh function also known as Tangent Hyperbolic Function, it can obtained from Sigmoid Function.

$$\tanh(x_i)=\frac{\sinh x_i}{\cosh x_i}=\frac{e^{x_i}-e^{-x_i}}{e^{x_i}+e^{-x_i}}=\frac{2}{1+e^{-2x_i}}-1=2sigmoid(2x_i)-1,\ \tanh'(x_i)=\frac{4e^{-2x_i}}{\left(1+e^{-2x_i}\right)^2} \tag{4}$$

Fig. 4 demonstrates its two curves. Compared with Sigmoid Function, Tanh function is also nonlinear, but unlike Sigmoid, its output is zero-centred, and it squashes a real-valued number to the range $[-1,1]$, so no worry of activations blowing up. And its gradient is stronger than Sigmoid, of course, it also has the problem of "vanishing gradients".



(a) Curve of function (b) Curve of derivative

Fig. 4 Tanh Function and its derivative

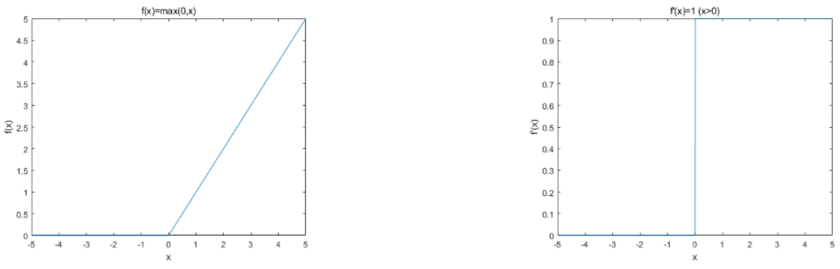
The range $\left[-1,1\right]$ makes the mean of the outputs come to be 0 or very close to 0, furthermore, makes the data more concentrated, and makes the learning much easier, thus it is usually used in hidden layers of ANNs.

2.2.3. ReLu

ReLu has become very popular in the last few years, it is usually implemented in hidden layers of ANNs, especially in almost all the Convolutional Neural Networks (CNN) or Deep Learning (DP). It is given by,

$$f\left(x_i\right)=\max\left(0,x_i\right)=\begin{cases}x_i, x_i>0\\0, x_i<0\end{cases}, \qquad f^{'}\left(x_i\right)=\begin{cases}1, x_i>0\\0, x_i<0\end{cases} \tag{5}$$

From Fig.5, it looks some like linear, but in fact it is nonlinear, because the output is 0 when $x_i<0$, and it avoids and rectifies vanishing gradient problem. Meanwhile, it is less computationally expensive than Sigmoid and Tanh.



(a) Curve of function (b) Curve of derivative

Fig.5 ReLu and its derivative

Of course, it is undeniable that ReLu no upper limit, it can blow up the activation. And because of the gradient is 0 for negative inputs, the weights will not get adjusted during descent, furthermore, the neurons which go into that state will stop responding to variations of inputs, they become stuck in a perpetually inactive state and “die”, this is called “Dying ReLu” problem.

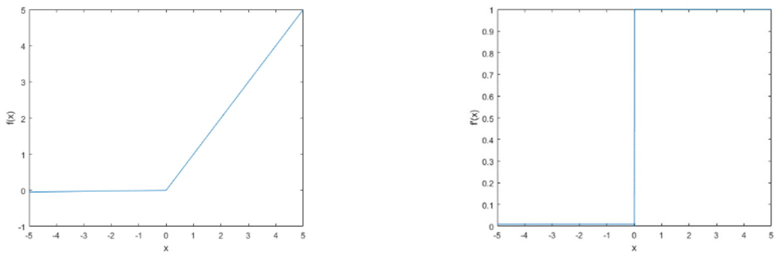
2.2.4. Leaky ReLu

Leaky ReLu is defined as

$$f\left(x_i\right)=\begin{cases}x_i, x_i>0\\\alpha_i x_i, x_i<0\end{cases}, \qquad f^{'}\left(x_i\right)=\begin{cases}1, x_i>0\\\alpha_i, x_i<0\end{cases} \tag{6}$$

Where α_i is the coefficient of i th channel for negative inputs, and it is a small constant, usually is 0.01 or so. Its output is not 0 for negative inputs, so it is the improvement of ReLu function for the problem of “Dying ReLu”. It is easy for us to get its derivative,

Fig. 6 shows its curve and derivative, because it can produce a constant times input value for negative inputs, so compared with ReLu, it will not saturate for both direction.



(a) Curve of function (b) Curve of derivative

Fig.6 Leaky ReLu and its derivative

2.2.5. PReLu
PReLu is given by,

$$f\left(x_i\right)=\begin{cases} x_i, x_i>0 \\ \alpha_i x_i, x_i<0 \end{cases}, \qquad f'\left(x_i\right)=\begin{cases} 1, x_i>0 \\ \alpha_i, x_i<0 \end{cases} \tag{7}$$

It looks like it is same as Leaky ReLu, but in fact, here the parameter α_i changes on different channels, and it can be learned in the networks by BP. Certainly, people can easily find that when $\alpha_i = 0$, PReLu will become ReLu; when α_i is a small and fixed value, it will become Leaky ReLu.

Due to its addition, it will increase the amount of parameters of the net, but according to K.M.He[11], it is a very small amount as opposed to the total number of weights, meanwhile, he used channel-shared variant let the α_i was shared by all channels of one layer, so there are only one parameter will be added to each layer.

2.2.6. RReLU
Like Leaky ReLu and PReLu, RReLU is also an improvement on the coefficients for negative inputs[10]. It is defined as,

$$f\left(x_{ji}\right)=\begin{cases} x_{ji}, x_{ji}\geq 0 \\ \alpha_{ji} x_{ji}, x_{ji}<0 \end{cases} \tag{8}$$

Where α_{ji} is the slope of negative inputs, and i refers to the channel and j refers to the example. It is subject to the following normal distribution,

$$\alpha_{ji} \sim U\left(l,u\right), l<u \ and \ l,u \in \left[0,1\right) \tag{9}$$

During the testing phase, α_{ji} is fixed, and the average of all α_{ji} is $\alpha_{ji}=\frac{l+u}{2}$. The motivation to introduce a random negative slope is to reduce overfitting. In the paper, they also found RReLu outperformed the others.

2.2.7. ELU
ELU means Exponential Linear Unit[16]. It is another type of activation function based on ReLu. It accelerates learning and alleviates the vanishing gradient problem. It is given as,

$$f\left(x_i\right)=\begin{cases} x_i, & x_i>0 \\ \alpha_i\left(e^{x_i}-1\right), & x_i\leq 0 \end{cases}, \qquad f'\left(x_i\right)=\begin{cases} 1, & x_i>0 \\ f\left(x_i\right)+\alpha_i, & x_i\leq 0 \end{cases} \tag{10}$$

According to Eq.10, the activations are very close to zero of negative inputs, their gradients similar to the natural gradient, so ELU speeds up learning. In [16], they found ELU was outstanding on CIFAR-10, CIFAR-100, ImageNet than ReLu and other improved activation functions.

3. Conclusion
In this work, we compare the characteristics of different activation functions, and explain their advantages and disadvantages. Each activation function has its own characteristics, so we cannot easily say that which one is the best, and the only thing we can do is choosing a suitable activation function based on our aim and our network structure.

Acknowledgments
This work is financially supported by the Key R&D Program Projects in Shaanxi Province (2018SF-409), and the authors would like to thank to all those who have made outstanding contributions to the Artificial Neural Networks.

References

[1] Jain Anil K., Mao Jianchang, Mohiuddin K.M. Artificial neural networks: A tutorial[J]. Computer, 1996, 29(3): 31-44.
[2] Cun Y.Le, Boser B., Denker J.S, et al. Handwritten digit recognition with a back-propagation network[C]. Advances in neural information processing systems. 1990: 396-404.
[3] Nair Vinod and Hinton Geoffrey E.. Rectified linear units improve restricted Boltzmann machines[C]. Proceedings of the 27th International Conference on Machine Learning (ICML10). 2010:807–814.
[4] Dahl George E., Sainath Tara N., Hinton Geoffrey E. Improving deep neural networks for LVCSR using rectified linear units and dropout[C]. Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013: 8609-8613.
[5] Hara Kazuyuki, Saito Daisuke, Shouno Hayaru. Analysis of function of rectified linear unit used in deep learning[C]. Neural Networks (IJCNN), 2015 International Joint Conference on. IEEE, 2015: 1-8.
[6] Glorot Xavier, Bordes Antoine, Bengio Yoshua. Deep sparse rectifier neural networks[C]. Proceedings of the fourteenth international conference on artificial intelligence and statistics. 2011: 315-323.
[7] Maas Andrew L., Hannun Awni Y., Ng Andrew Y. Rectifier nonlinearities improve neural network acoustic models[C]. Proceedings of the 30th International Conference on Machine Learning. 2013, 30(1)
[8] Han Yoonchang, Lee Kyogu. Convolutional neural network with multiple-width frequency-delta data augmentation for acoustic scene classification[J]. IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events, 2016.
[9] Li Peter, Qian Jiyuan, Wang Tian. Automatic instrument recognition in polyphonic music using convolutional neural networks[J]. arXiv preprint arXiv:1511.05520, 2015.
[10] Xu Bing, Wang Naiyan, Chen Tianqi, et al. Empirical evaluation of rectified activations in convolutional network[J]. arXiv preprint arXiv:1505.00853, 2015.
[11] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification[C]. Proceedings of the IEEE international conference on computer vision. 2015: 1026-1034.
[12] Russakovsky Olga, Deng Jia, Su Hao, et al. Imagenet large scale visual recognition challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.
[13] Agostinelli Forest, Hoffman Matthew, Sadowski Peter, et al. Learning activation functions to improve deep neural networks[J]. International Conference on Learning Representations (ICLR) 2015, 1-9
[14] Goodfellow Ian J., Warde-Farley David, Mirza Mehdi, et al. Maxout networks[J]. arXiv preprint arXiv:1302.4389, 2013.
[15] Lin Min, Chen Qiang, Yan Shuicheng. Network in network[J]. arXiv preprint arXiv:1312.4400, 2013.
[16] Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus)[J]. arXiv preprint arXiv:1511.07289, 2015