Retrospective Review of Activation Functions in Artificial Neural Networks

Manjunatha Badiger and Jose Alex Mathew

Abstract Deep learning is a subfield of machine learning and artificial intelligence technique. It employs neural network tasks like image processing, computer vision, voice recognition, machine translation, medical information processing, self-driving vehicles, predictive forecasting, robotics and control, cybersecurity, natural language processing, bioinformatics, and countless others. The performance of a neural network is determined by a variety of factors, and activation functions are an essential element in the design of a neural network. The hidden layer's activation feature defines the extent to which the network model learns the training data set. The type of prognostications made by the model is regulated by the activation functions present at the output layer. This paper presents a comprehensive review of research studies on different activation functions aimed toward deep learning applications.

Keywords Deep learning · Neural network · Machine learning · Artificial intelligence · Activation function

Introduction

Deep learning is a form of hierarchical learning. It comprises algorithms and topologies to solve a wide variety of issues. Deep learning is a feature-learning method with many levels of representation. Thus, it will be easy to understand nonlinear representations one layer at a time [1]. The lower-level features are minor details that are used to transform the representation at one level to build high-level features on top of it [2]. Thus, the complex functions can be learned using such transformations. Over the last few decades, deep learning has become a very popular and most powerful tool as it can handle a huge quantity of data. Deep learning architectures, which are outstandingly, expanded the number and types of problems that neural networks can

M. Badiger (✉)
Sahyadri College of Engineering & Management, Adyar, India

J. A. Mathew
Srinivas Institute of Technology, Mangalore, India

V. Bindhu et al. (eds.), *Proceedings of Third International Conference on Communication, Computing and Electronics Systems*, Lecture Notes in Electrical Engineering 844,
https://doi.org/10.1007/978-981-16-8862-1_59

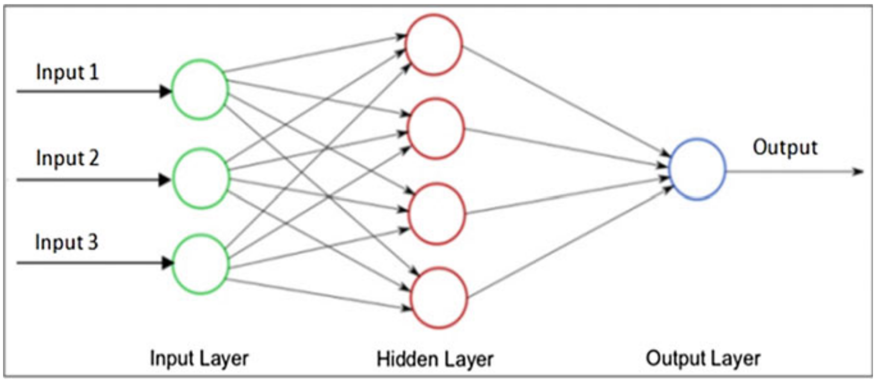

Fig. 1 Simple neural network

handle. Deep learning has shown several advancements by researchers and academicians in the last two decades [3]. Neural networks are a novel architecture imitating biological neural networks [4]. The rudimentary building block of artificial neural networks is a neuron. A neuron is a mathematical function that simulates the functioning of a biological neuron [5]. Neural networks are composite computer codes written with many basic, highly interconnected computing components that mimic human biological brain structure operations to simulate data models of human brain functioning and processing. Figure 1 shows the simple neural network. The neural network comprises three layers, namely activation function, learning technique, and weights. All these layers include neurons that are interlinked to form a network [6]. Its elements are validated depending on whether the neuron is used for input, output, or in one of the hidden layers.

The two key hyperparameters which control the architecture or topology of neural networks are the total number of layers and the number of total nodes in each hidden layer as shown in Fig. 2.

Input layer: It receives input either through an outside source or through other neighboring nodes. Every node is attached to another node of the succeeding layer. Each connection has a specific weight. Depending on its degree of importance, weights are assigned to a neuron in relation to other inputs. Once the entire node values from the input side are multiplied by their corresponding weights and totted up, the value for hidden layers is generated [6]. The output of the input layer can be given by the equation:

$$y_i = \sum(\text{weights} * \text{input} \ + \text{bias}) \tag{1}$$

It can range from −infinity to +infinity. So it is necessary to bound the output to get the desired prediction or generalized results.

Hidden layers: Hidden layers are always found between the input and output layers. It is always shrouded from the outside world. Hidden layers may vary from

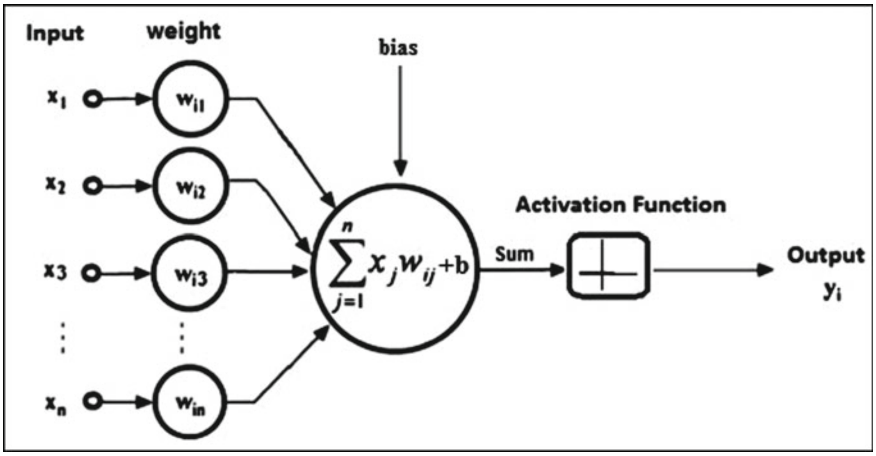

Fig. 2 Complex neural network

network to network that we selected. The number of hidden layers in a neural network is determined by the problem's nature and size. Once the hidden layer gets information from the input side, it performs all the computational tasks and provides the result [6, 7]. This result is then forwarded to the output layer. Hidden layers refine the input weightings until the marginal error of the neural network is small.

Output layer: The output nodes are known collectively as the output layer, and they are responsible for bringing out the final result. This output layer is designed differently to contour and improve the final results of the iterative task [8]. The output layer acquires the input from the hidden layers and uses its neurons to complete the computations, after which the output is generated.

Activation Function

In neural networks, activation functions are mathematical functions used to represent each neuron present in the network. The activation function of a neuron decides whether it should be turned on or turned off depending on the input or set of input values [9]. Activation functions facilitate normalizing the output of all the neurons and map them into a range within 1 and 0 or -1 and 1.

The activation functions can be classified mainly into the following types:

1. Identity or linear activation function
2. Nonlinear activation functions.

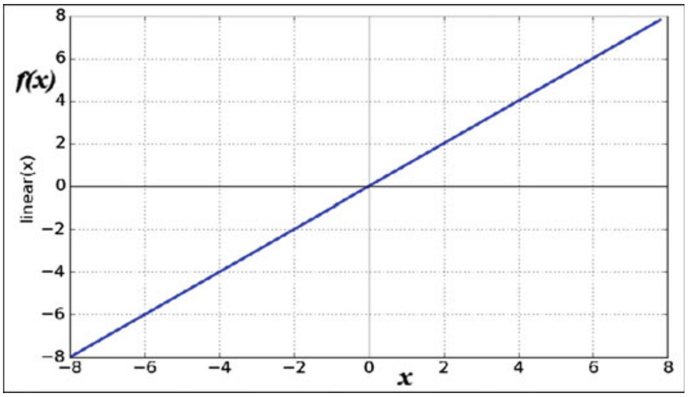

Fig. 3 Linear function

Identity or Linear Activation Function

The input values multiplied by their corresponding weights from each neuron are given as input to an activation function. The activation function produces an output corresponding to the input values [10]. A neural network is nothing but a linear regression model without a linear activation function. Linear activation function has confined power and also has limited capacity to solve the composite input data. The equation of a linear function is alike to that of a straight line, i.e., $f(x) = a * x$ and it ranges from $-\infty$ to ∞. The graphical representation is shown in Fig. 3.

When a linear activation function is employed in the multi-layered neural network, irrespective of how many layers are present in the network all the layers will be linear. Therefore, the final layer is purely a linear transformation of the final layer [11]. The linear function $f(x)$ has an invariable derivative and also it does not rely on the input value x. Thus, the linear function is unable to perform backpropagation every time to train the model. The gradient is pretty much the same hence it is not possible to improve the error.

Nonlinear Activation Function

Nonlinear activation functions can be used to represent any imaginable process as a computational function in the neural network [12]. They enable the model between the network inputs and outputs, to develop a composite mapping. These are essential to learn and model the nonlinear type of composite data such as images, video, audio, and data sets. Thus, to resolve the problems of a linear type of activation function, often nonlinear types of activation functions are used [13]. They have an input-related derivative function that permits backpropagation. Nonlinear functions often allow a deep neural network to be built up by stacking multiple hidden layers of neurons.

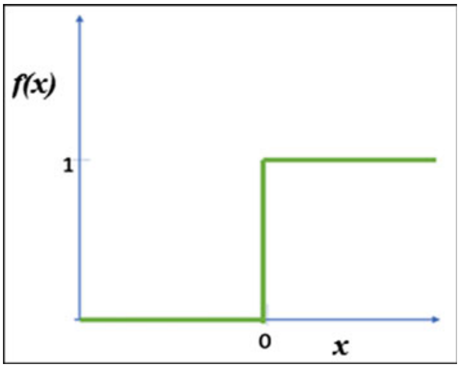

Fig. 4 Binary step activation function

Thus, it is possible to understand the composite data sets with high levels of precision using deep neural networks.

Binary Step Activation Function

The step function is among the most basic activation function available, which provides binary output [9]. That is why it is also called a binary step function. Here, we consider a threshold value, when the input passes the threshold limit the function produces a value 1(true) and then the neuron is activated. If the input does not pass the threshold value, the function produces a value 0(false) and then the neuron is deactivated. That is why they are very useful for binary classification studies. A graphical representation of the binary step activation function is shown in Fig. 4.

Mathematically, binary step activation function can be described as

$$f(x) = \left\{ \begin{matrix} 0, \text{ for } x < 0 \\ 1, \text{ for } x \geq 0 \end{matrix} \right\} \tag{2}$$

Sigmoid or Logistic Activation Function

The sigmoid function resembles an "*S*" shaped curve. It can be used to represent the anticipated values to probabilities. The sigmoid function distributes the input values of any size to output values in the interval between 0 and 1, normalizing the output of each neuron [14]. Graphical representation sigmoid activation function is shown in the below fig. Here, the output is not zero-centered as shown in Fig. 5.

Mathematically, sigmoid activation function can be described as

$$f(x) = \frac{1}{(1 + \mathrm{e}^{-x})} \tag{3}$$

The above function is exclusively monotonic in its entire region and it is easily differentiable. However, its derivative is not monotonic. There is almost no ambiguity in the estimation of very low or very high variation in the values of x, which creates

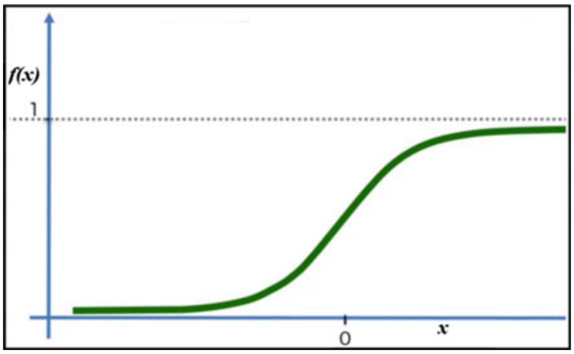

Fig. 5 Logistic/sigmoid function

a problem of vanishing gradient. That leads to a situation where the network refuses to learn further or being too sluggish to succeed in an accurate prediction. During the testing period, the logistic sigmoid function may end up causing the neural network to become stuck.

Tan*h* Function

Tan*h* activation function works almost always better than the sigmoid function. It is a mathematically modified version of the sigmoid function that transforms input to output with values ranging from -1 to 1 [15]. The gradient is stronger for tan*h* than the sigmoid activation function. The major advantage of the tan*h* activation function is that its negative inputs are always represented as strongly negative; zero inputs are represented near to zero which is not the same for sigmoid function as the range for it is between 0 and 1 as shown in Fig. 6.

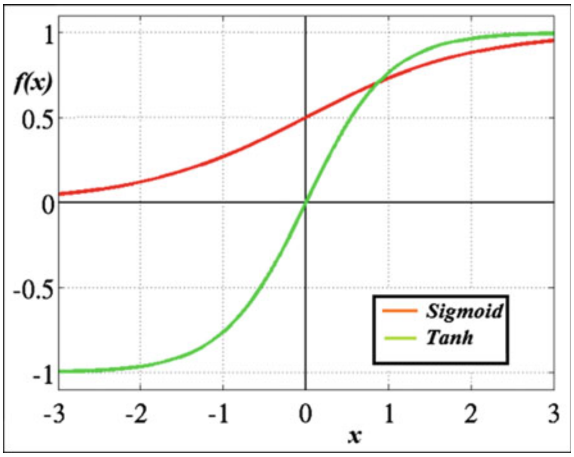

Fig. 6 Tan*h* function

The mean of the activations coming out of the hidden layer is closer to having a zero mean. Therefore, the data are more centered, making learning easier, and faster for the next layer. The function and its derivative are both monotonous.

Mathematically, tan*h* activation function can be described as

$$\tan h(x) = \frac{1 - \mathrm{e}^{-2x}}{1 + \mathrm{e}^{-2x}} \tag{4}$$

The main disadvantage of the tan*h* activation function is that its gradient will experience a very small value and even it may accomplish a near-zero value. This can thwart the gradient descent. The most difficult aspect of implementing these functions is that it demands the exponential term, which results in nonlinear behavior.

Arctan Function

The arctan function is similar to the sigmoid and tan*h* function and is obtained by the inverse of the tangent function. This activation function maps input to accelerating and decelerating output values ranging between ($-\pi/2$, $\pi/2$). The arctan function graph is a slightly flattered *S*-shape compared to the tan*h* function, which provides better classification power [11]. Arctan function can be mathematically described as

$$f(x) = \tan^{-1}(x) \tag{5}$$

Figure 7 shows the graphical representation of arctan function.

For the larger values of the input, its derivative converges to zero. Contrarily, the derivative of the sigmoid activation function converges exponentially to zero.

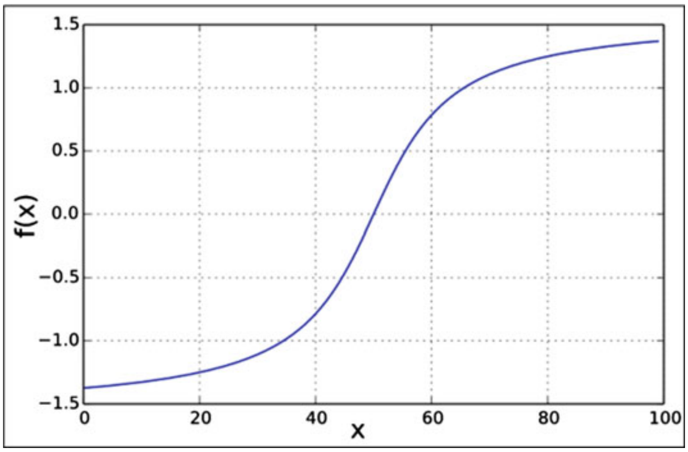

Fig. 7 Arctan function

Softmax Function

The Softmax function is another type of mathematical function that always produces an output ranging from 0 to 1 irrespective of whether the input values are positive, negative, zero, or greater than one. The cumulative of all the probabilities is equals to 1.Thus, the Softmax function is used to compute the normalized output probability distribution comprised of K probabilities from the input vector consisting of K real numbers. The formula for the Softmax functions can be specified as follows [16]:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{6}$$

The above expression calculates the input exponential value as well as the sum of all input exponential values. The Softmax function's output is proportional to the exponential input value as well as the sum of the exponential values.

The graph shown in Fig. 8 shows the variation of output probabilities corresponding to the variation in the input values. To overcome the issues of multi-class classification, the Softmax function can be used as the activation function in the output layer of neural network models.

Softsign Function

Softsign mathematical function is again a different type of activation function employed in neural networks. It almost resembles the hyperbolic tangent activation function but the main difference between them is that, unlike the tan*h* function which converges exponentially the softsign function converges in a polynomial form [17]. The value of softsign function is zero-centered and it ranges between -1 and $+1$, so the network learns effectively.

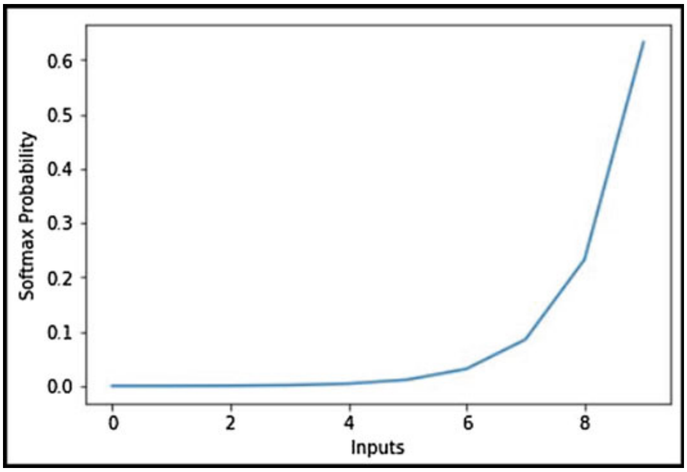

Fig. 8 Softmax function

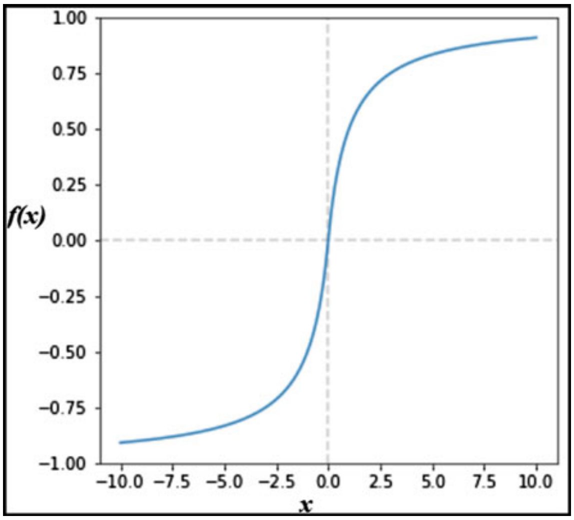

Fig. 9 Softsign function

Figure 9 shows the graphical interpretation of the softsign function. The formula for softsign functions can be specified as follows:

$$f(x) = \frac{1}{(1 + |x|)} \tag{7}$$

Softsign activation function is characterized by a high degree of nonlinearization and good error tolerance. Thus, it can be used in neural networks to transform the input into nonlinear distribution. The main advantage of the softsign function is that its output is centered on zero and its asymptote lines are smoother [11]. Thus, the output saturation reaches steadily to 0 on both sides. This alleviates the problem of gradient vanishing to some degree.

Rectified Linear Unit (ReLU) Function

In DL models, the rectified linear unit (ReLU) is one of the most famous and oftenly utilized activation functions. This function conserves the characteristics of a linear function. It also prevents the vanishing gradient problem seen in earlier forms of activation functions by rectifying the values of the inputs less than zero to zero otherwise; it will direct the input to output [18]. ReLU maps output ranging between 0 and 1 and it can be represented as

$$f(x) = \begin{Bmatrix} x, \text{ if } x \geq 0 \\ 0, \text{ if } x < 0 \end{Bmatrix} = \max(x, 0) \tag{8}$$

The below graph shown in Fig. 10 represents the equation of the ReLU activation function.

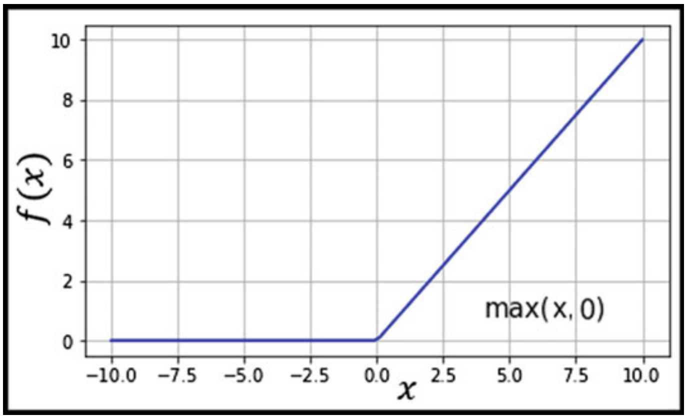

Fig. 10 ReLU function

ReLU incorporates faster AF learning that outperforms other AFs including the sigmoid and tan*h* functions in terms of efficiency and generalization [11]. It will be easier to train the varieties of neural network models using ReLU as a default activation function.

Exponential Linear Units (ELUs) Function

ELU also known as exponential linear unit is another form of activation function which is similar to the ReLU with certain variations. The ELU activation function can be specified mathematically as follows:

$$f(x) = \begin{Bmatrix} \alpha(e^x - 1), & x \le 0 \\ x, & x > 0 \end{Bmatrix} \tag{9}$$

The equation strictly outputs x-value for positive values of input x, which is the same as ReLU. In the case of negative input, the output will be α times $(e^x - 1)$. Where α is a hyperparameter that controls the value of negative inputs for which ELU saturates. This is an excellent way of handling the negative inputs [18]. The graphical contrast between the ReLU and ELU activation functions is shown in Fig. 11.

ELUs have negative values that try to bring the mean of the activations closer to 0. This enables quicker learning when the gradient is closer to the natural gradient. It does not experience the issue of dying neurons because the gradient of ELU is non-zero for all negative values. ELU is a steady and differentiable activation function at all points.

Swish Function

The swish activation function is an innovative activation function that can be effectively used in deep learning models across a variety of complicated data sets. Mathematically, the swish function can be defined as follows:

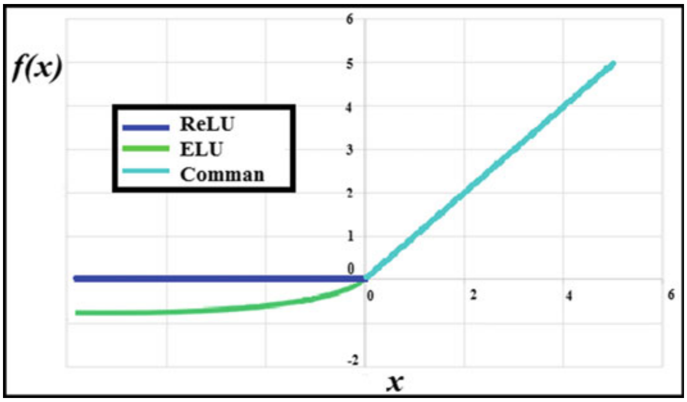

Fig. 11 Comparison between ReLU and ELU activation functions

$$f(x) = \frac{x}{(1 + e^{-x})} \tag{10}$$

The function is just the multiplication of the input x with the sigmoid function and its graphical representation is shown in Fig. 12.

Swish is unrestricted in the upper portion of the graph, therefore, the output would not be saturated to the maximum for large values of input [19]. However, it is restricted in the lower portion of the graph; therefore, for negative inputs, it does not return a zero as is the case for ReLU. Swish is smooth, non-monotonic, and continuous at all points; this differentiates it from most of the common activation functions.

Flatten-T Swish (FTS) Function

FTS, or flatten-T Swish (FTS), was introduced by Chieng as a novel activation function. Flatten-T swish incorporates activation features of both swish and rectified

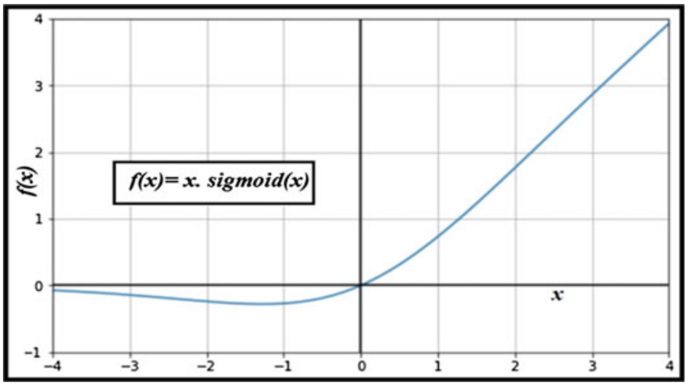

Fig. 12 Swish function

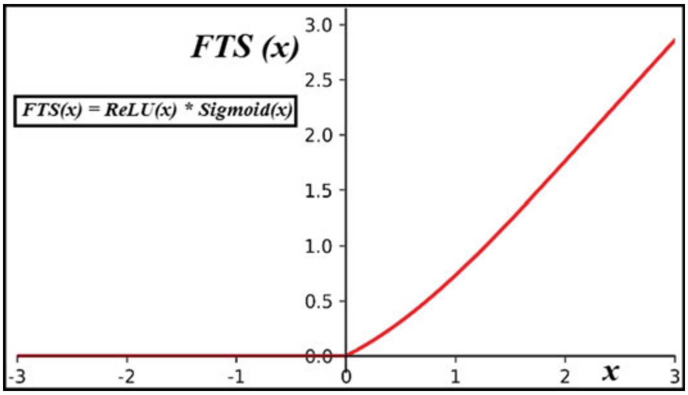

Fig. 13 Flatten-T swish function

linear units (ReLUs) activations functions together into an innovative one [20]. It is used to cope with the negative cancelation property in ReLU. Mathematically, FTS is formulated as follows:

$$\mathrm{FTS}(x) = \begin{Bmatrix} \frac{x}{1+\mathrm{e}^{-x}}, & x \geq 0 \\ 0, & x < 0 \end{Bmatrix} \tag{11}$$

When the value of $x \geq 0$, the FTS function has properties identical to that of the swish activation function. If the value of x is less than zero, then the function acts as ReLU. The graphical representation of the FTS function is shown in Fig. 13.

FTS has network limitations like dynamicity, pliability, and nonlinear representation capacity. Table 1 gives a summary of activation functions and their corresponding equations, derivatives, and applications.

Conclusion

Deep learning approaches employ neural networks consisting of several hidden layers to perform complex tasks. In the design of neural networks, the activation function plays a vital role. The hidden layer's activation function dictates how well the network model learns the training data set. The kind of predictions the model will offer is determined by the activation function employed in the output layer. Vanishing gradient is an unstable behavior that inhibits the training of deep neural networks with saturated activation functions. As the network's layers become deeper, the training efficiency and precision encounter numerous challenges which stimulate the development of different kinds of activation functions. Thus, activation functions are an important component of networks and selecting proper activation functions and analyzing their impact on the network will assist in optimizing the efficiency of the DL model.

Table 1 Activation functions and their corresponding equations

Function	Computation equation	Derivative	Application
Binary step activation function	$f(x) = \left\{\begin{array}{l} 0;\ \text{for}\, x < 0 \\ 1;\ \text{for}\, x \geq 0 \end{array}\right\}$	$f(x)' = \left\{\begin{array}{l} 0;\ \text{for}\, x \neq 0 \\ ?;\ \text{for}\, x = 0 \end{array}\right\}$	Perceptron linear classifier
Sigmoid activation function	$f(x) = \frac{1}{(1+\mathrm{e}^{-x})}$	$f(x)' = f(x)(1 - f(x))$	Logistic regression classification
Tan*h* function	$\tan h(x) = \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}}$	$f(x)' = 1 - f(x)^2$	Classification between two classes
Arctan function	$f(x) = \tan^{-1}(x)$	$f(x)' = \frac{1}{1+x^2}$	Learn complex patterns in the data
Softmax function	$\sigma(z_i) = \frac{\mathrm{e}^{z_i}}{\sum_{j=1}^{K} \mathrm{e}^{z_j}}$	$\sigma(z_j)' = \sigma(z_j)(1 - \sigma(z_j))$	Normalize the output of a network
Softsign function	$f(x) = \frac{1}{(1+\|x\|)}$	$f' = \frac{1}{(1+\|x\|)^2}$	Predict the multinomial probability distribution in the output layer
Rectified linear unit function	$f(x) = \left\{\begin{array}{l} x;\ \text{if}\, x \geq 0 \\ 0;\ \text{if}\, x < 0 \end{array}\right\} = \max(x, 0)$	$f(x)' = \left\{\begin{array}{l} 1;\ \text{if}\, x \geq 0 \\ 0;\ \text{if}\, x < 0 \end{array}\right\}$	Prevent the exponential growth in the computation
Exponential linear units function	$f(x) = \left\{\begin{array}{ll} \alpha(\mathrm{e}^x - 1), & x \leq 0 \\ x, & x > 0 \end{array}\right\}$	$f'(x) = \left\{\begin{array}{ll} f(x) + \alpha; & \text{if}\, x < 0 \\ 1 & \text{if}\, x \geq 0 \end{array}\right\}$	Introduces nonlinearity into the output of a neuron
Swish function	$f(x) = \frac{x}{(1+\mathrm{e}^{-x})}$	$f'(x) = f(x) + \frac{1}{1+\mathrm{e}^{-x}}(1 - f(x))$	Achieves higher test accuracy in very deep networks

(continued)

Table 1 (continued)

Function	Computation equation	Derivative	Application
Flatten-T swish function	$f(x) = \left\{\begin{matrix} \frac{x}{1+e^{-x}}, & x \geq 0 \\ 0, & x < 0 \end{matrix}\right\}$	$f'(x) = \left\{\begin{matrix} f(x) + \frac{1}{1+e^{-x}}(1 - f(x)); & x \geq 0 \\ 0; & x < 0 \end{matrix}\right\}$	Improved classification accuracy and converges twice as fast as ReLU

References

1. Min E, Guo X, Liu Q, Zhang G, Cui J, Long J (2018) A survey of clustering with deep learning: from the perspective of network architecture. IEEE Access 6:39501–39514. Author, F.: Article title. J 2(5):99–110 (2016)
2. Yan B, Han G (2018) Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system. IEEE Access 6:41238–41248
3. Hatcher WG, Yu W (2018) A survey of deep learning: platforms, applications and emerging research trends. IEEE Access 6:24411–24432
4. Gogoi P, Roy S, Bujarbaruah SM (2019) Modeling a H-H neuron based spiking neural network incorporating multiple pre-synaptic inputs. In: 2nd international conference on innovations in electronics, signal processing and communication, Shillong, India
5. Paugam-Moisy H, Bohte S (2010) Computing with spiking neuron networks. In: Rozenberg G, Back T, Kok JN (eds) Handbook of natural computing, 1st edn. vol. 1. SpringerVerlag, Heidelberg, Germany, pp 1–47
6. Shrestha A, Mahmood A (2019) Review of deep learning algorithms and architectures. IEEE Access 7:53040–53065
7. Xin Y et al (2018) Machine learning and deep learning methods for cybersecurity. IEEE Access 6:35365–35381
8. Zhao Z, Zheng P, Xu S, Wu X (2019) Object detection with deep learning: a review. IEEE Trans Neural Netw Learn Syst 30(11):3212–3232
9. Yu Y, Adu K, Tashi N, Anokye P, Wang X, Ayidzoe MA (2020) RMAF: Relu-memristor-like activation function for deep learning. IEEE Access 8:72727–72741
10. You W, Shen C, Wang D, Chen L, Jiang X, Zhu Z (2020) An intelligent deep feature learning method with improved activation functions for machine fault diagnosis. IEEE Access 8:1975–1985
11. Nwankpa C, Ijomah W, Gachagan A, Marshall S (2018) Activation functions: comparison of trends in practice and research for deep learning. arXiv:1811.03378
12. Wang Y, Li Y, Song Y, Rong X (2020) The influence of the activation function in a convolution neural network model of facial expression recognition. Appl Sci
13. Dlugosz Z, Dlugosz R (2018) Nonlinear activation functions for artificial neural networks realized in hardware. In: 25th international conference mixed design of integrated circuits and system (MIXDES), Gdynia, Poland, pp 381–384
14. Tan TG, Teo J, Anthony P (2011) A comparison of two sigmoidal-type activation functions in video game controller evolution. In: IEEE conference on sustainable utilization and development in engineering and technology
15. Pasca B, Langhammer M (2018) Activation function architectures for FPGAs. In: 28th international conference on field programmable logic and applications, Dublin, Ireland, pp 43–437
16. Wang M, Lu S, Zhu D, Lin J, Wang Z (2018) A high- speed and low-complexity architecture for softmax function in deep learning. In: IEEE Asia Pacific conference on circuits and systems (APCCAS), Chengdu, China, pp 223–226

17. Le P, Zuidema W (2015) Compositional Distributional semantics with long short term memory. arXiv1503.02510 [cs]
18. Qiumei Z, Dan T, Fenghua W (2019) Improved convolutional neural network based on fast exponentially linear unit activation function. IEEE Access 7:151359–151367. https://doi.org/10.1109/Access.2019.2948112
19. Tripathi GC, Rawat M, Rawat K (2019) Swish activation based deep neural network predistorter for RF-PA"TENCON 2019. In: 2019 IEEE region 10 conference, pp 1239–1242. https://doi.org/10.1109/TENCON.2019.8929500
20. Chieng HH, Wahid N, Pauline O, Perla SRK (2018) Flatten-T Swish: a thresholded RElU-swish-like activation function for deep learning. Int J Adv Intell Inform 4(2):76–86