Comparison study of Commonly Used Activation Functions for Deep Neural Networks

Dr.Hend Eissa
Electronic Technology College
"Electronic systems and programming center"
namarek2010@gmail.com

Dr. Naser Telesi
Electronic Technology College- Tripoli
NaserTelesi@Gmail.com

Dr. Sabri Mansour
College of applied administrative and financial sciences - Tripoli
Sabrikm_2012@yahoo.com

Abstract
Activation functions are the main decision-making components of neural networks. In adjunct, they assess the output of the neural node, making them critical to the performance of the overall network. thus it is essential to use the best activation function in the computation of neural networks. state that many recipes have been developed over time, although some are not very good. These days they are considered obsolete because sometimes they don't work properly. These functions have a number of qualities that are considered essential for successful learning. Their characteristics include monotonicity, individual derivatives and finite domain. This study will evaluate commonly used additive functions including swish, ReLU, sigmoid, and others. forward Their properties, their advantages and disadvantages and specific recommendations for applying the formula follow.
Key wards: deep learning, neural networks, activation function, classification, regression

Introduction
Deep learning neural networks have numerous uses, including object categorization, speech or pattern recognition, and voice analysis. This has been proven by its outstanding results in numerous domains. Additionally, it contains a structure with concealed layers, which suggests that there are multiple layers [1], [2]. However, according to research on deep learning structure, because of its prominent properties, it may be applied to natural scenarios in the actual world. The earliest deep learning model used for classification tasks only has a few layers; for instance, the LeNet5 model only has five levels.

Additionally, the depth has grown as a result of the demand for more complex network models, applications, and an increase in processing power [3]. Examples include twelve layers in Alex Net [4], nineteen or sixteen levels in VGG Net[5], depending on versions], twenty layers in Google Net[6], and one hundred and fifty-two layers in the largest Res Net architecture[7]. Last but not least, it has been demonstrated that stochastic depth networks have more than 1,200 layers. Therefore, delving deeply into neural networks will offer a greater knowledge of the hidden layers, enhancing their training and performance. The activation unit computes a neural cell output in the neural network. Later, the backpropagation algorithm makes use of the activation function derivative.

A differentiable hence So, the analysis must choose a differentiable activation function. This will make it possible for the function to be submitted to backpropagation weight

updates without zigzagging as in the case of the sigmoid function. [8], additional suggestion is that it ought to be simple to compute an activation function spare completing power, a crucial characteristic in massive neural networks with millions of nodes. It follows that the activation function is a crucial tool for mapping response variables and inputs for non-linear complex and sophisticated functions in artificial neural networks.

As a result, it demonstrates that non-linear systems are being introduced in a variety of domains [9]. However, the activation function's main duty is to transform an A-NN node's input signal into signal output. When a neural network contains several hidden layers, training it becomes tough and difficult. Some of these difficulties include zigzagging weight, vanishing gradient problem, overly convoluted formula, or saturation problem in the neural network of the activation function. This results in a long-term, continual learning process [10], [11]. The comparison of several activation functions made in this study paper, both practically and theoretically, is covered by Byrd et al. [12]. Soft plus, Tanh, and other activation features are among these. linear, sigmoid, Max out, swish, and Leaky Both ReLU and leaky ReLU A definition, a synopsis, and the advantages and disadvantages of each function are included in the study. This will make it possible for us to create rules for selecting the ideal activation function in each circumstance.

This work is distinctive because it discusses actual activation function applications. It's includes an overview of the most recent usage trends for these functions in comparison to cutting-edge research results from real-world deep learning deployments. The complexity presented in this study will allow proper decisions to be made regarding the selection of the best activation function and its implementation in any specific real-world application. As previously mentioned, it might be difficult to maintain sizable test data sets for various activation functions. Thus, Banerjee et al. [13] mention that some real-world uses of these services include maintaining training data, executing various experiments on various computers, and tracking experiment process. Following the investigation of various activation functions with specific real-world applications a summary is as shown below.

Activation Functions

In a neural network, activation functions are used to compute the weighted total of inputs and biases, which is then used to determine whether or not a neuron can be activated. It modifies the provided data and generates an output for the neural network that uses the data's parameters. In some literature, the activation functions are also referred to as transfer functions. These regulate the output of neural networks across several domains and can be either linear or nonlinear depending on the function they represent.

Before generating the final prediction for each label in a linear model, the hidden layers execute a linear mapping of an input function to an output. The formula for the input vector x transformation is

$$f(x) = w\,T.\,x + b,$$

where **x** is the input, **w** is the weight, and **b** is the bias.

The mappings of the aforementioned equation generate linear outcomes, and it is at this point that the activation function is required, first to transform these linear outputs into non-linear output for additional calculation, and secondly to identify patterns in the data. These models' outputs are given by:

$$y = (w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$

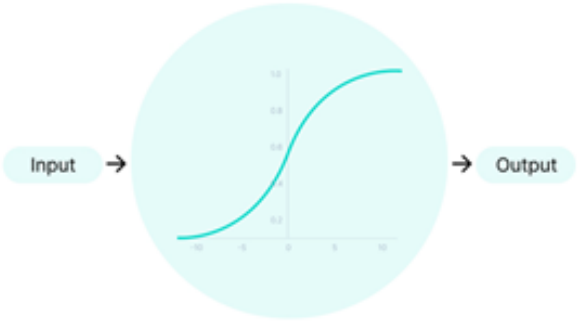

Fig1:Activation Function

Multilayered networks use these outputs from each layer, which are linear by default, to feed into the next layer until the ultimate output is reached. The kind of activation function that must be implemented in a specific network is said to depend on the desired output.

To change these linear inputs into non-linear outputs, however, requires the nonlinear activation functions because the outputs are linear in nature. These transfer functions were used to change the linear model outputs into their altered non-linear counterparts, which are now ready for processing. After applying the activation function, the non-linear output is provided by Fig2

$$y = (w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b).$$

where $\boldsymbol{\alpha}$ is the activation function.

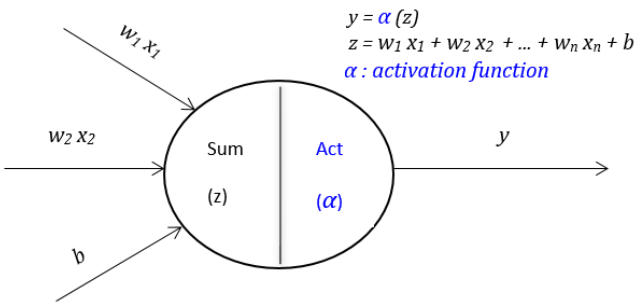

Fig2: Changing the linear model outputs into their altered non-linear counterparts

These activation functions are necessary because they facilitate the learning of high order polynomials for deeper networks by turning the linear input signals and models into non-linear output signals. Each neuron in a neural network performs two calculations:

• **Linear summing of inputs**: There are two inputs, x_1 and x_2, in the picture above, together with weights w1 and w2 and bias b. The linear addition

$$z = (w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b)$$

• **Activation computation:** By calculating the weighted total and then including bias with it, this calculation determines whether or not a neuron should be activated. The activation function's objective is to add non-linearity to a neuron's output. The weighted total of the inputs is often computed at the outset of a neural network. The layer's nodes can each have a different weighting. All nodes in the layer have the same activation function, though. While the weights are thought of as the learning parameters, they are typical of a fixed form. To enhance the output of neural network computing, the activation function must be chosen properly. For optimization reasons, all activation functions must be monotonic, differentiable, and immediately convergent with respect to the weights. The various forms of activation functions consist of:

1. **Linear Activation Functions**

The activation of a linear function, sometimes referred to as a straight-line function, is proportional to the input, or the weighted total of the input from the neurons. It has a straightforward function with the formula

$$f(x) = ax + c.$$

This activation's drawback is that it cannot be limited to a particular range. The activation function behaves like linear regression when this function is applied to every node. The neural network's last layer will operate as a linear function of the first layer. Another problem is that when differentiation is performed via gradient descent, the output is constant, which is undesirable because backpropagation's output and logic might be ruined by the constant rate of error change.

2. **Non-Linear Activation Functions**

The most popular activation functions are known to be non-linear ones. It makes it simple for a neural network model to discern between the results and adapt to a range of data. Based mostly on their range or curvature, these functions are categorized as follows:

a) **Functions of Sigmoid Activation**

The sigmoid function generates a value between 0 and 1 after accepting a real value as input. The input ranged in $(-\infty, +\infty)$ is converted to the range in via the sigmoid activation function (0,1) as shown below.

Fig3: Sigmoid/Logistic Activation Function

Sigmoid/Logistic Activation Function mathematically it can be represented as:

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

Here's why sigmoid/logistic activation function is one of the most widely used functions:

- It is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.
- The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

The limitations of sigmoid function are discussed below:

- The derivative of the function is f'(x) = sigmoid(x)*(1-sigmoid(x)).

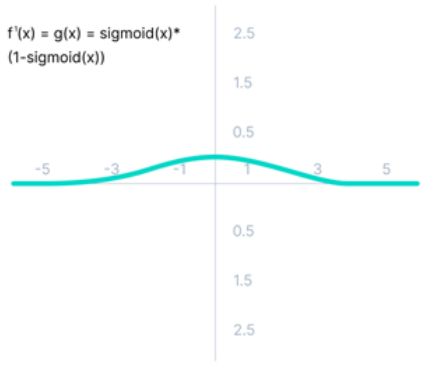

Fig4: The derivative of the Sigmoid Activation Function

As we can see from the above Fig4, the gradient values are only significant for range (-3 to 3), and the graph gets much flatter in other regions.

It implies that for values greater than (3 or less than -3), the function will have very small gradients. As the gradient value approaches zero, the network ceases to learn and suffers from the ***Vanishing gradient*** problem.

- The output of the logistic function is not symmetric around zero. So the output of all the neurons will be of the same sign. This makes the training of the neural network more difficult and unstable.

b) Tanh Function (Hyperbolic Tangent)

Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of (-1 to 1). In Tanh, the larger the input (more positive), the closer the output value will be to (1.0), whereas the smaller the input (more negative), the closer the output will be to (-1.0).

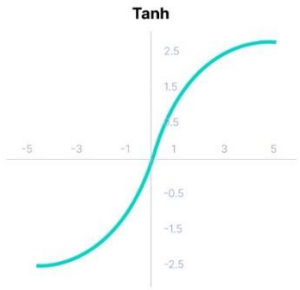

Fig5: Than Activation Function

Another potential function that can be utilized as a non-linear activation function between layers of a neural network is the Tanh function. There are some similarities between it and the sigmoid activation function. The Tanh function will map values between (-1 and 1), unlike a sigmoid function, which will map input values between (0 and 1). One of the intriguing characteristics of the Tanh function, like the sigmoid function, is that the derivative of Tanh may be stated in terms of the function itself.

Tanh Function (Hyperbolic Tangent), mathematically it can be represented as:

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

Advantages of using this activation function are:

- The output of the Tanh activation function is Zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers of a neural network as its values lie between -1 to; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.

Have a look at the gradient of the Tanh activation function to understand its limitations. See Fig6.

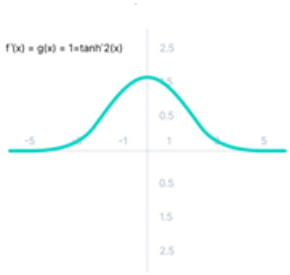

Fig6: Gradient of the Tanh (derivative) Activation Function

As you can see it also faces the problem of vanishing gradients similar to the sigmoid activation function. Plus, the gradient of the Tanh function is much steeper as compared to the sigmoid function. Although both sigmoid and Tanh face vanishing gradient issue, Tanh is zero centered, and the gradients are not restricted to move in a certain direction. Therefore, in practice, Tanh nonlinearity is always preferred to sigmoid nonlinearity.

c) Functions for *Rectified Linear Unit* (ReLU) Activation

ReLU stands for ***Rectified Linear Unit***. Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient. The main catch here is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0.

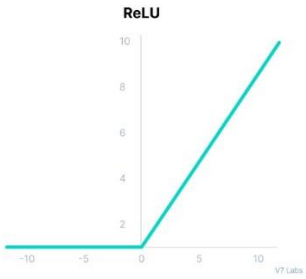

Fig7: ReLU Activation Function

Mathematically it can be represented as:

$$f(x) = \max(0, x)$$

Deceptively straightforward is the formula: max (0,z). Rectified Linear Units, despite its name, are not linear; they perform better than Sigmoid and offer the same advantages.

d) Maxout Activation Function

The ReLU and leaky ReLU functions are generalized by the Maxout activation. The dropout regularization method is intended to be used in conjunction with this piecewise linear function, which yields the maximum of inputs. ReLU and leaky ReLU are both unique examples of Maxout. Therefore, the Maxout neuron has all the advantages of a

ReLU unit without the drawbacks of a dying ReLU. A greater total number of parameters must be taught because it doubles the total number of parameters for each neuron.

e) ELU Activation Function

A function that has a propensity to converge more quickly and deliver more precise results is the exponential linear unit, or ELU. Which has an additional alpha constant that should be a positive number, unlike other activation functions. Except for negative inputs, ELU and ReLU are extremely similar. For non-negative inputs, they are both in the identity function form. As opposed to ReLU, which smoothest sharply, ELU becomes smooth gradually until its output equals.

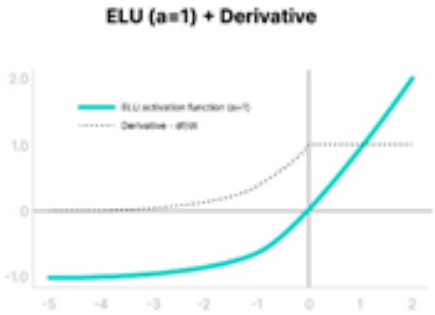

Fig8: ELU Activation Function

Mathematically it can be represented as:

$$f(x) = \begin{cases} 1 & for\ x \geq 0 \\ f(x) + \propto & for\ x < 0 \end{cases}$$

f) Softmax Activation Function

The probability distribution of an event across 'n' distinct events is calculated by the Softmax function. This function will, in general, determine the probabilities of each target class across all potential target classes. The target class for the supplied inputs will later be determined with the aid of the computed probability. Before exploring the ins and outs of the Softmax activation function, we should focus on its building block—the sigmoid/logistic activation function that works on calculating probability values.

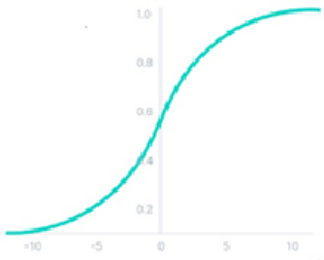

Fig9: Probability

The output of the sigmoid function was in the range of (0 to 1), which can be thought of as probability. But— This function faces certain problems. Let's suppose we have five output values of 0.8, 0.9, 0.7, 0.8, and 0.6, respectively. How can we move forward with it?

The answer is: We can't. The above values don't make sense as the sum of all the classes/output probabilities should be equal to 1.

You see, the Softmax function is described as a combination of multiple sigmoid. It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class. It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification.

Mathematically it can be represented as:

$$softmax(z) = \frac{\exp(z)}{\sum \exp(z)}$$

Choosing activation function in a neural network

It specifically depends on the kind of problem and the desired output's range of values. For instance, ReLU can be used to forecast values larger than 1 when tanh or sigmoid are inappropriate to utilize in the output layer. ReLU is not a good option, however, if the output values must fall between (0, 1) or (-1, 1); in this case, sigmoid or tanh can be used instead. The softmax activation function should be utilized in the last layer when performing a classification job and using the neural network to forecast a probability distribution over the mutually exclusive class labels. Use ReLU as an activation for the hidden layers as a general rule, though, when it comes to the surface layers.

The Sigmoid activation function should be applied in the case of a binary classifier. For the hidden layer, both the sigmoid and tanh activation functions perform dreadfully. ReLU or its superior variant leaky ReLU should be utilized for hidden layers. Softmax is the most effective activation function for multiclass classifiers. Even though there are more known activation functions, these are the ones that are reportedly used the most.

Right Activation Function

You need to match your activation function for your output layer based on the type of prediction problem that you are solving—specifically, the type of predicted variable.

Here's what you should keep in mind. As a rule of thumb, you can begin with using the ReLU activation function and then move over to other activation functions if ReLU doesn't provide optimum results. And here are a few other guidelines to help you out.

1. ReLU activation function should only be used in the hidden layers.
2. Sigmoid/Logistic and Tanh functions should not be used in hidden layers as they make the model more susceptible to problems during training (due to vanishing gradients).

3. Swish function is used in neural networks having a depth greater than 40 layers.

Finally, a few rules for choosing the activation function for your output layer based on the type of prediction problem that you are solving:

1. **Regression -** Linear Activation Function
2. **Binary Classification**—Sigmoid/Logistic Activation Function
3. **Multiclass Classification**—Softmax
4. **Multilabel Classification**—Sigmoid

The activation function used in hidden layers is typically chosen based on the type of neural network architecture.

5. **Convolutional Neural Network (CNN)**: ReLU activation function.
6. **Recurrent Neural Network**: Tanh and/or Sigmoid activation function.

Function	Comment	When to use?
Sigmoid	Prone to the vanishing gradient function and zigzagging during training due to not being zero centered	Can't into gates simulation
Tanh	Also prone to vanishing gradient network	In recurrent neural
RelU	The most popular function for hidden layers. Although, under rare circumstances, prone to the "dying ReLU" problem	First to go choice
ElU	ELU and ReLU are similar for non-negative inputs as opposed to ReLU which smoothest sharply ELU becomes smooth gradually until its output equals	Use only if You expect "dying ReLU" problem
Maxout	Far more advanced activation Function than ReLU, immune to "dying", but much more	Use as last resort expensive in case of computation
SoftMax		For output layer in classification

Conclusions

This study demonstrates that queries like "which activation function should I choose?" and "How to choose the right Activation Function?" do not have a single, definitive answer. However, based on the presented theory as indicated in the table above, and following this thorough overview of the activation functions employed in deep learning, we can make a few but firm recommendations. As a result, this study provides a thorough summary of the activation functions used in deep learning (DL) [Tab. 6], highlighting the most important and hitherto unreported application trends. The examination of various AFs and a discussion of specific applications domains in which these functions might be applied were

presented first, followed by a quick introduction to the activation function and deep learning. discusses the architectures and technologies used in the creation of deep neural networks. Additionally, because the activation functions can enhance the learning of specific data patterns, they can alter the process for the better or worse.

Finally, the functions evolve over time, but further research is still required before using any of them in deep learning for any given project. This work just serves as advice in this regard.

References

[1] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing* , vol. 3, p. e2, 2014.
[2] J. A. Hertz, *Introduction to the theory of neural computation* . CRC Press, 2018.
[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation* , vol. 1, no. 4, pp. 541–551, Dec. 1989.
[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Computer Vision and Pattern Recognition (CVPR)* , vol. 7, Dec. 2015.
[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proceedings of the 25th international conference on neural information processing systems - volume 1* , 2012, pp. 1097–1105.
[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR* , vol. abs/1409.1556, 2014.
[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer vision and pattern recognition (cvpr)* , 2015, pp. 1–17.
[8] K. J. Piczak, "Recognizing bird species in audio recordings using deep convolutional neural networks." in *CLEF (working notes)* , 2016, pp. 534–543.
[9] C. Y. M. Z. Alom T. M. Taha and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiV* , Dec. 2018.
[10] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth." in *ECCV (4)* , 2016, vol. 9908, pp. 646–661.
[11] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.* , vol. 22, no. 3, pp. 400–407, Sep. 1951.
[12] M. A. Nielsen, *Neural networks and deep learning* . Determination Press, 2015.
[13] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-newton method for large-scale optimization," *S IAM Journal on Optimization* , vol. 26, no. 2, pp. 1008–1031, 2016.
[14] A. Banerjee, A. Dubey, A. Menon, S. Nanda, and G. C. Nandi, "Speaker recognition using deep belief networks," *arXiv preprint arXiv:1805.08865* , 2018.
[15] José Naranjo , Marco Mora, Ruber Hernández. A Review of Convolutional Neural Network Applied to Fruit Image Processing. Published: 16 May 2020.
[16] Zewen Li; Fan Liu; Wenjie Yang; Shouheng Peng; Jun Zhou. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. **IEEE**. 10 June 2021.