Network and Complex Systems
ISSN 2224-610X (Paper) ISSN 2225-0603 (Online)
Vol.13, 2022

Evaluation of Sigmoid and ReLU Activation Functions Using Asymptotic Method

Sheriff Alimi[1*] Ogunyolu Olufunmilola Adnni[2*] Solanke Oluwole Babafemi[3*] Foladoyin Adegbie[4] Olutayo Ajayi[5]

1. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun, Nigeria.
2. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun, Nigeria.
3. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun, Nigeria.
4. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun, Nigeria.
5. Department of Computer Science, Babcock University, Ilisan-Remo, Ogun, Nigeria.

*E-mail of the corresponding author: alimi0356@pg.babcock.edu.ng

Abstract

The performance of two algorithms may be compared using an asymptotic technique in algorithm analysis, which focuses largely on the growth rate as the number of inputs grows. Sigmoid activation and ReLU activation functions are widely employed in ANNs (Yingying, 2020), and each has advantages and disadvantages that should be considered when designing ANN solutions for a given issue. This study aimed to compare the performance of sigmoid activation and ReLU activation function during training using an asymptotic approach. The work focuses on training time complexity as the basis of comparison of the two activation functions using an asymptotic approach. The result derived from this study showed that sigmoid activation function takes more computation time in performing forward path, loss computation and backward propagation than ReLU activation functions. The computation cost will become significant when dealing with deep neural networks with hundreds to thousands of neurons. Overall, the training time for ReLU based Neural network will be better than that of sigmoid based one. Sigmoid have higher computational cost compared to ReLU but the two algorithms have a linear growth rate.

Keyword: Back propagation, Loss computation, Sigmoid Activation, ReLU activation, ANNs

DOI: 10.7176/NCS/13-05
Publication date: June 30th 2022

1. Introduction

An artificial neural network (ANN) is a complex non-linear mathematical model which mimics the functioning of the brain in solving a complex problem. It has been applied in computer vision, speech processing, understanding and more (Gao et al, 2019). The human brain is an interconnection of biological neurons that accepts input signals through the dendrites and the nucleus transforms these inputs to the postsynaptic potential that is passed to the synapse through the axon. Similarly, the artificial neural network is an interconnection of several artificial neurons and each neuron takes inputs and transforms them to output using a mathematical function. The mathematical function used in the neural network is called the activation function (Yingying, 2020). There are different types of activation functions and the sigmoid or logistic function, function, tanh or hyperbolic tangent function, rectified linear Unit, and leaky rectified linear unit (leaky ReLU).

In building an ANN model, there are two phases, the first phase is the feed-forward path which is primarily concerned with the divergence between the prediction and actual value; this divergence is called loss and it is computed using a cost function. The second phase is the backpropagation (Song, 2016) where the ANN learns by adjusting the hyper-parameters to reduce the divergence. The whole training process is an iterative one in which a sequence of steps is used in the feed-forward path and another sequence of steps is used for the back-propagation path. The training process is an execution of an algorithm.

a. Problem Statement

In algorithm analysis, the performance of two algorithms can be compared using an asymptotic approach which focuses primarily on the growth rate as the number of input increases which can help neutralize the impact of the types system specifications in terms of CPU specification, memory size, operating the system, the programming language and the compiler type. Sigmoid activation and ReLU activation functions are used extensively in ANNs (Yingying, 2020), and each of them has its pros and cons which guide in the selection of activation functions used in the design of ANN solutions for a specific problem. It will also be of great value if the comparison of the two activation functions is performed using an asymptotic method and the outcome of that will be another fact that will help in the choice of activation function selection.

b. Aim and Objectives

The primary objective of this work is to compare the performance of sigmoid activation and ReLU activation function during training using an asymptotic approach

The objectives for achieving the aims are:

1) Implement a sigmoid-based neuron and ReLU-based neuron.
2) Train each of the neurons by varying the size of the input data.
3) Compare the performance of the neurons using the asymptotic method.

c. Scope of Work

The work focuses on training time complexity as the basis of comparison of the two activation functions using an asymptotic approach; accuracy of the functions is out of scope in this study.

2. Literature Review

Since deep learning was introduced, many researchers have continued to innovate., using several approaches to convolutional neural networks. According to (Yingying, 2020), the purpose of the activation function in a convolution activation layer is that the layer must process the result of the convolution operation severally and sum the convolution values in forward propagation, which would establish a non-linear relationship between input and output. This in overall improves the expressiveness of the convolutional layer.

(Akpinar et al., 2019), and (Bartlett et al., 2019) focused more on sample complexities. (Shen, 2019) and (Horel, 2020) focused on asymptotic properties and statistical tests of one-layer sigmoid networks. However, sigmoid networks are not often practicalised in comparison with ReLU networks; this is as a result of sigmoid having limited expression, also non- vanishing gradient expression to help speed up training procedure. Therefore, asymptotic properties of ReLU are clearer to investigate using standard approach such as consistency, probabilistic rate of convergence, and asymptotic normality.

Activation function can be either linear or non-linear depending on the function it represents which are used to control the outputs of neural networks across different domains, from machine translation to segmentation and other domains, this substantiates that a proper choice of activation function improves results in neural network computing (Chigozie, 2018).

The goal of neural networks is to modify input data into outputs by approximating time-varying unknown functions as well as objective functions which exhibit non-linearity. Individual layers of Neural networks can be assumed to be an estimation of an unknown objective function. For complex objective functions, the output of the first layer neural network conflicts the objective function. (Bartlett, 2019) explored several ways to navigate this issue, by increasing the number of nodes (neurons) or layers, this addition of new layers to the neural network improves the estimation value of the target function.

(Farrell et al, 2019) obtained a revolutionary result about asymptotic properties of deep neural networks. They demonstrate probabilistic convergence rates for multi- layers in the ReLU network, with the presumption that several hidden layers, i.e., the depth of the network, increases with sample size.

Researchers usually employ a pre-determined number of layers, as the increased depth makes training harder and increases the tendency to overtrain as shown by (Eldan et al., 2016). Contrary to (Farrell et al., 2019), asymptotic features of a neural network are derived by a fixed number of layers which doesn't get maximized as sample size increase. The neural network can be thought of as an estimation procedure where the basic functions are estimated from the data. The learning is done through an optimization problem on a flexible combination of simple functions.

A. Review of theoretical Concept of Sigmoid, ReLU activation function and Asymptotic Analysis

Like biological neurons as seen in figure 1 that has inputs represented by the dendrites, soma(nucleus) for aggregation of all inputs and transforming them as a postsynaptic voltage at the axion terminals, so does the artificial neuron. Of interest in this work are two of the activation functions used in artificial neurons for transforming the aggregated inputs to output and they are the sigmoid function and ReLU function.

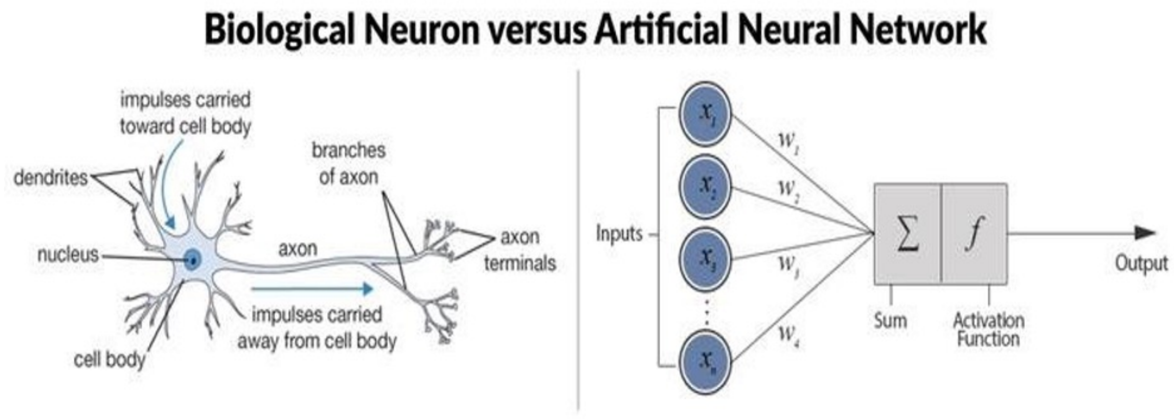


Figure 1 Biological and Artificial Neuron (Mwandau & Nyanchama, 2019)

Sigmoid Activation Function

A sigmoid activation function is a nonlinear function that bound the output to be between 0 and 1, it can be seen as a normalizing function that prevents jump in the output. Equations (1) and (2) are the mathematical representation of the function.

$$z = wx + b \qquad equation\ (1)$$

$$\hat{y} = \frac{1}{1 + e^{-z}} \qquad equation\ (2)$$

$\hat{y}$ is the predicted output in the feed-forward, where x is the input data, W is the weight and b is the activation bias.

The cost function represented by equation (3) is used to compute how the function models the supplied input data against the actual output and this is done in the feed-forward propagation phase. In the back-propagation phase, the weight (W) and bias (b) are adjusted using gradient-descent to improve the prediction and lower the variance between the hypothetical and actual output. Equation 4 is the path of backwards-propagation where W and b are adjusted as with a specified learning rate (α)

$$J(w,b) = y\ lg\ \hat{y} + (1 - y)\ lg(1 - \hat{y})\ equation(\ 3)$$

$$w_{t+1} = w_t - \alpha \frac{\partial J\langle w,b\rangle}{\partial w} \qquad equation\ 4(a)$$

$$b_{t+1} = b_t - \alpha \frac{\partial\langle w,b\rangle}{\partial b} \qquad equation\ 4(b)$$

Sigmoid activation function has some disadvantages, the major one is the vanishing gradient, a situation where there is no significant change in prediction when X is very small or very large. This activation function has been mentioned to be computationally expensive and this claim will be validated in the course of this study with respect to the ReLU activation function.

ReLU activation function

In a neural network, the transformation of the total weighted input arising from a node is done by activation function into the activation of a node or the output for the specified input. Rectified linear unit popularly called ReLU will output the input when its positive else output is zero. It is presently the most used activation function in the world mostly in Convolutional neural networks or deep learning. ReLU activation function is an easy function that comprises no heavy computation and so it takes minimal time to train. The ReLU activation function solely aims to capture interaction and non-linearities, It is a popular feed-forward neural network (Sütfeld, Brieger, Finger, Füllhase, & Pipa, 2020). The suggested neural network is for just a layer and a cost/loss function linked to a neuron, several weights, bias (b), and a ReLU activation. The cost function usually uses mean squared error and Gradient descent to minimize loss. It is important to train the neural network to identify the most appropriate weights and bias to enhance the efficiency of the network through the training procedure where Gradient Descents computes the slope of loss/Cost function moves the weights and biases based on the slope so as to minimize loss. (Wang, 2018)

$$\hat{y} = \max(0, z) \qquad \textit{equation } (5)$$

Recall equation (1) on how to compute z and x is the input to a neuron, The ReLU activation function returns 0 if it accepts any negative input but returns the same value if it is positive as depicted in equation (5)

Equation (6) is the cost function which is the mean square error (MSE), equation (7) is more elaborate

$$J(w,b) = (y - \hat{y})^2 \qquad \textit{equation} \quad (6)$$
$$J(w,b) = \frac{1}{N} \sum_{n=1}^{N} (y - \hat{y})^2 \ \textit{equation} \, (7)$$

where y= Vector of actual label and N is the input data size.

A vector input is an item which is multiplied by individual weight and summed together where bias is also included. ReLU then receives the value to give an activation output called W. It then converts any negative value to 0 then returns x if x is positive and 0 if negative, the summation of all gives the activation.

Researchers have observed that ReLU's performance is better than the sigmoidal activation function in Classification problems (Schmidt-Hieber, 2020), Other researchers have shown that the ReLU activation function is added in the Convolutional Neural network which enhances the efficiency of classification and increases training speed (Alhassan & Zainon, 2021). It is important to state that ReLU is not linear in nature but it is a true estimator and can merge with another function if already combined by ReLU (Kizrak, 2019).

In a neural network (Zhang, Weng, Chen, Hsieh, & Daniel, 2018) mentioned that an activation function like sigmoid based on efficient learning rate and weight decay features for training neural network models can provide a good accuracy as much as activation function like ReLu and an introduced framework like CROWN identifies the strength of neural network using ReLu activation function which enhances lower bounds as well as in sigmoid activation function.

Asymptotic Notation

Asymptotic notation helps to analyze the runtime of an algorithm with respect to its behavior as the input size increases. It measures the growth rate of the algorithm, it is a time complexity measure that determines the order of growth which could be constant, linear, logarithmic, polynomial (quadratic, cubic) or exponential.

In practice, the program that implements the algorithm of interest is executed several times and with various input data sizes. The execution time T(N) and input data size N are recorded for each instance of execution. Plotting T(time) against N (size) provides a visual understanding of the growth rate. The graph of log2 of running time T(N) against log2 input size (N) provides an easier way to estimate the order of growth mathematically. Equation 5 below is the linear representation of the log-log scale with b being the gradient of the slope.

$$\log_2(T(N)) = b \log_2 N + c \quad \textit{equation}\,(8)$$

From the equation above, we can deduce the running time T(N) as:

$$\mathrm{T(N)} = \mathrm{aN^b} \qquad \textit{equation}(9) \ \text{ where } a = 2^c \textit{ and } b \textit{ is the order of growth}$$

The doubling hypothesis makes the estimation of growth order b a lot easier. The size of input data is double for each instance of the execution of an algorithm and the log2 of the ratio of the consecutive running time is b.

The order of growth or growth rate is the system independent effects, as it is solely dependent on the algorithm and the input data. The coefficient a is the system-dependent effects that cover the hardware specifications (CPU, memory etc.) and the software factors such as the operating system, compiler and programming language.

3. Methodology

The section detailed the steps taken to achieve comparison of sigmoid and ReLU activation functions in terms of time complexity in asymptotic sense.

For the evaluation, the activation functions are used in implementing single neuron networks that has ten inputs, X= $\{X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8, X_9,\}$ and an out Y.
The methodology has seven stages which are highlighted below

1. Dataset generation
2. Training the two neurons with different problem size
3. Determination of growth order

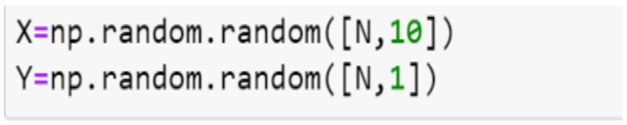
Dataset generation

To determine the time complexity which is recording how the training or running time changes with increasing input data size, a random dataset is generated for each training iteration such that the current data size is twice that of the previous.

Considering that X € $\boldsymbol{R}^{10}$,the sizes of the dataset in order of training are 20k,40k,80k,160k,320k,640k,1280k and 2560k.The dimension used for the

[20,000 x 10], [40,000 x 10], [80,000 x 10], [160,000 x 10], [320,000 x 10], [640,000 x 10], [1,280,000 x 10], and [2,560,000 x 10]. The sequence can be represented by $\boldsymbol{Dataset\ size} = [N * 10]$ where N = $\left\{\begin{matrix} 20000,40000,80000,160000,320000,640000, \\ 1280000, 2560000 \end{matrix}\right\}$

Python numpy library is used for the data generation for both training input and output as depicted in figure KK below



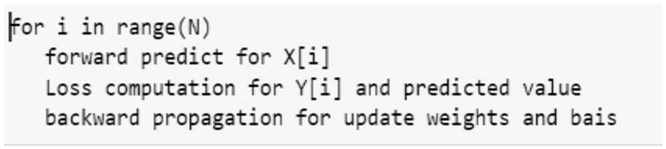
```
X=np.random.random([N,10])
Y=np.random.random([N,1])
```

Training the neurons

A training instance comprises of three phases

1. The forward phase which is the prediction
2. The loss computation
3. Backward propagation

For N input data size, the training algorithm is depicted below



```
for i in range(N)
   forward predict for X[i]
   Loss computation for Y[i] and predicted value
   backward propagation for update weights and bais
```

Regarding the sigmoid based single neuron network, the forward phase is represented by equations (1) and (2), the loss function is as well represented by equation (3). The backward propagation is the learning part where the weights and bias are update is represented by equation (4). Importantly, the $\frac{\partial J\langle w,b\rangle}{\partial w}$ and $\frac{\partial J\langle w,b\rangle}{\partial b}$ are not computed directly but rather through chain rule.

$$\frac{J\langle w,b\rangle}{\partial w} = \frac{\partial J\langle w,b\rangle}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

$$\frac{\partial J\langle w,b\rangle}{\partial y} = \frac{y}{\hat{}y} + \frac{1-y}{1-\hat{}y}, \frac{\partial y}{\partial z} = y(1-y), \frac{\partial z}{\partial w} = X$$

Likewise:

$$\frac{\partial J\langle w,b\rangle}{\partial b} - \frac{\partial J\langle w,b\rangle}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial b} \quad and \quad \frac{\partial z}{\partial b} = 1$$

Growth Rate

The log to base 2 of the running is plotted against the log to base of the input size, with a graphical method the necessary model parameters are obtained. Recall equation (5), the needed parameters are b (the growth rate) and c or a, the system factor which is the growth rate multiplier. Alternative to graphical method is the use of the linear regression formula to compute b which is the gradient and c that interception point.

4. Result and Findings

The two single neuron networks were trained with increasing the dataset size and the time taken for each instance of training were recorded. Table 1 below contains the time duration for training the neuron with different input sizes and it shows clearly that sigmoid takes more time than ReLU for all the input sizes used. Table 2 is the log2 of running time against log2 of input size, the table will be useful in computing the growth order.

Table 1 Training or running time for various input size

SN	N (Input size)	Sigmoid Time(sec)	ReLU Time(sec)
1	20,000	0.799751759	0.52575326
2	40,000	1.417845726	1.00128746
3	80,000	2.805955648	1.902569056
4	160,000	5.746193171	4.060643911
5	320,000	11.25614452	7.922258854
6	640,000	23.01362801	16.62149668
7	1,280,000	46.96020961	34.1930387
8	2,560,000	95.96643829	67.72208405

Table 2 log2(T(N)) versus log2(N)

SN	Log2(N) X	Sigmoid Log2(Time in sec) Y	ReLU Log2(Time in sec)) Y
1	14.2877	-0.322375835	-0.927542205
2	15.2877	0.503700563	0.001856218
3	16.2877	1.488492206	0.927948819
4	17.2877	2.522606493	2.021708519
5	18.2877	3.492640852	2.985911841
6	19.2877	4.524416531	4.05497839
7	20.2877	5.553366943	5.095630734
8	21.2877	6.584458045	6.081554466

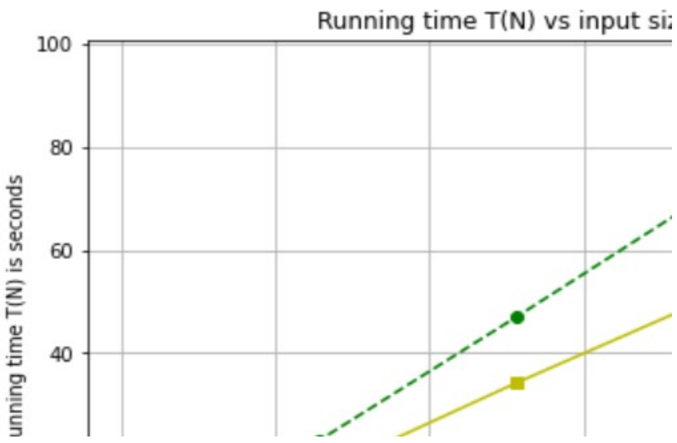
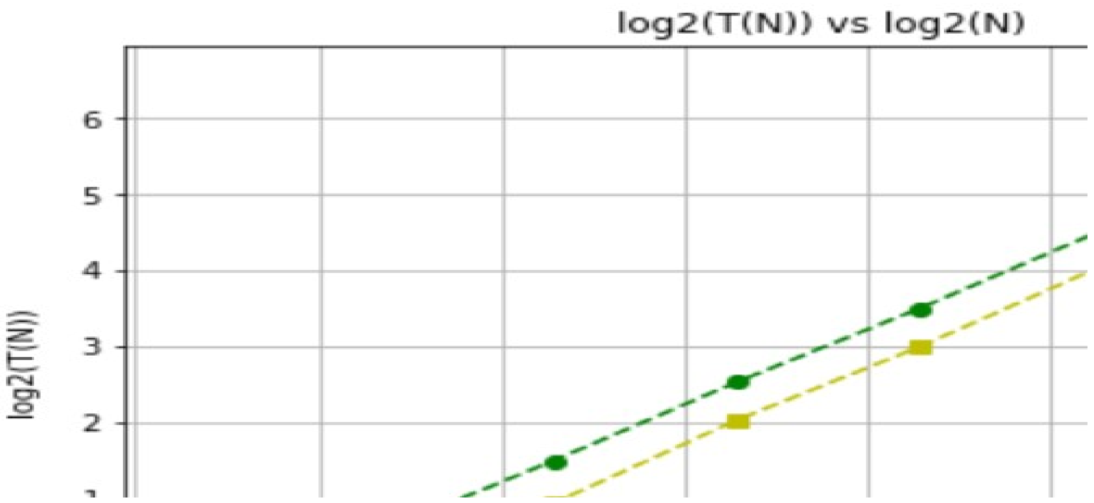


Figure 3 The graph of running time vs input size

Figure 3 above clearly shows that the running time T(N) for both sigmoid and ReLU are linear based on visual inspection**.**

Figure 4 log2(T(N)) versus log2(N)



With graphical method, the values of both b and c can be obtained, however, linear regression formulas are used. For the sigmoid, b= 0.996118767 and c= -14.67524856

$a = 2^c = 2^{-14.67524856}$ = 0.000038222

Therefore, the time complexity of single sigmoid neuron is expressed as:

$T_{sigmoid}(N) = 0.000038222\ N^{0.996118767}\ equation(7)$

Similarly, for ReLU, b = 1.052359123 and c= -16.12592958

$a = 2^c = 2^{-16.12592958}$ =0.000013983

Therefore, the time complexity of single ReLU neuron is expressed as:

$T_{ReLU}(N) = 0.000013983\ N^{1.052359123}\ equation(8)$

The gradient b for both sigmoid and ReLU can be approximated to be 1, equation (7) and equation (8) becomes:

$T_{sigmoid}(N) = 0.000038222 N equation(9)$
$T_{ReLU}(N) = 0.000013983\ N\quad equation(10)$

With the above equations (9 and 10) the order of growth for both Sigmoid and ReLU is linear.

5. Conclusion and Recommendation

The results show that sigmoid activation function takes more computation time in performing forward path, loss computation and backward propagation than ReLU activation functions. The computation cost will become significant when dealing with deep neural networks with several hundreds to thousands of neurons. Overall, the training time for ReLU based Neural network will be better than that of sigmoid based one. It is important to note that though sigmoid have higher computational cost compared to ReLU, the two algorithms have a linear growth rate.

It is recommended that asymptotic analysis of all the neural network activation function be carried-out to determine their respective growth rate as such information will be empirical facts for decision making in terms of training performance and where application scenario has constrained computational resources.

References

Akpinar. N, Kratzwald, and S. Feuerriegel(2019): "Sample Complexity Bounds for Recurrent Neural Networks with Application to Combinatorial Graph Problems," arXiv:1901.10289.

Alhassan, A. M., & Zainon, W. M. N. W. (2021). Brain tumor classification in magnetic resonance image using hard swish-based RELU activation function-convolutional neural network. *Neural Computing and Applications*, *33*(15), 9075–9087. https://doi.org/10.1007/s00521-020-05671-3

Bartlett.P, Harvey.N, Liaw.C, and Mehrabian. A (2019): "Nearly-tight VC- dimension and PseudodimensionBounds for Piecewise Linear Neural Networks.," *Journal of Machine Learning Research,* 20(63), 1–17.

Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. (2018)." *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*"

Eldan. R, and Shamir, O(2016): "The power of depth for feedforward neural networks," in Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, eds., 29th Annual Conference on Learning Theory, volume 49, *Proceedings of Machine Learning Research*, pp. 907–940.

Farrell, M, Liang. T, and Misra. S (2019): "Deep neural networks for estimation and inference: Application to causal effects and other semiparametric estimands," arXiv preprintarXiv:1809.09953.

Gao J, Chakraborty D, Tembine H and Olaleye O (2019) Nonparallel Emotional Speech Conversion. INTERSPEECH, Graz, Austria.

He, K, Zhang. X, Ren, S, and Sun, J (2016): "Deep residual learning for image recognition," in Proceedings of IEEE conference on computer vision and pattern recognition, pp. 770–778.

Horel, E, and Giesecke, K(2020): "Towards explainable AI: Significance tests for neural networks," *Journal of Machine Learning Research, forthcoming*.

Kizrak, A. (2019). Comparison of Activation Functions for Deep Neural Networks. *Towards Data Science*, (3).

Liu, M., Shi.J, LI, Z, LI, C, Zhu. J, and Liu. S (2016): "Towards better analysis of deep convolutional neural networks," *IEEE transactions on visualization and computer graphics,* 23(1), 91–100.

Mwandau, B., & Nyanchama, M. (2019). Investigating Keystroke Dynamics as a Two-Factor Biometric Security INVESTIGATING KEYSTROKE DYNAMICS AS A TWO-FACTOR BIOMETRIC SECURITY Brian Njogholo Mwandau *Submitted in partial fulfillment of the requirements for the Degree of Master of Science in Info*. (May).

Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with ReLU activation function.*Annals of Statistics*, *48*(4), 1875–1897. https://doi.org/10.1214/19- AOS1875

Shen, x., c. jiang, l. sakhanenko, and q. lu (2019): "asymptotic properties of Neural Network Sieve Estimators," arXiv preprint arXiv:1906.00875.

Song H, Sun X, Xiang S. Progress on the application of artificial neural network in chemical Industry, Chemical Industry and Engineering Progress, 2016, 35(12): 3755-3762.

Sütfeld, L. R., Brieger, F., Finger, H., Füllhase, S., & Pipa, G. (2020). Adaptive Blending Units: Trainable Activation Functions for Deep Neural Networks. *Advances in Intelligent Systems and Computing*, *1230 AISC*. https://doi.org/10.1007/978-3-030-52243-8_4

Sun, S, Chen, W, Wang, L, Liu. X, and Liu .T (2016): "On the depth of deep neural networks: A theoretical view," in Thirtieth AAAI Conference on Artificial Intelligence.

Schmidt-Hieber, J. (2020). Nonparametric regression using deep neural networks with ReLU activation function. *Annals of Statistics*, *48*(4), 1875–1897. https://doi.org/10.1214/19-AOS1875

Wang, C (2018) Finding the Cost Function of Neural Networks https:medium.com/Oreina wang

Yingying Wang , Yibin Li , Yong Song ,and Xuewen Rong (2020): " The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition "

Zhang, H., Weng, T. W., Chen, P. Y., Hsieh, C. J., & Daniel, L. (2018). Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, *2018-December*.