Activation Functions: Comparison of Trends in Practice and Research for Deep Learning

Chigozie E. Nwankpa
Dept. of DMEM
University of Strathclyde
Glasgow, UK
chigozie.nwankpa@strath.ac.uk

Winifred l.Ijomah
Dept. of DMEM
University of Strathclyde
Glasgow, UK
w.l.ijomah@strath.ac.uk

Anthony Gachagan
Dept. of EEE
University of Strathclyde
Glasgow, UK
a.gachagan@strath.ac.uk

Stephen Marshall
Dept. of EEE
University of Strathclyde
Glasgow, UK
stephen.marshall@strath.ac.uk

***Abstract*—Deep neural networks (DNN) have been successfully used in diverse emerging domains to solve real-world complex problems with may more deep learning(DL) architectures, being developed to date. To achieve this state-of-the-art (SOTA) performances, the DL architectures use activation functions (AFs), to perform diverse computations between the hidden layers and the output layers of any given DL architecture. This paper presents a survey on the existing AFs used in deep learning applications and highlights the recent trends in the use of the AFs for DL applications. The novelty of this paper is that it compiles the majority of the AFs used in DL and outlines the current trends in the applications and usage of these functions in practical deep learning deployments against the SOTA research results. This compilation will aid in making effective decisions in the choice of the most suitable and appropriate AF for a given application, ready for deployment.**

This paper is timely because majority of the research papers on AF highlights similar works and results while this paper will be the first, to compile the trends in AF applications in practice against the research results from the literature, found in DL research to date.

***Index Terms*—activation function, activation function types, activation function choices, deep learning, neural networks, learning algorithms**

1. Introduction

Deep learning algorithms are multi-level representation learning techniques that allow simple non-linear modules to transform representations from the raw input into the higher levels of abstract representations, with many of these transformations producing learned complex functions. The DL research was inspired by the limitations of the conventional learning algorithms especially being limited to processing data in raw form [1], and the human learning techniques by changing the weights of the simulated neural connections based on experiences, obtained from past data [2].

The use of representation learning, which is the technique that allows machines to discover relationships from raw data, needed to perform certain tasks like classification and detection. Deep learning, a sub-field of machine learning (ML), is more recently being referred to as representation learning in some literature [3]. The typical artificial neural networks (ANN) are biologically inspired computer programmes, designed by the inspiration of the workings of the human brain. These ANNs are called networks because they are composed of different functions [4], which gathers knowledge by detecting the relationships and patterns in data using past experiences known as training examples in most literature. The learned patterns in data are modified by an appropriate AF and presented as the output of the neuron as shown in Figure 1 The early DL algorithms

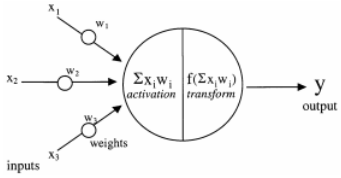

Figure 1. Typical biological inspired neuron [5]

used for recognition tasks had few layers in their entire architecture, with LeNet5, having just five layers [6]. These network layers have witnessed significant depth increases with researchers successfully training over one thousand two hundred layers in Stochastic Depth networks [7], and the layers still increasing. However, as these networks get deeper, the need to understand the makeup of the hidden layers and parameters within the layers becomes inevitable. Perhaps, a major issue of the deeper architectures is the difficulty of developing effective learning algorithms as the network gets deeper and studies on the issues associated with the training of the NNs have been a key research area.

To improve the performance of these learning algorithms, researchers highlighted that three key areas are usually of interest namely: to improve the model, to engineer better features, and to improve inference [8]. Furthermore, the use of better feature models have worked in many DL applications to obtain improved model features for the applications alongside the model architectures, however this paper focuses on inference improvement. Several techniques

for model improvement for DL algorithms exist in literature which includes the use of batch-normalisation and regularisation, [9], [10], [11], dropout [9], proper initialisation [12], good choice of AF to mention a few [10], [12], [13], [14]. However, these different techniques offer one form of improvement in the results or training improvement but our interest lies in the AFs used in deep learning algorithms, their applications, and the benefits of each function introduced in the literature.

Conversely, we summarize the trends in the application of existing AFs used in DL and highlight our findings as follows. This compilation is organized into six sections, with the first four sections introducing DL, AFs, the summary of AFs used in DL, and the application trends of AFs in deep architectures respectively. Section five discusses these functions and section six provides the conclusion and future work.

2. Activation Functions

AFs are functions used in NNs to compute the weighted sum of inputs and biases, which decides if a neuron can be fired or not. It manipulates the presented data through some gradient processing usually gradient descent to produce an output for the NN, that contains the parameters in the data. These AFs are often referred to as a transfer function in the literature, with early research results by [15], validating categorically that a proper choice of AF improves results in NN computing.

For a linear model, the linear mapping of an input function to output, is performed in the hidden layers before the final prediction of a class score for each label. It is computed by the affine transformation in most cases [4], where the input layer accepts the data for training the NN,in various formats including images, videos, texts, speech, sounds, or numeric data, and transforms it into vectors x, whose transformation is given by

$$f(x) = w^T x + b \tag{1}$$

where x = input, w = weights, b = biases. Furthermore, the NNs produce linear results from the mappings from equation 1, with the AFs converting these linear outputs into non-linear output for further propagation, to understand the patterns in the data. The outputs are obtained by

$$y = (w_1\, x_1 \; + \; w_2\, x_2 \; + .. + \; w_n\, x_n + \; b) \tag{2}$$

These outputs are fed into the next subsequent layer for multilayered networks like DNN until the final output is obtained. However, the expected output determines the type of AF to deploy in a given network. Conversely, since the outputs are linear, a nonlinear AF is required to convert them to non-linear outputs. The non-linear output after the application of the AF is given by

$$y = \alpha(w_1\, x_1 \; + \; w_2\, x_2 \; + .. + \; w_n\, x_n + \; b) \tag{3}$$

Where α is the AF. The AF aids the learning of high order polynomials beyond one degree for deeper networks, with the position-dependent on the function in the network structure. Besides, an understanding of the makeup of the multiple layers is our interest, with focus on the AF's.

3. Summary of the Major Activation Functions

This section highlights the different types of AFs and their evolution over the years. The AF research and applications in deep architectures, used in different applications has been a core research field. The SOTA research results are outlined as follows although, It is worthy to state categorically that this summary is not reported in chronological order but arranged with the main functions first, and their similar improvements as variants, alongside the combined derived AFs. These functions are detailed as follows:

3.1. Sigmoid Function

The Sigmoid AF is sometimes referred to as the logistic function or squashing function in some literature [8]. The Sigmoid function research results have produced several variants of the sigmoid AF, which are used in DL applications. The Sigmoid is a non-linear AF used mostly in feedforward NNs. It is a bounded differentiable real function, defined for real input values, with positive derivatives everywhere and some degree of smoothness [16]. The Sigmoid function is given by

$$f(x) = \quad \frac{1}{(1 + \; e^{-x})} \tag{4}$$

The sigmoid appears in the output layers of most DL architectures, and are used for predicting probability outputs. It has been applied successfully in binary classification problems, modeling logistic regression tasks, as well as other NN domains, with [17] highlighting it's main advantages as, being easy to understand and are used mostly in shallow networks. Moreover, [12] suggested that the Sigmoid AF should be avoided when initializing the NN from small random weights.

However, the Sigmoid AF suffers major drawbacks which include sharp damp gradients during backpropagation from deeper hidden layers to the input layers, gradient saturation, slow convergence, and non-zero centred output thereby causing the gradient updates to propagate in different directions. Other forms of AF including the hyperbolic tangent was proposed to remedy some of these drawbacks suffered by the Sigmoid.

3.1.1. Hard Sigmoid Function. The hard sigmoid activation is another variant of the sigmoid AF, given by

$$f(x) \; = \; max \; \left(0, min \left(1, \frac{(x+1)}{2}\right)\right) \tag{5}$$

A comparison of the hard sigmoid with the soft sigmoid shows that the hard sigmoid offers lesser computation cost when implemented both in a specialized hardware or software form as outlined [18], and the authors highlighted that it showed some promising results on DL based binary classification.

3.1.2. Sigmoid-Weighted Linear Units (SiLU). The Sigmoid-Weighted Linear Units is a reinforcement learning (RL) based approximation function. The SiLU function is computed as Sigmoid multiplied by its input [19]. The SiLU a_k is given by

$$a_k(s) = z_k \alpha(z_k) \tag{6}$$

where s = input vector, z_k= input to hidden units k. The input to the hidden layers is given by

$$z_k = \sum_i w_{ik} s_i + b_k \tag{7}$$

Where b_k is the bias and w_{ik} is the weight connecting to the hidden units k respectively. The SiLU function can only be used in the hidden layers of the DNNs and only for RL based systems. The authors also reported that the SiLU outperformed the ReLU function as outlined in Figure 2.

3.1.3. Derivative of Sigmoid-Weighted Linear Units (dSiLU). The derivative of the Sigmoid-Weighted Linear Units is the gradient of the SiLU function and referred to as dSiLU. The dSiLU is used for gradient-descent learning updates for the NN weight parameters, and the dSiLU is given by

$$a_k(s) = \alpha(z_k)(1 + z_k(1 - \alpha(z_k))) \tag{8}$$

The dSiLU response looks like an overshooting Sigmoid as shown in Figure 2. The authors highlighted that the dSiLU outperformed the standard Sigmoid function significantly [19].

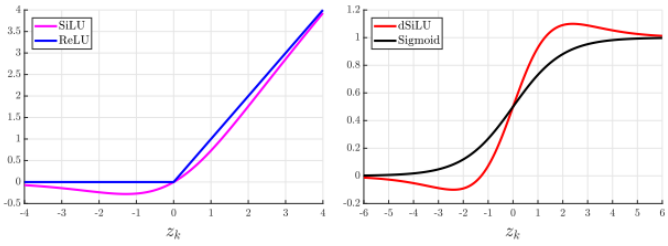

Figure 2. SiLU response comparison [19]

3.1.4. Logistic Sigmoid (lsigmoid). The lsigmoid proposes to address the problems of ReLU and LReLU when applied to DBN's, where these AF's are not able to use the pre-training effects of RBN's. The lsigmoid units resolve the inherent vanishing gradients during the back-propagation of Sigmoid by combining the benefits of unsaturation from LReLU [20]. The lsigmoid has a derivative and lsigmoid is given by

$$f_I(x) = \begin{cases} \alpha(x-a) + \text{Sigmoid}(a) & x >= a \\ \text{Sigmoid}(x) & -a < x < a \\ \alpha(x+a) + \text{Sigmoid}(a) & x <= -a \end{cases} \tag{15}$$

where a is the threshold and α is the slope, which is preset.

3.2. Softplus Function

The Softplus AF is a smooth version of the ReLU function which has smoothing and nonzero gradient properties, thereby enhancing the stabilization and performance of DNNs designed with Softplus units. It was proposed by [21] and it is a primitive of the sigmoid, given by

$$f(x) = \log(1 + e^x) \tag{9}$$

The Softplus was tested on statistical applications however, a comparison of the Softplus against the ReLU and Sigmoid functions by [22], showed an improved performance with lesser epochs to convergence during training. The Softplus was also applied to speech recognition systems [22], [23], of which the ReLU and sigmoid AF have been the dominant AFs, used to achieve speech recognition systems.

3.3. Hyperbolic Tangent Function (Tanh)

The Tanh is another AF used in DL with its main advantage being that it produces a smother zero centred output whose range lies between -1 to 1 thereby aiding the back-propagation. [1]. It is given by

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

The Tanh became the preferred function compared to the sigmoid in that it gives better training performance for multi-layer neural networks [15], [17]. However, Tanh suffers similar vanishing gradient problems like the sigmoid.

A property of the Tanh is that it can only attain a gradient of 1, only when the value of the input is 0, that is when x is zero. This makes the Tanh produce some dead neurons during computation. The dead neuron is a condition where the activation weight, rarely used as a result of zero gradients. This limitation of the Tanh function spurred further research in AFs to resolve the problem, and it birthed the ReLU AF. The Tanh functions have been used mostly in recurrent neural networks (RNN) for natural language processing [24] and speech recognition tasks [25].

3.3.1. Hard Hyperbolic Function. The Hardtanh function is another variant of the tanh AF used in DL applications. The Hardtanh represents a cheaper and more computational efficient version of tanh. The Hardtanh function lies within the range of -1 to 1 and it is given by

$$f(x) = \begin{pmatrix} -1 & if\ x < -1 \\ x & if\ 1 \leqslant x \leqslant 1 \\ 1 & if\ x > 1 \end{pmatrix} \tag{11}$$

The hardtanh function has been applied successfully in natural language processing [26], with the authors reporting that it provided both speed and accuracy improvements.

3.3.2. Linearly Scaled Hyperbolic Tangent (LiSHT). The LiSHT is proposed to address the problems of ReLU and Swish AF's where the negative gradients are cut off thereby causing dying gradient problems. It has unbounded upper limits, symmetric, smooth, and non-monotonic [27]. The non-parametric function LiSHT tries to scale non-linear Tanh function by a liner function thereby resolving the issue. The LiSHT is obtained by multiplying the $Tanh$ function by its input, given by

$$f(x) = Tanh(x).(x) \tag{12}$$

3.4. Softmax Function

The Softmax is another AF used in neural computing. It is used to compute probability distribution from a vector of real numbers. The Softmax produces an output which is a range of values between 0 and 1, with the sum of the probabilities been equal to 1. The Softmax function [4] is computed by

$$f(x_i) \ = \ \frac{e^{(x_i)}}{\sum\limits_{j} e^{(x_j)}} \tag{13}$$

The Softmax is used in multi-class models where it returns probabilities of each class, with the target class having the highest probability. It appears in almost all the output layers of the DL architectures, where they are used [28], [29], [30]. The main difference between the Sigmoid and Softmax AF is that the Sigmoid is used in binary classification while the Softmax is used for multivariate classification tasks.

3.5. Softsign

Softsign is another type of AF that is used in neural network computing. The Softsign function is another non-linear AF used in DL applications introduced by [8]. The Softsign function is a quadratic polynomial, given by

$$f(x) = \left(\frac{x}{|x| \ + \ 1}\right) \tag{14}$$

Where $|x|$ = absolute value of the input. The main difference between the Softsign and $Tanh$ function is that the Softsign converges in a polynomial, form unlike the $tanh$ which converges exponentially. It has been used mostly in regression computation problems [31] and DL based test to speech systems [32], with the authors reporting some promising results.

3.6. Rectified Linear Unit (ReLU) Function

The ReLU AF was proposed by [33] and ever since, is among the most widely used AFs for DL applications. The ReLU is a faster learning AF [1], which has proved to be the most successful and widely used function [13]. It offers better performance and generalization in DL compared to other AFs [34], [35]. The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimize, with gradient-descent methods [4]. The ReLU AF performs a threshold operation to each input element where values less than zero are set to zero thus the ReLU is given by

$$f(x) = \ max\,(0, x) = \ \begin{cases} x_i, & if\ x_i \ \geq \ 0 \\ 0, & if\ x_i < \ 0 \end{cases} \tag{15}$$

It rectifies the values of the inputs less than zero thereby forcing them to zero thereby eliminating observable vanishing gradient problem. The ReLU has been used within the hidden units of the DNNs with another AF, used in the output layers of the network with typical examples found in object classification [10], [36], and speech recognition [25].

The main advantage of using the ReLU is faster computation since it does not compute exponentials and divisions, with overall computation speed is enhanced [34]. Another property of the ReLU is that it introduces sparsity in the hidden units as it squishes the values between zero to maximum. However, it's limitation is the ease of overfitting compared to the sigmoid function although the dropout technique has been adopted to reduce this overfitting effect of ReLUs and [14] outlining that the ReLU improved performances of the DNNs. The ReLU and its variants have been used in different architectures of DL, which include the restricted Boltzmann machines [33] and the CNN architectures [28], [36], [37], [38], [39], although researchers [33] outlined that the ReLU has been used in numerous architectures because of its simplicity and reliability.

The limitation of ReLU is that it is sometimes fragile during training thereby causing some of the gradients to die. This causes the weight updates not to activate in future data points hindering learning as dead neurons give zero activation [4]. To resolve the dead neuron issues, the leaky ReLU was proposed.

3.6.1. Leaky ReLU (LReLU). The LReLU introduced some small negative slope to the ReLU to sustain and keep the weight updates alive during the entire propagation process [25]. The α parameter was introduced as a solution to the ReLUs dead neuron problems such that the gradients will not be zero at any time during training. The LReLU computes the gradient with a very small constant value for the negative gradient α in the range of 0.01 thus the LReLU AF is computed as

$$f(x) = \alpha x + x \ = \ \begin{cases} x & if\ x \ > \ 0 \\ \alpha x & if\ x \ \leq \ 0 \end{cases} \tag{16}$$

The LReLU has an identical result when compared to the standard ReLU with an exception that it has non-zero gradients over the entire duration thereby suggesting that there no significant result improvement except in sparsity and dispersion when compared to the standard ReLU and $tanh$ functions [25]. The LReLU was tested on the automatic speech recognition dataset.

3.6.2. Parametric Rectified Linear Units (PReLU). The PReLU is another variant of the ReLU AF proposed by [36]. It has the negative part of the function, being adaptively learned while the positive part is linear. PReLU is given by

$$f(x_i) = \begin{pmatrix} x_i \ , & if\ x_i\ >\ 0 \\ a_i x_i\ , & if\ x_i\ \leq 0 \end{pmatrix} \tag{17}$$

Where a_i is the negative slope controlling parameter and its learnable during training with back-propagation. If the term $a_i = 0$, the PReLU becomes ReLU. The authors reported that the PReLU performed better than ReLU in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and these results from the PReLU were the first to surpass human-level performance on visual recognition [36].

3.6.3. Randomized Leaky ReLU (RReLU). The randomized leaky ReLU is a dynamic variant of leaky ReLU where a random number sampled from a uniform distribution $U(l,u)$ is used to train the network. RReLU is given by

$$f(x_i) = \begin{pmatrix} x_{ji} \ , & if\ x_{ji}\ \geq\ 0 \\ a_{ji} x_{ji}\ , & if\ x_{ji}\ <\ 0 \end{pmatrix} \tag{18}$$

$Where\ a_i\ \sim\ U(l,u)\,,\ l\ <\ u\ and\ l,u\ \in [0,1]$. The test phase uses an averaging technique of all a_{ji} as in the training, without the dropout used in the learned parameter to a deterministic value, obtained as $a_{ji} = \frac{l+u}{2}$. Thus, the real test output is obtained using the following relationship.

$$y_{ji} = \frac{x_{ji}}{\frac{l\ +\ u}{2}} \tag{19}$$

The RReLU was tested on classification datasets and compared against the other variants of the ReLU AF and [40] validating that LReLU, RReLU, and PReLU perform better than the ReLU on classification problems.

3.6.4. S-Shaped ReLU (SReLU). The SReLU is another variant of the ReLU AF used to learn both convex and non-convex functions, inspired by the laws of neural sciences and psycho-physics. This SReLU AF was proposed by [41], and it consists of three-piece wise linear functions, formulated by four learnable parameters, which are learned during the training of the DNN using backpropagation. The SReLU is defined by the following mapping relationship

$$f(x) = \begin{pmatrix} t^r_i\ +\ a^r{}_i\,(x_i - t^r{}_i)\,,\ x_i\ \geq\ t^r{}_i \\ x_i\,, \qquad\qquad t^r_i > x_i\ >\ t^l_i \\ t^l{}_i\ +\ a^l{}_i\,(x_i - t^l{}_i)\,,\ x_i\ \leq\ t^l{}_i \end{pmatrix} \tag{20}$$

$where\ t^l{}_i,\ t^r{}_i,\ and\ a^l{}_i$ are learnable parameters of the network and it shows that the SReLU can vary in different channels. The parameter $a^r{}_i$ represents the slope of the right line with input above the set threshold $t^r{}_i$ and $t^l{}_i$ are thresholds in positive and negative directions respectively. The authors outlined that SReLU was tested on different CNN architectures, including the Network in Network and GoogLeNet, on CIFAR-10, ImageNet, and MNIST datasets, with improved results, compared to the other AFs [41].

3.6.5. Displaced Rectified Linear Units (DReLU). DReLU is another form of the ReLU AF seen referred to as the generalisation of SReLU, proposed by [42]. It conjectures the extending identity function of ReLU to the third quadrant thereby enhancing compatibility to batch normalisation. The DReLU is less computational than ELU, PReLU, and LReLU on learning speed and test accuracy on standard VGG and residual networks [42]. The DReLU is given by

$$f = \begin{cases} x & if x \geq -\delta \\ -\delta & if x\ <\ -\delta \end{cases} \tag{21}$$

3.6.6. ReLU Memristor-Like Activation Function (RMAF). The RMAF AF is another improvement of the ReLU that leverages the benefits of neural networks using two parameters, a constant α and a threshold parameters p to make the function smooth and monotonous while introducing non-linearity in the network [43]. The RMAF function is given by

$$RMAF = \left[j \left(\frac{1}{(0.25.(1+e^{-x}) + 0.75)^p} \right) \right] .\alpha x \tag{22}$$

where $p > 0$, and controls the flatness of region and the adjustment parameters j and p make the RMAF function like the ReLU and scale up any reference network. The operation range of the RMAF is important as the authors highlight that the RMAF has a similar property to Swish when $x \geq 0$.

The major highlight is the RMAF is tested on standard datasets and architectures of CNN's as well as considering both shallow multi-layer perceptrons and deep CNN's with the RMAF showing remarkable results compared to the standard ReLU, Swish, Tanh, Sigmoid AF's.

3.7. Exponential Linear Units (ELUs)

The ELU is another type of variant of ReLU AF proposed by [44]. They are used to speed up the training of DNN and have its main advantage that they can alleviate the vanishing gradient problem by using identity for positive values and also improves the learning characteristics. They also have negative values that allow for pushing of mean unit activation closer to zero thereby reducing computational complexity and improving learning speed. The ELU represents a good alternative to the ReLU as it decreases bias shifts by pushing mean activation towards zero during training. The ELU is given by

$$f(x) = \begin{pmatrix} x, & if\ x\ >\ 0 \\ \alpha\ exp(x)\ -\ 1, & if\ x \leq 0 \end{pmatrix} \tag{23}$$

The derivative or gradient of the ELU equation is given as

$$f' = \begin{pmatrix} 1, & if\ x\ >\ 0 \\ f(x)\ +\ \alpha, & if\ x\ \leq\ 0 \end{pmatrix} \tag{24}$$

Where α= ELU hyperparameter that controls the saturation point for negative net inputs which is usually set to 1.0.

The ELUs have a clear saturation plateau in its negative regime thereby learning more robust representations, and they offer faster learning and better generalisation compared to the ReLU and LReLU with specific network structure especially above five layers and guarantees SOTA results compared to ReLU variants. However, a critical limitation of the ELU is that it ELU does not centre the values at zero, and the parametric ELU was proposed to address this issue [45].

3.7.1. Parametric Exponential Linear Unit (PELU). The PELU is another parameterised version of the ELUs, which tries to address the zero centre issue found in the ELUs. The PELU was proposed by [45], and it uses the PELU in the context of vanishing gradient to provide some gradient-based optimization framework used to reduce bias shifts while maintaining the zero centre of values [45]. The PELU has two additional parameters compared to the ELU and the modified ELU is given by

$$f(x) = \begin{pmatrix} cx, & if\ x\ >\ 0 \\ \alpha\ e^{\frac{x}{b}}\ -\ 1, & if\ x \leq 0 \end{pmatrix} \tag{25}$$

Where a, b, and $c > 0$ and c causes a change in the slope in the positive quadrant, b controls the scale of the exponential decay, and α controls the saturation in the negative quadrant. The PELU is a good option for applications that require fewer bias shifts and vanishing gradients like CNNs.

3.7.2. Scaled Exponential Linear Units (SELU). The SELU is another variant of the ELUs, proposed by [46]. The SELU was introduced as a self-normalising NN that has a peculiar property of inducing self-normalising properties. It has a close to zero mean and unit variance that converges towards zero mean and unit variance when propagated through multiple layers during network training, thereby making it suitable for DL application, and with strong regularisation, learns robust features efficiently. The SELU is given by

$$f(x) = \tau \begin{pmatrix} x, & if\ x\ >\ 0 \\ \alpha\ e(x)\ -\ \alpha, & if\ x \leq 0 \end{pmatrix} \tag{26}$$

Where τ is the scale factor. The approximate values of the parameters of the SELU function are $\alpha \approx\ 1.6733\ and\ \lambda \approx 1.0507$. The SELUs are not affected by vanishing and exploding gradient problems and the authors have reported that they allow the construction of mappings with properties leading to self normalising NNs which cannot be derived using ReLU, scaled ReLU, sigmoid, LReLU, and even the $tanh$ functions. The SELU was tested on classification tasks [46] alongside some deep genetic mutation computation [47].

3.8. Maxout Function

The Maxout AF has a non-linearity applied as a dot product between the weights of a NN and data. The Maxout, proposed by [48] generalizes the leaky ReLU and ReLU where the neuron inherits the properties of ReLU and leaky ReLU where no dying neurons or saturation exist in the network computation. The Maxout function is given by

$$f(x) = max(w_1^T\ x + b_1\ \ ,\ w_2^T\ x\ + b_2\ ) \tag{27}$$

Where w = weights, b = biases, T = transpose. The Maxout was tested on phone recognition applications [49]. A drawback of the Maxout is that it is computationally expensive as it doubles the parameters used in all neurons thereby increasing the number of network parameters.

3.9. Swish Function

The Swish AF is one of the first compound AF proposed by the combination of the sigmoid AF and the input function, to achieve a hybrid AF. The Swish AF was proposed by [13], and it uses the reinforcement learning (RL) based automatic search technique to compute the function. The properties of the Swish include smoothness, non-monotonic, bounded below and unbounded in the upper limits. The smoothness property makes the Swish to produce better optimization and generalization results when used in training deep learning architectures [13]. The Swish is given by

$$f(x) = x\ \cdot\ sigmoid(x)\ =\ \frac{x}{1\ +\ e^{-x}} \tag{28}$$

The Swish advantage is the simplicity and improved accuracy as it does not suffer vanishing gradient but propagates reasonable information during training. The authors reported that the Swish outperformed ReLU on DL classification.

3.10. ELiSH

The Exponential Linear Squashing AF, known as the ELiSH function is another AF, proposed by [50]. The ELiSH shares common properties with the Swish function. The ELiSH function is a combination of the ELU and Sigmoid, given by

$$f(x)\ = \begin{cases} \left(\frac{x}{1+\ e^{-x}}\right), & x\ \geq\ 0 \\ \left(\frac{e^x\ -1}{1+\ e^{-x}}\right), & x\ <\ 0 \end{cases} \tag{29}$$

The properties of the ELiSH varies in both the negative and positive parts as defined by the limits. The Sigmoid part of the ELiSH function improves information flow while the linear parts eliminate the vanishing gradient issues. The ELiSH and HardELiSH tested successfully on the ImageNet dataset using different deep CNN architectures [50].

3.10.1. HardELiSH. The HardELiSH is the hard variant of the ELiSH activation function. It is a multiplication of the HardSigmoid and ELU in the negative part and a multiplication of the Linear and the HardSigmoid in the positive part [50]. The HardELiSH is given by

$$f(x) = \begin{cases} x \times max\left(0, min\left(1, \left(\frac{x+1}{2}\right)\right)\right), & x \geq 0 \\ (e^x - 1) \times max\left(0, min\left(1, \left(\frac{x+1}{2}\right)\right)\right), & x < 0 \end{cases} \tag{30}$$

3.11. Hexpo

The Hexpo is another learning speed improvement AF that differs from the rectified AFs. It has scalable limits on both negative and positive values and produces noise-robust output with identity mapping on the positive part [51]. The Hexpo input lies between $(-c, a)$ and it is computed as

$$f(x) = \begin{cases} -a(e^{-\frac{x}{b}} - 1), & \text{if } x \geq 0 \\ c(e^{\frac{x}{d}} - 1), & \text{otherwise} \end{cases} \quad (31)$$

The Hexpo outperformed ReLU on MNIST and CIFAR-10 and it produced comparable accuracy results to ELU [51].

4. Comparison of the Trends in Activation Functions Used in Deep Learning Architectures

The search for these AFs is based on the published research results of the winners of ILSVRC competitions alongside some cited research output from the AF research results found in the literature. The ImageNet competition was chosen for this trend comparison because it is the competition that produced the first DL success [28]. This forms the core basis for both the search of the functions outlined in this paper and the current trends in the use of AFs, although the scope of these trends goes beyond image recognition as other applications that birthed other DL AFs were included [8]. The numerous deep architectures include deep feedforward NNs, CNN, long short term memory, RNN, DBN, and deep generative models like deep Boltzmann machines, etc [1], [4]. The learning process is achieved with these specialized architectures.

The AFs used in DL architectures dates back from the beginning of the adoption of the deeper architectures for NN computations. Before 2012, image recognition challenges used shallow architectures and had few AFs embedded in them [6]. However, the adoption of the deeper architectures for neural computing was first explored by researchers in the ILSVRC [28], which is an annual event that started in 2010. Since then, the use of the deeper architectures has witnessed huge research advances in optimization techniques for training the deeper architectures, since the deeper the network, the greater the difficulty to train and optimize [37], [52], although performance enhancement is guarantee. The AF is a key component for training and optimization of NN, implemented on different layers of DL architectures, is used across domains including natural language processing, object detection, classification, segmentation, etc. These trends summarised in 1, outlining the remarkable architectural design improvements alongside the position of the respective AF's in the architecture from AlexNet; winner of ILSVRC 2012 [28] and the modified networks like SqueezeNet [53], which had multiple AFs embedded in it.

Besides, other architectures including AmoebaNet [54] have produced significant result improvement and researchers carefully exploring the balance of network depth, width, and resolution to enhance performance using a compound coefficient to produce a new architecture EfficientNet [55]. Further improvements on the training approach using unlabelled data produced another SOTA for ILSVRC [56] named Noisy Student. The Table 1 highlight the architectural trends of DNN that achieved significant improvement to the existing architectures, with the positions of the AF's used in those architectures outlined. Automatically designing the CNN architecture represents an emerging network design trend and researchers have suggested that it produces comparable results to the manually designed architectures [57].

TABLE 1. TYPES AND POSITIONS OF AFS USED IN DL ARCHITECTURES.

Architecture	Hidden Layers	Output Layer	Cross Reference
AlexNet	ReLU	Softmax	[28]
NiN	No activation	Softmax	[30]
ZFNet	ReLU	Softmax	[38]
VGGNet	ReLU	Softmax	[58]
SegNet	ReLU	Softmax	[29]
GoogleNet	ReLU	Softmax	[37]
ResNet	ReLU	Softmax	[59]
ResNeXt	ReLU	Softmax	[60]
SeNet	ReLU	Sigmoid	[61]
AmoebaNet	ReLU	Softmax	[54]
EfficientNet	Swish	SiLU	[55]

From Table 1 above, it is evident that the ReLU and Softmax AF are the dormant AFs used in practical DL applications, however more recent Sigmoid and SiLU AFs are finding applications in architecture design. Furthermore, the Softmax function is the dominant AF used in the output layer of most common practical DL applications while the ReLU AF's were used mostly in the hidden layers. In choosing the SOTA architecture for recognition tasks, it is handy from Table 1 to select the current and most recent architecture and understand the architectural make-up of the network. The EfficientNet architecture is the current SOTA architecture for recognition tasks as found in the literature.

5. Discussions

The identified crucial AFs used in DL research include the ReLU, with six variants, apart from the original ReLU including LReLU, PReLU, RReLU, SReLU, DReLU, and RMAF. Furthermore, the other AFs are the Sigmoid with four variants including HardSigmoid, SiLU, dSiLU and lsigmoid. The Hyperbolic Tangent has the Hardtanh and LiSHT variants while the ELU has PELU and SELU variants. The Softmax, Softsign, Softplus, Maxout, Swish,and Hexpo functions have no variants. Besides, the SOTA DL architectures have multiple AFs embedded to every given network and it is noteworthy to highlight that these multiple AFs are used at different layers, to perform gradient computations in a given network, thereby obtaining an effective learning, convergence and characterisation of the presented inputs. The rectified and exponential AF are the most researched AF used in DL applications. The ELUs has been highlighted as a faster learning AF compared to the ReLU [45], [62], and this assertion was validated by researchers, after an

extensive comparison of some variants of the ELU and ReLU AF on the MNIST recognition dataset [63]. The most significant AFs that have advanced DL research across different domains are outlined in Table 2.

TABLE 2. DOMINANT ACTIVATION FUNCTIONS ACROSS DOMAINS

AF	CNN	RNN	SAE
Swish	*		
ReLU	*		*
Softmax	*		
SiLU	*		
Sigmoid		*	*

The PELU and PReLU are two different parametric AFs that evolved from the exponential and linear units with some learnable parameters. It is also evident that there are other AFs that researchers have suggested that they perform better than the ReLU. However, the ReLU has been the most consistent AF for DL applications since invention [13], [36]. Other researchers validated that the other variants of the ReLU including LReLU, PReLU and RReLU performed better than the ReLU but some of these functions lack theoretical justifications to support their SOTA results [40]. Furthermore, the parametric AFs have been a development in emerging applications where the AF was used as a learnable parameter from the dataset, thus looks to be a promising approach in the new functions developed most recently as observed in SReLU, PELU, and PReLU. Also, the majority of these AFs were developed and tested on CNN architecture and image recognition datasets. This further demonstrated that DL tasks are thriving mostly in classification tasks, using CNNs.

The most notable observation on the use of AFs for DL applications is that the newer AFs seem to outperform the older ones like the ReLU, yet even the latest top-performing architectures rely on the ReLU in the hidden layers except EfficientNet, which had the Swish and SiLU AFs. [55]. However, We can suggest that the current practices do not use the newly developed SOTA AFs but depends on the tested and proven AFs, thereby underlining the fact that the newer AFs are seldom used in practice.

Finally, to choose an appropriate AF for a given DL architecture requires testing these AF's heuristically on the architecture, and observing the performance and training results. This approach ensures that the most appropriate AF is selected for a given DL application.

6. Conclusion

This paper provides a comprehensive summary of AFs used in DL and most importantly, highlights the current trends in the use of these functions in practice for the development of the SOTA architectures used in DL research. It provided a brief introduction to DL and AFs, discussed the different types of AFs and outlined the specific applications where these functions were tested in the development of DL architectures. The specific architectural use of the AFs was presented in tabular form for easy identification. These AFs can improve the learning of the patterns in data thereby automating the process of features detection and justifying their use in both the hidden and output layers of the neural networks and their usefulness across domains.

The AF's have developed over the years with the compounded AFs being the future of AFs research. Besides, it is also worthy to state that there are other AFs that have not been discussed in this literature as we focused was on the AFs, used in DL applications. Future work would be to compare all these SOTA functions on the most recent architectures, using standard datasets.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] N. Jones, "Computer science: The learning machines," *Nature News*, vol. 505, no. 7482, p. 146, 2014.

[3] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[5] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[6] 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, 1989.

[7] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European conference on computer vision*. Springer, 2016, pp. 646–661.

[8] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, 2009, pp. 245–248.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[13] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[14] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[15] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[16] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*. Springer, 1995, pp. 195–201.

[17] R. M. Neal, "Connectionist learning of belief networks," *Artificial intelligence*, vol. 56, no. 1, pp. 71–113, 1992.

[18] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[19] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3–11, 2018.

[20] Y. Qin, X. Wang, and J. Zou, "The optimized deep belief networks with improved logistic sigmoid units and their application in fault diagnosis for planetary gearboxes of wind turbines," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 5, pp. 3814–3824, 2018.

[21] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," in *Advances in neural information processing systems*, 2001, pp. 472–478.

[22] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li, "Improving deep neural networks using softplus units," in *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–4.

[23] A. Senior and X. Lei, "Fine context, low-rank, softplus deep neural networks for mobile speech recognition," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 7644–7648.

[24] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International conference on machine learning*, 2017, pp. 933–941.

[25] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[26] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[27] S. K. Roy, S. Manna, S. R. Dubey, and B. B. Chaudhuri, "Lisht: Non-parametric linearly scaled hyperbolic tangent activation function for neural networks," *arXiv preprint arXiv:1901.05894*, 2019.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[29] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[30] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[31] P. Le and W. Zuidema, "Compositional distributional semantics with long short term memory," *arXiv preprint arXiv:1503.02510*, 2015.

[32] W. Ping, K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller, "Deep voice 3: 2000-speaker neural text-to-speech," *Proc. ICLR*, pp. 214–217, 2018.

[33] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[34] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, "On rectified linear units for speech processing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3517–3521.

[35] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8609–8613.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[38] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[39] S. S. Md Noor, J. Ren, S. Marshall, and K. Michael, "Hyperspectral image enhancement and mixture deep-learning classification of corneal epithelium injuries," *Sensors*, vol. 17, no. 11, p. 2644, 2017.

[40] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[41] X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep learning with s-shaped rectified linear activation units," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[42] D. Macêdo, C. Zanchettin, A. L. Oliveira, and T. Ludermir, "Enhancing batch normalized convolutional networks using displaced rectifier linear units: A systematic comparative study," *Expert Systems with Applications*, vol. 124, pp. 271–281, 2019.

[43] Y. Yu, K. Adu, N. Tashi, P. Anokye, X. Wang, and M. A. Ayidzoe, "Rmaf: Relu-memristor-like activation function for deep learning," *IEEE Access*, vol. 8, pp. 72 727–72 741, 2020.

[44] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[45] L. Trottier, P. Gigu, B. Chaib-draa *et al.*, "Parametric exponential linear unit for deep convolutional neural networks," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2017, pp. 207–214.

[46] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[47] J. Lehman, J. Chen, J. Clune, and K. O. Stanley, "Safe mutations for deep and recurrent neural networks through output gradients," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2018, pp. 117–124.

[48] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *International conference on machine learning*, 2013, pp. 1319–1327.

[49] L. Tóth, "Phone recognition with hierarchical convolutional deep maxout networks," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2015, no. 1, pp. 1–13, 2015.

[50] M. Basirat and P. M. Roth, "The quest for the golden activation function," *arXiv preprint arXiv:1808.00783*, 2018.

[51] S. Kong and M. Takatsuka, "Hexpo: A vanishing-proof activation function," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 2562–2567.

[52] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[53] V. Golkov, A. Dosovitskiy, J. I. Sperl, M. I. Menzel, M. Czisch, P. Sämann, T. Brox, and D. Cremers, "Q-space deep learning: twelve-fold shorter and model-free diffusion mri scans," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1344–1351, 2016.

[54] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.

[55] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[56] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le, "Self-training with noisy student improves imagenet classification," in *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[57] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, 2020.

[58] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[60] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[61] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4438–4446.

[62] Z. Deng, Z. Wang, and S. Wang, "Stochastic area pooling for generic convolutional neural network," in *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, 2016, pp. 1760–1761.

[63] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on mnist classification task," *arXiv preprint arXiv:1804.02763*, 2018.