Adaptive Rectified Linear Unit (AReLU) for Classification Problems to Solve Dying Problem in Deep Learning

Ibrahim A. Atoum
Department of Computer Science and Information Systems
College of Applied Sciences, Al Maarefa University
Riyadh, Saudi Arabia

***Abstract*—A convolutional neural network (CNN) is a subset of machine learning as well as one of the different types of artificial neural networks that are used for different applications and data types. Activation functions (AFs) are used in this type of network to determine whether or not its neurons are activated. One non-linear AF named as Rectified Linear Units (ReLU) which involves a simple mathematical operations and it gives better performance. It avoids rectifying vanishing gradient problem that inherents older AFs like tanh and sigmoid. Additionally, it has less computational cost. Despite these advantages, it suffers from a problem called Dying problem. Several modifications have been appeared to address this problem, for example; Leaky ReLU (LReLU). The main concept of our algorithm is to improve the current LReLU activation functions in mitigating the dying problem on deep learning by using the readjustment of values (changing and decreasing value) of the loss function or cost function while number of epochs are increased. The model was trained on the MNIST dataset with 20 epochs and achieved lowest misclassification rate by 1.2%. While optimizing our proposed methods, we received comparatively better results in terms of simplicity, low computational cost, and with no hyperparameters.**

Keywords*—Rectified Linear Unit (ReLU); Convolutional Neural Network; activation function; deep learning; MNIST dataset; machine learning

I. Introduction

The concept of Artificial Intelligence (AI) revolves around creating intelligent machines that are able to simulate human thinking while Machine Learning (ML) is a branch of this concept that allows these intelligent machines to learn the hidden patterns from the input data [1]. Neural networks (NNs) as a subset of ML simulate the human brain using a set of algorithms. These networks consist of input, hidden, and output layers. These layers consist of neurons that mimic the structure of a biological neuron, where each neuron has inputs that are processed to give outputs, which in turn will be input to another neuron. When neural networks consist of more than three layers then they can be called Deep Learning Networks (DLNs) [2].

AFs play a critical role in DLNs to extract the results from the input values and thus determine whether the underline neuron is activated or not [3]. DLNs can be considered as just a linear regression without AFs, so appropriate AFs must be used to model a nonlinear DLNs. AFs classified as binary step, linear activation and nonlinear activation functions. Binary step function is a basic threshold classifier where some threshold value is decided to choose which output neurons should be activated or deactivated. Linear activation function is a simple straight line activation function that converts linear input signals into non-linear output signals. Nonlinear AFs are what make it easier for the DLNs model to adapt to a variety of data and to distinguish between outcomes; examples are: ReLU, Leaky ReLU, Sigmoid, Tanh and Softmax [4]. Some of these are suited to be used in hidden layers and others in output layers.

There are two terms used in training the model, the first is the term feedforward, which is used in NNs to refer to the transition with specified weights from input to output, while the term backpropagation, as the name suggests, moves from output to input with readjustment of weights depending on loss values and then propagation processes straight ahead. This approach allows the use of gradient methods, such as gradient descent or stochastic gradient descent, to train multi-layer networks and update weights to reduce loss [5] [6]. ReLU as a nonlinear AF has gained a lot of interest in research due to its simplicity, low computation cost and it avoids the vanishing gradient problem that inherent to the earliest AFs like tanh and sigmoid [7]. Despite all the previous advantages of this function, it has a problem called the Dying ReLU problem, which indicates that the neuron becomes inactive and outputs zero only for any input. This problem has been attributed to a high learning rate and a high negative bias [8].

ReLU was initially introduced by [9]; the researchers designed an electronic circuit to simulate a hybrid slug in which the latent cortex combines the digital selection of an active cluster of neurons with an analog response, and this behavior is achieved by dynamically changing the positive feedback inherent in recurrent cortical connections, this behavior, according to the researchers, created computational capabilities creates the process of stimulus selection, conferring the ability to modify and generate a spatio-temporal pattern in this cortex.

ReLU was later used in object recognition by [10], and researchers summarized the three stages used to extract object features such as filter bank, nonlinear transformation and a kind of feature pooling layer emphasizing that most systems use one or two of these stages, assuming that the use of two stages gives more accurate results. The study demonstrated the

accuracy of this hypothesis by using nonlinear layers and pooling layers on different object data sets through either supervised optimization or unsupervised pre-training.

ReLU was also popularized by [11] in the context of Restricted Boltzmann Machines. The study demonstrated how to create a more powerful type of hidden units for Restricted Boltzmann Machines (RBM) in object recognition and face comparison by combining weights and biases for an infinite set of binary units with approximating these stepped sigmoid units with noisy corrected linear units.

Leaky ReLU [12] has added a slight slop in the negative range; this modification on ReLU ends the presence of dead neurons in the negative region by using a hyperparameter. Thereafter many leaky ReLU variants have been appeared like Parametric Rectified Linear Unit (PReLU) [13] which introduces a new learnable parameter as a slope for the negative part and Exponential linear unit (ELU) has used an exponential function to transition from the positive to small negative values [14].

The value of the loss function is related to the results of the model. If the value of the loss function is low, this means that the model will give good results [16]. Loss functions are divided into two types, classification and regression. Classification functions also divided into binary entropy loss/log loss and hinge loss. During the execution of AReLU the first function was used [15]. AReLU is applied on MNIST dataset that contains 70000 images of black and white handwritten digits divided into 60000 images for training and 10000 images for Testing [17].

In this study, the decreasing value of the used loss function was exploited as an adaptive parameter to keep the network active. The study is presented into four sections: section two introduces the idea of ReLU, section three identifies the ReLU dying problem, and section four introduces the AReLU. Section five presents the results and finally section six is the conclusion.

II. Rectified Linear Unit (ReLU) Activation Function

Artificial neurons are mathematical model that mimic human biological neurons and they are the basic building blocks of neural networks as shown in Fig. 1.

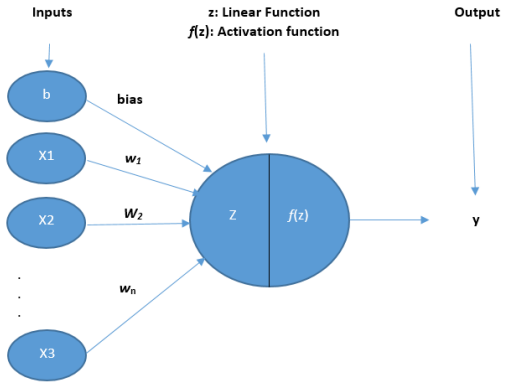


Fig. 1. Artificial neurons representation

Where $z = ((w_1 \times x_1) + (w_2 \times x_2) + \ldots\ldots\ldots + (w_n \times x_n)) + b$ states the linear function, accordingly the activation function $y = f(z)$. $x_1 \; to \; x_n$ represents the inputs, $w_1 \; to \; w_n$ illustrates the weights that connect inputs with perceptions and they measure the significance level of each input. The bias value (b) is added to the weighted sum of the inputs to prevent the activation function from getting a zero value. This linear results in linear modeling come from the linear mapping of the input function to output in hidden layers. The role of activation function is to convert these linear outputs into non-linear outputs for further computation as in

$y = \alpha * (((w_1 \times x_1) + (w_2 \times x_2) + \ldots\ldots\ldots + (w_n \times x_n)) + b)$; where α is the activation function. The literature has introduced many activation functions such as Sigmoid, binary step, Tanh, ReLU, Leaky ReLU, identity and Softmax.

ReLU activation function can be described mathematically as in Eq. (1) and graphically as in Fig. 2, where x is the input to the neuron. The function f(x) equal zero for all negative input values and equal original input value for all positive input values as in Eq. (1).

$$f(x) = \max(0,x) \quad (1)$$

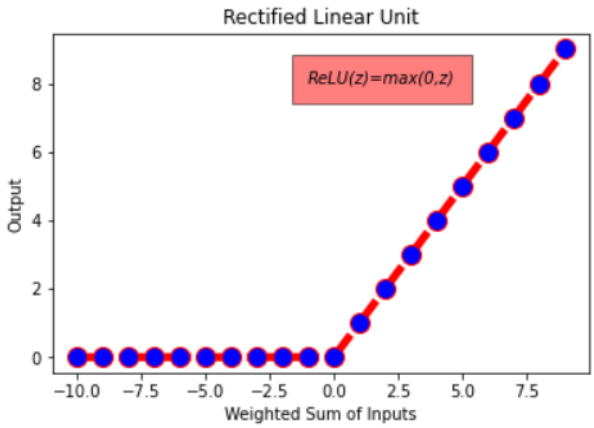


Fig. 2. ReLU representation

ReLU avoids vanishing gradient problem occurred with other activation functions by preserving the gradient [18]. This problem is formed when the gradients of deep neurons vanish or becomes zero, this means that the deep layers of the network may not learn or learn very slowly [19]. Derivative Activation function is fundamental to optimizing neural network, the ReLU (x) can be expressed as:

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

It can be simplified as follows:

$$\max(0, x) = \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$$

The first order derivative of this function is:

$$\frac{d}{dx} f(x) = \frac{d}{dx} \max(0, x) = \frac{d}{dx} \begin{cases} 0, x < 0 \\ x, x \geq 0 \end{cases}$$

And can be illustrated as:

$$\frac{d}{dx}f(x) = \frac{d}{dx}\begin{cases}\frac{d}{dx}(0), x < 0\\ \frac{d}{dx}(x), x \geq 0\end{cases}$$

The final derivative is:

$$\frac{d}{dx}f(x) = \frac{d}{dx}\begin{cases}0\ , x < 0\\ x, x \geq 0\end{cases}$$

III. ReLU Dying Problem

Dying ReLU problem is one limitation for ReLU where its neurons output is zero as illustrated by the red outline in Fig. 3.

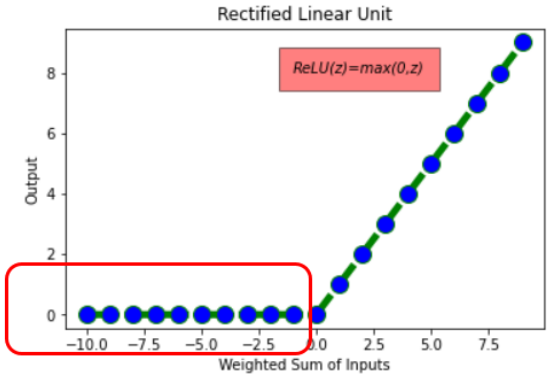


Fig. 3. Red outline where ReLU outputs 0

The normal situation for ReLU neurons is to stay active, update weights, and keep learning. Although this feature provides the power to ReLU through the sparsity of the network, it poses a problem when most of the inputs of these ReLU neurons are in the negative range, and the issue becomes more complicated when the output of most Neurons is zero, making their task so abnormal that they become inactive and unlearning. This inevitably causes gradients to fail to flow during backpropagation.

The cause of this problem is due to two main factors: High Learning Rate and a Large Negative Bias. The former one allows faster learning with the possibility of a numerical overflow, while its very small value may never converge or stumble on a suboptimal solution. So choosing an average rate that is neither too large nor too small ensures an optimal approximation of the mapping problem as represented by the training data set. The best way to discover the value of the learning rate is through trial and error, not analytically for a particular model on a particular data set. This can be illustrated by the update process in backpropagation as shown in Eq. (2).

$$newW_{ij} = oldW_{ij} - LR(\left(\frac{\partial Error}{\partial oldW_{ij}}\right) \quad (2)$$

where $\frac{\partial Error}{\partial oldW_{ij}}$ is the derivative of error with respect to weight. We can see from Eq. (2) that giving a high value of the learning rate (LR) will cause a high value for the last part of Eq. (2) $LR * (\left(\frac{\partial Error}{\partial oldW_{ij}}\right)$, so subtracting large number from $oldW_{ij}$ will end up with highly negative $newW_{ij}$. These negative results cause negative inputs for ReLU, therefore generating the dying ReLU problem.

Biases are extra inputs that ensure neurons are activated regardless of the input. Changing the value of the weights in the neuron changes the steepness of the curve without the ability to change it to the right or left, to change the curve to the left or right the value is changed. Giving a high negative bias value makes the ReLU activation input negative. To mitigate Dying ReLU problem, several techniques have emerged, all trying to keep the network active when the input is negative or zero.

Leaky ReLU [12] demolished dead neurons in the negative part by adding a slight slop in the negative range using a hyperparameter (α=0.1 or more) as shown in Eq. (3) and illustrated in Fig. 4.

$$f(x) = \max(ax, x), where\ a > 0 \quad (3)$$

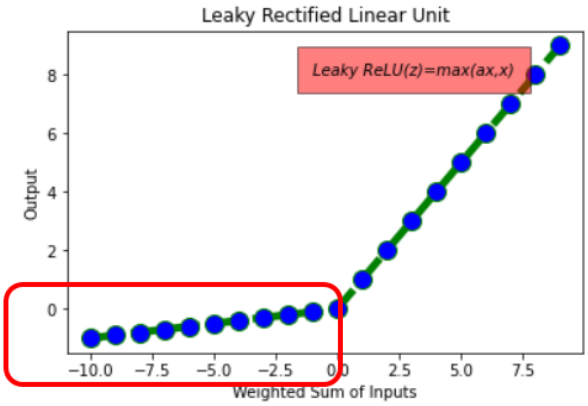


Fig. 4. Red outline where leaky ReLU outputs less than zero

Parametric Rectified Linear Unit (PReLU) [13] thereafter presented a new learnable parameter as a slope for the negative part as in Eq. (4):

$$f(x) = \max(ax, x), \text{where a is a Learnable Parametere} \quad (4)$$

And Exponential linear unit (ELU) used an exponential function $\propto (e^x - 1)$ to transition from the positive to small negative values [14] as shown in Eq. (5).

$$f(x) = \begin{cases}x\ , x > 0\\ \propto (e^x - 1), x \leq 0\end{cases} \quad (5)$$

IV. Adaptive Rectifier Linear Unit (ARELU) on MNIST Dataset

The study used the MNIST dataset of handwritten greyscale images, these images were size-normalized and centered in a fixed-size image available from NIST [20] as shown in Fig. 5 which shows the first 25 images of MNIST.

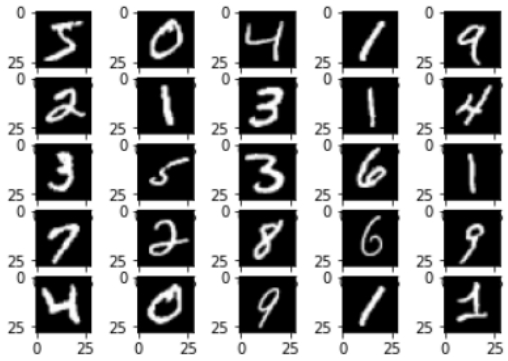


Fig. 5. Subset of MNIST dataset

This dataset composed of approximately 70,000 handwritten monochrome images of 0 to 9 (10 digits), each of which is 784 pixels in size, so that the input data is in pairs (70,000,784) and output (70,000, 10) as shown in Fig. 6.

To form the network, the AReLU activation function were used in the hidden layer and softmax in the output layer. The used loss function is categorical_crossentropy and the optimizer is Adamax. The batch size is adopted to 128 and the number of epochs to 20.

Once the output is generated from the final neural net layer, loss function (input vs output) is calculated and backpropagation process is performed where the weights are adjusted to get the minimum loss. Neural Networks are trained using the gradient descent process. This process consists of the backward propagation step which is basically chain rule to get the change in weights in order to reduce the loss after every epoch.

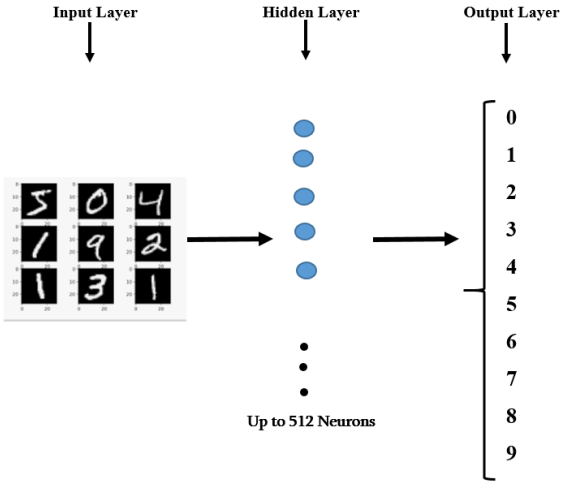


Fig. 6. MNIST neural network

The primary goal of AReLU is to mitigate Dying ReLU problem by improving the previous methods by using the adaptive Loss Function (l) parameter instead of hyperparameter one. l is multiplied by the input value as shown in Eq. (6) to transit from the positive to small negative values.

$$f(x) = \begin{cases} x, x > 0 \\ l * x, x \leq 0 \end{cases} \quad (6)$$

The AReLU has implemented by using Python programming language according to the algorithm shown in Fig. 7 and more illustrated in Fig. 8. It is noticed from the equation that there is no change in the case of the positive values, but only the change in the negative inputs, as we notice this in Fig. 9(a).

Step1: START AReLU

Step2: *Import the required libraries.*

keras, matplotlib, mnist, etc…

Step3: *Loading the built-in MNIST Dataset.*

mnist.load_data()

Step4: *Creating the model and add the layers: Input, Hidden and Output using dense layer from keras*

Step5: *Compile the model by using the defined Loss Function, Optimizer and the metrics*

Step6: for i in range of epochs

6.1 Fitting the Model by using the training set.
6.2 Update AReLU parameter= new Loss value

Step7: *Evaluate the model on the testing set.*

Step8: STOP

Fig. 7. AReLU algorithm

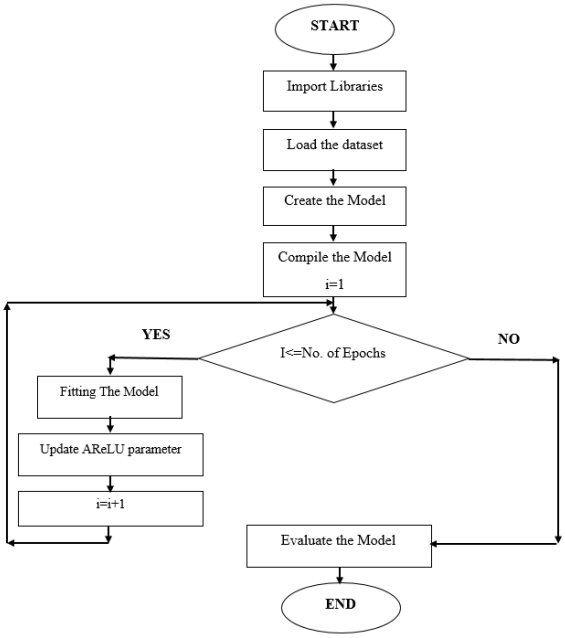


Fig. 8. AReLU flowchart

The used structure of the deep neural network is shown in Fig. 9 that composed of four layers, one input layer represents the input shape as 784 image pixels, two hidden layers each composed of 512 neurons and the final 10 neurons layer that characterize the output layer.

Binary Cross-Entropy Loss/Log Loss has been used as loss function in the model compilation process; where in this phase the loss function, the optimizer and the metrics are defined. This function is defined in Eq. (7); where N is the number of rows and M the number of classes. p_{ij} are the corrected probabilities, a negative average is used to compensate for negative values resulted from calculating log value of

corrected probabilities because their values range between 0 and 1. It is one of the most common loss functions used in multiclass classification problems. The value of this function decreases as the predicted probability converges to the actual label.

$$Loss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij}) \quad (7)$$

Backpropagation in a network aims to make a change in the error value with respect to weights and this process is called derivative because its goal is to make a change in one value with respect to another. The first derivative of this function is:

$$\frac{d}{dx}f(x) = \frac{d}{dx}\begin{cases}\frac{d}{dx}(x), x > 0\\ \frac{d}{dx}(l * x), x \leq 0\end{cases}$$

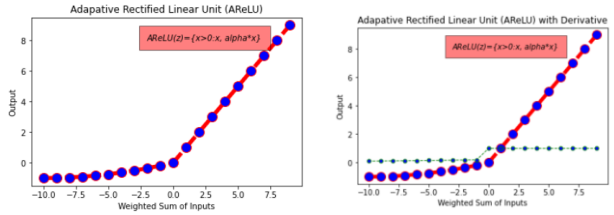
$$\frac{d}{dx}f(x) = \frac{d}{dx}\begin{cases}1, x > 0\\ x, x \leq 0\end{cases}$$

The function starts with any initial l value; say 0.1 and then it is automatically adapted according to the initial loss function value. Fig. 9(b) shows the Graphical Representation of AReLU Derivative. It is evident that the values of the derivative are close to zero but are not zero in the case of negative values.

The most effected activation function used in the output layer in the case of multi-layer classification problems is Softmax, which converts the raw outputs of a neural network into a vector of probability scores between 0 and 1. Its equations is defined in:

$$Softmax\ (o)_i = e^{o_i} \Big/ \sum_{j-1}^{N} e^{o_j}$$

Where **o** is the input vector, e^{o_i} is the standard exponential function for o_i, N is the number of classes in the multiclass classifier and e^{o_j} is the standard exponential function for output vector and e is the exponential which is equal nearly 2.718.



(a)AReLU (b)AReLU and its Derivative

Fig. 9. AReLU graphical representation and its derivative

V. Results

Fig. 10 illustrates the relationship between the training and validation accuracy over 20 epochs, the accuracy escalations are noticed in the first three epochs, indicating that the network is learning fast, thereafter the curve flattens, indicating that there is no need for more epochs to further training the model. The model accuracy was 98.8% (meaning 9880 of the 10000 images were predicted correctly!) and 120 images were wrongly tagged (1.2%).

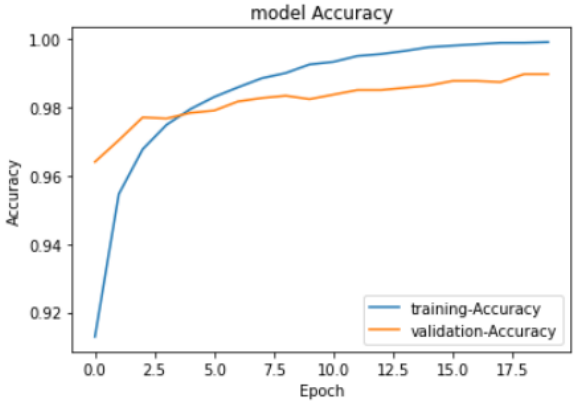


Fig. 10. AReLU model accuracy

AReLU gives better results based on the Misclassification Rate (MR) which is a measure of the percentage of observations that were incorrectly predicted by some classification model [21] and it's calculated as in

$$MR = Incorrect\ Predictions \Big/ Total\ Predictions$$

The MR for our model was 1.2% where for PReLU is 1.62 according to [22] as shown in Table I. This study measured the MR for different adaptive ReLUs including Sigmoid, tanh, MSAF, MSAf_Symmetrical, ReLU, LReLU, ELU and adaptive tanh.

TABLE I. MR Measurements for Activation Functions on MNIST Dataset

AF	MR
Sigmoid	7.01
Hyperbolic Tangent	1.86
MSAF	12.59
MSAF_Symmetrical	11.28
ReLU	2.08
LReLU	1.68
PReLU	1.6
ELU	1.88
Adaptive tanh	2.93

Reading Fig. 11 which illustrates the relationship between training and validation loss, we can see the rapid loss in the training set at the first two epochs while validation loss remained almost constant for several epochs, in contrast to the loss level of the training set, which means that the model can be generalized to unseen data.

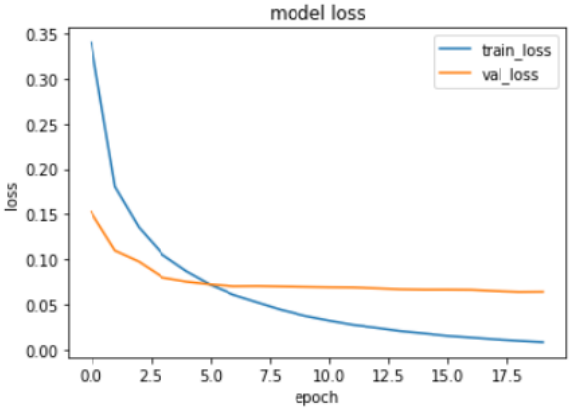


Fig. 11. AReLU model loss

VI. Conclusion

This article produced automatic and adaptive activation function in which it retained inherent characteristics of ReLU with simplicity, high accuracy, speed and low loss ratio. Expected diminishing characteristic of Loss function value has exploited in implementing the AReLU, this function is used to measure the difference between the current output and the expected output. Cross-entropy type is used in developing the ReLU as one of the most widely used loss functions in machine learning due to its role in better generalization and faster model training. This function is used in binary and multi-class classification cases. AReLU is implemented by using Python programming language on MNIST dataset of handwritten digits to get 1.2% classification Rate. The model maintained the gains that the previous methods indicated, such as simplicity, low computational cost, no fixed coefficients, and adaptation in nature. In the future, AReLU will be applied to different data sets and work to reduce the rate of misclassification while maintaining the characteristics of simplicity, low computational cost, and no hyperparameters.

Acknowledgment

The Author would like to express his gratitude to AlMaarefa University, Riyadh, Saudi Arabia, for providing funding to do this research.

Funding

This research was funded by AlMaarefa University, Riyadh, Saudi Arabia

References

[1] Nichols, James A., Hsien W. Herbert Chan, and Matthew AB Baker. "Machine learning: applications of artificial intelligence to imaging and diagnosis." *Biophysical reviews* 11.1 (2019): 111-118.

[2] Sarker, Iqbal H. "Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions." *SN Computer Science* 2.6 (2021): 1-20.

[3] Dubey, Shiv Ram, Satish Kumar Singh, and Bidyut Baran Chaudhuri. "Activation functions in deep learning: a comprehensive survey and benchmark." *Neurocomputing* (2022).

[4] Fan, Jianqing, Cong Ma, and Yiqiao Zhong. "A selective overview of deep learning." *Statistical science: a review journal of the Institute of Mathematical Statistics* 36.2 (2021): 264.

[5] Shaik, Nagoor Basha, et al. "A feed-forward back propagation neural network approach to predict the life condition of crude oil pipeline." *Processes* 8.6 (2020): 661.

[6] Xie, Jingyi, and Sirui Li. "Training Neural Networks by Time-Fractional Gradient Descent." *Axioms* 11.10 (2022): 507.

[7] Li, Yanyi, Jian Shi, and Yuping Li. "Real-Time Semantic Understanding and Segmentation of Urban Scenes for Vehicle Visual Sensors by Optimized DCNN Algorithm." *Applied Sciences* 12.15 (2022): 7811.

[8] Chai, Enhui, et al. "An Efficient Asymmetric Nonlinear Activation Function for Deep Neural Networks." *Symmetry* 14.5 (2022): 1027.

[9] Hahnloser, Richard HR, et al. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit." *nature* 405.6789 (2000): 947-951.

[10] Jarrett, Kevin, et al. "What is the best multi-stage architecture for object recognition?." *2009 IEEE 12th international conference on computer vision*. IEEE, 2009.

[11] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Icml*. 2010.

[12] Maas, Andrew L., Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models." *Proc. icml*. Vol. 30. No. 1. 2013.

[13] He, Kaiming, et al. "." *Proceedings of the IEEE international conference on computer vision*. 2015.

[14] Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)." *arXiv preprint arXiv:1511.07289* (2015).

[15] Hajiabadi, Moein, et al. "Deep learning with loss ensembles for solar power prediction in smart cities." *Smart Cities* 3.3 (2020): 842-852.

[16] Chadha, Ankita, Azween Abdullah, and Lorita Angeline. "A Comparative Performance of Optimizers and Tuning of Neural Networks for Spoof Detection Framework." *International Journal of Advanced Computer Science and Applications* 13.4 (2022).

[17] Deng, Li. "The mnist database of handwritten digit images for machine learning research [best of the web]." *IEEE signal processing magazine* 29.6 (2012): 141-142.

[18] Razak, H. A., et al. "Detection of Criminal Behavior at the Residential Unit based on Deep Convolutional Neural Network." *International Journal of Advanced Computer Science and Applications* 13.2 (2022).

[19] Tan, Hong Hui, and King Hann Lim. "Vanishing gradient mitigation with deep learning neural network optimization." *2019 7th international conference on smart computing & communications (ICSCC)*. IEEE, 2019.

[20] Cohen, Gregory, et al. "EMNIST: Extending MNIST to handwritten letters." *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017.

[21] Maach, Anas, et al. "An Intelligent Decision Support Ensemble Voting Model for Coronary Artery Disease Prediction in Smart Healthcare Monitoring Environments." *arXiv preprint arXiv:2210.14906* (2022).

[22] M. M. Lau and K. Hann Lim, "Review of Adaptive Activation Function in Deep Neural Network," *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, 2018, pp. 686-690, doi: 10.1109/IECBES.2018.8626714.