Grenze International Journal of Engineering and Technology, Jan Issue

Learning Combination of Activation Functions and Various Optimizers with Loss Function for Improved Bi-LSTM Lexicon Embedding Model

Nibedita P[1] and Dr. Asha T[2]
[1]Dept. of CSE, Bangalore Institute of Technology, Bangalore, India
Email: nibedita.kuni@gmail.com
[2]Prof. & Head, Dept. of CSE, Bangalore Institute of Technology, Bangalore, India
Email: asha.masthi@gmail.com

***Abstract*—Deep Learnings has got a lot of prominence due to its advance results in various fields like Computer Vision, Natural Language Processing, Time Series Analysis, Health Care etc. In the last decade, an active area of research has been devoted to design novel activation functions that are able to help deep neural networks to converge, obtaining better performance. The training procedure of these architectures usually involves optimization of the weights of their layers only, while non-linearity's are generally pre-specified and their (possible) parameters are usually considered as hyper-parameters to be tuned manually. Earlier, the Deep Learning was implemented using the batch and stochastic gradient descent algorithms and some optimizers which lead to very less performance of the models. But today, lot of work is going on for the enhancement of the performance of Deep Learning using various optimization techniques. So, in this context, it is proposed to build a Deep Learning model using various Optimizers (Adagrad, RmsProp, Adam, Adamax, SGD), Loss functions (mean squared error, binary cross entropy) and various activation functions (Relu, Elu, Softmax, Sigmoid, Exponential, Tanh) with different input-size and no. of epochs, for the Convolutional neural networks and Recurrent neural networks and verify the performance such as Accuracy and Loss of the model. The proposed model has achieved maximum Accuracy when Adam optimizer and binary cross entropy loss function and Relu activation function with an input size of 250 and 20 epochs are applied on Recurrent neural networks. This paper reports a behavioural analysis and accuracy comparison of improved Bi-LSTM (Long-Short Term Memory) and LSTM models. The objective is to explore to what extend additional layers of training of data would be beneficial to tune the involved parameters. Here we have Proposed comparison between different algorithms and the experiment results are shown the recurrent neural network using LSTM for the trained data model that is scored the highest accuracy of 97%, 86%, and 89% for the Amazon Cell review, IMDB and Yelp dataset respectively. Experimental result of three public benchmark datasets show our proposed model yields obvious performance improvement.**

***Index Terms*— Activation function, Deep learning, Optimizers, Loss Function, Recurrent Neural Networks(RNN), Bi-LSTM.**

Grenze ID: 01.GIJET.8.1.73

I. INTRODUCTION

A. Deep Learning

In the last decade, deep learning [1][7][8] has achieved remarkable results, which has been applied in various domains like computer vision, natural language processing, robotics, health care and artificial intelligence etc. The roots of recent successes of deep learning can be found in: (i) the increase of the data available for training neural networks, (ii) the rising of commodity computational power needed to crunch the data, (iii) the development of new techniques, architectures, and activation functions that improve convergence during training of deeper networks, overcoming the obstacle of vanishing/exploding gradient [9], [10]. In Deep Learning, the learning is achieved at various levels. The inputs are processed at the initial level, transform into an abstract form, at the next level, still processed into more precise form and likewise the learning is achieved deeper. In this paper, the model is built using the Deep Learning algorithms such as Convolutional neural networks(CNN), Recurrent neural networks(RNN).

For many years, neural networks have usually employed logistic sigmoid activation functions. Unfortunately, this activation is affected by saturation issues. This problem reduces its effectiveness and, nowadays, its usage in feedforward networks is discouraged [11]. To overcome such weakness and improve accuracy results, an active area of research has been devoted to design novel activation functions. The training procedure of these architectures usually involve optimization of the weights of their layers only, while non-linearity's are generally pre-specified and their (possible) parameters are usually considered as hyper-parameters to be tuned manually. In this paper, we introduce few approaches able to automatically learn combinations of base activation functions (such as: Relu, Elu, Softmax, Sigmoid, Exponential, Tanh) and model using various Optimizers (Adagrad, RmsProp, Adam, Adamax, SGD), Loss functions (mean squared error, binary cross entropy during training; our aim is to identify a search space for the activation function and the optimizers by means of convex combination or affine combination of the base functions. Here we tested different well-known networks employing customary activation functions and Optimizers with loss function on three standard datasets and we compared their results with those obtained applying our novel approach (Improved Bi-LSTM). The techniques proposed in this paper outperformed the baselines in all the experiments, even when using deep architectures.

B. Optimization in Deep Learning

The Gradient Descent [12][13][14] technique is one of the most popular algorithms to perform optimization for the neural networks. Here, we need to find the best parameters for our learning algorithm and its corresponding performance of the algorithm for different values of the parameters. The Gradient descent is having various variants such as Batch, Stochastic and mini Batch. The Batch Gradient descent uses the entire training data set and computes the gradient of the cost function with respect to the parameters. The Stochastic Gradient descent is the one which performs the parameter update for every training example. And the mini Batch takes the best of above two strategies by performing the parameter update for every mini batch of training examples. Initially the deep learning has used the basic gradient descent algorithms with which the models used to run with very less performance. In this paper, it is proposed to use few gradient descent optimization algorithms such as Adagrad, RmsProp and ADAM for both convolutional neural network and recurrent neural network models.

Optimizers are the expanded class, which includes the method to train your deep learning model. Right optimizers are necessary for our model as they improve training speed and performance, now there are many optimizers algorithms we have in Tensor Flow library but here we will be discussing how to initiate Tensor Flow Keras optimizers.

Before optimizers, it's good to have some preliminary exposure in loss functions as both works parallel in deep learning projects. Loss functions are just a mathematical way of measuring how good your machine/deep learning model performs.

During the training of the model, we tune the parameters and weights to minimize the loss and try to make our prediction accuracy as correct as possible. Now to change these parameters the optimizer's role came in, which ties the model parameters with the loss function by updating the model in response to the loss function output. Simply optimizers shape the model into its most accurate form by playing with model weights. The loss function just tells the optimizer when it's moving in the right or wrong direction. Optimizers are Classes or methods used to change the attributes of our deep learning model such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster.

C. Nonlinearity Through Activation

A lot of theory and mathematical machines behind the classical ML (regression, support vector machines, etc.) were developed with linear models in mind. However, practical real-life problems are often nonlinear in nature and therefore cannot be effectively solved using those ML methods. Deep learning models are inherently better to tackle such nonlinear classification tasks.

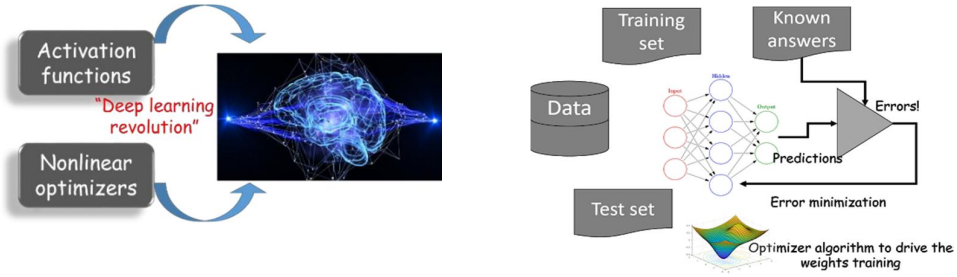

Fig. 1: Activation function and Optimizer algorithm in deep learning

However, at its core, structurally, a deep learning model consists of stacked layers of linear perceptron units and simple matrix multiplications are performed over them. Matrix operations are essentially linear multiplication and addition. It can also be shown, mathematically, that the universal approximation power of a deep neural network, the ability to approximate any mathematical function to sufficient degree and it does not hold without these nonlinear activation stages in between the layers.

D. Reducing Errors with Optimizers

Practically, we need to compare the predictions to the ground truth and turning the parameters of the model. The difference between the prediction and the ground truth is called the classification error. So, the two components, activation functions and nonlinear optimizers are at the core of every deep learning architecture. However, there is considerable variety in the specifics of these components and here we go over the latest developments.

In artificial neural networks, we extend this idea by shaping the outputs of neurons with activation functions. They push the output signal strength up or down in a nonlinear fashion depending on the magnitude. High magnitude signals propagate further and take part in shaping the final prediction of the network whereas the weakened signal die off quickly. ere we have discussed some common activation functions.

Sigmoid (Logistic)

$$f(x) = \frac{1}{1+e^{-x}}$$
$$f'(x) = \frac{\partial f(x)}{\partial x} = f(x) * (1 - f(x)) \quad . (1)$$

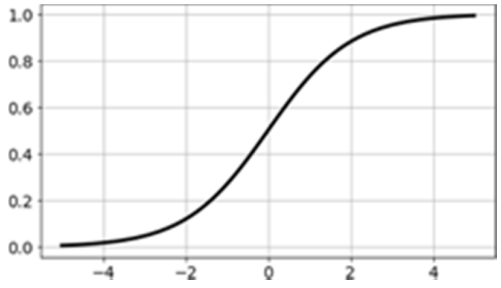

Fig. 2: Sigmoid function graph

In the logistic function, a small change in the input only causes a small change in the output as opposed to the stepped output. Hence, the output is smoother than the step function output. While, sigmoid functions were one of the first ones used in early neural network research, Other functions have been shown to produce the same performance with less iterations. This is because of the output range of [0,1] which can be interpreted as probability values.

Tanh (Hyperbolic Tangent)

Tanh is a non-linear activation function that compresses all its inputs to the range [-1, 1]. The mathematical representation is given below,

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$f'(x) = \frac{\partial f(x)}{\partial x} = (1 - (f(x))^2) \quad (2)$$

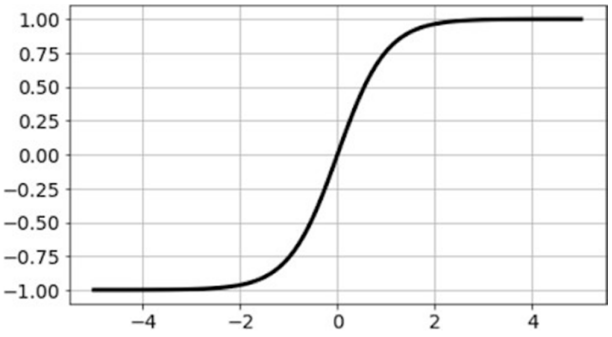

Fig. 3: Tanh function graph

ReLU(Rectified Linear Unit)

It is a non-linear activation function which was first popularized in the context of a convolution neural network (CNN). If the input is positive then the function would output the value itself, if the input is negative the output would be zero.In the process of ReLu function evaluation is computationally efficient as it does not involve computing exp(x) and therefore, in practice, it converges much faster than logistic/Tanh for the same performance (classification accuracy for example). For this reason, ReLU has become de-facto standard for large convolutional neural network architectures such as Inception, ResNet, MobileNet, VGGNet, etc.In ReLU, instead of producing zero for negative inputs, it will just produce a very small value proportional to the input i.e 0.01 , as if the function is 'leaking' some value in the negative region instead of producing hard zero values.

$$f(x) = max(0.01x, x)$$
$$f'(x) = \frac{\partial f(x)}{\partial x} = \begin{cases} 0.01 & if \quad x < 0 \\ 1 & if \quad x > 0 \end{cases} \quad (3)$$

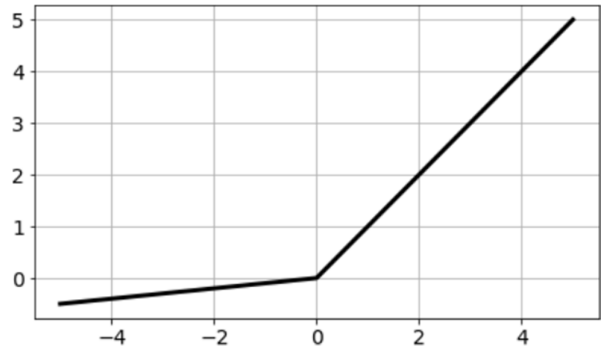

Fig. 4: ReLU function graph

Learning Rate

Learning rate controls how much we should adjust the weights with respect to the loss gradient. Learning rates are randomly initialized. Lower the value of the learning rate, slower will be the convergence to global minima. A higher value for learning rate will not allow the gradient descent to converge.

Basically, the update equation for weight optimization is,

$$w := w - \alpha \frac{\partial c}{\partial \omega}$$ (4)

Here, α is the learning rate, C is the cost function and w and ω are the weight vectors. We update the weights proportional to the negative of the gradient (scaled by the learning rate).

Gradient descent optimization algorithms

In the following, we will figure out some popular algorithms that are widely used by the Deep Learning community to deal with the various challenges.

Adam Optimizer

Adam stands for Adaptive moment estimation, which is another way of using past gradients to calculate current gradients. Adam utilizes the concept of momentum by adding fractions of previous gradients to the current one, it is practically accepted in many projects during training neural nets.

RMSprop Optimizer

It is an exclusive version of **Adagrad** developed by Geoffrey Hinton

(learn more), now the thinking behind this optimizer was pretty straight forward: instead of letting all of the gradients accumulate for momentum, it only accumulates gradients in a specific fix window. It is exactly like **Adaprop** which is an updated version of Adagrad with some improvement.

Adammax

As the name suggests AdaMax is an adaption of **Adam optimizer,** by the same researchers who wrote the Adam algorithm. It is a variant of Adam based on the infinity norm. Sometimes it is considered superior to Adam, especially in models with embedding.

SGD Optimizer

Stochastic gradient descent(SGD) optimization algorithm in contrast performs a parameter update for each training example.SGD performs redundant computations for bigger datasets, as it recomputes gradients for the same example before each parameter update. It performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

AdaGrad Optimizer

Adagrad adapts the learning rate specifically with individual features: it means that some of the weights in our dataset have different learning rates than others. It always works best in a sparse dataset where a lot of inputs are missing. It is a parameter specific learning rate, adapts with how frequently a parameter gets updated during training. Parameters we pass with these optimizers are learning_rate, initial_accumulator_value, epsilon, name.

E. Loss functions

It is proposed to use the popular Loss functions [16] such as mean squared error and binary cross entropy along with the above optimizers. The mean squared error computation is done in the following manner. For the Input layer, the Inputs or attributes of the first training example is taken, multiply with their weights, apply the activation function and get the output for the hidden layer. Again the same process is applied on this hidden layer neuron inputs and the final output is computed. Now the difference between the final output and the Actual value (class label of data set) is computed which is termed as the Error. Now, the Error is computed for all the training examples, square all the errors, and calculate the mean of the total error. This is the mean squared error with which the initial weights are updated and again the iterations start from the input layer till the error converges to a minimum threshold. Let us suppose that xi be the vector denoting n number of prediction values. Also, xi be a vector denoting n number of true values, then the mean squared error MSE is given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{x_i} - x_i)^2$$

The other standard loss function is binary cross entropy. The binary cross entropy computes the difference between true and predicted probability distributions. The final output is calculated in the same way as it is done for the mean squared error and this it is termed as predicted distribution and the class label of the data set is termed as true distribution. Here, the partial derivatives are computed over the predicted and true distributions and the error is computed with the difference of both the distributions.

$$J = \frac{i}{m}\sum_{i=1}^{m} y_i log\hat{y}_i + (1 - y_i)\log(1 - \hat{y_i})$$

When the activation function is sigmoid

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

Then the loss function is given by

$$J = \frac{1}{n}\sum_x x_j(\sigma(z) - y)$$

This paper is organized as follows: in Section II the related works are summarized; in Section III the experimental results are presented; finally, conclusions and future works are summarized in Section IV.

II. Related Work

Our focus in this section of the related work was based on the standard activation functions and different variations of the optimizers with loss functions that has recently been proposed.

S.V.G. Reddy et. al. [6] proposed to build a Deep Learning model using various Optimizers (Adagrad, RmsProp, Adam), Loss functions (mean squared error, binary cross entropy) and Dropout concept for the Convolutional neural networks and Recurrent neural networks and verify the performance such as Accuracy and Loss of the model. The proposed model has achieved maximum Accuracy when Adam optimizer and mean squared error loss function are applied on convolutional neural networks and the model is run with minimum Loss when the same Adam optimizer and mean squared error loss function are applied on Recurrent neural networks.

Sebastian Ruder et. al. [2] investigated algorithms that are most commonly used for optimizing SGD: Momentum, Nesterov accelerated gradient, Adagrad, Adadelta, RMSprop, Adam, AdaMax, Nadam, as well as different algorithms to optimize asynchronous SGD.Finally, we've considered other strategies to improve SGD such as shuffling and curriculum learning,batch normalization, and early stopping, and Introduce the most common optimization algorithms, review architectures in a parallel and distributed setting, and investigate additional strategies for optimizing gradient descent.

Franco Manessi et. al. [3] introduced two approachesto automatically learn different combinations of base activation functions (such as the identity function, ReLU, and tanh) during the training phase. We present a thorough comparison of our novel approaches with well-known architectures (such as LeNet-5, AlexNet, and ResNet-56) on three standard datasets (Fashion-MNIST, CIFAR-10, and ILSVRC-2012), showing substantial mprovements in the overall performance, such as an increase in the top-1 accuracy for AlexNet on ILSVRC-2012 of 3.01 percentage points.

Vladimir Kunc et. al. [4] proposed an improved neural network achieves an average mean absolute error of 0.1340, which is a significant improvement over our reimplementation of the original D–GEX, which achieves an average mean absolute error of 0.1637. The proposed transformative adaptive function enables a significantly more accurate reconstruction of the full gene expression profiles with only a small increase in the complexity of the model and its training procedure compared to other methods.

Yongbin Yu et. al. [5] used activation function called ReLU-Memristor-like Activation Function (RMAF) is proposed to leverage benefits of negative values in neural networks. RMAF introduces a constant parameter and a thresholdparameter

(p) making the function smooth, non-monotonous, and introduces non-linearity in the network. Our experiments show that, the RMAF works better than ReLU and other activation functions on deeper models and across number of challenging datasets.

Designing activation functions that enable fast training of accurate deep neural networks is an active area of research. The rectified linear activation function, introduced in [15] and argued in [16] to be better biological model than logistic sigmoid activation function, has eased the training of deep neural networks by alleviating the problems related to weight initialization and vanishing gradient. Slight variations of ReLU have been proposed over the years, such as leaky ReLU (LReLU), [17], which addresses dead neuron issues in ReLU networks, thresholder ReLU [18], which tackles the problem of large negative biases in auto encoders, and parametric ReLU (PReLU) [19], which treats the leakage parameter of LReLU as a per-filter learnable weight.

Clevert et al., [20]. Proposed that ELU becomes smooth slowly until its output equals to a constant negative value unlike ReLU which sharply becomes smooth and it blows activation which leads to gradient exploding problem for a given positive inputs.

Kalash, M et al.,[21] proposed for training deep networks where the function has breakthrough against the problem termed as vanishing gradient which is in the case of sigmoid and tanh.The ReLU is an identity function for the non-negative inputs and zero function for the negative inputs.

V. Nair and G. E. Hinton [22] noted that, the better solution to gradient vanishing from sigmoid and tanh is to maintain the resultant values to 1 to prevent changes when they are multiplied. This is generally the work of ReLU, which has a gradient of 1 for positive inputs and 0 for negative inputs.

However, all the previous learning techniques are limited by the fact that the family of functions over which the learning takes place is either finite or a simple parameterization of customary activation functions.

III. Experimental Results

Here we have taken three standard datasets (Amazon dataset, IMDB data Set, Yelp dataset) for analysing various optimizers and activation functions with loss functions. Initially we have fixed the number of epochs to 150 and input data size as 150, then recorded the corresponding accuracy of the standard dataset for various optimizers and activation functions. Gradually we increased the input size as 250 and number of epochs to 20, our results are slightly. Further we tried to increase the number of epochs, but gradually the accuracy id reducing. So the best result in accuracy is built at 3o epochs and 250 as input size. Out of all the optimizers it is shown here that Adam with Relu activation function and Binary Cross entropy loss Function has given the best performance. The following tables have shown the accuracy with various optimizers and activation functions and with different loss functions.

Here Table I shows the accuracy with 15 number of epochs and input data size as 150 along with the binary cross-entropy loss function for the Amazon dataset. Similarly, Fig 5 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE I: ACCURACY WITH EPOCHS:15, INPUT SIZE-150 AND BINARY CROSS-ENTROPY LOSS FUNCTION (AMAZON DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	96	91	74	56	44
Elu()	92	89	69	56	43
Softmax()	85	88	45	35	32
Exponential()	88	83	68	66	45
Tanh()	86	84	56	34	43

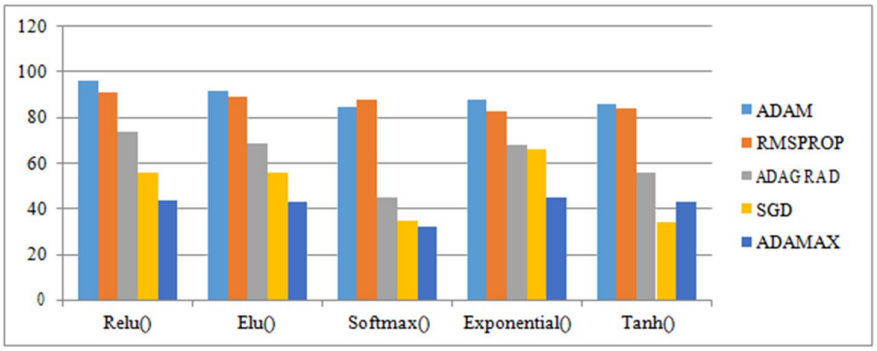

Fig. 5: epochs:15, input size-150 with binary cross-entropy loss function (Amazon dataset).

Here Table II shows the accuracy with 20 number of epochs and input data size as 250 along with the binary cross-entropy loss function for the Amazon dataset. Similarly, Fig 6 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE II: ACCURACY WITH EPOCHS:20, INPUT SIZE-250 AND BINARY CROSS-ENTROPY LOSS FUNCTION (AMAZON DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	97	90	77	69	45
Elu()	93	89	69	56	43
Softmax()	84	88	45	40	32
Exponential()	88	83	68	66	45
Tanh()	86	84	56	32	44

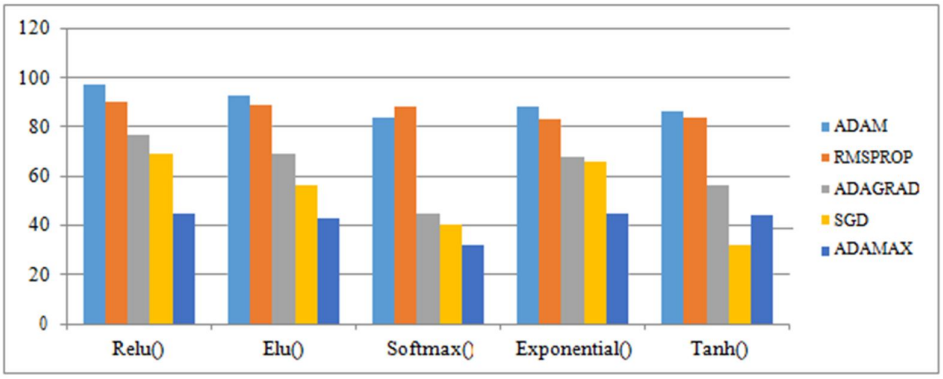

Fig. 6: epochs:20 , input size-250 with binary cross-entropy loss function (Amazon dataset)

Here Table III shows the accuracy with 15 number of epochs and input data size as 150 along with the Binary cross-entropy loss function for the Amazon dataset. Similarly, Fig 7 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE III: ACCURACY WITH EPOCHS:15, INPUT SIZE-150 AND BINARY CROSS-ENTROPY LOSS FUNCTION (YELP DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	78	72	56	50	56
Elu()	70	67	56	55	47
Softmax()	62	88	50	42	31
Exponential()	63	83	70	63	44
Tanh()	65	84	45	35	44

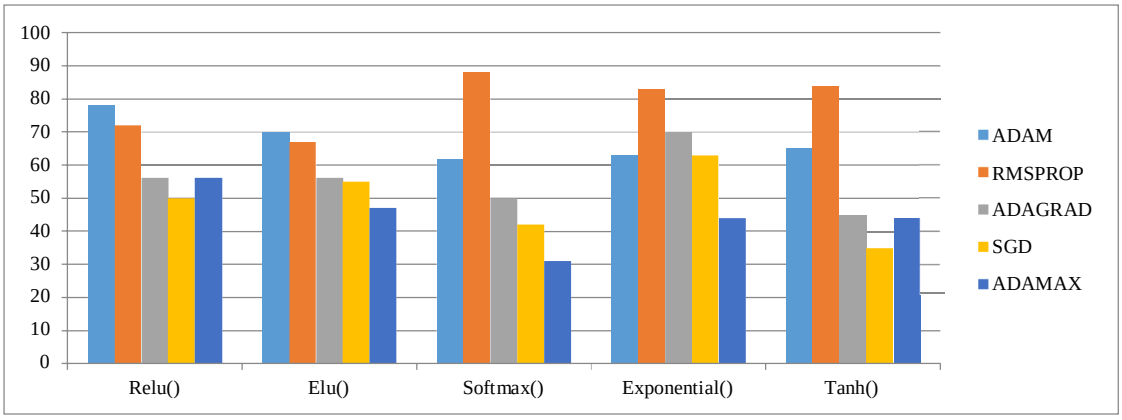

Fig. 7 : EPOCHS:20 , Input Size-250 with Binary Cross-entropy Loss Function (Yelp Dataset)

Here Table IV shows the accuracy with 20 number of epochs and input data size as 250 along with the binary cross-entropy loss function for the Amazon dataset. Similarly, Fig 8 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE IV: ACCURACY WITH EPOCHS:20, INPUT SIZE-250 AND BINARY CROSS-ENTROPY LOSS FUNCTION (YELP DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	89	77	69	67	56
Elu()	67	65	55	56	50
Softmax()	66	76	56	44	35
Exponential()	66	71	67	63	47
Tanh()	65	84	48	46	44

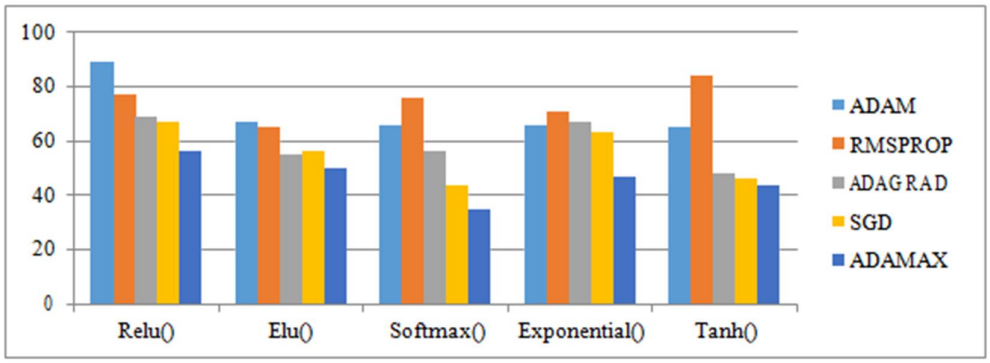

Fig. 8: epochs:20 , input size-250 with binary cross-entropy loss function (Yelp dataset)

Here Table V shows the accuracy with 15 number of epochs and input data size as 150 along with the binary cross-entropy loss function for the Amazon dataset. Similarly, Fig 9 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE V: ACCURACY WITH EPOCHS:15, INPUT SIZE-150 AND BINARY CROSS-ENTROPY LOSS FUNCTION (IMDB DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	85	78	67	69	55
Elu()	68	67	56	56	45
Softmax()	67	78	59	50	37
Exponential()	66	77	69	63	44
Tanh()	68	70	49	49	45

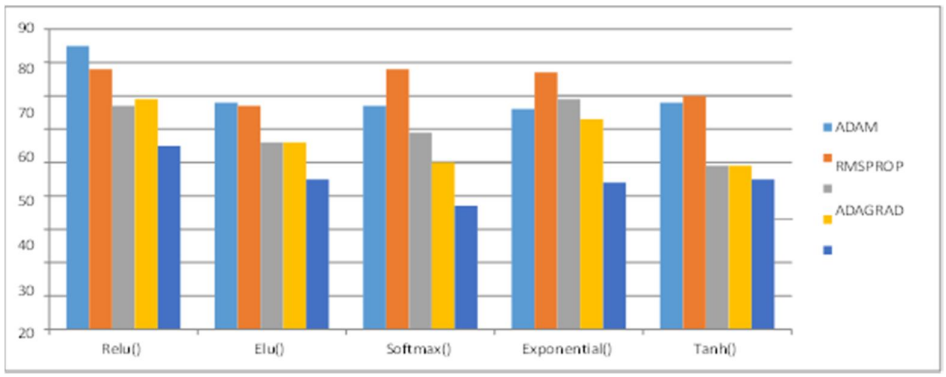

Fig. 9: epochs:15, Input Size-150 with Binary Cross-entropy Loss Function (IMDB Dataset)

Here Table VI shows the accuracy with 20 number of epochs and input data size as 250 along with the binary cross-entropy loss function for the Amazon dataset. Similarly, Fig 10 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE VI: ACCURACY WITH EPOCHS:20, INPUT SIZE-250 AND BINARY CROSS-ENTROPY LOSS FUNCTION (IMDB DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	85.7	76	65	66	60
Elu()	67	68	59	55	48
Softmax()	65	77	58	56	45
Exponential()	64	73	66	63	44
Tanh()	69	72	46	45	45

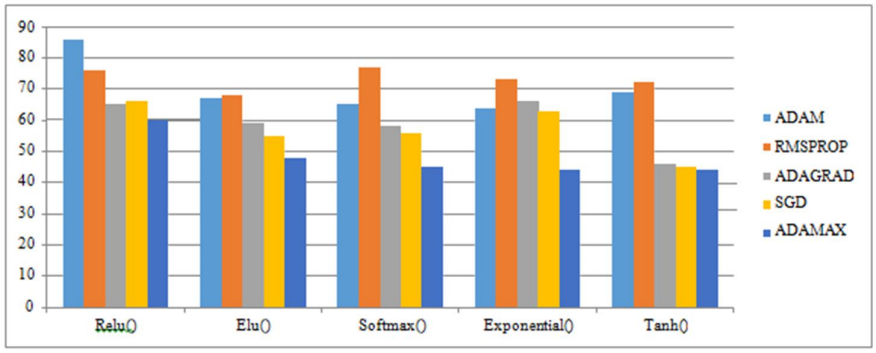

Fig. 9: epochs:20, Input Size-250 with Binary Cross-entropy (IMDB Dataset)

Here Table VII shows the accuracy with 15 number of epochs and input data size as 150 along with the Mean squared error loss function for the Amazon dataset. Similarly, Fig 11 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE VII: ACCURACY WITH EPOCHS:15, INPUT SIZE-150 AND MEAN SQUARED ERROR LOSS FUNCTION (AMAZON DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	90	75	63	66	60
Elu()	84	68	59	55	48
Softmax()	80	77	58	56	45
Exponential()	86	73	66	63	44
Tanh()	83	72	35	44	45

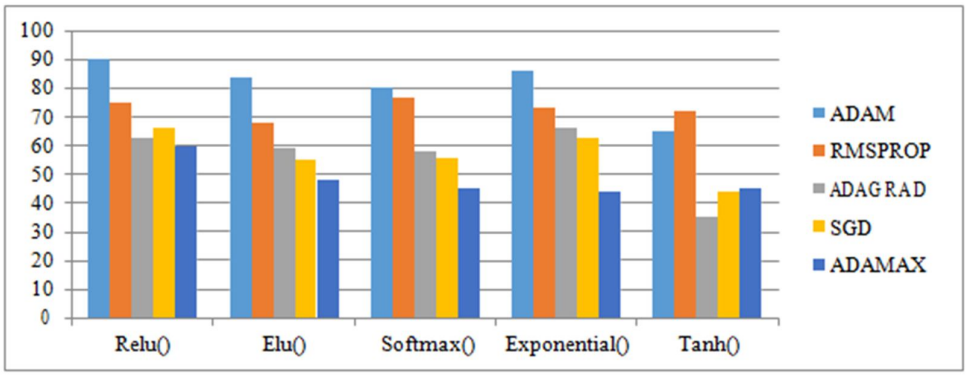

Fig. 11: : epochs:15, Input Size-150 with Mean squared Error Loss Function (Amazon Dataset)

Here Table VIII shows the accuracy with 20 number of epochs and input data size as 250 along with the Mean squared error loss function for the Amazon dataset. Similarly, Fig 12 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE VIII: ACCURACY WITH EPOCHS:20, INPUT SIZE-250 AND MEAN SQUARED ERROR LOSS FUNCTION (AMAZON DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	81	72	56	60	63
Elu()	82	64	57	55	48
Softmax()	80	77	58	56	45
Exponential()	86	73	66	63	44
Tanh()	83	72	35	44	45

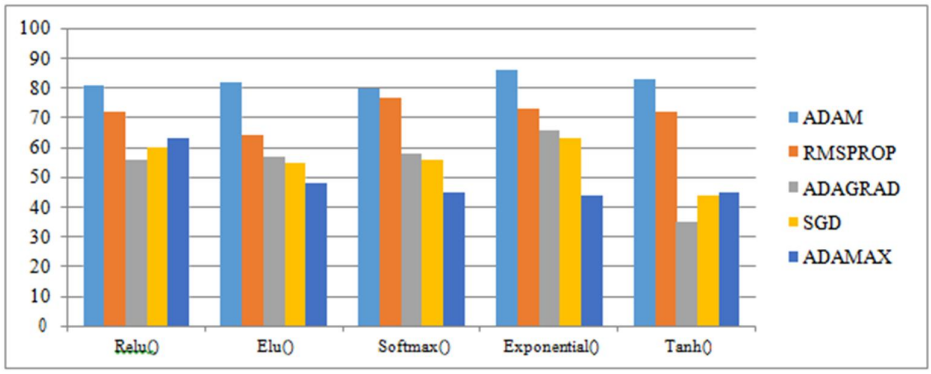

Fig. 12: epochs:20, Input Size-250 with Mean squared Error Loss Function (Amazon Dataset)

Here Table IX shows the accuracy with 15 number of epochs and input data size as 150 along with the Mean squared error loss function for the Amazon dataset. Similarly, Fig 13 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE IX: ACCURACY WITH EPOCHS:15, INPUT SIZE-150 AND MEAN SQUARED ERROR LOSS FUNCTION (YELP DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	70	68	55	46	53
Elu()	69	65	54	53	48
Softmax()	63	82	48	42	35
Exponential()	63	83	70	63	44
Tanh()	65	84	45	35	44

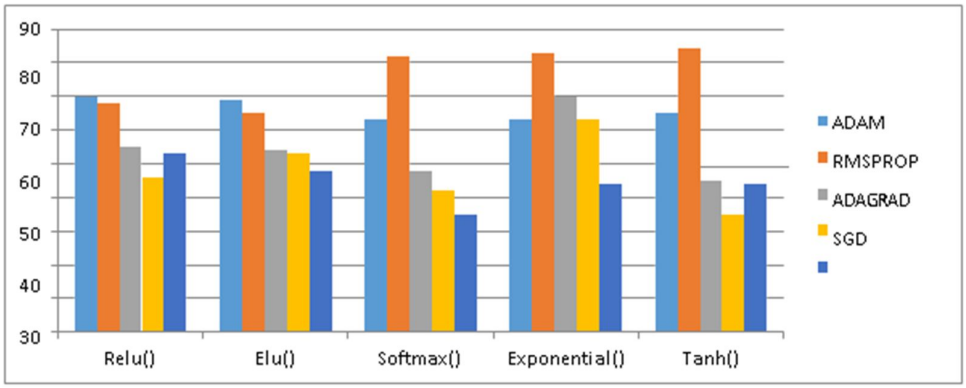

Fig. 13: epochs, Input Size-150 with Mean squared Error Loss Function (Yelp Dataset)

Here Table X shows the accuracy with 20 number of epochs and input data size as 250 along with the Mean squared error loss function for the Amazon dataset. Similarly, Fig 14 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE X: ACCURACY WITH EPOCHS:20, INPUT SIZE-250 AND MEAN SQUARED ERROR LOSS FUNCTION (YELP DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	65	68	56	44	35
Elu()	66	64	55	49	46
Softmax()	63	82	48	42	35
Exponential()	63	83	70	63	44
Tanh()	65	84	45	35	44

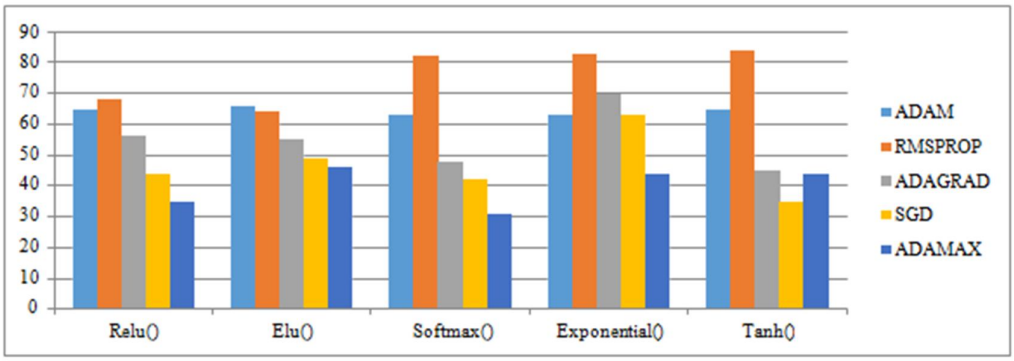

Fig. 14: EPOCHS:20, Input Size-250 with Mean squared Error Loss Function (Yelp Dataset)

Here Table XI shows the accuracy with 15 number of epochs and input data size as 150 along with the Mean Squared error loss function for the Amazon dataset. Similarly, Fig 15 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE XI: ACCURACY WITH EPOCHS:15, INPUT SIZE-150 AND MEAN SQUARED ERROR LOSS FUNCTION (IMDB DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	83	70	63	62	36
Elu()	63	62	51	54	50
Softmax()	68	72	55	53	44
Exponential()	68	71	68	66	39
Tanh()	69	72	46	45	45

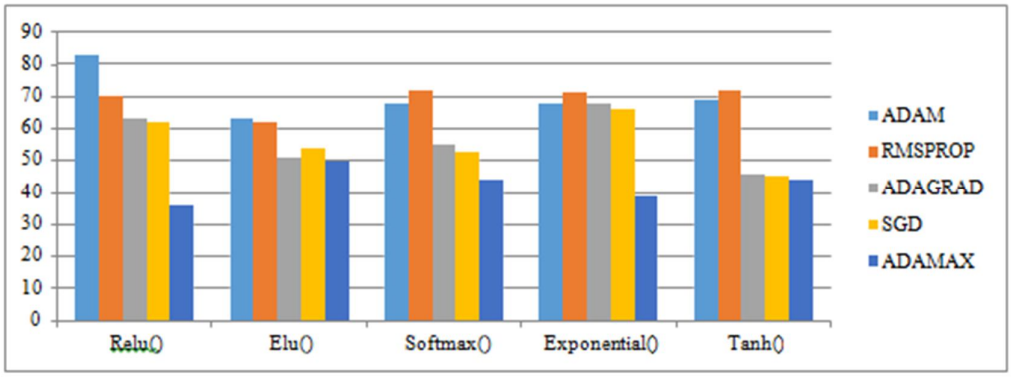

Fig. 15: epochs:15, Input Size-150 with Mean Squared error Loss Function (IMDB Dataset)

Here Table XII shows the accuracy with 20 number of epochs and input data size as 250 along with the Mean Squared error loss function for the Amazon dataset. Similarly, Fig 16 displays the accuracy comparison of all the optimizers performance in terms of percentage.

TABLE XII: ACCURACY WITH EPOCHS:20, INPUT SIZE-250 AND MEAN SQUARED ERROR LOSS FUNCTION (IMDB DATASET)

	Optimizers				
Activation function	ADAM	RMSPROP	ADAGRAD	SGD	ADAMAX
Relu()	84.2	71	60	61	52
Elu()	66	64	55	56	48
Softmax()	65	77	58	56	45
Exponential()	64	73	66	63	44
Tanh()	63	73	36	49	43

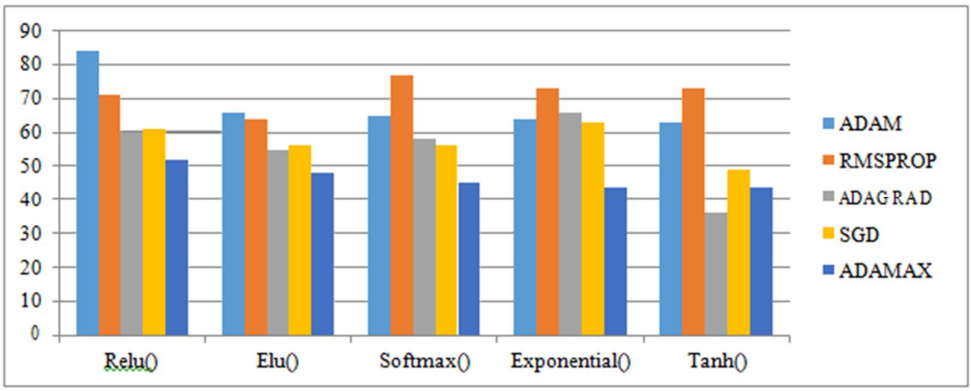

Fig. 16: epochs:20, Input Size-250 with Mean Squared error Loss Function (IMDB Dataset)

V. Conclusion

In this article, we have initially looked at the many activation functions which is most popular. We have then investigated algorithms that are most commonly used for optimizing SGD, Adagrad, RMSprop, Adam, AdaMax, as well as different loss functions. finally, in all our experiments, the Adam optimizer with mean Squared error loss function discussed in this paper achieved the best performance and the combined activation functions learned using our approaches usually outperform the corresponding base components. The effectiveness of the proposed techniques is further proved by the increase in performance achieved using networks with different depths and architectures.

Acknowledgement

I am very much thankful to my Supervisor Dr. Asha T, for her guidance and constant monitoring of my PhD work and heartfelt gratitude to my parents and the Almighty.

References

[1] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, And A. Torralba, "Learning Deep Features for Discriminative Localization," In Proceedings of The IEEE Conference On Computer Vision and Pattern Recognition, 2016, Pp. 2921–2929.
[2] Sebastian Ruder, "An Overview Of Gradient Descent Algorithms", Cornell University Library, Arxiv: 1609.04747[Cs. Lg] .
[3] Franco Manessi "Learning Combinations of Activation Functions." 2018 , 24th International Conference on Pattern Recognition (ICPR), Volume-1 Issue-1, Beijing, 2018, pp. 61-66.
[4] Vladimı´r KuncOn "Transformative adaptive activation functions in neural networks for gene expression inference" in the article PLOS ONE | https://doi.org/10.1371/journal.pone.0243915 ,January 14, 2021.
[5] YONGBIN YU, "RMAF: ReLU-Memristor-like Activation Function for Deep Learning" in IEEE Access, volume 4 issue-1, 2016.
[6] S.V.G.Reddy, "Optimization of Deep Learning using various Optimizers, Loss functions and Dropout" in International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-7 Issue-4S2, December 2018.
[7] J. Tang, C. Deng, and G.B. Huang," Extreme Learning Machine for multilayer Perceptron," IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no-4, pp. 809-821,2016.
[8] L.-W. Kim, "Deepx: Deep learning accelerator for restricted Boltzmann machine artificial neural networks," IEEE transactions on neural networks and learning systems, vol. 29, no. 5, pp. 1441–1453, 2018.
[9] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE transactions on neural networks, vol. 5, no. 2, pp. 157–166, 1994.
[10] 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.
[11] AsmelashTekaHadgu; Aastha Nigam ; Ernesto Diaz-Aviles, "Large-scale learning with Adagrad on Spark, 2015 IEEE International Conference on Big Data (Big Data), DOI: 10.1109/BigData.2015.7364091.
[12] Zijun Zhang, "Improved Adam Optimizer for Deep Neural Networks", 978-1-5386-2542-2/18/ ©2018 IEEE .

[13] AnirbanSarkar, AdityaChattopadhyay, PrantikHowlader, V. Balasubramanian, Grad-Cam++: “Generalized Gradient-Based Visual Explanations For Deep Convolutional Networks”, Proceedings Of Ieee Winter Conference On Applications Of Computer Vision (Wacv'18), Mar 2018. [Arxiv]
[14] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, And D. Batra, “Grad-Cam: Why Did You Say That? Visual Explanations From Deep Networks Via Gradient-Based Localization,” ArxivPreprint Arxiv:1610.02391 , 2016.
[15] K. Jarrett, K. Kavukcuoglu, Y. LeCun et al., “What is the best multistage architecture for object recognition?” in Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009, pp. 2146–2153.
[16] 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.
[17] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” in ICML Workshop on Deep Learning for Audio, Speech and Language Processing, 2013.
[18] K. Konda, R. Memisevic, and D. Krueger, “Zero-bias autoencoders and the benefits of co-adapting features,” arXiv preprint arXiv:1402.3337,2014.
[19] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,”in Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), ser. ICCV ’15. Washington, DC, USA:IEEE Computer Society, 2015, pp. 1026-1034.
[20] 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.
[21] Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D. B., Wang, Y., and Iqbal, F. (2018). Malware Classification with Deep Convolutional Neural Networks. In 2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018
[22] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.
[23] A. Shukla, R. Tiwari, and P. Kaur, —Knowledge Based Approach for Diagnosis of Breast Cancer, in 2009 IEEE International Advance Computing Conference, 2009, pp. 6–12.
[24] M. Ewertz et al.,—Effect of obesity on prognosis after early-stage breast cancer., J. Clin. Oncol., vol. 29, no. 1, pp. 25–31, 2011.
[25] Nhs Choices, —Breast Cancer Symptoms, WebMD, pp. 1–5, 2010.
[26] L. Zhang and P. N. Suganthan, “Random forests with ensemble of feature spaces,” Pattern Recognition, vol. 47, no. 10, pp. 3429–3437, 2014.
[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. pp. 2278–2324, 1998.
[28] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, —[3] Strukov,"The missing memristor found", Nature, vol. 453, no. 7191, pp.80–3, 2008.
[29] P. S. Georgiou, S. N. Yaliraki, E. M. Drakakis, and M. Barahona,—Window functions and sigmoidal behaviour of memristive systems.pp. 1–12.
[30] M. E. Turner, E. L. Bradley, K. A. Kirk, and K. M. Pruitt, —A theory of growth Math. Biosci., vol. 29, no. 3–4, pp. 367–373, 1976.