http://dx.doi.org/10.5755/j02.eie.34105 ELEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392-1215, VOL. 29, NO. 3, 2023

Modified Backpropagation Algorithm with Multiplicative Calculus in Neural Networks

Serkan Ozbay
Department of Electrical and Electronics Engineering, Gaziantep University, 27310 Gaziantep, Turkey
sozbay@gantep.edu.tr

***Abstract*—Backpropagation (BP) is one of the most widely used algorithms for training feedforward deep neural networks (NNs). The algorithm requires a differentiable activation function and it performs computations of the gradient proceeding backward through the feedforward deep NN from the last layer through to the first layer. To calculate the gradient at a specific layer, the gradients of all layers are combined using the chain rule of calculus. One of the biggest disadvantages of BP is that it requires a large amount of training time. To overcome this issue, this paper proposes a modified BP algorithm with multiplicative calculus. Multiplicative calculus provides an alternative to the classical calculus, and it defines new kinds of derivative and integral forms in multiplicative form rather than addition and subtraction forms. The performance analyses are discussed in various case studies, and the results are given comparatively with the classical BP algorithm. It is found that the proposed modified BP algorithm converges in less time to the solution and thus provides fast training in the given case studies. It is also shown that the proposed algorithm avoids the problem of local minima.**

***Index Terms*—Backpropagation; Local minima problem; Multiplicative calculus; Neural networks.**

I. Introduction

The backpropagation (BP) algorithm was first proposed by Werbos in his doctoral dissertation in 1974 [1]. The thesis introduced the algorithm in the general context so that it was not widely known and publicised in the neural network (NN) community until the mid 1980s. The algorithm was then independently presented by Rumelhart, Hinton and Williams [2], Parker [3], and Lecun [4], but it became popularised by Rumelhart and McClelland [5]. After the publication of this study, there was a huge explosion of scientific work in the field of NNs. Most multilayer NNs have been trained by the BP algorithm.

Although the BP algorithm has numerous advantages in many successful applications, there have been some drawbacks such as the BP needs a lot of training time and therefore the convergence tends to be significantly slow [6] and the learning process is stuck at a local minimum [7]. This study addresses a novel solution to overcome the slow convergence rate problem and avoid local minima problem, and this paper introduces a modified BP algorithm with multiplicative calculus. To my knowledge, this is the first study on the use and application of multiplicative calculus in this field.

Manuscript received 15 February, 2023; accepted 10 May, 2023.

The proposed algorithm uses the multiplicative derivative of the activation function in the NN architecture. The proposed algorithm is the modified BP algorithm that still performs a backward pass while adjusting the learnable parameters (weights and biases). The difference from classical one is the use of multiplicative form of the derivative in the computations and also the use of new derivative form on only the last layer not in the hidden layers.

Some linearly inseparable problems are studied through shallow NN models with the proposed algorithm, the performance of the algorithm is analysed in different case studies, and the obtained results are provided comparatively with the classical BP algorithm. It is shown that the proposed modified BP algorithm converges quickly to the optimal solution compared to the classical one. The initial values of the adjustable parameters are changed and the potential local minima convergence is created for the classical BP algorithm, and then the proposed algorithm is implemented through the same network and it is indicated that the proposed algorithm avoids the local minima problem.

The rest of the paper is presented as follows. Section II gives an overview of the classical BP algorithm. In Section III, the multiplicative calculus description is introduced. Section IV describes the proposed algorithm, and the performance analyses are discussed in Section V. Finally, Section VI provides the conclusions of the paper.

II. An Overview of Backpropagation

In feedforward NNs, when one of the inputs x in the training data set comes to produce an output y, the initial information flows to the hidden units in each layer, and this process is called “forward propagation”. In training NNs, forward propagation lasts until it produces a cost (or error). The backpropagation (BP) algorithm simply enables the information flow from the cost and then propagates back through the network to compute the gradient [8]. The BP algorithm to train the feedforward NN is summarised in Algorithm 1 [9].

As illustrated, the BP algorithm requires differentiable activation functions to carry out gradient computations. To implement the computations of the gradient at a specific layer, the gradients of all layers are merged via the chain rule of calculus.

Algorithm 1. Standard BP algorithm framework.

```
Given the training data set
$\{\boldsymbol{inputs}, \boldsymbol{target\ values}\} = \{\boldsymbol{x's}, \boldsymbol{t's}\}$,
1. Initialise the weights, including biases.
2. Take one input from the training data set $\{\boldsymbol{x}, \boldsymbol{t}\}$ and obtain the network output, y.
3. Calculate the error, e, of the network output to the target and the delta, $\boldsymbol{\delta}$, of the output nodes.
    $\boldsymbol{e} = \boldsymbol{t} - \boldsymbol{y}$ and $\boldsymbol{\delta} = \boldsymbol{\varphi}'(\boldsymbol{v})\boldsymbol{e}$
    Note that v represents the weighted sum, $\boldsymbol{\varphi}$ represents the activation function, and $\boldsymbol{\varphi}'(\boldsymbol{v})$ represents the first derivative of the output $\boldsymbol{y} = \boldsymbol{\varphi}(\boldsymbol{v})$.
4. Propagate the output node delta, $\boldsymbol{\delta}$, in the backward direction and then calculate the deltas of the consecutive left nodes.
    $\boldsymbol{e}^{(k)} = \boldsymbol{W}^T\boldsymbol{\delta}$ and $\boldsymbol{\delta}^{(k)} = \boldsymbol{\varphi}'(\boldsymbol{v}^{(k)})\boldsymbol{e}^{(k)}$
5. Repeat the previous step up to the layer which is on the immediate right of the input layer.
6. Update the adjustable parameters according to the following rule.
    $\Delta \boldsymbol{w}_{ij} = \boldsymbol{\alpha}\boldsymbol{\delta}_i \boldsymbol{x}_j$ and $\boldsymbol{w}_{ij}(\boldsymbol{new}) = \boldsymbol{w}_{ij}(\boldsymbol{old}) + \Delta \boldsymbol{w}_{ij}$
    Note that α represents the learning rate.
7. Repeat the stages explained in the previous steps for each training data point.
8. Continue updates until the network parameters are adjusted accordingly and the NN is properly trained.
```

III. MULTIPLICATIVE CALCULUS

There are two main tools in calculus: the derivative with differentiation operation and the integral with integration operation. The process of finding a derivative is called "differentiation" and the process of finding an integral is called "integration". Those two operations are fundamental and essential operations of calculus. The differentiation and integration are infinitesimal versions of the subtraction operation and addition operation on numbers, respectively. The close connection between the derivative and the integral was first observed independently by Newton and Leibniz in the second half of the 17th century [10].

It is known that many physical quantities in nature are of an exponentially varying type. As an alternative to classical calculus, multiplicative calculus was proposed for exponentially varying functions [11]. Multiplicative calculus introduces new kinds of derivative and integral forms in division and multiplication forms rather than addition and subtraction. This alternative calculus was extensively described in [12]–[14], with use on real-valued functions [15], and its extension to complex-valued functions [11], [16].

The multiplicative derivative of a function $f(x) \in \mathbb{R}^+$ symbolised by $\frac{d^*}{dx^*} f(x)$ or $f^*(x)$ is given by (1)

$$\frac{d^*}{dx^*} f(x) = f^*(x) = \lim_{h \to 0} \left(\frac{f(x+h)}{f(x)} \right)^{1/h}. \tag{1}$$

Similarly, the multiplicative integral of a function $f(x) \in \mathbb{R}^+$ represented by $\int_a^b f(x)^{dx}$ or $\prod_a^b f(x)^{dx}$ is defined as in (2)

$$\int_a^b f(x)^{dx} = \prod_a^b f(x)^{dx} = exp\left(\int_a^b ln\left(f(x) \right) dx \right). \tag{2}$$

In these definitions, $\mathbb{R}^+$ indicates the positive real numbers and *ln*(*f*(*x*)) represents the natural logorithm of the given function. It is known that the classical derivative denoted by $\frac{d}{dx} f(x)$ or $f'(x)$ of a function $f(x) \in \mathbb{R}^+$ is defined as following [(3)]

$$\frac{d}{dx} f(x) = f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}. \tag{3}$$

Comparing the multiplicative derivative equation with this classical definition of derivative, we observe that the difference f(x + h) – f(x) is replaced by the ratio $\frac{f(x+h)}{f(x)}$, and also the division by h is replaced by raising the reciprocal power 1/h.

Additionally, the multiplicative derivative and the classical derivative are related by (4) and (5):

$$f^*(x) = \lim_{h \to 0} \left(\frac{f(x+h)}{f(x)} \right)^{1/h} = \lim_{h \to 0} (1 + \\ + \frac{f(x+h) - f(x)}{f(x)})^{\frac{f(x)}{f(x+h)-f(x)} \times \frac{f(x+h)-f(x)}{h} \times \frac{1}{f(x)}}, \tag{4}$$

$$f^*(x) = exp\left(\frac{f'(x)}{f(x)} \right) = exp\left(\frac{d}{dx} ln\left(f(x) \right) \right). \tag{5}$$

IV. THE PROPOSED ALGORITHM

The proposed algorithm is a modified version of BP for training neural networks (NNs). The algorithm still makes backward passes during update processes of adjustable parameters.

The novelty and improvement of the proposed algorithm is the use of multiplicative form of the derivative in the computations rather than the classical derivative. The type of sigmoid function is selected as the activation function forms on the layers of NN but the new derivative form is applied only on the last layer not in the hidden layers. The sigmoid function is defined by the following formula [(6)]

$$\frac{1}{1+e^{-x}} = \frac{e^x}{e^x + 1}. \tag{6}$$

The natural logarithm of the sigmoid function is calculated as given in (7), below

$$ln\left(\frac{1}{1+e^{-x}} \right) = ln\left(\frac{e^x}{e^x + 1} \right) = ln(e^x) - ln(e^x + 1) = \\ = x - ln(e^x + 1). \tag{7}$$

Then the classical derivative for the natural logarithm of the sigmoid function is evaluated as given in (8) and (9), below:

$$\frac{d}{dx}ln\left(\frac{1}{1+e^{-x}}\right)=\frac{d}{dx}ln\left(\frac{e^{x}}{e^{x}+1}\right)=$$
$$=\frac{d}{dx}\left\{x-ln(e^{x}+1)\right\}, \quad (8)$$

$$\frac{d}{dx}\left\{x\right\}-\frac{d}{dx}\left\{ln(e^{x}+1)\right\}=1-\frac{1}{e^{x}+1}\times e^{x}=\frac{1}{e^{x}+1}. \quad (9)$$

Finally, the multiplicative derivative of the sigmoid function is formulated by (10), below

$$\frac{d^{*}}{dx^{*}}\frac{1}{1+e^{-x}}=exp\left(\frac{d}{dx}ln\left(\frac{1}{1+e^{-x}}\right)\right)=exp\left(\frac{1}{e^{x}+1}\right). \quad (10)$$

The proposed algorithm is now described as in Algorithm 2.

Algorithm 2. Proposed algorithm framework.

```
Given the training data set
$\{\boldsymbol{inputs}, \boldsymbol{target\ values}\}=\{\boldsymbol{x's}, \boldsymbol{t's}\}$,
1. Initialise the weights, including biases.
2. Take one input from the training data set $\{\boldsymbol{x}, \boldsymbol{t}\}$ and obtain the network output, $\boldsymbol{y}$.
3. Calculate the exp raised to the power of $\boldsymbol{t}$ and the exp raised to the power of $\boldsymbol{y}$ to obtain the error, $\boldsymbol{e}$, of the output of the network to the target and the delta, $\boldsymbol{\delta}$, of the output nodes using the multiplicative derivative of the sigmoid function.
    $\boldsymbol{e}=\boldsymbol{exp}(\boldsymbol{t})-\boldsymbol{exp}(\boldsymbol{y})$ and $\boldsymbol{\delta}=\left(\frac{\boldsymbol{d}^{*}}{\boldsymbol{dx}^{*}}\frac{\boldsymbol{1}}{\boldsymbol{1}+\boldsymbol{e}^{-\boldsymbol{y}}}\right)\boldsymbol{e}$
4. Propagate the output node deltas, $\boldsymbol{\delta}$, in backward direction but now use the classical derivative of the sigmoid function.
    $\boldsymbol{e}^{(\boldsymbol{k})}=\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{\delta}$ and $\boldsymbol{\delta}^{(\boldsymbol{k})}=\boldsymbol{\varphi}'\left(\boldsymbol{v}^{(\boldsymbol{k})}\right)\boldsymbol{e}^{(\boldsymbol{k})}$
5. Repeat the previous step up to the layer that is on the immediate right of the input layer.
6. Update the adjustable parameters according to the following rule.
    $\Delta\boldsymbol{w}_{\boldsymbol{ij}}=\boldsymbol{\alpha}\boldsymbol{\delta}_{\boldsymbol{i}}\boldsymbol{x}_{\boldsymbol{j}}$ and $\boldsymbol{w}_{\boldsymbol{ij}}(\boldsymbol{new})=\boldsymbol{w}_{\boldsymbol{ij}}(\boldsymbol{old})+\Delta\boldsymbol{w}_{\boldsymbol{ij}}$
7. Repeat the stages explained in the previous steps for each training data point.
8. Continue updates until the network parameters are adjusted accordingly and the NN is properly trained.
```

V. RESULTS AND DISCUSSION

To analyse the performance of the proposed modified backpropagation (BP) algorithm, various cases of linearly inseparable problems are examined through some shallow neural network (NN) models.

– *Case I: Training XOR classification task with two input nodes, one output node, and one hidden layer of two nodes. The size of the training data set is four.*

The XOR, or "exclusive OR", classification problem is a classic and common problem in NN research areas [17]. The two-input XOR problem is shown in Table I.

Minsky and Papert [18] first showed that it was impossible for a single-layer perceptron network to solve the XOR classification problem. In this experiment, a shallow NN with two input nodes, one hidden layer with two nodes, and one output node is used. The architecture is demonstrated in Fig. 1. Each node in the hidden layer and the output layer includes a bias term.

In the first phase of the experiment, the network is trained with the standard BP algorithm. The sigmoid function is chosen as the transfer function in each node unit. Next, the network is trained with the proposed algorithm. The weights and biases are initially selected as 0.5 for each node as follows (11)–(14):

$$W_{hidden}=\begin{bmatrix}0.5 & 0.5\\ 0.5 & 0.5\end{bmatrix}, \quad (11)$$

$$b_{hidden}=\begin{bmatrix}0.5\\ 0.5\end{bmatrix}, \quad (12)$$

$$W_{output}=\begin{bmatrix}0.5 & 0.5\end{bmatrix}, \quad (13)$$

$$b_{output}=\begin{bmatrix}0.5\end{bmatrix}. \quad (14)$$

The learning rate is chosen as 0.2.

TABLE I. TWO-INPUT XOR CLASSIFICATION PROBLEM.

Input 1	**Input 2**	**Output**
0	0	0
0	1	1
1	0	1
1	1	0

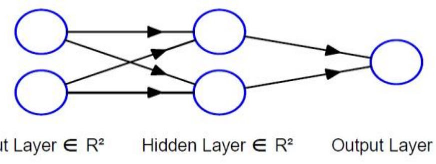

Fig. 1. NN architecture in Case I for the two-input XOR classification problem.

Figure 2 illustrates the comparison of performances between the proposed algorithm and standard BP. The figure specifically includes the mean error at each epoch.

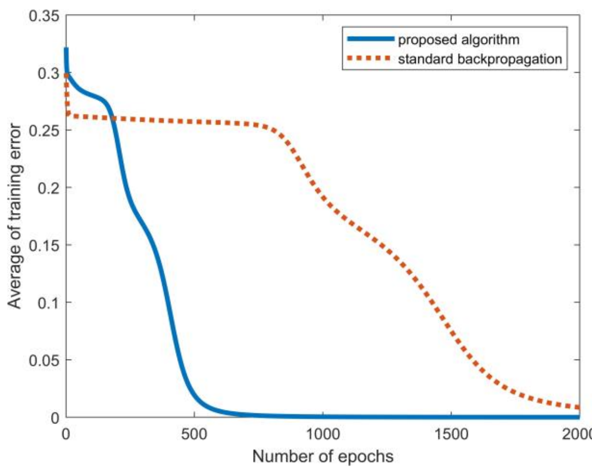

Fig. 2. Performance comparison of standard BP and the algorithm proposed in Case I with learning rate = 0.2.

In the second phase of the experiment, the learning rate is increased and chosen as unity - 1, but the weights and biases are initially selected the same as in the first phase of the experiment, 0.5 for each node.

The corresponding performance comparison is given in Fig. 3. The results of the training the XOR classification task demonstrate that the proposed modified BP algorithm provides faster learning than the classical BP with given problem specifications and used network architecture.

– *Case II: Training a linearly inseparable classification task with three input nodes, one output node, and one hidden layer of four nodes. The size of the training data set is four.*

In this experiment, a linearly inseparable problem shown in Table II is chosen. To solve the classification problem, the NN is designed with three input nodes, one hidden layer with four nodes, and one output node. The NN architecture designed is represented in Fig. 4. All nodes in the hidden layer and output node do not involve bias terms in this case.

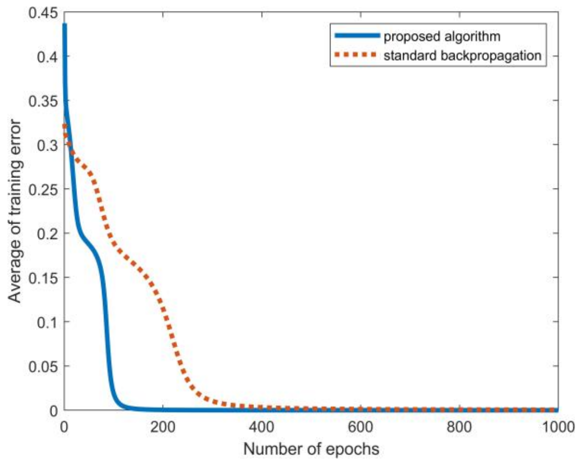

Fig. 3. Performance comparison of standard BP and the algorithm proposed in Case I with learning rate = 1.

TABLE II. A LINEARLY INSEPARABLE CLASSIFICATION PROBLEM.

Input 1	Input 2	Input 3	Output
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

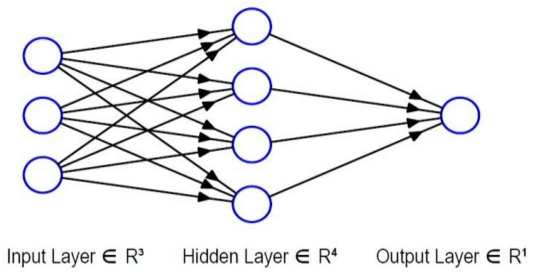

Fig. 4. NN architecture in Case II for the three-input linearly inseparable classification problem.

In the first stage of the experiment in Case II, the network is trained with a standard BP algorithm. The sigmoid function is chosen as the transfer function in each node unit. Next, the network is trained with the proposed algorithm. The learning rate is chosen as 0.1 in both stages. To keep the equality in analysing both algorithms, classical BP and the proposed algorithm, the weight matrices for the hidden layer and the output layer are selected the same as given by (15) and (16):

$$W_{hidden} = \begin{bmatrix} -0.35 & 0.30 & -0.60 \\ -0.55 & 0.20 & -0.80 \\ -0.45 & 0.10 & -0.40 \\ -0.15 & 0.40 & -0.50 \end{bmatrix}, \tag{15}$$

$$W_{output} = \begin{bmatrix} -0.30 & 0.20 & 0.30 & -0.20 \end{bmatrix}. \tag{16}$$

Figure 5 shows the average training errors of the proposed algorithm and the standard BP.

In the second stage of the experiment in Case II, the learning rate is changed and selected as one while the initial weights and biases are chosen equally as in the first phase of the experiment. The performance of the standard BP and the proposed algorithm are indicated in Fig. 6.

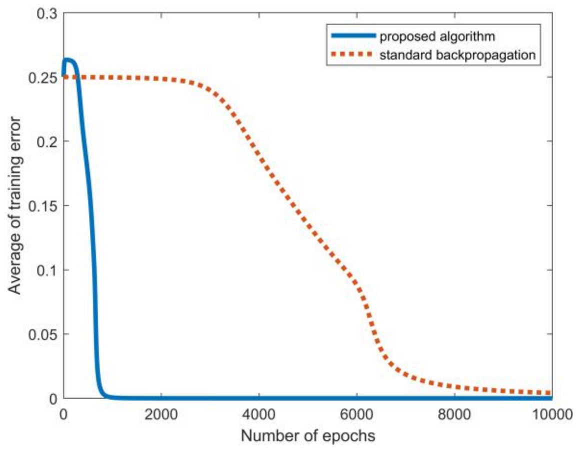

Fig. 5. Performance comparison of standard BP and the algorithm proposed in Case II with learning rate = 0.1.

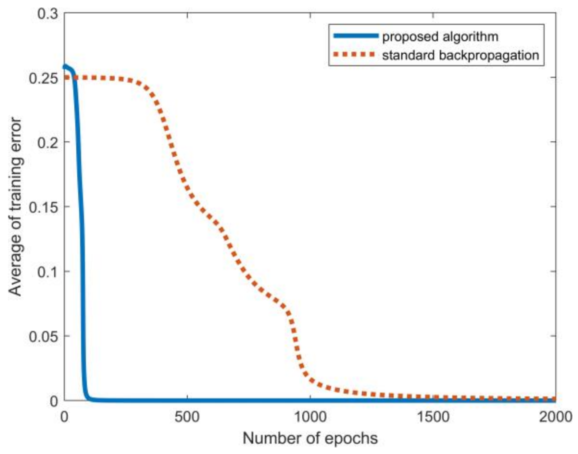

Fig. 6. Performance comparison of standard BP and the algorithm proposed in Case II with learning rate = 1.

From the experimental results in Case II, it is clear that for the given linearly inseparable problem, the proposed algorithm outperforms the classical BP considering the convergence speed to minimum loss.

– Case III: Training a linearly inseparable classification task with three input nodes, one output node, and one hidden layer of four nodes. The size of the training data set is eight.

In Case III, another inseparable problem with the size of eight training pairs is selected as given in Table III. To solve the given problem, the same NN architecture is used as in Case II.

In this experiment, the network performance is analysed both for the standard BP algorithm and for the proposed algorithm. The sigmoid function is set as the transfer function in each node unit. The learning rate is chosen as one in both algorithms, and the weights are initially selected same as in Case II. The resulting average training errors of the standard BP algorithm and the proposed algorithm are presented in Fig. 7.

The results obtained from the experiments conducted in Case III specify that the proposed algorithm shows superior performance compared to standard BP.

As stated above, one of the important pitfalls of BP is the tendency to a slow convergence. Taking into account the empirical evidence from the experiments conducted in all cases, the modified algorithm provides faster training with an improvement in the risk of slow convergence.

TABLE III. A LINEARLY INSEPARABLE CLASSIFICATION PROBLEM.

Input 1	Input 2	Input 3	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

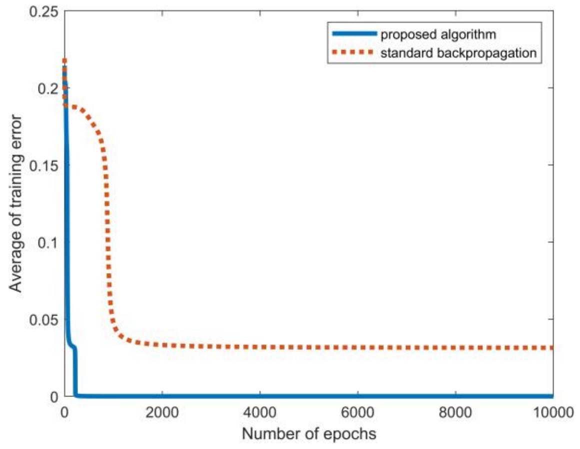

Fig. 7. Performance comparison of the standard BP and the algorithm proposed in Case III with learning rate = 1.

Another significant risk in training NNs by BP is the local minima problem [19], [20] and this threat is more common in linearly inseparable situations [7]. The local minima problem usually occurs due to the saturation of nodes in the hidden layers of feedforward NNs. In the case of saturation, a lack of harmony would occur in the weights connecting the hidden layer to the output layer [6] and the network may no longer be trained.

To determine the performance of the proposed algorithm to alleviate the local minima problem in nonlinearly separable classification tasks, the initial weights and bias values are set to one of the potential local minima point for standard BP algorithm in two-input XOR classification problem, and then the proposed algorithm is tested through the same network with same initial values.

The weights and biases for each node are initially selected as given by (17)–(20):

$$W_{hidden} = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}, \tag{17}$$

$$b_{hidden} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \tag{18}$$

$$W_{output} = \begin{bmatrix} 4 & 4 \end{bmatrix}, \tag{19}$$

$$b_{output} = \begin{bmatrix} 1 \end{bmatrix}. \tag{20}$$

The learning rate is chosen as unity (1).

Figure 8 shows the performance of the proposed algorithm in the case of a local minimum point of standard BP.

A similar experiment is conducted using the NN architecture to solve the three-input linearly inseparable classification problem in Case II. When a local minimum is chosen with the selected initial weights given by (21) and (22) and the learning rate is chosen as unity - 1, the corresponding analysis is presented in Fig. 9:

$$W_{hidden} = \begin{bmatrix} 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \\ 4 & 4 & 4 \end{bmatrix}, \tag{21}$$

$$W_{output} = \begin{bmatrix} -4 & -4 & -4 & 4 \end{bmatrix}. \tag{22}$$

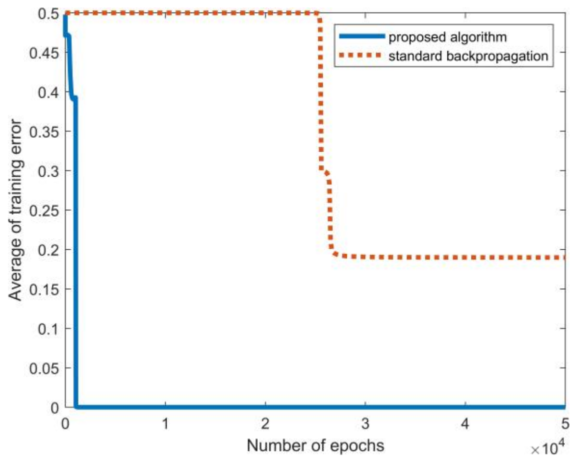

Fig. 8. Performances of the proposed algorithm and standard BP to a specific local minimum problem, two-input XOR classification problem.

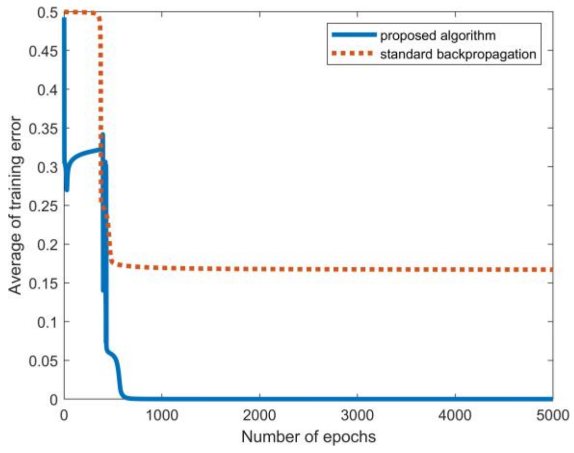

Fig. 9. Performances of the proposed algorithm and standard BP to a specific local minimum problem, three-input linearly inseparable classification problem in Case II.

For the classification task given in Case III, Fig. 7 also gives this comparison with respect to a local minima problem. All the experimental evidence states that the proposed algorithm is less prone to the common local minima problems.

It is known that, in training the NNs, initial weights, biases, and also learning rate affect the performance of learning process directly on convergence characteristics (speed, training time, and avoiding local minima).

To obtain a generalised performance evaluation, different initial weights and biases are randomly selected, and the algorithms with different learning rates are compared. Table IV illustrates the experimental results for the two-input XOR problem in Case I.

Both algorithms are run with three different learning rates ($\alpha = 0.1$, $\alpha = 0.5$, and $\alpha = 1$) and the algorithms are implemented with 100 different random initial weights and biases. Random values are generated in the interval of [0, 1]. The table provides the averages of all mean square error (MSE) values in total of 100 experiments after completing the 1000th, 5000th, and 10000th training cycles.

TABLE IV. EXPERIMENTAL RESULTS FOR TWO-INPUT XOR PROBLEM.

Algorithm	MSE after 1000th epoch	MSE after 5000th epoch	MSE after 10000th epoch
Proposed algorithm ($\alpha = 0.1$)	0.1177	0.0339	0.0226
BP algorithm ($\alpha = 0.1$)	0.2510	0.1876	0.0917
Proposed algorithm ($\alpha = 0.5$)	0.0265	0.0166	0.0142
BP algorithm ($\alpha = 0.5$)	0.2160	0.0320	0.0172
Proposed algorithm ($\alpha = 1$)	0.0184	0.0161	0.0153
BP algorithm ($\alpha = 1$)	0.0863	0.0238	0.0189

Similar analysis is performed to test the proposed algorithm on avoiding the local minima problem. The two-input XOR problem and the same NN architecture as in Case I are selected and the learning rate is taken as one. 100 different random initial weights and biases are created in the interval of [0, 1] and the standard BP and the proposed algorithm are tested in 100 experiments.

The results are recorded after 50,000 epochs and the experiments that converge to a local minimum are labelled as learning failures. Success rate is calculated as the number of experiments that converges to the global minimum. Table V shows the success rates for avoiding the local minima for the two-input XOR problem in Case I.

TABLE V. SUCCESS RATES OF ALGORITHMS TO AVOID THE LOCAL MINIMA PROBLEM FOR TWO-INPUT XOR PROBLEM.

Algorithm	Number of learning failures (out of 100 experiments)	Success rate
Proposed algorithm	4 (out of 100)	96 %
BP algorithm	15 (out of 100)	85 %

To make a comparative analysis of the proposed algorithm with improved versions of BP, two benchmark problems are selected. Those are standard two-input XOR problem as in Case I and modified XOR problem [21]. The training performances of the algorithms are evaluated in terms of the convergence speeds in the simulations. In comparisons, the BP algorithm with momentum (BP-M), BP with ESP function (BP-ESP) for the output nodes, BP with ESP function for hidden output nodes (BP-ESP-H) [22], BP with gain (BP-G) [23], BP with adaptive momentum (BP-AM) [24], BP with adaptive gain (BP-AG) [25], and the Nguyen-Widrow weight initialisation technique (NG-W) [26] are selected as improved versions of BP. To be parallel and consistent with the methods mentioned in [21], in the proposed algorithm, the weights and biases are initialised to random values in the range of (-0.5, 0.5), the learning rate is set to 0.5, and the performance measure is chosen as MSE. 30 independent trials are conducted and the numbers of epochs required for convergence are recorded. Then the mean of the epochs (# of epochs required to converge) is calculated. The termination condition for convergence is chosen as the MSE of 0.001.

First, to train the two-input XOR problem, 2-2-1 network (two input nodes, one output node, and one hidden layer of two nodes) is used as in Case I. The results of the corresponding comparisons on this problem are given in Table VI. Performance evaluation results for the improved versions of BP are taken from [21].

TABLE VI. COMPARISON OF THE ALGORITHMS ON TWO-INPUT XOR PROBLEM.

Algorithm	# of epochs required to converge
BP-M	5971
BP-AM	4996
BP-ESP	2220
BP-ESP-H	2256
NG-W	1127
BP-G	5805
BP-AG	6885
Proposed algorithm	1070

Second, to train the modified XOR problem, the same 2-2-1 network is used. The truth table for the modified XOR problem is given in Table VII.

TABLE VII. MODIFIED XOR PROBLEM [21].

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0
0.5	0.5	1

The performance comparisons of the improved versions of BP and the proposed algorithm are presented in Table VIII.

TABLE VIII. COMPARISONS OF THE ALGORITHMS ON A MODIFIED XOR PROBLEM.

Algorithm	# of epochs required to converge
BP-M	6621
BP-AM	5100
BP-ESP	2718
BP-ESP-H	2269
NG-W	1143
BP-G	5960
BP-AG	2389
Proposed algorithm	1356

VI. CONCLUSIONS

In this paper, a modified backpropagation (BP) algorithm with multiplicative calculus is proposed for feedforward neural networks (NNs). The proposed algorithm contains general characteristics of standard BP that make backward passes during update processes of learnable parameters. On the other hand, the originality and enhancement are the utilisation of the multiplicative form of the derivative in the computations rather than the classical derivative.

The sigmoid function is preferred as the activation functions on the hidden layer of NN and the new multiplicative derivative is applied only on the last layer not in the hidden layer.

Standard BP is very common in many NN applications but it has two major negative issues: convergence speed can be slower and the training algorithm can converge to a local minimum. The proposed algorithm introduces a novel solution to overcome the slow convergence rate problem and avoid the local minima problem. Many different tasks are chosen, and several experiments are conducted to measure the performance of the proposed algorithm. Experimental results show that the proposed algorithm with multiplicative calculus yields outstanding success at both convergence speed and avoiding local minima. Simulations carried out in Case I have demonstrated that when the learning rate is chosen, $\alpha = 1$, after the 1000th epoch, a reduction of approximately 80 % in mean square error is obtained as compared to the standard BP. Additionally, a 96 % success rate has been achieved in avoiding the local minima problem while a success rate is 85 % in standard BP.

CONFLICTS OF INTEREST

The author declares that he has no conflict of interest.

REFERENCES

[1] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences", Ph.D. dissertation, Harvard University, Cambridge, MA, 1974.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[3] D. B. Parker, "Learning-logic: Casting the cortex of the human brain in silicon", Technical Report TR-47, Center for Computational Research in Economics and Management Science, Cambridge, MA, MIT, 1985.

[4] Y. Lecun, "Une procedure d'apprentissage pour reseau a seuilassymetrique", in *Proc. of Cognitiva 85*, 1985, pp. 599–604.

[5] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA: MIT Press, 1986. DOI: 10.7551/mitpress/5236.001.0001.

[6] W. Bi, X. Wang, Z. Tang, and H. Tamura, "Avoiding the local minima problem in backpropagation algorithm with modified error function", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E88-A, no. 12, pp. 3645–3653, 2005. DOI: 10.1093/ietfec/e88-a.12.3645.

[7] M. Gori and A. Tesi, "On the problem of local minima in backpropagation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 1, pp. 76–86, 1992. DOI: 10.1109/34.107014.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[9] P. Kim, Matlab Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence. Apress, 2017. DOI: 10.1007/978-1-4842-2845-6_1.

[10] S. K. Stein and A. Barcellos, *Calculus and Analytic Geometry*, 5th ed. McGraw-Hill, 1992.

[11] A. Uzer, "Multiplicative type complex calculus as an alternative to the classical calculus", *Computers and Mathematics with Applications*, vol. 60, no. 10, pp. 2725–2737, 2010. DOI: 10.1016/j.camwa.2010.08.089.

[12] M. Grossman and R. Katz, *Non-Newtonian* Calculus. Lee Press, Massachusetts, 1972.

[13] J. Grossman, *Meta-Calculus: Differential and Integral*. University of Michigan, 1981. DOI: 10.1016/B978-0-12-304360-3.50010-4.

[14] A. Laucka, V. Adaskeviciute, D. Andriukaitis, "Research of the Equipment Self-Calibration Methods for Different Shape Fertilizers Particles Distribution by Size Using Image Processing Measurement Method," *Symmetry*, vol. 11, no. 7, p. 838, Jun. 2019. DOI: 10.3390/sym11070838.

[15] A. E. Bashirov, E. M. Kurpınar, and A. Özyapıcı, "Multiplicative calculus and its applications", *Journal of Mathematical Analysis and Applications*, vol. 337, no. 1, pp. 36–48, 2008. DOI: 10.1016/j.jmaa.2007.03.081.

[16] A. E. Bashirov and M. Rıza, "On complex multiplicative differentiation", *TWMS Journal of Applied and Engineering Mathematics*, vol. 1, no. 1, pp. 75–85, 2011.

[17] A. Mishra, J. Cha, and S. Kim, "Single neuron for solving XOR like nonlinear problems", *Computational Intelligence and Neuroscience*, vol. 2022, art. ID 9097868, 2022. DOI: 10.1155/2022/9097868.

[18] M. L. Minsky and S. A. Papert, *Perceptrons: An introduction to Computational Geometry*. MIT Press, 2017. DOI: 10.7551/mitpress/11301.001.0001.

[19] L. F. A. Wessels and E. Barnard, "Avoiding false local minima by proper initialization of connections", *IEEE Transactions on Neural Networks*, vol. 3, no. 6, pp. 899–905, 1992. DOI: 10.1109/72.165592.

[20] I. G. Sprinkhuizen-Kuyper and E. J. W. Boers, "A local minimum for the 2-3-1 XOR network", *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 968–971, 1999. DOI: 10.1109/72.774274.

[21] Saduf and M. A. Wani, "Comparative study of high speed back-propagation learning algorithms", *International Journal of Modern Education and Computer Science*, vol. 6 no. 12, pp. 34–40, 2014. DOI: 10.5815/ijmecs.2014.12.05.

[22] H.-M. Lee, C.-M. Chen, and T.-C. Huang, "Learning efficiency improvement of back-propagation algorithm by error saturation prevention method", *Neurocomputing*, vol. 41, nos. 1–4, pp. 125–143, 2001. DOI: 10.1016/S0925-2312(00)00352-0.

[23] X. G. Wang, Z. Tang, H. Tamura, M. Ishii, and W. D. Sun, "An improved backpropagation algorithm to avoid the local minima problem", *Neurocomputing*, vol. 56, pp. 455–460, 2004. DOI: 10.1016/j.neucom.2003.08.006.

[24] H. M. Shao and G. F. Zheng, "A new BP algorithm with adaptive momentum for FNNs training", in *Proc. of 2009 WRI Global Congress on Intelligent Systems*, 2009, pp. 16–20. DOI: 10.1109/GCIS.2009.136.

[25] N. M. Nawi, R. S. Ransing, and M. R. Ransing, "A new method to improve the gradient based search direction to enhance the computational efficiency of back propagation based neural network algorithms", in *Proc. of 2008 IEEE Second Asia International Conference on Modelling & Simulation*, 2008, pp. 546–552. DOI: 10.1109/AMS.2008.70.

[26] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights", in *Proc. of 1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 21–26, vol. 3. DOI: 10.1109/IJCNN.1990.137819.