Scientific Research Publishing

American Journal of Computational Mathematics, 2026, 16(3), 151-167
https://www.scirp.org/journal/ajcm
ISSN Online: 2161-1211
ISSN Print: 2161-1203

A Mathematical Analysis of the Backpropagation Algorithm in Artificial Neural Networks

Olaniyi Sadik Maliki[1], Jane Ngozi Oruh[2], Patrick Agwu Okpara[3]

[1]Department of Mathematics, Michael Okpara University of Agriculture, Umudike, Nigeria
[2]Department of Computer Science, Michael Okpara University of Agriculture, Umudike, Nigeria
[3]Department of Mathematics and Health Statistics, David Umahi Federal University of Health Sciences, Uburu, Nigeria
Email: maliki.niyi@mouau.edu.ng, janeoruh@mouau.edu.ng, paokpara@yahoo.com

How to cite this paper: Maliki, O.S., Oruh, J.N. and Okpara, P.A. (2026) A Mathematical Analysis of the Backpropagation Algorithm in Artificial Neural Networks. *American Journal of Computational Mathematics*, **16**, 151-167.
https://doi.org/10.4236/ajcm.2026.163009

Received: April 20, 2026
Accepted: July 19, 2026
Published: July 22, 2026

Abstract

Any new mathematical concept in Machine Learning can often times prove difficult until it is approached in an algorithmic format, enabling manual step-by-step computations to be possible. This type of computation-based approach from first principles can greatly enhance our understanding of the subject. Backpropagation is very basic to Machine Learning; however, it may not be easy to fully comprehend how it functions. This article provides a detailed, step-by-step pedagogical exposition of the backpropagation algorithm for multi-layer perceptrons (MLPs). Starting from first principles, we derive the necessary gradient expressions using the chain rule, illustrate the calculations manually on a small numerical example, and provide an annotated MATLAB implementation. The contribution is a self-contained tutorial that bridges the gap between mathematical formalism and executable code, enabling learners to trace forward and backward propagation without external references. Our main objective is to minimize the sum of squared errors involving the input values and the target values. Once this error is very close to or equal to zero, we conclude that the neural network has learned the given task. We demonstrate convergence of the sum-of-squares error over 10,000 iterations.

Keywords

Backpropagation Algorithm, MLP, Sigmoid Function, Sum of Squares Error, Matlab Code

1. Introduction

An artificial neural network (ANN) is an information–processing system that has

certain performance characteristics in common with biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of human cognition or neural biology [1], based on the assumptions that:

- Information processing occurs at many simple connections called neurons.
- Signals are passed between neurons over connection links.
- Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
- Each neuron applies an activation function to its net input to determine its output signal.

The basic component of an artificial neural network is the artificial neuron, similar to a biological neuron [2] in a biological neural network. A biological neuron may be modeled artificially to perform computation, and then the model is termed an artificial neuron.

1.1. Brief Literature

The beginning of Neurocomputing dates back to 1943 when McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logic function [1]. The Backpropagation Algorithm, which is a major tool in minimizing error in ANN, was first proposed by Werbos (1974) [2], and since then has been employed in a number of fields for solving problems that would be quite difficult using conventional computer science techniques. In Nielson (2023) [3], it is shown that it is possible to adjust the weight of a multilayer feed-forward neural network in a systematic way to learn the implicit mapping in a set of input-output pattern pairs. The learning law is called the generalized delta rule or error backpropagation.

This manuscript does not propose a new algorithm or a theoretical advance. Instead, it serves as an educational resource in the form of a fully explicit, manually verified walkthrough of backpropagation with accompanying MATLAB code. All mathematical derivations follow classical references [2] and [3], but the step-by-step numerical presentation with all intermediate values is intended to reduce the learning curve for students and practitioners.

1.2. Multilayer Perceptrons (MLP)

Multilayer perceptrons (MLP) are the most popular type of neural networks [4]. They belong to a general class of structures called feedforward neural networks, a basic type of neural network capable of approximating generic classes of functions, including continuous and integrable functions. MLP neural networks have been used in a variety of microwave modelling and optimization problems.

1.3. MLP Structure

The Multilayer Perceptron (MLP) neural network structure is depicted below in **Figure 1**, incorporating the input layer, hidden layers and output layer.

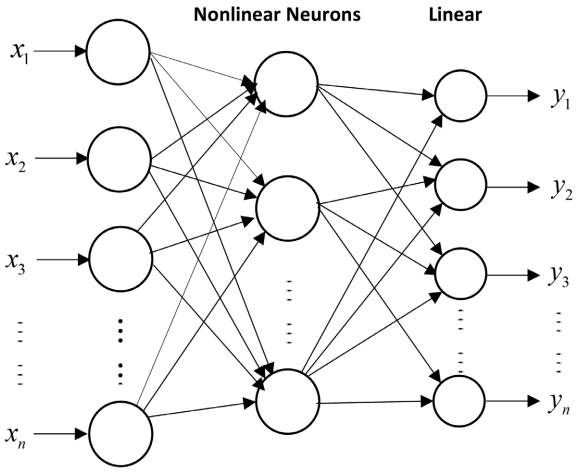

Figure 1. General neural network (Multilayer perceptron).

This particular model can accommodate many inputs $(x_1, x_2, \cdots, x_n)$, and the outputs $(y_1, y_2, \cdots, y_n)$ can each be related to the inputs in the following way, $y_i = \phi_i(x_1, x_2, \cdots, x_n)$, $i = 1, \cdots, n$, where the ϕ_i's are the respective activation functions.

1.4. Activation Functions

Activation functions are very important in Machine Learning because they incorporate *nonlinearity* into our ANN [2]. This enables the neural network to learn complicated tasks; without it, the neural network can only perform *linear regression*. The most popular hidden neuron activation functions are the sigmoid, arctangent, and the hyperbolic tangent functions given respectively by (**Figure 2**).

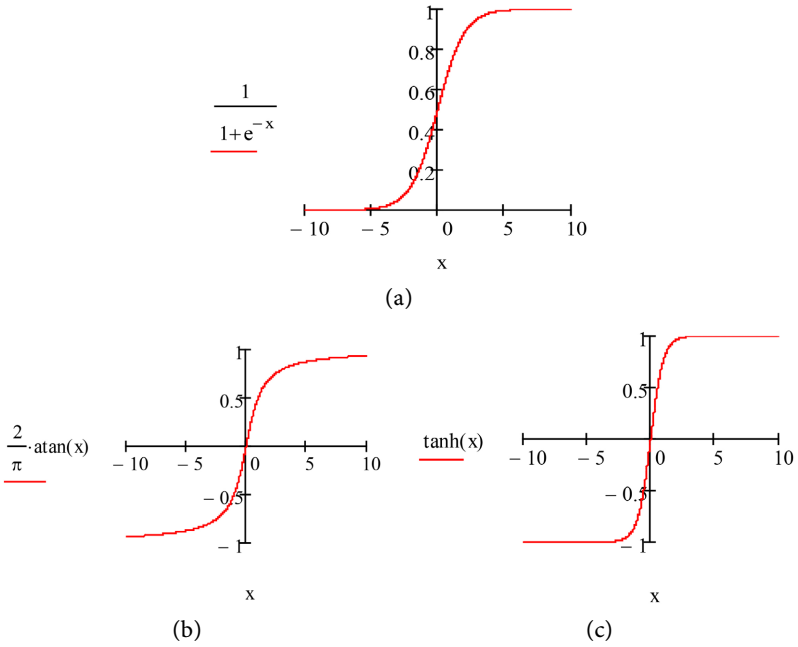

Figure 2. Activations functions. (a) $\sigma_1(x)$, (b) $\sigma_2(x)$, (c) $\sigma_3(x)$.

$$\sigma_1(x)=\frac{1}{1+e^{-x}},\quad \sigma_2(x)=\frac{2}{\pi}\arctan x \quad \text{and} \quad \sigma_3(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\tanh(x) \tag{1}$$

The sigmoid function $\sigma_1(x)$ is a smooth switch function having the following property;

$$\sigma_1(x)\to\begin{cases}1 & \text{as} \quad x\to+\infty\\ 0 & \text{as} \quad x\to-\infty\end{cases} \tag{2}$$

All these logistic functions are bounded, continuous, monotonic, and continuously differentiable.

In this work, we require the derivative of the sigmoid function. Thus given

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

We have

$$\frac{d}{dx}\sigma(x)=\frac{d}{dx}\left(1+e^{-x}\right)^{-1}=-\left(1+e^{-x}\right)^{-2}\left(-e^{-x}\right)=\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}$$

Now

$$\frac{e^{-x}}{\left(1+e^{-x}\right)^{2}}=\frac{1+e^{-x}-1}{\left(1+e^{-x}\right)^{2}}=\frac{\left(1+e^{-x}\right)-1}{\left(1+e^{-x}\right)^{2}}=\frac{1}{1+e^{-x}}-\frac{1}{\left(1+e^{-x}\right)^{2}}=\sigma(x)-\left(\sigma(x)\right)^{2}$$

$$\therefore \quad \frac{d}{dx}\sigma(x)=\sigma(x)-\left(\sigma(x)\right)^{2}=\sigma(x)\left(1-\sigma(x)\right) \tag{3}$$

1.5. Objective and Contribution

The exact contribution of this paper is a fully worked numerical example of backpropagation for a 3-2-2 network (One input layer with three neurons, one hidden layer with two neurons and one output layer with two neurons, **Figure 6**), including all intermediate gradient calculations, followed by a verifiable MATLAB implementation. No novel algorithmic modifications are proposed; instead, we prioritize transparency and reproducibility for educational purposes.

2. Universal Approximation Theorem

The universal approximation theorem [4] for MLP is stated as follows.

2.1. Universal Approximation

Let $\boldsymbol{I}_n$ represent an n-dimensional unit cube containing all possible input samples $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$ with $x_i\in[0,1]$, $i=1,2,\cdots,n$. Let $C(\boldsymbol{I}_n)$ be the space of continuous functions on $\boldsymbol{I}_n$, given a continuous sigmoid function $\sigma(\cdot)$, then universal approximation theorem states that the finite sums of the form

$$y_k=y_k(\boldsymbol{x},\boldsymbol{w})=\sum_{i=1}^{N_2}w_{ki}^{3}\sigma\left(\sum_{j=0}^{n}w_{ki}^{2}x_j\right)\quad k=1,2,\cdots,m \tag{4}$$

are dense in $C(\boldsymbol{I}_n)$. This simply means that given any function $f \in C(\boldsymbol{I}_n)$ and $\varepsilon > 0$, there is a sum $y(\boldsymbol{x}, \boldsymbol{w})$ of the above form that satisfies

$$\left| y(\boldsymbol{x}, \boldsymbol{w}) - f(\boldsymbol{x}) \right| < \varepsilon \ \forall \boldsymbol{x} \in \boldsymbol{I}_n \tag{5}$$

2.2. Remark

The universal approximation theorem expressed in Equation (4) simply states that there always exists a 3-layer perceptron that can approximate an arbitrary nonlinear, continuous, multidimensional function *f* with any desired accuracy. However, the theorem does not state how many neurons are needed by the three-layer MLP to approximate the given function; as such, failure to develop an accurate model can be attributed to an inadequate number of hidden neurons, inadequate learning or training, or perhaps the presence of a stochastic rather than a deterministic relation between inputs and outputs [5].

2.3. Model of a Neuron

The computational structure of a neuron (**Figure 3**) consists of a set of synapses, each of which is characterized by

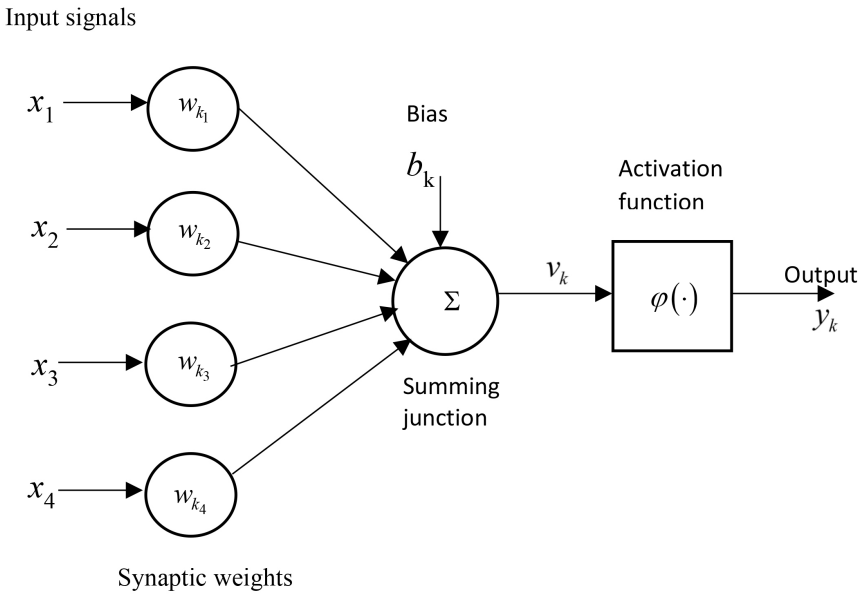

Figure 3. Computational structure of a neuron.

- A weight or strength of its own.
- An adder, an activation function and a bias.

In mathematical terms, a neuron *k* can be described by:

$$v_k = \sum_{j=1}^{m} w_{k_j} x_j \quad \text{and} \quad y_k = \varphi(u_k + b_k) \tag{6}$$

where u_k is the linear combiner output due to input signals. Moreover, $v_k = u_k + b_k$.

The bias is an external parameter of artificial neuron and can be included into Equation (6) as follows:

$$v_k = \sum_{j=0}^{m} w_{k_j} x_j \quad \text{and} \quad y_k = \varphi(v_k) \tag{7}$$

2.4. Manual Computation of the MLP Model

We consider a Multilayer Perceptron Feed Forward Neural Network with binary inputs. The network diagram is given below (Figure 4).

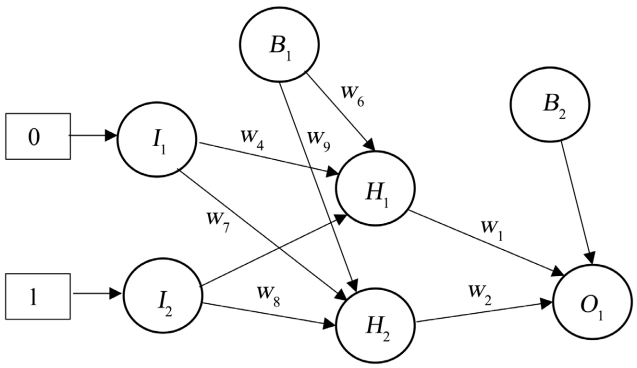

Figure 4. MLP Neural Network with one hidden layer containing two neurons H_1, H_2.

In the above network diagram I_1, I_2 are the inputs, B_1, B_2 the bias, H_1, H_2 are the two neurons in the single hidden layer and O_1 is the output. The distribution of weights is given in Table 1. Furthermore, we employ as activation function the sigmoid function

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

Table 1. Weight distribution.

w	Value	Connections
w_1	10	$H_1 \to O_1$
w_2	10	$H_2 \to O_1$
w_3	−5	$B_2 \to O_1$
w_4	5	$I_1 \to H_1$
w_5	−6	$I_2 \to H_1$
w_6	−3	$B_1 \to H_1$
w_7	−6	$I_1 \to H_2$
w_8	6	$I_2 \to H_2$
w_9	−3	$B_1 \to H_2$

The computations are as follows:

$$H_1 = \phi\left(I_1 w_4 + I_2 w_5 + w_6\right) = \phi\left[0\times 5 + 1\times(-6) + (-3)\right]$$

$$= \phi(-9) = \frac{1}{1+\mathrm{e}^{-(-9)}} = 1.234\times 10^{-4}$$

$$H_2 = \phi\left(I_1 w_7 + I_2 w_8 + w_9\right) = \phi\left[0\times(-6) + 1\times 6 + (-3)\right] = \phi(3) = \frac{1}{1+\mathrm{e}^{-3}} = 0.953$$

$$O_1 = \phi\left(H_1 w_1 + H_2 w_2 + w_3\right) = \phi\left[\left(1.234\times 10^{-4}\right)\times 10 + (0.953)\times 10 + (-5)\right]$$

$$= \phi(4.531) = \frac{1}{1+\mathrm{e}^{-4.531}} = 0.989$$

The final output of the multilayer network is thus 0.989.

2.5. Training

All neural networks must be trained, and the common training algorithm is depicted in the flowchart below (Figure 5).

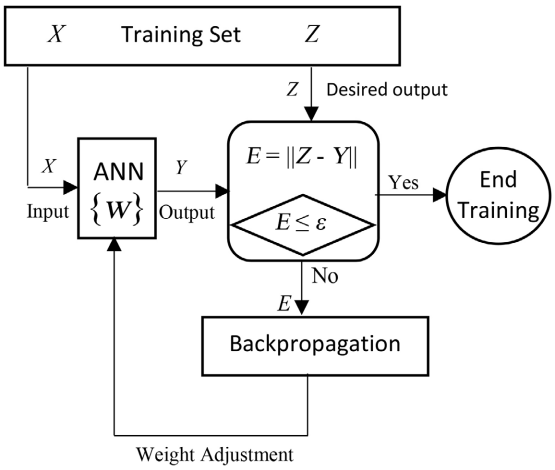

Figure 5. Network training (One training iteration).

2.6. Mathematical Analysis of Backpropagation

Backpropagation actually stands for *backward propagation of errors*, and is commonly used technique for training neural network. It is a mechanism employed to update the weights using *gradient descent*. It computes the gradient of the error function with respect to the neural network's weights. The calculation proceeds backwards [5] through the network. We note that gradient descent is an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize the error function [6]. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

2.7. Backpropagation Algorithm

The Backpropagation algorithm is a supervised learning method for multilayer

feed-forward neural networks. Backpropagation is the algorithm that runs deep learning [7].

The principle of the backpropagation approach is to model a given function by modifying internal weights of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system's output and a known expected output is presented to the system and used to modify its internal state. Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer. In the following demonstration, we use a single training example ($x_1 = 1$, $x_2 = 4$, $x_3 = 5$) with target outputs ($t_1 = 0.1$, $t_2 = 0.05$). The network is not intended for generalization but to illustrate one complete forward-backward-update cycle.

Choice of Hyperparameters and Activation Function

The sigmoid activation is chosen because it is differentiable, bounded, and produces a simple derivative $\sigma'(x) = \sigma(x)(1-\sigma(x))$, which simplifies manual calculations. The sum-of-squares error (SSE) is standard for regression-like tasks and yields clean gradient expressions. A learning rate of 0.01 is sufficiently small to avoid divergence but large enough to show parameter changes within a few decimal places. Initial weights are arbitrarily selected small numbers (0.1 to 0.9) to avoid saturation of the sigmoid at the start. The iteration count of 10,000 ensures the error has visibly converged; this is empirically determined from the MATLAB output.

2.8. Manual Computation of the Backpropagation Algorithm

In this section, we consider an example of one iteration of the backpropagation algorithm employing appropriate formulae from basic principles and actual values. The neural network will consist of three input neurons, one hidden layer with two neurons, and an output layer with two neurons (**Figure 6**).

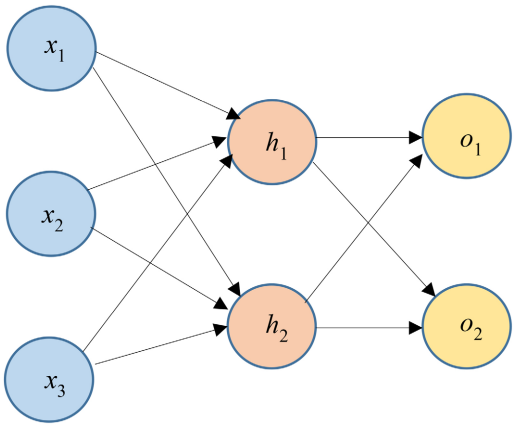

Figure 6. Schematic for the Network training (One training iteration).

We will take into consideration the following important steps.

1) Initialize weights for the parameters we want to train.

2) Forward propagate through the network to get the output values.

3) Define the error or cost function and its first derivatives.

4) Backpropagate through the network to determine the error derivatives.

5) Update the parameter estimates using the error derivative and the current value.

The input and target values for this problem are $x_1 = 1$, $x_2 = 4$, $x_3 = 5$ and $t_1 = 0.1$, $t_2 = 0.05$. We initialize the weights as depicted in the diagram below. In general, we assign them randomly for illustration purposes. We chose the numbers as shown (Figure 7).

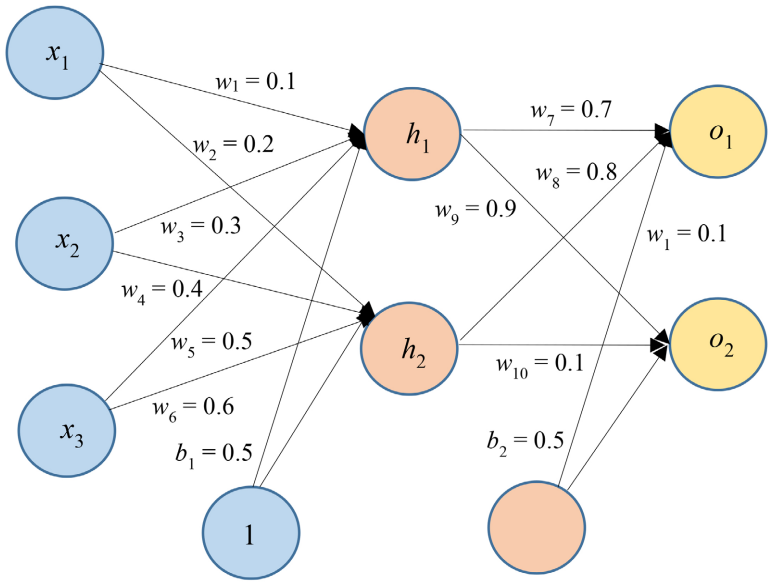

Figure 7. Schematic with parameter values of the network (one training iteration).

Mathematically, we have the following relationships between nodes in the networks. For the input and output layer, we prefer the convention of setting $z_{h_1}, z_{h_2}, z_{o_1}, z_{o_2}$ to denote the values before the activation function is applied and the notation of h_1, h_2, o_1, o_2 to denote the values after application of the activation function.

Input to hidden layer

$$\begin{aligned} &w_1x_1 + w_3x_2 + w_5x_3 + b_1 = z_{h_1},\ \ w_2x_1 + w_4x_2 + w_6x_3 + b_1 = z_{h_2}, \\ &h_1 = \sigma\left(z_{h_1}\right),\ \ h_2 = \sigma\left(z_{h_2}\right) \end{aligned} \tag{8}$$

Hidden layer to output layer

$$w_7h_1 + w_9h_2 + b_2 = z_{o_1},\ \ w_8h_1 + w_{10}h_2 + b_2 = z_{o_2},\ \ o_1 = \sigma\left(z_{o_1}\right),\ o_2 = \sigma\left(z_{o_2}\right) \tag{9}$$

We can use Equations (8) and (9) above to forward propagate through the network. We exhibit up to four decimal places below, but maintain all decimals in actual calculations.

$$w_1x_1 + w_3x_2 + w_5x_3 + b_1 = z_{h_1} = 0.1(1) + 0.3(4) + 0.5(5) + 0.5 = 4.3$$
$$h_1 = \sigma\left(z_{h_1}\right) = \sigma(4.3) = 0.9866$$
$$w_2x_1 + w_4x_2 + w_6x_3 + b_1 = z_{h_2} = 0.2(1) + 0.4(4) + 0.6(5) + 0.5 = 5.3$$
$$h_2 = \sigma\left(z_{h_2}\right) = \sigma(5.3) = 0.9950$$
$$w_7h_1 + w_9h_2 + b_2 = z_{o_1} = 0.7(0.9866) + 0.9(0.9950) + 0.5 = 2.0862$$
$$o_1 = \sigma\left(z_{o_1}\right) = \sigma(2.0862) = 0.8896$$
$$w_8h_1 + w_{10}h_2 + b_2 = z_{o_2} = 0.8(0.9866) + 0.1(0.9950) + 0.5 = 1.3888$$
$$o_2 = \sigma\left(z_{o_2}\right) = \sigma(1.3888) = 0.8004$$

We now define the sum of squares error (SSE) using the target values and the results from the last layer from forward propagation.

$$E = \frac{1}{2}\left[(o_1 - t_1)^2 + (o_2 - t_2)^2\right] \tag{10}$$

$$\Rightarrow \quad \frac{\partial E}{\partial o_1} = o_1 - t_1, \quad \frac{\partial E}{\partial o_2} = o_2 - t_2 \tag{11}$$

We can now backpropagate through the network to compute all the error derivatives with respect to the parameters. Note that in what follows we employ only the basic principles of calculus, which is the *chain rule* or function of a function rule. Furthermore, given that $w_7h_1 + w_9h_2 + b_2 = z_{o_1}$ and $w_8h_1 + w_{10}h_2 + b_2 = z_{o_2}$, we have the simple partial derivatives;

$$\frac{\partial z_{o_1}}{\partial w_7} = h_1, \frac{\partial z_{o_2}}{\partial w_8} = h_1, \frac{\partial z_{o_1}}{\partial w_9} = h_2, \frac{\partial z_{o_2}}{\partial w_{10}} = h_2, \frac{\partial z_{o_1}}{\partial b_2} = 1 \text{ and } \frac{\partial z_{o_2}}{\partial b_2} = 1 \tag{12}$$

$$\Rightarrow \quad \frac{\partial E}{\partial o_1} = o_1 - t_1, \quad \frac{\partial E}{\partial o_2} = o_2 - t_2 \tag{13}$$

We are now ready to compute $\partial E/\partial w_7$, $\partial E/\partial w_8$, $\partial E/\partial w_9$ and $\partial E/\partial w_{10}$ employing our previous derivatives; we have

$$\frac{\partial E}{\partial w_7} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial z_{o_1}}\frac{\partial z_{o_1}}{\partial w_7} = (o_1 - t_1)\left[o_1(1 - o_1)\right]h_1$$
$$= (0.8896 - 0.1)\left[0.8896(1 - 0.8896)\right](0.9866)$$
$$\therefore \quad \frac{\partial E}{\partial w_7} = 0.0765$$

Similarly, we have;

$$\frac{\partial E}{\partial w_8} = \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial z_{o_2}}\frac{\partial z_{o_2}}{\partial w_8} = (0.7504)(0.1598)(0.9866) = 0.1183$$

$$\frac{\partial E}{\partial w_9} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial z_{o_1}}\frac{\partial z_{o_1}}{\partial w_9} = (0.7896)(0.0983)(0.9950) = 0.0772$$

$$\frac{\partial E}{\partial w_{10}} = \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial z_{o_2}}\frac{\partial z_{o_2}}{\partial w_{10}} = (0.7504)(0.1598)(0.9950) = 0.1193$$

The error derivative of b_2 is a little bit more involved since changes to b_2 affect the error through both o_1 and o_2.

$$\frac{\partial E}{\partial b_2} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial z_{o_1}}\frac{\partial z_{o_1}}{\partial b_2} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial z_{o_2}}\frac{\partial z_{o_2}}{\partial b_2}$$
$$= (0.7896)(0.0983)(1) + (0.7504)(0.1598)(1) \quad (14)$$
$$\therefore \ \frac{\partial E}{\partial b_2} = 0.1975$$

In summary, we have computed numerical values for the error derivatives with respect to $w_7, w_8.w_9, w_{10}$, and b_2. We will now back-propagate one layer to compute the error derivatives of the parameters connecting the *input layer* to the *hidden layer*. These error derivatives are seven in number, namely;

$$\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \frac{\partial E}{\partial w_3}, \frac{\partial E}{\partial w_4}, \frac{\partial E}{\partial w_5}, \frac{\partial E}{\partial w_6} \text{ and } \frac{\partial E}{\partial b_1} \quad (15)$$

We will compute $\partial E/\partial w_1$, $\partial E/\partial w_3$ and $\partial E/\partial w_5$ first since they all flow through the h_1 node.

$$\therefore \ \frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial z_{h_1}}\frac{\partial z_{h_1}}{\partial w_1} \quad (16)$$

The computation of the first term on the right-hand side of the equation above is a bit more involved than previous calculations since h_1 affects the error through both o_1 and o_2. As usual chain rule, we have;

$$\frac{\partial E}{\partial h_1} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial z_{o_1}}\frac{\partial z_{o_1}}{\partial h_1} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial z_{o_2}}\frac{\partial z_{o_2}}{\partial h_1} \quad (17)$$
$$= (0.7896)(0.0983)(0.7) + (0.7504)(0.1598)(0.8) = 0.1502$$

Now, we proceed with the numerical values for the error derivatives above. These derivatives have already been computed above. Substituting the above into the formula for $\partial E/\partial w_1$, we obtain;

$$\frac{\partial E}{\partial w_1} = (0.1502)(0.0132)(1) = 0.0020 \quad (18)$$

The calculations for $\partial E/\partial w_3$ and $\partial E/\partial w_5$ are as follows

$$\frac{\partial E}{\partial w_3} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial z_{h_1}}\frac{\partial z_{h_1}}{\partial w_3} = (0.1502)(0.0132)(4) = 0.0079 \quad (19)$$

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial z_{h_1}}\frac{\partial z_{h_1}}{\partial w_5} = (0.1502)(0.0132)(5) = 0.0099 \quad (20)$$

We now compute $\partial E/\partial w_2$, $\partial E/\partial w_4$ and $\partial E/\partial w_6$ since they all flow through the h_2 node. We have;

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial z_{h_2}}\frac{\partial z_{h_2}}{\partial w_2} \quad (21)$$

The computation of the first term on the right hand side of the equation above is a bit more involved since h_2 affects the error through both o_1 and o_2.

$$\frac{\partial E}{\partial h_2} = \frac{\partial E}{\partial o_1}\frac{\partial o_1}{\partial z_{o_1}}\frac{\partial z_{o_1}}{\partial h_2} + \frac{\partial E}{\partial o_2}\frac{\partial o_2}{\partial z_{o_2}}\frac{\partial z_{o_2}}{\partial h_2} \tag{22}$$
$$= (0.7896)(0.0983)(0.9) + (0.7504)(0.1598)(0.1) = 0.0818$$

Substituting the above into Equation (17) for $\partial E/\partial w_2$ we obtain

$$\frac{\partial E}{\partial w_2} = (0.0818)(0.0049)(1) = 0.0004 \tag{23}$$

The computations for $\partial E/\partial w_4$ and $\partial E/\partial w_6$ are as follows;

$$\frac{\partial E}{\partial w_4} = \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial z_{h_2}}\frac{\partial z_{h_2}}{\partial w_4} = (0.0818)(0.0049)(4) = 0.0016 \tag{24}$$

$$\frac{\partial E}{\partial w_6} = \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial z_{h_2}}\frac{\partial z_{h_2}}{\partial w_6} = (0.0818)(0.0049)(5) = 0.0020 \tag{25}$$

The final error derivative we have to compute is $\partial E/\partial b_1$, which is done as follows;

$$\frac{\partial E}{\partial b_1} = \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial z_{h_1}}\frac{\partial z_{h_1}}{\partial b_1} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial z_{h_2}}\frac{\partial z_{h_2}}{\partial b_2} \tag{26}$$
$$= (0.1502)(0.0132)(1) + (0.0818)(0.0049)(1) = 0.00238$$

We now have computed all error derivatives and now proceed to make the parameter updates after the first iteration of backpropagation. We employ the learning rate of $\alpha = 0.01$. Thus;

$$w_1 = w_1 - \alpha\frac{\partial E}{\partial w_1} = 0.1 - (0.01)(0.0020) = 0.1,$$

$$w_2 = w_2 - \alpha\frac{\partial E}{\partial w_2} = 0.2 - (0.01)(0.0004) = 0.2$$

$$w_3 = w_3 - \alpha\frac{\partial E}{\partial w_3} = 0.3 - (0.01)(0.0016) = 0.2999,$$

$$w_4 = w_4 - \alpha\frac{\partial E}{\partial w_4} = 0.4 - (0.01)(0.0099) = 0.4$$

$$w_5 = w_5 - \alpha\frac{\partial E}{\partial w_5} = 0.5 - (0.01)(0.0099) = 0.4999,$$

$$w_6 = w_6 - \alpha\frac{\partial E}{\partial w_6} = 0.6 - (0.01)(0.0020) = 0.6$$

$$w_7 = w_7 - \alpha\frac{\partial E}{\partial w_7} = 0.7 - (0.01)(0.0765) = 0.6992,$$

$$w_8 = w_8 - \alpha\frac{\partial E}{\partial w_8} = 0.8 - (0.01)(0.1183) = 0.7988$$

$$w_9 = w_9 - \alpha\frac{\partial E}{\partial w_9} = 0.9 - (0.01)(0.0772) = 0.8992,$$

$$w_{10} = w_{10} - \alpha \frac{\partial E}{\partial w_{10}} = 0.1 - (0.01)(0.1193) = 0.0988$$

$$b_1 = b_1 - \alpha \frac{\partial E}{\partial b_1} = 0.5 - (0.01)(0.00238) = 0.499,$$

$$b_2 = b_2 - \alpha \frac{\partial E}{\partial b_2} = 0.5 - (0.01)(0.1975) = 0.4980$$

We remark that the quantities $\partial E/\partial w_j \quad j = 1, 2, \cdots, 10$ and $\partial E/\partial b_i \quad i = 1, 2$ are respectively the rate at which the error E is changing with respect to the weights w_j and the biases b_i. Having done all the above computations, it is necessary to repeat them many times until the error (SSE) decreases and the parameter estimates stabilize or converge to some values, as shown in Figure 8. This task, however, cannot be achieved manually as it is computationally intensive. We will therefore rely on a MATLAB program to execute these complex iterations for speed and accuracy. The MATLAB code is given in the Appendix below.

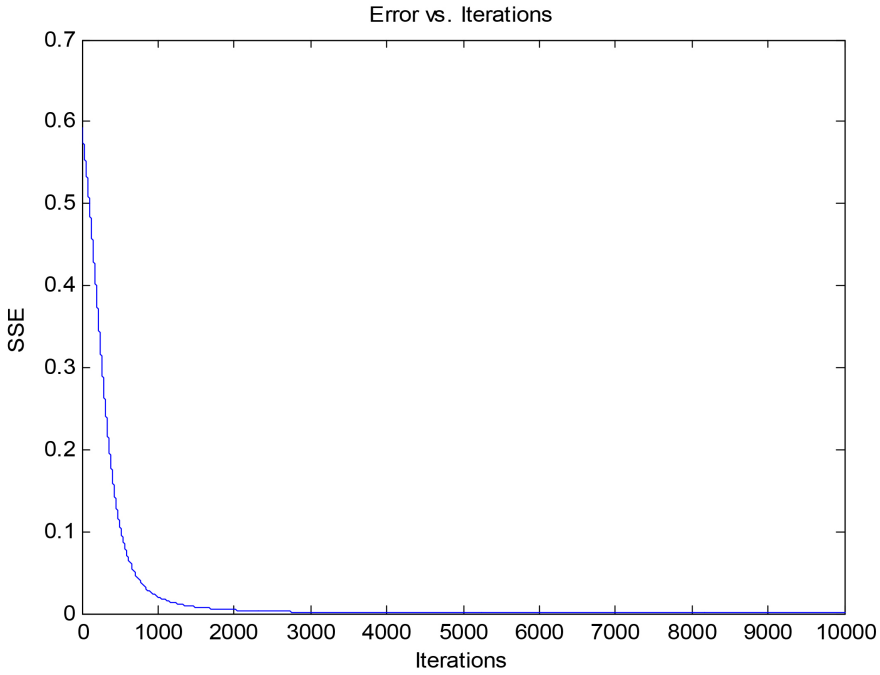

Figure 8. Sum of squares error after 10,000 iterations.

3. Conclusions

Backpropagation is central to Machine Learning, but the computational procedure can be daunting. In this research work, we have provided a detailed step-by-step computational procedure. Because this involves a lengthy iterative process, it is more convenient to codify these computations in MATLAB software, thereby enabling us to perform very high-order iterations. Our objective is to minimize the sum of squared errors involving the input values and the target values. This was achieved as displayed in the graphical profile of the SSE shown (Figure 8). Running the easy to understand Matlab program provides the parameter estimates for

the backpropagation algorithm (Appendix).

While backpropagation is extensively documented, tutorial implementations often skip intermediate gradient values or use vectorized code that obscures per-weight updates. Our contribution is pedagogical transparency, not algorithmic novelty.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Ian, G., Yoshua, B. and Aaron, C. (2020) Deep Learning. MIT Press.

[2] Werbos, P.J. (1974) Beyond Recognition, New Tools for Prediction and Analysis in the Behavioural Sciences. Ph.D. Thesis, Harvard University.

[3] Nielsen, M.A. (2023) Neural Networks and Deep Learning: A Textbook. Springer.

[4] Hinton, G.E. and Oriol, V. (2021) Deep Learning with Backpropagation: Progress and Challenges. *IEEE Transactions on Neural Networks and Learning Systems*, **32**, 1011-1026.

[5] Bengio, Y. and LeCun, Y. (2019) Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, **2**, 1-127. https://doi.org/10.1561/2200000006

[6] LeCun, Y., Bottou, L., Orr, G.B. and Müller, K.R. (1998) Efficient Backprop. In: *Lecture Notes in Computer Science*, Springer, 9-50. https://doi.org/10.1007/3-540-49430-8_2

[7] Kirsch, L. and Schmidhuber, J. (2021) Meta Learning Backpropagation and Improving It. *Proceedings of the* 35*th International Conference on Neural Information Processing Systems* (*NeurIPS* 2021), Online, 6-14 December 2021.

Appendix

Matlab Code for the Backpropagation Algorithm

```
numIter = 10000;
% Initialize variables
w1 = 0.1; w2 = 0.2; w3 = 0.3; w4 = 0.4; w5 = 0.5; w6 = 0.6; w7 = 0.7; w8 = 0.8; w9 = 0.9; w10 = 0.1;
wList = [w1, w2, w3, w4, w5, w6, w7, w8, w9, w10];
b1 = 0.5; b2 = 0.5;
bList = [b1, b2];
% Input and target values
x1 = 1; x2 = 4; x3 = 5;
xList = [x1, x2, x3];
t1 = 0.1; t2 = 0.05;
tList = [t1, t2];
% Learning rate
alpha = 0.01;
% Sigmoid function
sigmoid = @(x) 1 ./ (1 + exp(-x));
% Forward propagation function
forwardProp = @(xList, wList, bList) deal( ...
    sigmoid(wList(1) * xList(1) + wList(3) * xList(2) + wList(5) * xList(3) + bList(1)), ... % h1
    sigmoid(wList(2) * xList(1) + wList(4) * xList(2) + wList(6) * xList(3) + bList(1)), ... % h2
    sigmoid(wList(7) * sigmoid(wList(1) * xList(1) + wList(3) * xList(2) + wList(5) * xList(3) + bList(1)) + wList(9) * sigmoid(wList(2) * xList(1) + wList(4) * xList(2) + wList(6) * xList(3) + bList(1)) + bList(2)), ... % o1
    sigmoid(wList(8) * sigmoid(wList(1) * xList(1) + wList(3) * xList(2) + wList(5) * xList(3) + bList(1)) + wList(10) * sigmoid(wList(2) * xList(1) + wList(4) * xList(2) + wList(6) * xList(3) + bList(1)) + bList(2)) ...
);
% Error function (Sum of Squared Errors)
error = @(oList, tList) 0.5 * (sum((oList - tList).^2));
% Error list for tracking error over iterations
errList = [];
for i = 1:numIter
    % Forward propagation
    [h1, h2, o1, o2] = forwardProp(xList, wList, bList);
    % Compute error (SSE)
    sse = error([o1, o2], tList);
    errList = [errList, sse];
    % Display progress
    disp(['Running ' num2str(i) ' of ' num2str(numIter)]);
```

```
disp(['o1: ' num2str(o1)]);
disp(['t1: ' num2str(t1)]);
disp(['o2: ' num2str(o2)]);
disp(['t2: ' num2str(t2)]);
disp(['error: ' num2str(sse)]);
disp('    ');
% Backpropagation: Compute gradients
% dE/do1 and dE/do2 (Error derivatives w.r.t outputs)
dE_do1 = o1 - t1;
dE_do2 = o2 - t2;
% Derivatives of sigmoid (activation functions)
do1_dzo1 = o1 * (1 - o1);
do2_dzo2 = o2 * (1 - o2);
% Derivatives w.r.t weights from output layer to hidden layer
dzo1_dw7 = h1;
dzo2_dw8 = h1;
dzo1_dw9 = h2;
dzo2_dw10 = h2;
% Gradients for output layer weights and biases
dE_dw7 = dE_do1 * do1_dzo1 * dzo1_dw7;
dE_dw8 = dE_do2 * do2_dzo2 * dzo2_dw8;
dE_dw9 = dE_do1 * do1_dzo1 * dzo1_dw9;
dE_dw10 = dE_do2 * do2_dzo2 * dzo2_dw10;
% Gradients for bias at output layer
dE_db2 = dE_do1 * do1_dzo1 + dE_do2 * do2_dzo2;
% Gradients w.r.t hidden layer weights
dzo1_dh1 = w7;
dzo2_dh1 = w8;
dE_dh1 = dE_do1 * do1_dzo1 * dzo1_dh1 + dE_do2 * do2_dzo2 *
dzo2_dh1;
dh1_dzh1 = h1 * (1 - h1);
dzh1_dw1 = x1;
dzh1_dw3 = x2;
dzh1_dw5 = x3;
dE_dw1 = dE_dh1 * dh1_dzh1 * dzh1_dw1;
dE_dw3 = dE_dh1 * dh1_dzh1 * dzh1_dw3;
dE_dw5 = dE_dh1 * dh1_dzh1 * dzh1_dw5;
% Gradients for second hidden layer
dzo1_dh2 = w9;
dzo2_dh2 = w10;
dE_dh2 = dE_do1 * do1_dzo1 * dzo1_dh2 + dE_do2 * do2_dzo2 *
dzo2_dh2;
dh2_dzh2 = h2 * (1 - h2);
dzh2_dw2 = x1;
```

```
        dzh2_dw4 = x2;
        dzh2_dw6 = x3;
        dE_dw2 = dE_dh2 * dh2_dzh2 * dzh2_dw2;
        dE_dw4 = dE_dh2 * dh2_dzh2 * dzh2_dw4;
        dE_dw6 = dE_dh2 * dh2_dzh2 * dzh2_dw6;
        % Gradients for bias at hidden layer
        dE_db1 = dE_dh1 * dh1_dzh1 + dE_dh2 * dh2_dzh2;
        % Update all parameters
        w1 = w1 - alpha * dE_dw1;
        w2 = w2 - alpha * dE_dw2;
        w3 = w3 - alpha * dE_dw3;
        w4 = w4 - alpha * dE_dw4;
        w5 = w5 - alpha * dE_dw5;
        w6 = w6 - alpha * dE_dw6;
        w7 = w7 - alpha * dE_dw7;
        w8 = w8 - alpha * dE_dw8;
        w9 = w9 - alpha * dE_dw9;
        w10 = w10 - alpha * dE_dw10;
        b1 = b1 - alpha * dE_db1;
        b2 = b2 - alpha * dE_db2;
        wList = [w1, w2, w3, w4, w5, w6, w7, w8, w9, w10];
        bList = [b1, b2];
    end
    % Plot error graph
    figure;
    plot(1:numIter, errList);
    xlabel('Iterations');
    ylabel('SSE');
    title('Error vs. Iterations');
```

Cross section of output

Runs: 9255 of 10,000	Runs: 9256 of 10,000	Runs: 9257 of 10,000	Runs: 9258 of 10,000
o1: 0.101	o1: 0.101	o1: 0.10099	o1: 0.10099
t1: 0.1	t1: 0.1	t1: 0.1	t1: 0.1
o2: 0.065792	o2: 0.06579	o2: 0.065789	o2: 0.065787
t2: 0.05	t2: 0.05	t2: 0.05	t2: 0.05
error: 0.00012519	error: 0.00012516	error: 0.00012513	error: 0.0001251

Runs: 9259 of 10,000	Runs: 9260 of 10,000	Runs: 9261 of 10,000	Runs: 9258 of 10,000
o1: 0.10099	o1: 0.10099	o1: 0.10099	o1: 0.10099
t1: 0.1	t1: 0.1	t1: 0.1	t1: 0.1
o2: 0.065785	o2: 0.065783	o2: 0.065781	o2: 0.065787
t2: 0.05	t2: 0.05	t2: 0.05	t2: 0.05
error: 0.00012507	error: 0.00012505	error: 0.00012502	error: 0.0001251