Applied Science Innovation (ASI)
Published by College of Physical and Applied Sciences, MOUAU

- *Received:* 25 May 2026
- *Revised*: 30 May 2026
- *Accepted:* 10 June 2026
- *Published:* 16 June 2026

A NEW PERSPECTIVE FOR SOLVING POISSON EQUATIONS VIA ARTIFICIAL NEURAL NETWORK

Roseline N. Okereke

Department of Mathematics, Michael Okpara University of Agriculture, Umudike, Abia State, Nigeria.

***Corresponding author:** okereke.roseline@mouau.edu.ng

ABSTRACT

In this research work, a new method of solving Poisson equations using Artificial Neural Networks (ANN) was employed. Literature reveals that many works have been done in this area, but most conventional methods of solution depend on weight updating to finding approximate solutions, which actually are cumbersome in nature. The new method uses a systematic procedure in obtaining the weights, and bypasses weight updating. As the method aims at minimizing the error in approximation, which is a function of the network parameters, weights and biases, the use of MathCAD 14 software slightly adjusted the intrinsic biases only, which are involved in solving Poisson equations. Feed-forward Multilayer Perceptron Neural Network (MLPNN) was employed, but this method avoids the standard back-propagation algorithm which involves weight updating. Here, a three-layer perceptron neural network, with two input units, nine hidden units and one output unit was adopted. The neural network is trained to achieving convergence with an acceptable minimum error. Poisson equation was considered, and Gaussian Radial Basis Function (GRBF) approximation model was applied to achieve the objective. The result of the research work was compared with exact and existing neural network results which are in good agreement. The major advantage here is that the novel artificial neural networks procedure applied in this study demonstrated quite clearly the function approximation capabilities in maintaining satisfactory accuracy.

Keywords: Poisson Equations, GRBF, Artificial Neural Network, MLPNN, MathCAD 14.

1. Introduction

A new perspective for solving Poisson equations using Artificial Neural Network (ANN) is presented. This is an extension of the procedure developed by (Okereke, 2019), (Okereke and Maliki, 2020, 2021), (Okereke et al, 2021), and (Okereke, 2024). This novel approach was extensively dealt with, but was limited to ordinary differential equations and boundary value problems. This research work extends this new approach to partial differential equations (PDE), which has a number of advantages over standard numerical methods. Neural network methods rely on the function approximation seen in feed- forward neural networks that results in a solution written in an analytic form which has unlimited differentiability (Principe *et al.*, 1997). So, Poisson equations are being solved via neural network, with the help of GRBF approximation.

The major advantage here is that the method reduces considerably the computational complexity involved in weight updating, while maintaining satisfactory accuracy.

1.1 The Structure of Neural Network

A neural network is an inter-connection of processing elements, units or nodes, whose functionality resemble that of the human neurons. The processing ability of the network is stored in the connection strengths, simply called weights, which can be obtained by a process of adaptation to, a set of training patterns (Graupe, 2007). Neural network methods can solve both ordinary and partial differential equations. Furthermore, it relies on the function approximation property of feed-forward neural networks which results in a solution written in a closed analytic form. This form employs a feed-forward neural network as a basic approximation element. Training of the neural network can be done either by any optimization technique which in turn requires the computation of the gradient of the error with respect to the network parameters, by regression-based model or by basis function approximation.

1.2 Mathematical Model Of ANN

ANNs are constructed with layers of units, which are termed *multilayer* ANNs. A layer of units in an ANN is made up of units that perform similar tasks. First layer of a multilayer ANN consists of input units. These units perform similar tasks as independent variables in statistical literature. Last layer contains output units known as dependent or response variables in statistical nomenclature. Between input and output units in the model are other units called hidden units and constitute hidden layers.

Each input is weighted before reaching the main body of a neuron N_i by connection strength or weight factor w_{ij} for $j = 1, 2, ..., n$. In addition, it has a bias b or w_0, which helps the model to shift the activation function towards the positive or negative side in order to incorporate functions passing outside the origin. A function $f(s)$ acts on the produced weighted signal. This function is called an *activation function***.** Mathematically, the output of the i^{th} neuron N_i is given by;

$$O_i = f\left[w_0 + \sum_{j=1}^{n} w_{ij} x_j \right] \qquad (1)$$

(Yadav *et al*., 2015).

1.3 Function Approximation

The potentials of function approximation is used in this work which attempts to describe the behaviour of very complicated functions by sets of simpler functions. The concept of function approximation can be stated briefly as follows;

Let $f(x)$ be a real function of a real valued vector $x=[x_1\ x_2 \ldots x_d]^T$ that is square integrable in $\mathbb{R}$.
The goal of function approximation is to describe the behaviour of $f(x)$ in a compact area $S \subset \mathbb{R}$ of the input space, by a combination of simpler functions $\phi_i(x)$, i.e.

$$\hat{f}(x,w)=\sum_{i=1}^{N} w_i \phi_i(x) \tag{2}$$

where w_i are real valued constants such that

$$\left| f(x)-\hat{f}(x,w) \right| < \varepsilon \tag{3}$$

and where ε can be made arbitrarily small. The function $\hat{f}(x,w)$ is called an approximant to $f(x)$. When we obtain the elementary functions, $\{\phi_i(x)\}$, we linearly combine them such that

$$\left| f(x)-\hat{f}(x,w) \right| < \varepsilon$$

is made as small as possible (Principe *et al.*, 1997)

1.4 Universal Approximation Theorem for Multi-Layer Perceptron (MLP)

The universal approximation theorem for MLP, which was proved by (Cybenco,1989) and (Hornik *et al.*, 1989) in 1989 is stated as follows: Let I_n represents n – dimensional unit cube containing all possible input samples *x*, that is, $x_i \in [0,1], i=1,2,...,n$, and $C(I_n)$ be the space of continuous functions on I_n. If $\sigma(.)$ is a continuous sigmoid function, then the finite sums of the form

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

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum $y(x,w)$ of the form (4) that satisfies $\left| y(x,w)-f(x) \right| < \varepsilon$ for all $x \in I_n$. As such, there always exists a 3-layer perceptron that can approximate an arbitrary nonlinear, continuous, multi-dimensional function f with any desired accuracy (Zhang, 2003).

1.5 Computation of Gradients with respect to Network Inputs

Boundary conditions in BVPs sometimes involve the derivative of the unknown function. This directly affects the neural network solution which will require the computation of the derivative of the neural network algorithm with respect to its inputs (Majidzadeh, 2011).
We now consider a multilayered perceptron with two input nodes, a hidden layer with m nodes, and one output unit. For the given inputs $x=(x_1,x_2,...,x_n)$, the output is given by

$$N(x,p)=\sum_{j=1}^{m} v_j \sigma(z_j) \tag{5}$$

where $z_j=\sum_{i=1}^{n} w_{ji}x_i+u_j, w_{ji}$ denotes the weight from input unit *i* to the hidden unit *j*, v_j denotes weight from the hidden unit *j* to the output unit, u_j denotes the biases, and $\sigma(z_j)$ is the sigmoid activation function. The derivative of the network output $N(x,p)$ with respect to input vector x_i is given by

$$\frac{\partial N}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\sum_{j=1}^{m} v_j \sigma(z_j)\right) = \sum_{j=1}^{m} v_j w_{ji} \sigma_j^{(1)}, \tag{6}$$

Where $\sigma^{(1)} \equiv \frac{\partial \sigma(z)}{\partial x}$. Likewise, the k^{th} derivative of N is computed as follows:

$$\frac{\partial^k N}{\partial x_i^k} = \sum_{j=1}^{m} v_j w_{ji}^k \sigma_j^{(k)}, \tag{7}$$

where $\sigma \equiv \sigma\left(z_j\right)$ and $\sigma^{(k)}$ denotes the k^{th} order derivative of sigmoid activation function.

2 General Formulation For Differential Equations

We consider the following general differential equations which represent both ordinary and partial differential equations Majidzadeh (2011):

$$G\left(x, \psi(x), \nabla\psi(x), \nabla^2\psi(x), ...\right) = 0, \quad \forall x \in D, \tag{8}$$

subject to some initial or boundary conditions, where $x = (x_1, x_2, ..., x_n) \in D \subset \mathbb{R}^n$ denotes the domain, and $\psi(x)$ is the unknown scalar-valued solution to be computed. Here, G is the function which defines the structure of the differential equation, and ∇ is a differential operator. Let $\psi_t(x, p)$ denote the trial solution with parameters (weights, biases) p . Lagaris *et al.* (1997) gave the following as the general formulation for the solution of the differential equations (8) using ANN: The trial solution, $\psi_t(x, p)$ may be written as the sum of two terms

$$\psi_t(x, p) = A(x) + F\left(x, N(x, p)\right), \tag{9}$$

where $A(x)$ satisfies initial or boundary conditions and contains no adjustable parameters, whereas $N(x, p)$ is the output of feed forward neural network with the parameters p and input data x. The function $F\left(x, N(x, p)\right)$ is actually the operational model of the neural network. Feed forward neural network (FFNN) converts differential equation problem to function approximation problem. The neural network $N(x, p)$ is given by

$$N(x, p) = \sum_{j=1}^{m} v_j \sigma\left(z_j\right) \tag{10}$$

where

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + u_j, \tag{11}$$

w_{ji} denotes the weight from input unit i to the hidden unit j , v_j denotes weight from the hidden unit j to the output unit, u_j denotes the biases, and $\sigma\left(z_j\right)$ is the sigmoid activation function.

2.1 Neural Network Training

The neural network weights determine the closeness of predicted outcome to the desired outcome. If the neural network weights are not able to make the correct prediction, then only the biases need to be

adjusted. The basis function we shall apply in this work in training the neural network is the sigmoid activation function given by

$$\sigma\left(z_j\right)=\frac{1}{1+e^{-z_j}} \tag{12}$$

3. Materials, Theory and Methods

3.1 Neural Network Method of Solving Poisson Equations

Partial differential equations involve two or more independent variables. Here, only two-dimensional problems involving only two independent variables are considered.

Lagaris *et al* (1997) proposed a fixed or Dirichlet boundary conditions (BC) defined on the domain $[0,1]\times[0,1]$ for such problems, which was also adopted in this new method. Also, a mesh of 100 points which resulted in considering 10 equidistant points of the domain $[0,1]$ of each variable was considered. Similar to the work done by (Okereke, 2024), a MLP was adopted, but here with two input units (accepting the coordinates of x and y of each point), 9 hidden units of the hidden layer, and one output unit. For Poisson equation

$$\frac{\partial^2}{\partial x^2}\varphi(x,y)+\frac{\partial^2}{\partial y^2}\varphi(x,y)=f(x,y), \tag{13}$$
$$x\in[0,1], y\in[0,1]$$

With Dirichlet BCs: $\varphi(0,y)=f_0(y), \varphi(1,y)=f_1(y)$ *and* $\varphi(x,0)=g_0(x),\ \varphi(x,1)=g_1(x)$, (14)

Lagaris *et al*, (1997) proposed a trial solution written as:

$$\varphi_t(x,y)=A(x,y)+x(1-x)y(1-y)N\left(x,y,\vec{p}\right) \tag{15}$$

where $A(x,y)$ is chosen so as to satisfy the BCs:

$$A(x,y)=(1-x)f_0(y)+xf_1(y)+((1-y)\left\{g_0(x)-\left[(1-x)g_0(0)+xg_0(1)\right]\right\} \tag{16}$$
$$+y\left\{g_1(x)-\left[(1-x)g_1(0)+xg_1(1)\right]\right\},$$

and $N\left(x,y,\bar{p}\right)$ is the neural output of the feed forward network with two input data x and y and parameters p (p represents the parameters – weights and biases). To solve this problem using neural network (NN), a NN architecture with three layers - one input layer with two neurons; one hidden layer with nine neurons and one output layer with one output unit was employed as shown in figure 3.1. Each neuron is connected to other neurons of the previous layer through adaptable synaptic weights w_{ij}, from the input layer to the hidden layer, v_j, from hidden layer to the output layer and biases u_j. With

$$N(x_i,y_i,p)=\sum_{j=1}^{9}v_j\sigma\left(z_j\right), \tag{17}$$

$$\sum_{j=i}^{9} v_j \sigma(z_j) = v_1\sigma(z_1) + v_2\sigma(z_2) + v_3\sigma(z_3) + \ldots + v_9\sigma(z_9), \tag{18}$$

where

$$\begin{aligned} z_1 &= xw_{11} + yw_{21} + u_1,\ z_2 = xw_{12} + yw_{22} + u_2, \\ z_3 &= xw_{13} + yw_{23} + u_3,\ \ldots,\ z_9 = xw_{19} + yw_{29} + u_9 \end{aligned} \tag{19}$$

In finding solutions to any partial differential equation, a check that the trial solution satisfies the boundary conditions, in this case the Dirichlet conditions is carried out. $f(x)$ is found such that $w = \varphi^{-1} f$; and $v = \varphi^{-1} f$; where w and v become vectors with the coefficients, f is a matrix composed of the values of the functions at the N points, and φ the matrix with entries given by values of the elementary functions at each of the N points in the domain.

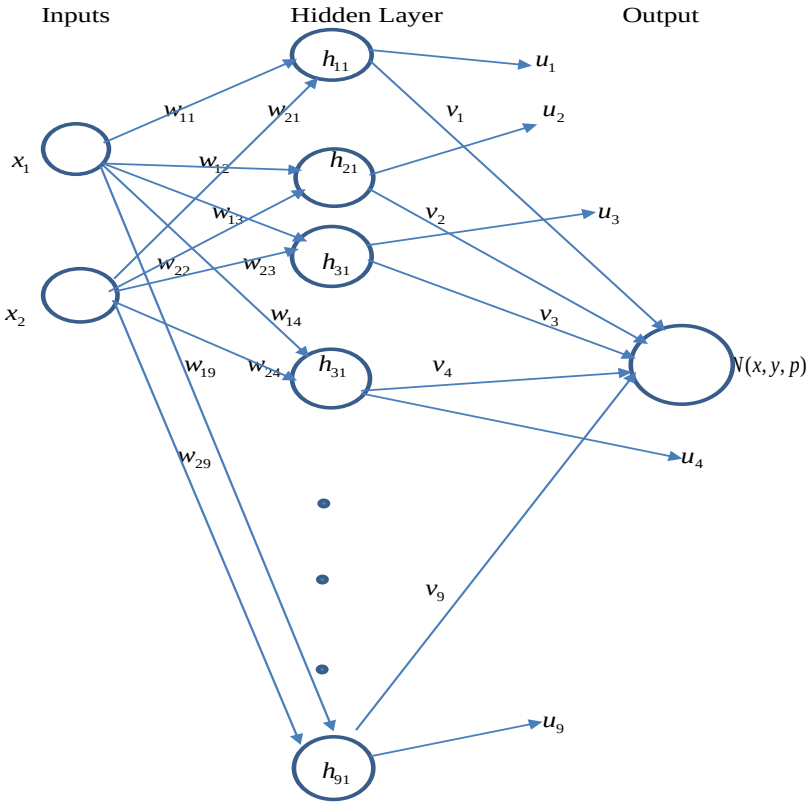


Figure 3.1 Schematic for $N(x,y,p)\ [x_1,x_2,x_3]\times[y_1,y_2,y_3] = [0.0, 0.5, 1.0]\times[0.0, 0.5, 1.0]$.

To compute the weights $w_{ij}, i, j = 1,2,3$ from the input layer to the hidden layer (Figure 3.1), a function $F_1(x,y)$ is constructed such that $w = \varphi^{-1} F_1$, where $(x,y) = [x_1,x_2,x_3]^T \times [y_1,y_2,y_3]^T$ are the coordinates of

$$\begin{bmatrix} w_1 \\ \vdots \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_N(x_1) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_1(x_N) & \varphi_2(x_N) & \cdots & \varphi_N(x_N) \end{bmatrix}^{-1} \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ f_{N1} & f_{12} & \cdots & f_{NN} \end{bmatrix} \tag{20}$$

To obtain the weights, v_j, $j=1,2,...,9$ from hidden layer to the output layer, another function $F_2(x,y)$ also is constructed such that $v=\varphi^{-}F_2$, $(x,y)=[x_1,x_2,x_3]^T\times[y_1,y_2,y_3]^T$ And

$$\begin{bmatrix} v_1 \\ \vdots \\ \vdots \\ v_N \end{bmatrix} = \begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_N(x_1) \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_1(x_N) & \varphi_2(x_N) & \cdots & \varphi_N(x_N) \end{bmatrix}^{-1} \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1N} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ f_{N1} & f_{12} & \cdots & f_{NN} \end{bmatrix} \tag{21}$$

The exact form of $F_1(x,y)$ depends on the nature of a given differential equation. So, an algebraic equation is formed based on the default sign of the differential equation. Therefore,
$F_1(x,y)=ax^2+bxy+cy^2+dx+ey+g$, *where* $a,b,c,d,e,$ and g are arbitrary constants. Also, $F_2(x,y)$ is computed using the forcing function of the differential equation.
And Using a set of functions, $\{\varphi_i(x)\}$ called the elementary functions, yields.

$$\varphi_i(x)=\exp\left[-\left(\frac{\left(|x-x_i|\right)^2}{2\sigma^2}\right)\right], \tag{22}$$

where $\sigma^2=\frac{1}{N}\sum_{i=1}^{n}(x_i-\bar{x})^2=\frac{1}{N}\sum_{i=1}^{n}(y_i-\bar{y})^2$ (23)

is the variance of the input vectors x,y, and

$$\bar{x}=\frac{1}{N}\sum_{i=1}^{n}x_i=\bar{y} \tag{24}$$

The above is the so-called Gaussian Radial Basis function (GRBF) approximation model.

4. Results and Discussion

The main results which demonstrated the efficiency of the proposed neural network in solving Poisson Equations is presented here. Numerical Example solved by Lagaris et al, (1997) using neural network algorithm that updates all the parameters- weights and biases, is presented here with a method that bypasses weights updating, but slightly adjusts the biases. Solve the equation

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=e^{-x}\left(x-2+y^3+y\right),\ x,y\in[0,1] \tag{25}$$

with the Dirichlet BCs:

$$u(0,y)=y^3,\ u(1,y)=(1+y^3)e^{-1}\ \ and\ \ u(x,0)=xe^{-x},\ u(x,1)=e^{-x}(x+1) \tag{26}$$

Implementing the BC below:

$u(0,y)=f_0(y),\ u(1,y)=f_1(y)$ *and* $u(x,0)=g_0(x),\ u(x,1)=g_1(x)$, we have exact solution:

$$u_d(x,y)=e^{-x}(x+y^3) \tag{27}$$

Neural network trial solution:

$$u_p(x,y) = A(x,y) + x(1-x)y(1-y)N(x,y,p) \tag{28}$$

where

$$A(x,y) = (1-x)f_0(y) + xf_1(y) + (1-y)\left\{g_0(x) - \left[(1-x)g_0(0) + xg_0(1)\right]\right\} + y\left\{g_1(x) - \left[(1-x)g_1(0) + xg_1(1)\right]\right\} \tag{29}$$

Applying the BCs gives

$$u_p(x,y) = (1-x)y^3 + x(1+y^3)e^{-1} + (1-y)x(e^{-x} - e^{-1}) + y\left[(1+x)e^{-x} - (1-x+2xe^{-1})\right] + x(1-x)y(1-y)N(x,y,p) \tag{30}$$

The trial solution satisfies the BCs, therefore we consider the left hand side of the partial differential equation (25) and form an algebraic equation with the default sign. We shall use the GRBF here for the weights as demonstrated by (Okereke, 2019).

First, let $F_1(x) = ax^2 + bxy + cy^2 + dx + ey + g$, *where* $a,b,c,d,e,$ and g are arbitrary constants.

From the left-hand side of equation (i), $a=1,\ b=0,\ c=1,\ d=0,\ e=0,$ and $\mathrm{g}=0$

Therefore, $F_1(x_i,y_j) = x_i^2 + y_j^2,\ \ i,j=1,2,3;\ x_i = 0.0,0.5,1.0;\ y_j = 0.0,0.5,1.0$.

With $(x_i,y_j) = [x_1,x_2,x_3]\times[y_1,y_2,y_3] = [0.0,0.5,1.0]\times[0.0,0.5,1.0]$

$$F_1(x,y) = \begin{bmatrix} 0.00 & 0.25 & 1.00 \\ 0.25 & 0.50 & 1.25 \\ 1.00 & 1.25 & 2.00 \end{bmatrix}$$

$F_2(x_i,y_j) = e^{-x_i}\left(x_i - 2 + y_j^3 + 6y_j\right),\ \ i,j=1,2,3;\ x_i = 0.0,0.5,1.0;\ y_j = 0.0,0.5,1.0$

Therefore,

$$F_2(x_i,y_j) = \begin{bmatrix} -2.0000 & 1.1200 & 5.0000 \\ -0.9098 & 0.9856 & 3.3359 \\ -0.3679 & 0.7817 & 2.2073 \end{bmatrix}$$

$\varphi_{ij}\left(x_i,y_j\right) = 2\exp\left(\dfrac{-\left(\left|x_i - y_j\right|\right)^2}{2\sigma^2}\right), i,j=1,2,3;\ x_i = 0.0,0.5,1.0;\ y_j = 0.0,0.5,1.0$. Thus,

$$\varphi_{ij}\left(x_i,y_j\right) = \begin{pmatrix} 2.0000 & 0.4395 & 4.6660 \\ 0.4395 & 2.0000 & 0.4395 \\ 4.6660 & 0.4395 & 2.0000 \end{pmatrix},\quad \varphi^{-1}\left(x_i,y_j\right) = \begin{pmatrix} -0.11 & -0.034 & 0.265 \\ -0.034 & 0.515 & -0.034 \\ 0.265 & -0.034 & -0.11 \end{pmatrix}.$$ And

$$w_{ij} = \varphi^{-1}f_1 = \begin{pmatrix} -0.11 & -0.034 & 0.265 \\ -0.034 & 0.515 & -0.034 \\ 0.265 & -0.034 & -0.11 \end{pmatrix}\begin{bmatrix} 0.00 & 0.25 & 1.00 \\ 0.25 & 0.50 & 1.25 \\ 1.00 & 1.25 & 2.00 \end{bmatrix} = \begin{bmatrix} 0.256 & 0.286 & 0.377 \\ 0.095 & 0.207 & 0.542 \\ -0.119 & -0.089 & 0.0017 \end{bmatrix},\ i=1,2,\ j=1,2,...,9$$

Where $f_1 = F_1(x_i,y_j)$. Also,

$$v_j = \varphi^{-1} f_2 = \begin{pmatrix} -0.11 & -0.034 & 0.265 \\ -0.034 & 0.515 & -0.034 \\ 0.265 & -0.034 & -0.11 \end{pmatrix} \begin{bmatrix} -2.0000 & 1.1200 & 5.0000 \\ -0.9098 & 0.9856 & 3.3359 \\ -0.3679 & 0.7817 & 2.2073 \end{bmatrix} = \begin{bmatrix} 0.154 & 0.05 & -0.08 \\ -0.388 & 0.443 & 1.473 \\ -0.458 & 0.177 & 0.967 \end{bmatrix}, \; j = 1,2,...9$$

where $f_2 = F_2(x_i, y_j)$ and i, j represents the number of input units and the number of neurons in the hidden layer respectively. The biases are fixed between -1.0 and 2.0. The network is trained with the available parameters using the MathCAD 14 algorithm (computer output). The MathCAD 14 Algorithm is displayed in Appendix A, for a better visibility and understanding of the equations. The result from the developed model which was executed by MathCAD 14 shows almost a perfect match with the exact result as can be seen in Table 4.1 below. Looking closely at Table 4.1, it can be seen that it is only at point (0.5,0.5) that a small deviation occurred. This result was also compared with the existing neural network result mentioned in this work, and found to be a better approximation. See also Table 4.2. Let the existing neural network solution be denoted by $U_e(x, y)$.

Table 4.1 Comparison of Exact and Predicted Results

x	y	$U_d(x,y)$	$U_p(x,y)$	Deviations $U_d - U_p$
0.0	0.0	0.0	0.0	0.00
0.0	0.5	0.125	0.125	0.00
0.0	1.0	1.00	1.00	0.00
0.5	0.0	0.303281	0.303281	0.00
0.5	0.5	0.379101	0.3791	1.5×10^{-12}
0.5	1.0	0.909843	0.909843	0.00
1.0	0.0	0.367918	0.367918	0.00
1.0	0.5	0.413907	0.413907	0.00
1.0	1.0	0.735835	0.735835	0.00

The result in Table 4.2 was also simulated to have better view of the comparisons. See Figure 4.1. The closeness of the result from the novel approach to the exact was established.

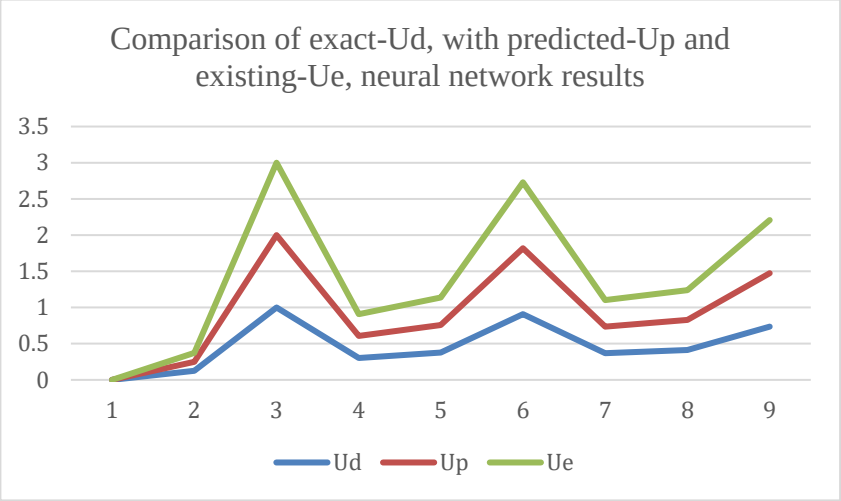


Figure 4.1 Comparison of Exact Result with Predicted and Existing Neural Network Result

Table 4.2 Comparison of exact, predicted and existing neural network results

x	y	$U_d(x,y)$	$U_p(x,y)$	Deviations $U_d - U_p$	$U_e(x,y)$	Deviations $U_d - U_e$
0.0	0.0	0.0	0.0	0.00	0.000001	1.0×10^{-6}
0.0	0.5	0.125	0.125	0.00	0.124998	2.0×10^{-6}
0.0	1.0	1.00	1.00	0.00	0.999996	4.0×10^{-6}
0.5	0.0	0.303281	0.303281	0.00	0.303261	4.0×10^{-6}
0.5	0.5	0.379101	0.3791	1.5×10^{-12}	0.379076	5.0×10^{-6}
0.5	1.0	0.909843	0.909843	0.00	0.909789	7.0×10^{-6}
1.0	0.0	0.367918	0.367918	0.00	0.367873	6.0×10^{-6}
1.0	0.5	0.413907	0.413907	0.00	0.413857	7.0×10^{-6}
1.0	1.0	0.735835	0.735835	0.00	0.735750	9.0×10^{-6}

5. Conclusion

The novel neural network method presented here, which bypasses weights updating, closely approximates the exact solution to the given Poisson equation, and performed even better than the convectional result obtained by weights updating, as seen in the tables and the figure. However, this work can still be extended to incorporate other types of partial differential equations not covered here, like heat, wave and Navier-Stokes equation. It can also be extended to solving integral equations with some additional effort.

Conflicts of Interest
The author declares that there is no conflicts of interest regarding the publication of this paper.

Acknowledgement: I appreciate Dr. S. O. Maliki who guided me on the simulation of the Table of comparison

References

Cybenco, G. (1989). Approximation by Superposition of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2, 303-314. https://doi.org/10.1007/BF02551274Graupe, D. (2007). *Principles of Artificial Neural Networks (2nd Edition). Advanced Series on Circuits and Systems* – World Scientific Publishing Co. Pte. Ltd., Singapore, Vol. 6

Hornic, K., Stinchcombe, M. and White, H. (1989). Multilayer Feedforward Networks Are Universal Approximators. Neural Networks, 2, 359-366. https://doi.org/10.1016/0893-6080(89)90020-8

Lagaris, I.E., Likas, A.C. and Fotiadis, D.I. (1997). Artificial Neural Network for Solving Ordinary and Partial Differential Equations. Department of Computer Science, University of Ioannina, Greece. GR 45110 *arXiv: physics/9705023v1*.

Lagaris, I. E., Likas, A. C. and Papageorgiou, D. G. (2004). Neural network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11: 1041–1049

Majidzadeh, K. (2011). Inverse Problem with Respect to Domain and Artificial Neural Network Algorithm for the Solution. *Mathematical Problems in Engineering*, 2011, Article ID: 145608, 16 p.

Okereke, R.N. (2019) A New Perspective to the Solution of Ordinary Differential Equations using Artificial Neural Networks. [Unpublished Ph.D Dissertation] Mathematics Department, Michael Okpara University of Agriculture, Umudike, Nigeria.

Okereke, R. N. and Maliki, O. S. (2020). A New Perspective for Solving Third Order Initial Value Problems Using Artificial Neural Networks. *Transactions of the Nigerian Association of Mathematical Physics*, Volume 11,(January-June 2020 Issue),pp 123-130. Published by NAMP

Okereke, R. N. and Maliki, O. S. (2020). A Novel Approach for Solving Variable Coefficient Initial Value Problems Using Artificial Neural Networks. *Transactions of the Nigerian Association of Mathematical Physics*, Volume 12,(July- Sept., 2020 Issue), pp 89-96. Published by NAMP

Okereke, R. N., Maliki, O. S. and Oruh, B. I. (2021). A Novel Method for Solving Ordinary Differential Equations with Artificial Neural Networks. *Applied Mathematics Electronic Journal. Scientific Research P u b l i s h i n g I n c , U S A* . https://doi.org/10.4236/am.2021.1210059

Okereke, R. N. and Maliki, O. S. (2021). Solving Riccati-Type Nonlinear Differential Equations with Novel Artificial Neural Networks. *Applied Mathematics Electronic Journal. Scientific Research P u b l i s h i n g I n c , U S A.*

Okereke, R. N. (2024). A New Method for Solving Boundary Value Problems Via Artificial Neural Network. *Ambrose Alli University Journal of Physical and Applied Sciences (AAUJPAS). Published by Faculty of Physical Sciences*, Ambrose Ali University, Ekpoma. May 2024

Principe, J. C., Euliano, N. R. and Lefebvre, W. C. (1997). Neural and Adaptive Systems: Fundamentals Through Simulation. *PTC (Parametric Technology Corporation*) (2007) Mathcad Version 14.

Yadav, N., Yadav, A. and Kumar, M. (2015). *An Introduction to Neural Network Methods for Differential Equations.* Springer Zill, Dennis G. and Cullen, Michael R. (2009). *Differential Equations with Boundary Value Problems*. Seventh Edition, Brooks/Cole, Cengage Learning, USA.

APPENDIX A

MathCAD 14 Algorithm is displayed here for a better visibility and understanding of the equations.

$w_{11} := 0.256 \quad w_{12} := 0.286 \quad w_{13} := 0.377 \quad w_{14} := 0.095 \quad w_{15} := 0.207 \quad w_{16} := 0.542$

$w_{17} := -0.119 \quad w_{18} := -0.089 \quad w_{19} := 0.0017$

$w_{21} := 0.256 \quad w_{22} := 0.286 \quad w_{23} := 0.377 \quad w_{24} := 0.095 \quad w_{25} := 0.207 \quad w_{26} := 0.542$

$w_{27} := -0.119 \quad w_{28} := -0.089 \quad w_{29} := 0.0017$

$v_1 := 0.154 \quad v_2 := 0.05 \quad v_3 := -0.08 \quad v_4 := -0.388 \quad v_5 := 0.443 \quad v_6 := 1.473$

$v_7 := -0.458 \quad v_8 := 0.177 \quad v_9 := 0.967$

$e := 2.718$

$x := 0.0 \quad y := 0.5 \quad u_1 := 1 \quad u_2 := -1 \quad u_3 := 0.39201 \quad u_4 := 1 \quad u_5 := 0.5 \quad u_6 := -1$

$u_7 := 0.7439 \quad u_8 := 1 \quad u_9 := 1$

$z_1 := w_{11} \cdot x + w_{21} \cdot y + u_1 = 1.128 \quad z_2 := w_{12} \cdot x + w_{22} \cdot y + u_2 = -0.857 \quad z_3 := w_{13} \cdot x + w_{23} \cdot y + u_3 = 0.58051$

$z_4 := w_{14} \cdot x + w_{24} \cdot y + u_4 = 1.0475 \quad z_5 := w_{15} \cdot x + w_{25} \cdot y + u_5 = 0.6035 \quad z_6 := w_{16} \cdot x + w_{26} \cdot y + u_6 = -0.729$

$z_7 := w_{17} \cdot x + w_{27} \cdot y + u_7 = 0.6844 \quad z_8 := w_{18} \cdot x + w_{28} \cdot y + u_8 = 0.9555 \quad z_9 := w_{19} \cdot x + w_{29} \cdot y + u_9 = 1.00085$

$$\sigma(z_1) := \frac{1}{1 + \exp(-w_{11} \cdot x - u_1 - w_{21} \cdot y)} \qquad \frac{1}{1 + \exp(-w_{11} \cdot x - u_1 - w_{21} \cdot y)} = 0.75547$$

$$\sigma(z_2) := \frac{1}{1 + \exp(-w_{12} \cdot x - u_2 - w_{22} \cdot y)} \qquad \frac{1}{1 + \exp(-w_{12} \cdot x - u_2 - w_{22} \cdot y)} = 0.297967$$

$$\sigma(z_3) := \frac{1}{1 + \exp(-w_{13}\cdot x - u_3 - w_{23}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{13}\cdot x - u_3 - w_{23}\cdot y)} = 0.641185$$

$$\sigma(z_4) := \frac{1}{1 + \exp(-w_{14}\cdot x - u_4 - w_{24}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{14}\cdot x - u_4 - w_{24}\cdot y)} = 0.740295$$

$$\sigma(z_5) := \frac{1}{1 + \exp(-w_{15}\cdot x - u_5 - w_{25}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{15}\cdot x - u_5 - w_{25}\cdot y)} = 0.646457$$

$$\sigma(z_6) := \frac{1}{1 + \exp(-w_{16}\cdot x - u_6 - w_{26}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{16}\cdot x - u_6 - w_{26}\cdot y)} = 0.325414$$

$$\sigma(z_7) := \frac{1}{1 + \exp(-w_{17}\cdot x - u_7 - w_{27}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{17}\cdot x - u_7 - w_{27}\cdot y)} = 0.66472$$

$$\sigma(z_8) := \frac{1}{1 + \exp(-w_{18}\cdot x - u_8 - w_{28}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{18}\cdot x - u_8 - w_{28}\cdot y)} = 0.72222$$

$$\sigma(z_9) := \frac{1}{1 + \exp(-w_{19}\cdot x - u_9 - w_{29}\cdot y)} \qquad \frac{1}{1 + \exp(-w_{19}\cdot x - u_9 - w_{29}\cdot y)} = 0.731226$$

$$\sigma(z_1) := \frac{1}{1 + \exp(-z_1)} \quad \sigma(z_2) := \frac{1}{1 + \exp(-z_2)} \quad \sigma(z_3) := \frac{1}{1 + \exp(-z_3)} \quad \sigma(z_4) := \frac{1}{1 + \exp(-z_4)} \quad \sigma(z_5) := \frac{1}{1 + \exp(-z_5)}$$

$$\sigma(z_6) := \frac{1}{1 + \exp(-z_6)} \quad \sigma(z_7) := \frac{1}{1 + \exp(-z_7)} \quad \sigma(z_8) := \frac{1}{1 + \exp(-z_8)} \quad \sigma(z_9) := \frac{1}{1 + \exp(-z_9)}$$

$$\sigma(z_1) = 0.75547 \quad \sigma(z_2) = 0.297967 \quad \sigma(z_3) = 0.641185 \quad \sigma(z_4) = 0.740295 \quad \sigma(z_5) = 0.646457$$

$$\sigma(z_6) = 0.325414 \quad \sigma(z_7) = 0.66472 \quad \sigma(z_8) = 0.72222 \quad \sigma(z_9) = 0.731226$$

$$N := v_1\cdot\sigma(z_1) + v_2\cdot\sigma(z_2) + v_3\cdot\sigma(z_3) + v_4\cdot\sigma(z_4) + v_5\cdot\sigma(z_5) + v_6\cdot\sigma(z_6) + v_7\cdot\sigma(z_7) + v_8\cdot\sigma(z_8) + v_9\cdot\sigma(z_9)$$

$$U_d(x,y) := e^{-x}\cdot\left(x + y^3\right) \qquad U_d(x,y) = 0.125 \qquad N = 1.088913$$

$$U := (1-x)\cdot y^3 + x\cdot\left(1+y^3\right)\cdot e^{-1} + (1-y)\cdot x\cdot\left(e^{-x} - e^{-1}\right) + y\cdot\left[(1+x)\cdot e^{-x} - \left(1 - x + 2\cdot x\cdot e^{-1}\right)\right] + x\cdot(1-x)\cdot y\cdot(1-y)\cdot N = 0.125$$

$$U_p(x,y) := (1-x)\cdot y^3 + x\cdot\left(1+y^3\right)\cdot e^{-1} + (1-y)\cdot x\cdot\left(e^{-x} - e^{-1}\right) + y\cdot\left[(1+x)\cdot e^{-x} - \left(1 - x + 2\cdot x\cdot e^{-1}\right)\right] + x\cdot(1-x)\cdot y\cdot(1-y)\cdot N$$

$$E := 0.5\cdot\left(U_d(x,y) - U_p(x,y)\right)^2 \qquad E = 0$$