Transactions of the Nigerian Association of Mathematical Physics
Volume 12, (July – Sept., 2020 Issue), pp89 – 96

A NOVEL APPROACH FOR SOLVING VARIABLE COEFFICIENT INITIAL VALUE PROBLEMS USING ARTIFICIAL NEURAL NETWORKS

Okereke R.N. and Maliki O.S.

Department of Mathematics Michael Okpara University of Agriculture Umudike, Nigeria

Abstract

A novel approach for solving ordinary differential equations with variable coefficients using Artificial Neural Networks (ANN) stems from the fact that most conventional methods of solutions rely on cumbersome weight updating to finding approximate solutions. In this paper, we develop a Neural Network algorithm using MathCAD 14 software, which enables us to slightly adjust the intrinsic biases involved in solving ordinary differential equations with variable coefficients. For this purpose, we employ a Gaussian Radial Basis Function (GRBF) to obtain the weights, which need not be adjusted, both from input layer to the hidden layer, and from the hidden layer to the output layer of the network. This also involves the use of the Statistical Package for Social Sciences (SPSS 23) software. We compare exact results with the neural network results for our example ODE problems and find the results to be in good agreement. Furthermore, this compares favourably with existing neural network methods of solution.

Keywords: Artificial Neural Network, Weights, Biases, IVP, MathCAD 14, SPSS 23, GRBF.

1.0 Introduction

An Artificial neural network (ANN) is fundamentally a mathematical model, and its structure consists of a series of processing elements which are inter-connected and their operation resemble that of the human neurons. These processing elements are also known as units or nodes. The ability of the network to process information is embedded in the connection strengths, simply called weights, which, when exposed to a set of training patterns, adapts to it [1]. The conventional way of solving differential equations using artificial neural network involves updating of all the parameters, weights and biases, during the neural network training [2 - 7]. This is caused by the inability of the neural network to predict a solution with an acceptable minimum error. In order to reduce the error, the error function is minimized. Minimizing the error function demands finding its gradient. This gradient involves the computation of multivariate partial derivatives of the error function with respect to all the parameters, weights and biases, and the independent variable. This computational complexity is quite evidentwhen handling first order ordinary differential equations, as shown in [8,9]. It is even more difficult when solving higher order ODE where you need to compute higher order derivatives of the error function.

2.0 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. There are two functions that govern the behaviour of a unit in a particular layer, which normally are the same for all units within the whole ANN. These are;

1) The input function and
2) The output/activation function

Input into a node is a weighted sum of output from nodes connected to it.

A neuron receives a set of n inputs, $S = \{x_j \mid j = 1, 2, ..., n\}$. In fig. 1, 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, a threshold value θ_k, which has to be reached or exceeded for the neuron to produce an output signal [10].

2.1 Activation Function

The activation of a neuron is computed by applying a threshold function (popularly known as activation function) to the

Correspondence Author: Okereke R.N., Email: okerekeroseline@yahoo.com, Tel: +2347036838240

weighted sum of the inputs plus a bias [10]. 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;

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

and figure 1 shows detailed computational steps of the working principle of an artificial neuron (AN) in a neural network. Now the input signal for the i^{th} neuron N_i is,

$$s_i = w_0 + \sum_{j=1}^{n} w_{ij} x_j \quad (2)$$

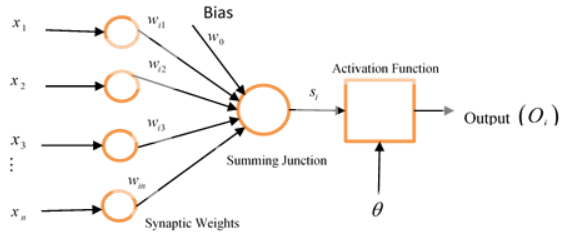

Fig.1. Mathematical Model of Artificial Neural Network. [8]

All inputs are multiplied by their weights and added together to form the net input to the neural called net. Mathematically, we can write

$$net = w_{i1}x_1 + w_{i2}x_2 + \ldots + w_{in}x_n + \theta \quad (3)$$

where θ is a threshold value that is added to the neurons. The neuron behaves as activation or mapping function $f(net)$ to produce an output y which can be expressed as:

$$y = f(net) = f\left(\sum_{j=1}^{n} w_{ij} x_j + \theta \right) \quad (4)$$

where $f(net)$ is called the neuron activation function or the neuron transfer function [10].

In this paper we employ the sigmoid activation function given by

$$\sigma(x) = \left(1 + e^{-Tx}\right)^{-1} \quad (5)$$

It is an *S* shaped smooth function, where input is mapped into values between +1 and 0.

2.2Function Approximation

Function approximation attempts to describe the behaviour of very complicated functions by sets of simpler functions, therefore we shall exploit its potentials in this paper.

2.3 Theorem (Universal approximation theorem for MLP)

Let I_n represent an *n*-dimensional unit cube containing all possible input samples *x*, that is, $x_i \in [0,1], \quad i = 1,2,\ldots,n$, 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

$$y_k = y_k(x,w) = \sum_{i=1}^{N_2} w_{ki}^3 \sigma\left(\sum_{j=0}^{n} w_{ij}^2 x_j \right) \quad k = 1,2,\ldots,m \quad (6)$$

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 (6) that satisfies $|y(x,w) - f(x)| < \varepsilon$ for all $x \in I_n$. As such, there always exists a 3-layer perception that can approximate an arbitrary nonlinear, continuous, multi-dimensional function f with any desired accuracy [11].Thus, any continuous function can be approximated to a given precision using artificial neural networks with just one hidden layer.

3.0Neural Network methodfor solving Variable Coefficient ODE

Lagaris *et al.* [4] gave the following as the general formulation for the solution of first order ODE

$$\frac{d\psi}{dx} = f(x,\psi), \quad x \in [a,b] \quad (7)$$

with initial condition $\psi(a) = A$ .To employ ANN for the solution, we define a trial solution, $\psi_t(x, p)$ written as the sum of two terms, i.e.

$$\psi_t(x, p) = A(x) + F(x, N(x, p)), \tag{8}$$

where $A(x)$ satisfies the initial conditions and contains no adjustable parameters, and $N(x, p)$ is the output of feed forward neural network with the parameters p and input data x. The function $F(x, N(x, p))$ 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

$$N(x, p) = \sum_{j=1}^{m} v_j \sigma(z_j) \tag{9}$$

where

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + u_j, \tag{10}$$

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.

To solve this problem using neural network (NN), a NN architecture with three layers - one input layer with one neuron; one hidden layer with three neurons and one output layer with one output unit, as shown in figure 2, is employed. Each neuron is connected to other neurons of the previous layer through adaptable synaptic weights w_j and biases u_j . Lagaris *et al.* [4] applied the same procedure to second order IVP.

$$\frac{d^2\psi}{dx^2} = f\left(x, \psi, \frac{d\psi}{dx}\right), \ \psi(0) = A, \psi'(0) = B \tag{11}$$

with the trial solution cast as

$$\psi_t(x) = A + Bx + x^2 N(x_i, p) \tag{12}$$

where $A, B \in \mathbb{R}$ and $N(x_i, p) = \sum_{j=1}^{3} v_j \sigma(z_j)$.

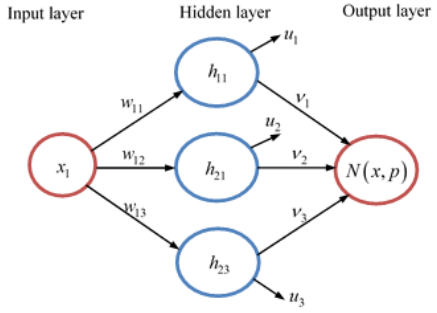

Fig.2. Schematic for $N(x, p)$

Each neuron is connected to other neurons of the previous layer through adaptable synaptic weights w_j and biases u_j . Lagaris *et al.*[4] applied the same procedure to second order initial value problem.

$$\frac{d^2\psi}{dx^2} = f\left(x, \psi, \frac{d\psi}{dx}\right), \quad \psi(0) = A, \ \psi'(0) = B, \tag{13}$$

with the trial solution cast as

$$\psi_t(x) = A + Bx + x^2 N(x_i, p), \ A, B, \in \mathbb{R}$$

where $A, B \in \mathbb{R}$ and

$$N(x_i, p) = \sum_{j=1}^{3} v_j \sigma(z_j),$$

The task of obtaining the weights from hidden layer to the output layer is done by finding $f(x)$, a real function of a real valued vector $x = [x_1, x_2, ..., x_d]^T$ and a set of functions, $\{\varphi_i(x)\}$ called the elementary functions such that

$$\hat{f}(x,v)=\sum_{i=1}^{N} v_i\varphi_i(x) \tag{14}$$

is satisfied, where v_i are real valued constants such that

$$\left|f(x)-\hat{f}(x,v)\right|<\varepsilon \tag{15}$$

When one can find coefficients v_i that make ε arbitrarily small for any function $f(\cdot)$ over the domain of interest, we say that the elementary function set $\{\varphi_i(\cdot)\}$ has the property of universal approximation over the class of functions $f(\cdot)$ [8]. We shall use the GRBF given by

$$\varphi_i(x)=\exp\left[-\frac{|x-x_i|^2}{2\sigma^2}\right],\quad \sigma^2=\frac{1}{N}\sum_{i=1}^{n}(x_i-\bar{x})^2 \tag{16}$$

as our elementary function and find $f(x)$ such that;

$$v=\phi^{-1}f \tag{17}$$

Here v becomes a vector with the coefficients, f is a vector composed of the values of the function 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. The major task here is finding $f(x)$. We shall apply what [8,9] proposed for first and second order constant coefficient ode to this effect. Now to compute the weights from hidden layer to the output layer, we find a function $F(x)$ such that $v=\phi^{-1}F$ where $F(x)=\left[f(x_1),\mathrm{f}(x_2),\mathrm{f}(x_3)\right]^T$. We form a linear function based on the default sign of the differential equation, i.e. $f(x)=ax+b$, where a is the coefficient of the derivative of y and b is the coefficient of y. Extending this to the second order IVP, $f(x)=ax^2+bx+c$, where $a,b,c\in\square$. Then we employ MathCAD 14 [12], a numerical algebra software, to slightly adjust the biases. These are the only parameters to be adjusted according to this new procedure. The Mean-Squared Error (MSE), given by:

$$E_n=\frac{1}{n}\sum_{i=1}^{n}(y_i-f(x_i))^2 \tag{18}$$

will be employed to analyze the error. A demonstration of how the neural network solutions agree with exact solutions are presented in the subsequent tables and simulations.

4.0 Main results

Variable coefficient differential equations are usually difficult to solve. Majority have to be transformed into another form before they can be solved. In this section we present our main results which demonstrate the efficiency of our proposed neural network method [8], in solving variable coefficient initial value problems.

4.1 First and second order linear ODE with variable coefficients

In this section we illustrate the foregoing with two examples on second order variable coefficient IVP.

4.1.1 Problem1

We consider the initial value problem[13]:

$$y''-2\tan(x)y'=1,\ y(0)=1,\ y'(0)=2,\ x\in[0,1]. \tag{19}$$

The trial solution is given by $y_t(x)=A+Bx+x^2N(x,p)$ and $y_t'(x)=B+2xN(x,p)+x^2N'(x,p)$.

Hence applying the initial conditions gives $y_t(0)=A=1$, $y_t'(0)=B=2\Rightarrow y_t(x)=1+2x+x^2N(x,p)$. The exact solution: $y(x)=2\tan(x)+1+\frac{1}{2}x\tan(x)$. Next we choose $f(x)=ax^2-bx+c$, and make the identifications $a=1,b=-2\tan(x),c=0$. This implies that $f(x)=x^2+2\tan(x)$. Hence,

$$F_1(x)=\left(f(x_1),f(x_2),f(x_3)\right)^T=(0.030067,0.121080,0.275602)^T \text{ for } x=(0.1,0.2,0.3)^T \tag{20}$$

As usual with $w=\phi^{-1}F_1$, we have;

$$\begin{bmatrix}w_1\\w_2\\w_3\end{bmatrix}=\begin{bmatrix}1&0.94&0.78\\0.94&1&0.94\\0.78&0.94&1\end{bmatrix}^{-1}\begin{bmatrix}0.030067\\0.121080\\0.275602\end{bmatrix}=\begin{bmatrix}41.335&-73.437&36.79\\-73.437&139.062&-73.437\\36.79&-73.437&41.335\end{bmatrix}\begin{bmatrix}0.030067\\0.121080\\0.275602\end{bmatrix}=\begin{bmatrix}2.49\\-5.61\\3.606\end{bmatrix} \tag{21}$$

Giving the weights from the input layer to the hidden layer as: $w_1 = 2.49,\ w_2 = -5.61,\ w_3 = 3.606$.

Next we form the constant function $g(x) = 1$, corresponding to the nonhomogeneous term. Thus

$$F_2(x) = \left(g(x_1), g(x_2), g(x_3)\right)^T = (1,1,1)^T \text{ for } x = (0.1, 0.2, 0.3)^T \tag{22}$$

We now use the relationship $v = \phi^{-1} F_2$ to compute the weights from the hidden layer to the output layer as;

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 0.94 & 0.78 \\ 0.94 & 1 & 0.94 \\ 0.78 & 0.94 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 41.335 & -73.437 & 36.79 \\ -73.437 & 139.062 & -73.437 \\ 36.79 & -73.437 & 41.335 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4.687 \\ -7.812 \\ 4.687 \end{bmatrix} \tag{23}$$

The weights from the hidden layer to the output layer are thus: $v_1 = 4.687,\ v_2 = -7.812,\ v_3 = 4.687$.

The biases are fixed between -1 and 10. We now train the network with the available parameters using MathCAD 14 model as follows:

$w_1 := 2.49 \quad w_2 := -5.61 \quad w_3 := 3.606 \quad x := 1$

$v_1 := 4.687 \quad v_2 := -7.812 \quad v_3 := 4.687 \quad u_1 := -1 \quad u_2 := 7.034 \quad u_3 := -1$

$z_1 := w_1 \cdot x + u_1 = 1.49 \quad z_2 := w_2 \cdot x + u_2 = 1.424 \quad z_3 := w_3 \cdot x + u_3 = 2.606$

$\sigma(z_1) := \left[1 + \exp(-z_1)\right]^{-1}$

$\sigma(z_2) := \left[1 + \exp(-z_2)\right]^{-1}$

$\sigma(z_3) := \left[1 + \exp(-z_3)\right]^{-1}$

$N := v_1 \cdot \sigma(z_1) + v_2 \cdot \sigma(z_2) + v_3 \cdot \sigma(z_3)$

$N(x) := 4.687 \cdot \left[1 + \exp(-2.49 \cdot x + 1)\right]^{-1} - 7.812 \cdot \left[1 + \exp(5.61 \cdot x - 7.0341)\right]^{-1} + 4.687 \cdot \left[1 + \exp(-3.606 \cdot x + 1)\right]^{-1}$

$N(1) = 1.8935$

$y_p(x) := 1 + 2x + x^2 N$

$y_p(1) = 4.8935$

$y_d(x) := 2\tan(x) + 1 + \frac{1}{2} x \tan(x)$

$y_d(1) = 4.8935$

$E := 0.5 \cdot \left(y_d(x) - y_p(x)\right)^2 = 5.915416 \times 10^{-12}$

4.1.2 Problem 2

We consider the initial value problem[13]:

$$x^2 y'' - 4xy' + 6y = x^4,\ y(1) = y'(1) = 0,\ x \in (0,1]. \tag{24}$$

The trial solution is given by $y_t(x) = A + Bx + x^2 N(x, p)$. Applying the initial conditions gives:

$$y_t'(x) = B + 2xN(x,p) + x^2 N'(x,p),\ \therefore y_t'(1) = B + 2N(1) + N'(1) = 0 \ \Rightarrow\ B = -2N(1) - N'(1)$$

$$y_t(1) = A + B + N(1) = 0 \ \Rightarrow\ A = 2N(1) + N'(1) - N(1) = N(1) + N'(1)$$

Therefore

$$y_t(x) = \left(N(1) + N'(1)\right) + \left(-2N(1) - N'(1)\right)x + x^2 N(x,p) \tag{25}$$

The exact solution is $y(x) = \frac{1}{2}x^2 - x^3 + \frac{1}{2}x^4$. Choosing $f(x) = ax^2 - bx + c$ with $a = x^2, b = -4x, c = 6$.

Hence; $f(x) = x^4 + 4x^2 + 6$, then we define;

$$F_1(x) = \left(f(x_1), f(x_2), f(x_3)\right)^T = (6.04, 6.162, 6.368)^T, \text{ for } x = (0.1, 0.2, 0.3)^T \tag{26}$$

With $w = \phi^{-1} F_1$, we compute the weights from the input layer to the hidden layer as;

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} 1 & 0.94 & 0.78 \\ 0.94 & 1 & 0.94 \\ 0.78 & 0.94 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 6.04 \\ 6.162 \\ 6.368 \end{bmatrix} = \begin{bmatrix} 41.335 & -73.437 & 36.79 \\ -73.437 & 139.062 & -73.437 \\ 36.79 & -73.437 & 41.335 \end{bmatrix} \begin{bmatrix} 6.04 \\ 6.162 \\ 6.368 \end{bmatrix} = \begin{bmatrix} 31.42 \\ -54.309 \\ 32.911 \end{bmatrix} \tag{27}$$

The weights from input layer to the hidden layer are: $w_1 = 31.42, w_2 = -54.309, w_3 = 32.911$.

To compute the weights from the hidden layer to the output layer. We consider the function $g(x) = x^4$ according to the right hand side of the given ODE. We now define;

$$F_2(x) = \left(g(x_1), g(x_2), g(x_3)\right)^T = (0.0001, 0.0016, 0.0081)^T \text{ for } x = (0.1, 0.2, 0.3)^T \tag{28}$$

Hence the weights from the hidden layer to the output layer given by $v = \phi^{-1} F_2$, is computed as;

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 0.94 & 0.78 \\ 0.94 & 1 & 0.94 \\ 0.78 & 0.94 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0.0001 \\ 0.0016 \\ 0.0081 \end{bmatrix} = \begin{bmatrix} 41.335 & -73.437 & 36.79 \\ -73.437 & 139.062 & -73.437 \\ 36.79 & -73.437 & 41.335 \end{bmatrix} \begin{bmatrix} 0.0001 \\ 0.0016 \\ 0.081 \end{bmatrix} = \begin{bmatrix} 0.185 \\ -0.38 \\ 0.221 \end{bmatrix} \quad (29)$$

Thus the weights from the hidden layer to the output layer are: $v_1 = 0.185, v_2 = -0.38, v_3 = 0.221$.

The biases are fixed between -50 and 50. We now train the network with the available parameters using our MathCAD 14 algorithm as follows:

$w_1 := 31.42 \quad w_2 := -54.309 \quad w_3 := 32.911 \quad x := 1$

$v_1 := 0.185 \quad v_2 := -0.38 \quad v_3 := 0.221 \quad u_1 := 1 \quad u_2 := -1 \quad u_3 := 1$

$z_1 := w_1 \cdot x + u_1 = 32.42 \quad z_2 := w_2 \cdot x + u_2 = -55.309 \quad z_3 := w_3 \cdot x + u_3 = 33.911$

$\sigma(z_1) := \left[1 + \exp(-z_1)\right]^{-1}$

$\sigma(z_2) := \left[1 + \exp(-z_2)\right]^{-1}$

$\sigma(z_3) := \left[1 + \exp(-z_3)\right]^{-1}$

$N := v_1 \cdot \sigma(z_1) + v_2 \cdot \sigma(z_2) + v_3 \cdot \sigma(z_3)$

$N(x) := 0.185 \cdot \left[1 + \exp(-31.42 \cdot x - 1)\right]^{-1} - 0.38 \cdot \left[1 + \exp(54.309 \cdot x + 1)\right]^{-1} + 0.221 \cdot \left[1 + \exp(-32.911 \cdot x - 1)\right]^{-1}$

$N(1) = 0.406$

$N' := v_1 \cdot \frac{d}{dx}\sigma(z_1) + v_2 \cdot \frac{d}{dx}\sigma(z_2) + v_3 \cdot \frac{d}{dx}\sigma(z_3)$

$$N'(x) := \frac{5.8127 \cdot \exp(-31.42 \cdot x - 1)}{\left[1 + \exp(-31.42 \cdot x - 1)\right]^2} + \frac{20.63742 \cdot \exp(54.309 \cdot x + 1)}{\left[1 + \exp(54.309 \cdot x + 1)\right]^2} + \frac{7.273331 \cdot \exp(-32.911 \cdot x - 1)}{\left[1 + \exp(-32.911 \cdot x - 1)\right]^2}$$

$N'(1) = 6.199 \times 10^{-14}$

$y_p(x) := \left(N(1) + N'(1)\right) + \left(-2N(1) - N'(1)\right)x + x^2 N(x) = 0.406 - 0.812 \cdot x + x^2 \cdot N(x)$

$y_p(1) = 0$

$y_d(x) := \frac{1}{2}x^2 - x^3 + \frac{1}{2}x^4$

$y_d(1) = 0$

$E := 0.5 \cdot \left(y_d(x) - y_p(x)\right)^2 = 0$

5.0 Simulations

We present the graphical demonstration of how our method of solution compares with the analytical (exact) solutions. The conventional method, as we pointed out earlier, involves complicated multivariate partial derivatives, while adjusting the parameters (weights and biases). This becomes even more apparent when second and higher order differential equations with variable coefficients are involved. Tables 1 and 2 provide the exact and predicted values for problems 1 and problem 2. The graphical profiles for these are displayed in figures 3 and 4.

Table 1 Comparison of the results of Problem 1

Input Data (X)	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y Exact	1	1.206	1.426	1.665	1.930	2.229	2.600	2.979	3.471	4.087	4.894
Y Predicted	1	1.206	1.426	1.665	1.930	2.229	2.600	2.979	3.471	4.087	4.894

Table 2 Comparison of the results of Problem 2

Input Data (X)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Y Exact	0.00405	0.0128	0.02205	0.0288	0.03125	0.0288	0.02205	0.0128	0.00405	0
Y Predicted	0.00405	0.0128	0.02205	0.0288	0.03125	0.0288	0.02205	0.0128	0.00406	0

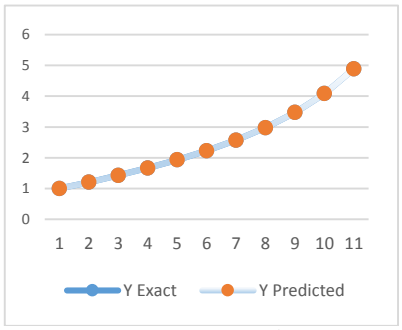

Fig 3. Y Exact and Y Predicted for Problem 1

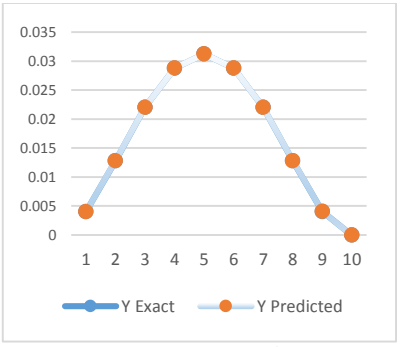

Fig 4. Y Exact and Y Predicted for Problem 2

5.1 Solving nth order linear ordinary differential equations

From the results established in this paper, we claim that it is possible to solve any nth order linear, variable and constant coefficient ordinary differential equations with our neural network method. Thus, given an nth order linear ODE,

$$p_n(x)y^n(x) + p_{n-1}(x)y^{n-1}(x) + p_{n-2}(x)y^{n-2}(x) + \ldots + p_0(x)y = g(x), \quad (44)$$

where $p_n(x),\ldots,p_0(x),g(x)$ are all functions of x only and are continuous in the given interval, form an algebraic equation $\Phi(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0$, with the default sign of the differential equation, (where $a_n,\ldots,a_0$ are the coefficients of the respective orders of the dependent variable, which in this case is *y*), of the same degree with the order of the given ODE and create the vector function $F_1(x) = (\Phi(x_1),\cdots,\Phi(x_n))^T$ evaluated at some relevant point $x = (x_1,\cdots,x_n)^T$.Then apply the GRBF matrix equation $w = \phi^{-1}F_1$ to get the weights from the input layer to the hidden layer. Subsequently, following the form of the right hand side of the given ODE, we set $\Theta(x) = g(x)$, and create a new function $F_2(x) = (\Theta(x_1),\cdots,\Theta(x_n))^T$ evaluated at same point $x = (x_1,\cdots,x_n)^T$. We then apply the GRBF matrix equation $v = \phi^{-1}F_2$ and get weights from the hidden layer to the output layer. Finally, use MathCAD 14 algorithm to implement the neural network model and compute the error.

6.0 Conclusion

In this paper we extended the novel approach developed by [8,9] to linear ordinary differential equations with variable coefficients. We employed MathCAD 14 algorithm and GRBF matrix model to achieve this task. Our results are validated by the perfect approximations achieved in comparison with the exact solutions, as well as demonstrating the function approximation capabilities of ANN. This then proves the efficiency of our neural network procedure.

References

[1] Graupe,D. (2007). Principles of Artificial Neural Networks (2nd Edition). Advanced Series on Circuits and Systems – World Scientific Publishing Co. Pte. Ltd., Singapore, Vol. 6

[2] Rumelhart, D. E. and McClelland, J. L. (1986). Parallel Distributed Processing, Explorations in the Microstructure of Cognition I and II. MIT Press, Cambridge.

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

[4] Lagaris, I. E., Likas A. C. and Fotiadis D.I. (1997). Artificial Neural Network for Solving Ordinary and Partial Differential Equations. arXiv:physics/9705023v1, Greece.

[5] Mall, S. and Chakraverty, S. (2013). Comparison of Artificial Neural Network Architecture in Solving Ordinary Differential Equations. Hindawi Publishing Corporation.

[6] Otadi, M. and Mosleh M. (2011). Numerical Solution of Quadratic Riccati Differential Equations by Neural Network. *Mathematical Sciences*, 5:249-257.

[7] Tian Qi, C., Yulia, R., Jesse, B. and David, D. (2018). Neural Ordinary Differential Equations. arXiv:1806.07366v1, Canada.

[8] Okereke, R. N. (2019). A new perspective to the Solution of ordinary differential Equations using Artificial Neural Networks. Ph.D Dissertation, Mathematics Department, Michael Okpara University of Agriculture Umudike, Nigeria.

[9] Okereke R. N, Maliki O. S.,Oruh B. I (2020).A novel method for solving ordinary differential equations with artificial neural networks. Applied Mathematics (AP), (to appear).
[10] Yadav, N., Yadav, A. and Kumar, M. (2015). An Introduction to Neural Network Methods for Differential Equations. Springer.
[11] Principe J. C., Euliano N. R., Lefebvre W. Curt, (1997), Neural and Adaptive Systems: Fundamentals Through Simulation
[12] Mathcad Version 14 (2007) PTC (Parametric Technology Corporation) Software Products. http://communications@ptc.com.
[13] Ricardo, H. J. (2009).A Modern Introduction to Differential Equations, 2nd edition. Elsevier