Volume 6, No.3, March 2019

Journal of Global Research in Mathematical Archives

ISSN 2320 – 5822

RESEARCH PAPER

Available online at http://www.jgrma.info

NUMERICAL SOLUTION OF FIRST ORDER INITIAL VALUE PROBLEMS BY CROSSBRED NEURAL NETWORKS

Mazin Hashim Suhhiem
Department of Statistics, University of Sumer, Alrifaee, Iraq
mazin.suhhiem@yahoo.com

Abstract: In this work, anovel numerical method based on crossbred neural networkis proposed to solve the first order ordinary differential equation. Here crossbred neural network is considered as a part of large field called neural computing or soft computing. The crossbred feed forward neural network based on replacing each element in the training set by a polynomial of third degree. The model finds the approximated solution of the first orderinitial value problems inside its domain for the close enough neighborhood of the initial point. Thismethod, in comparison with existing numerical methods, shows that the use of crossbred neural networks provides solutions with good generalization and high accuracy.

Keywords: first order ordinary differential equation, crossbred neural network, trial solution, minimized error function, hyperbolic tangent activation function.

1. INTRODUCTION

Many methods have been developed so far for solving ordinary differential equations since it is utilized widely for the purpose of modeling problems in science and engineering.

Most of the practical problems require the solution of the ordinary differential equation which satisfies initial conditions , therefore, the ordinary differential equation must be solved .Many ordinary differential equation could not be solved exactly ,thus considering their approximate solutions is becoming more important.

In 1990 researchers began using artificial neural network (ANN) for solving ordinary differential equation such as : lee , Kang in [1]; Meade , Fernandez in [2,3] ;Lagaris , Likasin [4] ; Liu ,Jammes in [5] ; Tawfiq in [6] ;malek , shekari in [7] ; Pattanaik , Mishra in [8];Baymani ,Kerayechian in [9] ;Suhhiem in[10] and other researchers .

In this work, we have used crossbred feed forward artificial neural network to find the numerical solution of the first order initial value problems. The crossbred neural network based on replacing each element in the training set by a polynomial of third degree. This polynomial can be written as: $k(x) = \epsilon(x^3 + x^2 + x + 1), \epsilon \in (0,1)$.

in this method we test different values for ϵ in the interval (0,1) which contains many infinitely suitable and not suitable chosen values for ϵ .Therefore,finding the suitable value of ϵ is not easily .Our numerical results showed that this method is better and much accuracy in comparison with other numerical methods. In general, the modified method in this work iseffective for solving the first order ordinary differential equation.

In the proposed method, the crossbred neural network model is applied as universal approximator . We use trial function, this trial function is a combination of two terms. The first term is responsible for the initial condition while the second term contains the crossbred neural network adjustable parameters to be calculated. Our crossbred neural network is a three-layer feed forward neural network where the connections weights, biases and inputs are given as real numbers.

The trial solution of the first order initial value problem is written as a sum of two parts. The first part satisfies the initial condition, it contains no adjustable parameters. While the second part involves crossbred feed-forward neural networks which containing adjustable parameters.

2. ARTIFICIAL NEURAL NETWORK

Artificial neural networks are learning machines that can learn any arbitrary functional mapping between input and output. They are fast machines and can be implemented in parallel, either in software or in hardware. In fact, the computational complexity of ANN is polynomial in the number of neurons used in the network .Parallelism also brings with it the advantages of robustness and fault tolerance.

(i.e.) ANN is a simplified mathematical model of the human brain. It can be implemented by both electric elements and computer software. It is a parallel distributed processor with large numbers of connections It is an information processing system that has certain performance characters in common with biological neural networks[1,2].

3. CROSSBRED NEURAL NETWORK

In this section , we introduce a novel method to modify the artificial neural networks .This new method is based on replacing each x in the input vector (training set)$\vec{x} = (x_1 , x_2 , \dots , x_n)$, $x_j \in [a, b]$ by a polynomial of third degree . .

We have used the function:

$k(x) = \epsilon(x^3 + x^2 + x + 1), \epsilon \in (0,1)$.

Then the input vector will be:

$(k(x_1) , k(x_2) , \dots , k(x_n)), k(x_j) \in (a , b)$ and j=1,2,…,n

Using crossbred neural network makes that training points should be selected over the open interval (a , b) without training the neural network in the range of first and end points. Therefore, the calculating volume involving computational error is reduced. In fact, the training points depending on the distance $[a , b]$ selected for training neural network are converted to similar points in the open interval (a , b) by using the new approach, then the network is trained in these similar areas [10].

4. DESCRIPTION of THE METHOD

In this section we illustrate how the proposed method can be used to find the approximate solution of the first order ordinary differential equation:

$G(x , \Psi(x) , \nabla \Psi(x) , \nabla^2 \Psi(x), \dots) = 0$, $x \in D$(1)

Where$x = (x_1, x_2 , \dots , x_n) \in R^n$, $D \subset R^n$denotes the domain and $\Psi(x)$ is the computed solution.

To obtain a solution to the above differential equation, the collocation method is adopted which assumes a discretization of the domain D into a set points $\widehat{D}$. The problem is then transformed into the following system of equations:

$G(x_i , \Psi(x_i) , \nabla \Psi(x_i) , \nabla^2 \Psi(x_i), \dots) = 0 \quad ,\forall x_i \in \widehat{D}$ (2)

If $\Psi_t(x_i , p)$ denotes a trial solution with adjustable parameters p, the problem is transformed to a discretize form :

$$\min_p \sum_{x_i \in \widehat{D}} \left(G\left(x_i, \Psi_t(x_i, p), \nabla \Psi_t(x_i, p), \nabla^2\Psi_t(x_i, p), \dots\right)\right)^2 (3)$$

The trial solution Ψ_t employs a feed forward crossbredneural network and the parameters p correspond to the weights and biases of the neural architecture . We choose a form for the trial function $\Psi_t(x)$ such that it satisfies the initial condition. This is achieved by writing it as a sum of two terms.

$$\Psi_t(x) = A(x) + F\left(x , N(k(x) , p)\right)(4)$$

where $N(k(x) , p)$ is a single-output feed forward crossbred neural network with parameters p and n input units fed with the input vector k(x).

The term $A(x)$ contains no adjustable parameters and satisfies the initial condition. The second term F is constructed so as not to contribute initial condition, since $\Psi_t(x)$ satisfy them. This term can be formed by using crossbred neural network whose weights and biases are to be adjusted in order to deal with the minimization problem [3,4].

5. COMPUTATION OF THE GRADIENT

An efficient minimization of eq.(3) Can be considered as a procedure of training the crossbred neural network, where the error corresponding to each input vector k(x) is the value E(x) which has to force near zero. Computation of this error value involves not only the crossbred neural network output but also the derivatives of the output with respect to any of its inputs [3,9].

Considering a multilayer crossbred neural network with n input units, one hidden layer with H sigmoid units and a linear output unit . The extension to the case of more than one hidden layers can be obtained accordingly.

For a given input vector $k(x) = \left(k(x_1)\, , k(x_2)\, , \ldots , k(x_n)\right)$ the output of the crossbred neural network is:

$$N = \sum_{i=1}^{H} v_i \, s(z_i) \qquad (5)$$

$$z_i = \sum_{j=1}^{n} w_{ij} k(x_j) + b_i \qquad (6)$$

w_{ij} denotes the weight connecting the input unit j to the hidden unit i,v_i denotes the weight connecting the hidden unit i to the output unit,b_i denotes the bias of hidden unit i, and $s(z)$ is the hyperbolic tangent activation function.

The gradient of N with respect to the parameters of the crossbred neural network can be easily obtained as:

$$\frac{\partial N}{\partial v_i} = s(z_i)(7)$$

$$\frac{\partial N}{\partial b_i} = v_i s'(z_i)(8)$$

$$\frac{\partial N}{\partial w_{ij}} = v_i s'(z_i) k(x_j)(9)$$

Once the derivative of the error with respect to the network parameters has been defined, then it is a straightforward to employ any minimization technique and we have used BFGS quasi-Newton method (For more details, see [10]) .

6. ILLUSTRATION OF THE METHOD

To illustrate the proposed method, we consider the first order ordinary differential equation:

$$\frac{dy(x)}{dx} = f(x\, , y)(10)$$

where $x \in [a\, , b]$,$k(x) \in (a,b)$ and the initial condition $y(a) = A$.

The trial solution can be written as:

$$y_t(x) = A + (x-a)\, N(k(x)\, , p)(11)$$

where $N(k(x)\, , p\,)$ is the output of the crossbred neural network with one input unit for k(x) and weights p .

Note that $y_t(x)$ satisfies the initial condition by construction. The error function that must be minimized is given by[6,10]:

$$E[p] = \sum_{i=1}^{n} \left[\frac{dy_t(x_i)}{dx} - f\left(x_i\, , y_t(x_i)\right)\right]^2 \qquad (12)$$

where the x_i's are points in $[a\, , b]$ and $k(x_i)$ are points in (a,b) .

From above, we have :

For a given input vector $\left(k(x_1)\, , k(x_2)\, , \ldots , k(x_n)\right)$, $k(x_j) \in (a,b)$ and j=1,2,…,n

The output of the crossbred neural network is:

$$N = \sum_{i=1}^{H} v_i \; s\,(z_i) \quad (13)$$

where

$z_i = \sum_{j=1}^{n} w_{ij}k(x_j) + b_i$ (14)

$k(x_j) = \epsilon(x_j^3+x_j^2+x_j + 1), \epsilon \in (0,1)$

where $x_j \in [a , b]$ and $k(x_j) \in (a , b)$, j=1,2,…,n

then the equations (7 − 9) will be :

$\frac{\partial N}{\partial v_i} = s\left(w_{ij}k(x_j) + b_i\right) = s\left(\epsilon\, (x_j^3+x_j^2+x_j + 1)\, w_{ij} + b_i\right)$(15)

$\frac{\partial N}{\partial b_i} = v_i s'\left(w_{ij}k(x_j) + b_i\right) = v_i s'\left(\epsilon\, (x_j^3+x_j^2+x_j + 1)\, w_{ij} + b_i\right)$(16)

$\frac{\partial N}{\partial w_{ij}} = v_i k(x_j) s'\left(w_{ij}k(x_j) + b_i\right)$

$= \epsilon\, (x_j^3+x_j^2+x_j + 1) v_i\, s'\left(\epsilon\, (x_j^3+x_j^2+x_j + 1)\, w_{ij} + b_i\right)$(17)

where s′ is the first derivative of the activation function.

7. SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

To find the approximate solution of the first order ordinary differential equations by using the crossbred neural network we use $(1 \times m \times 1)$ feed-forward crossbredneural network (Fig. 1) which contains one input unit, m hidden units and one output unit.

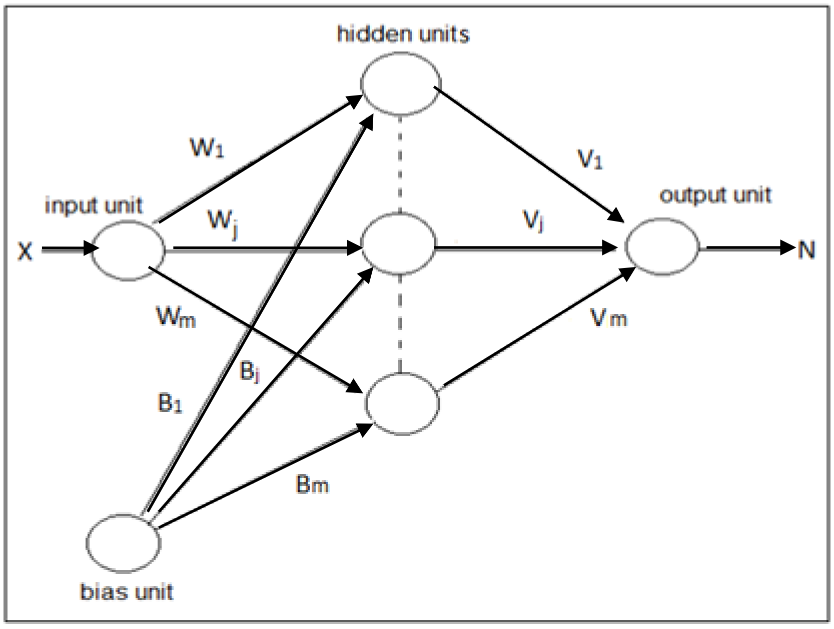

Fig. (1) $(1 \times m \times 1)$feed-forward neural network .

For every entry x the input neuron makes no changes in its input, so the input to the hidden neurons is:

$net_j = k(x)w_j + b_j$, j = 1,2, …m(18)

where w_j is a weight parameter from input layer to the jth unit in the hidden layer, b_j is an jth weight bias for the jth unit in the hidden layer.

The output, in the hidden neurons is:

$z_j = s(net_j)$, $j = 1,2, \dots m$ (19)

where s is the hyperbolic tangent activation function. The output neuron make no changes in its input, so the input to the output neuron is equal to output:

$N = \sum_{j=1}^{m} v_j z_j$ (20) where v_j is a weight parameter from the jth unit in the hidden layer to the output layer .

Then the equations (18-20) can be written as :

$net_j = k(x)w_j + b_j = \epsilon(x^3+x^2+x+1)w_j + b_j$(21)

$z_j = s(net_j) = s(\epsilon(x^3+x^2+x+1)w_j + b_j)$ (22)

$N = \sum_{j=1}^{m} v_j z_j = \sum_{j=1}^{m} v_j s\,(\epsilon(x^3+x^2+x+1)w_j + b_j)$ (23)

where $j = 1,2, \dots m$, and $\epsilon \in (0,1)$, $k(x) \in (a , b)$.

. Numerical Example

In the section, we have solved initial value problem with different values of ϵ.We have used a three-layer feed forward crossbred neural network having one input unit, one hidden layer with 10 hidden units (neurons) and one output unit, and hyperbolic tangent activation function.

For the numericalproblem, the analytical solution $y_a(x)$has been known in advance, therefore we test the accuracy of the obtained solutions by computing the deviation: $\Delta y(x) = |y_t(x) - y_a(x)|$.

To minimize the error function we have used BFGS quasi-Newton method (For more details, see[10]) . The computer programs which we have used in this work are coded in MATLAB 2015 .

Example (1): Consider the first order initial value problem :

$y'(x) = 2y(x) - y^2(x) + 1$, with$y(0) = 0$ and $x \in [0,1]$.

The analytical solution for this problem is:

$$y_a(x) = 1 + \sqrt{2}\tanh\left(\sqrt{2}\,x + \frac{1}{2}\log\left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\right)\right).$$

The trial solution for this problem is :

$y_t(x) = x\,N(k(x)\,,p)$

The crossbredneural network trained using a grid of ten equidistant points in the interval $[0\,,1]$,(i. e.) the input vector $\vec{x}$(training set) is:

$\vec{x} = \{0\,,0.1\,,0.2\,,0.3\,,0.4\,,0.5\,,0.6\,,0.7\,,0.8\,,0.9\,,1\}$.

Now, to find the error function E that must be minimized for this problem, we apply the following steps:

$$\frac{\partial y_t(x)}{\partial x} = N(k(x)\,,p) + x\frac{\partial N(k(x)\,,p)}{\partial x}$$

and

$$E = \sum_{i=1}^{11}\left[\frac{\partial y_t(x_i)}{\partial x} - \left(2y_t(x_i) - \left(y_t(x_i)\right)^2 + 1\right)\right]^2$$

then we get

$$E=\sum_{i=1}^{11}\left[N\big(k(x_i),p\big) + x_i\frac{\partial N(k(x_i),p)}{\partial x} - 2x_iN\big(k(x_i),p\big) + \big(x_iN\big(k(x_i),p\big)\big)^2 - 1\right]^2$$

where

$$N(k(x),p) = \sum_{j=1}^{10} v_j\, s\big(k(x)\,w_j + b_j\big)$$

$$\frac{\partial N(k(x),p)}{\partial x} = \sum_{j=1}^{10} \epsilon\, v_jw_j\,(3x^2+2x+1)s'\big(k(x)\,w_j + b_j\big)$$

since $s'(\alpha) = 1 - s^2(\alpha)$,

then we get :

$$\frac{\partial N(k(x),p)}{\partial x}=\sum_{j=1}^{10}\Big(\epsilon\, v_jw_j\,(3x^2+2x+1) - \epsilon\, v_jw_j\,(3x^2+2x+1)\, s^2\big(k(x)\,w_j + b_j\big)\Big)$$

Therefore we have :

$$E = \sum_{i=1}^{11}\Big[\sum_{j=1}^{10} v_js\,(k(x_i)w_j + b_j) + x_i\sum_{j=1}^{10}\Big(\epsilon\, v_jw_j\,(3x^2+2x+1) - \epsilon\, v_jw_j\,(3x^2+2x+1)\, s^2\big(k(x)\,w_j + b_j\big)\Big) - 2x_i\sum_{j=1}^{10} v_j\, s\,(k(x_i)w_j + b_j) + \Big(x_i\sum_{j=1}^{10} v_j\, s\,(k(x_i)w_j + b_j)\Big)^2 - 1\Big]^2 . \quad (24)$$

Then we use (24) to update the weights and biases .

Suhhiem in [10] solved this problem by using usual neural network, in this work we solved this problem with four values of $\epsilon$and then we compared the results in this work with the result in [10] to show the accuracy of the proposed method .

Analytical and trial solutions for this problem can be found in table (1),table(2),table(3) and table(4) .

9. CONCLUSION

In this work, we have introduced a modified method to find the numerical solution of the first order ordinary differential equations. This method based on crossbred neural network to approximate the solution of the first order initial value problems. The crossbred neural network based on replacing each element in the training set by a polynomial of third degree, This polynomial is defined as: $k(x) = \epsilon(x^3 + x^2 + x + 1)$, $\epsilon \in (0,1)$.in this method we test different values for ϵ in the interval (0,1) which contains many infinitely suitable and not suitable chosen values for ϵ .Therefore, the accuracy of the results depends on the chosen value of ϵ . For future studies, one can extend this method to find a numerical solution of the higher order ordinary differential equations. Also, one can use this method for solving partial differential equation.

REFERENCES

[1] Lee H. , Kang I. S. , "Neural Algorithms For Solving Differential Equations" , Journal of Computational Physics , 91 ,110-131,1990 .

[2] Meade A. J. , Fernandes A. A. ,"The Numerical Solution of Linear Ordinary Differential Equations by Feed-Forward Neural Networks" , Mathematical and Computer Modelling , Vol. 19 , No. 12 , 1-25 , 1994.

[3] Meade A. J. , Fernandes A. A. ," Solution of Nonlinear Ordinary Differential Equations by Feed-Forward Neural Networks" , Mathematical and Computer Modelling , Vol. 20 , No. 9 , 19-44 , 1994 .

[4] Lagaris I. E. , Likas A. , et al. ,"Artificial Neural Networks For Solving Ordinary and Partial Differential Equations", Journal of Computational Physics, 104 , 1-26 , 1997 .

[5] Liu B. , Jammes B. ,"Solving Ordinary Differential Equations by Neural Networks" , Warsaw , Poland , 1999 .

[6] Tawfiq L. N. M. ,"On Design and Training of Artificial Neural Network For Solving Differential Equations",Ph.D. Thesis , College of Education Ibn AL-Haitham,University of Baghdad , Iraq , 2004 .

[7] Malek A. , Shekari R. ,"Numerical Solution For High Order Differential Equations by Using a Hybrid Neural Network Optimization Method " ,Applied Mathematics and Computation, 183 , 260-271 , 2006 .

[8] Pattanaik S. , Mishra R. K. ,"Application of ANN For Solution of PDE in RF Engineering",International Journal on Information Sciences and Computing , Vol. 2 , No. 1 , 74-79, 2008 .

[9] Baymani M. , Kerayechian A. , et al. ,"Artificial Neural Networks Approach For Solving Stokes Problem" , Applied Mathematics , 1 , 288-292,2010 .

[10] Suhhiem M. H. ,"Fuzzy Artificial Neural Network For Solving Fuzzy and Non-Fuzzy Differential Equations ",Ph.D. Thesis , College of Sciences, AL-Mustansiriyah University, Iraq , 2016 .

Table (1): Numerical result for example (1)

x	$y_a(x)$	$y_t(x)$,$\epsilon = 0.25$	error	$y_t(x)$ in[10]	error
0	0	0	0	0	0
0.1	0.110295196	0.110294331	8.6553e-7	0.110293475	1.7210e-6
0.2	0.241976799	0.241976056	7.4322e-7	0.241970548	6.2510e-6
0.3	0.395104848	0.395104229	6.1966e-7	0.395104269	0.5790e-6
0.4	0.567812166	0.567807230	4.9369e-6	0.567802288	9.8780e-6
0.5	0.756014393	0.756010753	3.6408e-6	0.756011576	2.8170e-6
0.6	0.953566216	0.953556801	9.4150e-6	0.953493355	7.2861e-5
0.7	1.152948967	1.152948182	7.8532e-7	1.152913777	3.5190e-5
0.8	1.346363655	1.346362694	9.6157e-7	1.3462634330	1.0022e-4
0.9	1.526911313	1.526911084	2.2967e-7	1.526859080	5.2233e-5
1	1.689498392	1.689497500	8.9214e-7	1.689452714	4.5678e-5

Table (2): Numerical result for example (1)

x	$y_a(x)$	$y_t(x)$,$\epsilon = 0.45$	error	$y_t(x)$ in[10]	error

0	0	0	0	0	0
0.1	0.110295196	0.110294711	4.8521e-7	0.110293475	1.7210e-6
0.2	0.241976799	0.241976096	7.0382e-7	0.241970548	6.2510e-6
0.3	0.395104848	0.395103899	9.4977e-7	0.395104269	0.5790e-6
0.4	0.567812166	0.567812031	1.3598e-7	0.567802288	9.8780e-6
0.5	0.756014393	0.756014279	1.1405e-7	0.756011576	2.8170e-6
0.6	0.953566216	0.953565807	4.0926e-7	0.953493355	7.2861e-5
0.7	1.152948967	1.152948535	4.3226e-7	1.152913777	3.5190e-5
0.8	1.346363655	1.346362960	6.9511e-7	1.3462634330	1.0022e-4
0.9	1.526911313	1.526910613	7.0059e-7	1.526859080	5.2233e-5
1	1.689498392	1.689498112	2.8040e-7	1.689452714	4.5678e-5

Table (3): Numerical result for example (1)

x	$y_a(x)$	$y_t(x)$,$\epsilon = 0.65$	error	$y_t(x)$ in[10]	error
0	0	0	0	0	0
0.1	0.110295196	0.110295110	8.6773e-8	0.110293475	1.7210e-6
0.2	0.241976799	0.241976755	4.4179e-8	0.241970548	6.2510e-6
0.3	0.395104848	0.395104757	9.1638e-8	0.395104269	0.5790e-6
0.4	0.567812166	0.567812141	2.5566e-8	0.567802288	9.8780e-6
0.5	0.756014393	0.756013978	4.1535e-7	0.756011576	2.8170e-6
0.6	0.953566216	0.953566025	1.9190e-7	0.953493355	7.2861e-5
0.7	1.152948967	1.152948219	7.4858e-7	1.152913777	3.5190e-5
0.8	1.346363655	1.346362946	7.0991e-7	1.3462634330	1.0022e-4
0.9	1.526911313	1.526911229	8.4950e-8	1.526859080	5.2233e-5
1	1.689498392	1.689498314	7.8303e-8	1.689452714	4.5678e-5

Table (4): Numerical result for example (1)

x	$y_a(x)$	$y_t(x)$,$\epsilon = 0.85$	error	$y_t(x)$ in[10]	error

0	0	0	0	0	0
0.1	0.110295196	0.110286055	9.1419e-6	0.110293475	1.7210e-6
0.2	0.241976799	0.241968385	8.4149e-6	0.241970548	6.2510e-6
0.3	0.395104848	0.395096346	8.5023e-6	0.395104269	0.5790e-6
0.4	0.567812166	0.567802605	9.5614e-6	0.567802288	9.8780e-6
0.5	0.756014393	0.756007523	6.8704e-6	0.756011576	2.8170e-6
0.6	0.953566216	0.953480016	8.6200e-5	0.953493355	7.2861e-5
0.7	1.152948967	1.152878518	7.0449e-5	1.152913777	3.5190e-5
0.8	1.346363655	1.346304977	5.8678e-5	1.3462634330	1.0022e-4
0.9	1.526911313	1.526906124	5.1894e-6	1.526859080	5.2233e-5
1	1.689498392	1.689493801	4.5912e-6	1.689452714	4.5678e-5