Open Access
Baghdad Science Journal **Vol.16(1)2019**

DOI: http://dx.doi.org/10.21123/bsj.2019.16.1.0116

Solving Mixed Volterra - Fredholm Integral Equation (MVFIE) by Designing Neural Network

Nahdh S. M. Al-Saif* *Ameen Sh. Ameen*

Received 29/3/2018, Accepted 20/1/2019, Published 11/3/2019

Abstract:

In this paper, we focus on designing feed forward neural network (FFNN) for solving Mixed Volterra – Fredholm Integral Equations (MVFIEs) of second kind in 2–dimensions. in our method, we present a multi – layers model consisting of a hidden layer which has five hidden units (neurons) and one linear output unit. Transfer function (Log – sigmoid) and training algorithm (Levenberg – Marquardt) are used as a sigmoid activation of each unit. A comparison between the results of numerical experiment and the analytic solution of some examples has been carried out in order to justify the efficiency and the accuracy of our method.

Key words: Feed Forward neural network, Levenberg – Marquardt (trainlm) training algorithm, Mixed Volterra - Fredholm integral equations.

Introduction:

Integral equation is one of the main branches of modern mathematics that appear in various applied areas including mechanics, physics and engineering …etc. Here, we are concerned with the numerical solution of the following (MVFIEs) that has the formula below:

$$u(x,y) = f(x,y) + \int_0^y \int_M K(x,y,u(t,s))dtds\,,$$
$$(x,y) \in I = M \times [0,T] \ldots (1)$$

where $u(x,y)$ is an unknown function that should to be found and $f(x,y)$ and $K(x,y,u(t,s))$ are given analytical real valued function defined on I = $M \times [0,T]$ and M is a compact subset on R^n (n=1,2,3), with convenient norm $\|.\|$.

In literatures, different numerical methods used to solve 2-dimensional (MVFIEs) have been reported. In (1) Babolian et al. applied block pulse function and their operational matrix to solve (MVFIEs) in 2-dimensional spaces. By using Hybrid Legendre Functions, Nemati et al. introduced a numerical method for the (MVFIEs)(2). Shahooth presented a Numerical Solution for Mixed (MVFIEs) of the second kind using Bernstein Polynomial Method (3), this method is used to obtain the system of algebraic equations from the integral equation. In 2015, Ahmadabadi used a Meshless Method for solving (MVFIEs) of Urysohn type on non – rectangular regions numerically (4).

Department of mathematic, College of Science, University of Anbar, Anbar, Iraq.
*Corresponding author: nn_ss_m68@yahoo.com

Ibrahim et al. using the New Iterative Method for solving (MVFIEs) (5). This work is structured as follows:

In Section 2 we introduce the definition of Artificial Neural Network (ANN). The structure of a Neural Network presented in Section 3. Section 4 is concerned with Levenberg – Marquardt Algorithm (LM). Section 5 demonstrates conducting our proposed approach to calculate the approximation solution for (MVFIEs), and in Section 6 the proposed method is applied in some examples, to clarify the efficiency and accuracy of this method.

Artificial Neural Network (ANN):

An (ANN) is formed from many artificial neurons joined together depending on particular network architecture. The goal of the neural network is to transform the inputs into significant outputs.

In other words, (ANN) is an interconnected system of nodes ('equivalent to neurons of a human brain') by weighted arrows ('equivalent to synapses between neurons').The outcome of (ANN) is altered by changing the arrow's weights. The result of the network for the data fed to the input layer is displayed by the output layer. Dependent variables have been estimated from the input nodes, which represent the independent or predictor variables.

Hristev in (6) characterizes ANN as follows,

i) Its pattern of connections between the neurons (called its architecture).

ii) The method by which the connections weight is calculated (training or learning algorithm).
iii) Its activation function .

Neural Network Structure (6):

The structure or topology of an artificial neural network means the way of regulation of neuronal computational cell in the network. That is, how the nodes are connected and how the information is transmitted through the network. The architecture can be classified in terms of three aspects (Number of levels or layers, Connection pattern and Information flow).see fig(1).

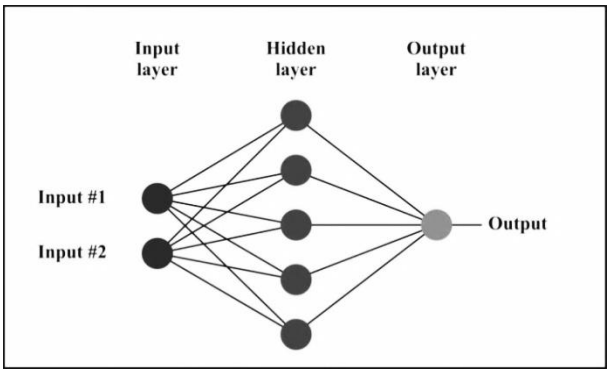

Figure 1. Structure of a Neural Network (Topology)

Levenberg–Marquardt Algorithm(LMA) (7)

"The Levenberg-Marquardt algorithm is a variation of Newton's method that designed for minimizing functions that are sums of squares of other nonlinear functions. This is very well suited to neural network training where the performance index is the mean squared error.

In order to optimize LMA performance index, If we assume that F(w) is a sum of squares function we define it as,

$$F(w) = \sum_{p=1}^{p}\left[\sum_{k=1}^{k}\left(d_{kp} - o_{kp}\right)^2\right] \dots (2)$$

where $w = [w_1 w_2 \cdots w_N]^T$ includes all network's weights. d_{kp}, o_{kp}, K, P and N are the desired value of the k^{th} output and the p^{th} pattern, the actual value of the k^{th} output and the p^{th} pattern, the number of the network output, the number of pattern and the number of the weights, respectively. Equation (2) can be written as follows

$$F(w) = E^T E \quad \dots (3)$$

Where

$E = \left[e_{11} \cdots e_{k1} e_{12} \cdots e_{k2} \cdots e_{1p} \cdots e_{kp}\right]^T$, $E =$, $e_{kp} = d_{kp} - o_{kp}$, $k = 1, \cdots, K$, $p = 1, \cdots, P$.

Where E is the cumulative error vector (for all pattern). From equation (3) the weights are calculated using the following equation

$$w_{t+1} = w_t - (J_t^T J_t + \mu_t I)^{-1} J_t^T E_t \dots \quad (4)$$

and the jacobian matrix is define as

$$J = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} \cdots \frac{\partial e_{11}}{\partial w_N} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} \cdots \frac{\partial e_{21}}{\partial w_N} \\ \vdots & \vdots \quad \vdots \\ \frac{\partial e_{k1}}{\partial w_1} & \frac{\partial e_{k1}}{\partial w_2} \cdots \frac{\partial e_{k1}}{\partial w_N} \\ \vdots & \vdots \quad \vdots \\ \frac{\partial e_{1p}}{\partial w_1} & \frac{\partial e_{1p}}{\partial w_2} \cdots \frac{\partial e_{1p}}{\partial w_N} \\ \frac{\partial e_{2p}}{\partial w_1} & \frac{\partial e_{2p}}{\partial w_2} \cdots \frac{\partial e_{2p}}{\partial w_N} \\ \vdots & \vdots \quad \vdots \\ \frac{\partial e_{kp}}{\partial w_1} & \frac{\partial e_{kp}}{\partial w_2} \cdots \frac{\partial e_{kp}}{\partial w_N} \end{bmatrix} \dots (5)$$

where I, μ and J are identity unit matrix, the learing parameter and jacobian of m out- put error of the neural network with respect to n weights, respectively. At each iteration the μ parameter automatically adjusted in order to secure convergence, the calculation of the jacobian matrix J and the inverse of $J^T J$ square matrix of order $N \times N$ at each iteration step are the requirement of LMA.

Description of the Method:

In the current section, we demonstrate conducting our approach to calculate the approximation solution of the (MVFIEs):

$u(x,y) = f(x,y) + \int_0^y \int_M K(x,y,u(t,s))dtds$,
$(x,y) \in I = M \times [0,T]$
where $x,y \in I \subseteq R^2$, and $u(x,y)$ the solution to be found. When a trial solution with adjustable parameters p denoted by $u_t(x,y,p)$, then the problem is transformed to the following discretize form:

$$Min \sum_{x_i, y_i \in M} f(x_i, y_i) +$$

$$\int_0^{y_i} \int_M K(x_i, y_i, t, s, u_t(t,s,))dtds \dots (6)$$

In our approach, the trial solution $u_t(x,y,p)$ employs a fast feed neural network (FFNN)and the parameters p correspond to the weight and biases of the neural architecture , we choose form for the trial function $u_t(x,y)$

$$u_t(x_i, y_i, p) = G\big(x, y, N(x,y,p)\big) \quad \dots (7)$$

where $N(x,y,p)$ is a single - output (FFNN) with parameter p and n input unit fed with the input vectors (x,y). The term G is constructed, since $u_t(x,y)$ satisfy them. This term can be formed by using an (ANN) whose weight and biases are to be adjusted in order to deal with the minimization problem. The Minimized is given by

$$E(p) = \left\{ u(x,y) - \sum_{i=1}^{n} \left(f(x_i,y_i) + \int_0^{y_i} \int_M k(x_i,y_i,u_t(s,t))\, dt\, ds \right) \right\}^2 \quad \ldots(8)$$

Numerical Examples:

In this section we report numerical result, we use a multi-layer FFNN having one hidden layer with **5** hidden units (neurons) and one linear output unit. The sigmoid activation of each hidden unit is Log – sigmoid and training algorithm (Levenberg – Marquardt) used as a sigmoid activation of each unit, in each example the mean square error (MSE) is used to test the accuracy of obtained solutions.

Example 6.1:

Let the following (MVFIE).

$$u(x,y) = e^x y^2 - \tfrac{2}{3}x^2y^3 + \int_0^y \int_{-1}^1 x^2 e^{-t}\, u(t,s)dtds,\ (x,y) \in [0,y] \times [-1,1] \quad \ldots(9)$$

whose analytic solution $u(x,y) = e^x\, y^2$

We applied the present method to solve equation (9). Table (1) shows the analytic, neural result and its error. Table (2) gives the weight, bias, Epoch, time and performance of the designer network.

Table 1. Analytic, Neural and Accuracy of solution Equation (9).

X↓	Y	Analytic $u_a(x,y)$	Trainlm $u_t(x,y)$	Error = $\lvert u_t(x,y) - u_a(x,y)\rvert$
-1	0.0	0.0000e+000	1.1709e-007	1.1709e-007
-0.8	0.2	1.7973e-002	1.7973e-002	2.4602e-007
-0.6	0.4	8.7810e-002	8.7810e-002	1.6134e-007
-0,4	0.6	2.4132e-001	2.4132e-001	5.1660e-007
-0.2	0.8	5.2399e-001	5.2399e-001	1.0256e-006
0.0	1.0	1.0000e+000	1.0000e+000	3.1978e-007
0.2	1.2	1.7588e+000	1.7588e+000	1.4060e-006
0.4	1.4	2.9240e+000	2.9240e+000	1.9408e-006
0.6	1.6	4.6646e+000	4.6646e+000	4.6404e-007
0.8	1.8	7.2108e+000	7.2108e+000	1.8224e-007
1	2	1.0873e+001	1.0873e+001	4.4215e-009
			MSE 7.30e-013	

Table 2. Weight, Bias, Epoch, Time and Performance of the Network.

Weight and Bias				Epoch, Time and Performance		
Net-IW{1,1}		Net-LW{1,2}	Net-B{1}	Epoch	Time	Performance
0.9991	0.6692	0.1564	0.4283			
0.1711	0.1904	0.8555	0.4820	20000	0:04:15	6.21e-14
0.0326	0.3689	0.6448	0.1206			
0.5612	0.4607	0.3763	0.5895			
0.8819	0.9816	0.1909	0.2262			

Example 6.2:

Consider the following (MVFIE).

$$u(x,y) = e^{xy} - x^2y + \int_0^y \int_0^1 x^2 e^{-st}\, u(t,s)dtds \quad (x,y) \in [0,y] \times [0,1] \quad \ldots(10)$$

Which has the analytic solution below $u(x,y) = e^{xy}$

Using the same present method for solving equation (10). Table (3) shows the analytic, neural result and its error. Table (4) gives the weight, bias, Epoch, time and performance of the designer network.

Table 3. Analytic, Neural and Accuracy of Solution Equation (10).

X↓	Y	Analytic $u_a(x,y)$	Trainlm $u_t(x,y)$	Error = $\lvert u_t(x,y) - u_a(x,y)\rvert$
0.0	0.0	1.0000e+000	1.0000e+000	9.4199e-010
0.1	0.1	1.0101e-000	1.0101e-000	2.8402e-008
0.2	0.2	1.0408e-000	1.0408e-000	1.3063e-008
0.3	0.3	1.0942e-000	1.0942e-000	5.2128e-008
0.4	0.4	1.1735e-000	1.1735e-000	3.2270e-009
0.5	0.5	12840e+000	12840e+000	1.1274e-008
0.6	0.6	1.4333e+000	1.4333e+000	2.2061e-008
0.7	07	1.6323e+000	1.6323e+000	7.e-0518508
0.8	0.8	1.8965e+000	1.8965e+000	5.8855e-008
0.9	0.9	2.2479e+000	2.2479e+000	1.3026e-008
1	1	2.7183e+001	2.7183e+001	2.2140e-010
			MSE 1.28e-013	

Table 4. Weight, Bias, Epoch, Time and Performance of the Network.

Weight and Bias				Epoch, Time and Performance		
Net-IW{1,1}		Net-LW{1,2}	Net-B{1}	Epoch	Time	Performance
0.0842	0.5399	0.9742	0.3307			
0.1639	0.0954	0.5708	0.4300	9684	0:01:51	4.55e-16
0.3242	0.1465	0.9969	0.4918			
0.3017	0.6311	0.5535	0.0710			
0.0117	0.8593	0.5155	0.8877			

Example 6.3:

Suppose the following (MVFIE).

$u(x,y) =$

$f(x,y) + \int_0^y \int_M K(x,y,t,s)\ u(t,s)dtds,\ \ (x,y) \in M \times [0,2]$ ……(11)

with $M = [0,2]$, and $K(x,y,t,s) = -\cos(x-t)\exp(s-y)$

$f(x,y) = \exp(-y)(\cos(x) + ycos(x) + \frac{1}{2}ycos(x-2)\sin(2))$.

their analytic solution is $u(x,y) = \cos(x)\exp(-y)$

Using our method to solve equation (11). Table (5) shows the analytic, neural result and its error. Table (6) gives the weight, bias, Epoch, time and performance of the designer network.

Table 5. Analytic, Neural and Accuracy of Solution Equation (11).

X↓	Y	Analytic $u_a(x,y)$	Trainlm $u_t(x,y)$	Error = $\lvert u_t(x,y) - u_a(x,y)\rvert$
0.0	0.0	1.0000 e+000	1.0000 e+000	9.9680e-008
0.2	0.2	8.0241e-001	8.0241e-001	2.4528e-006
0.4	0.4	6.1741e-001	6.1741e-001	6.4110e-007
0.6	0.6	4.5295e-001	4.5295e-001	-2.4407e-007
0.8	0.8	3,1305e-001	3,1305e-001	-1.0810e-007
1.0	1.0	1.9877e-001	1.9877e-001	2.4741e-007
1.2	1.2	1.0913e-001	1.0913e-001	1.6111e-007
1.4	1.4	4.1913e-002	4.1913e-002	1.1516e-007
1.6	1.6	-5.8953e-003	-5.8953e-003	1.2807e-008
1.8	1.8	-3.7556e-002	-3.7556e-002	1.2018e-007
2	2	-5.6319e-002	-5.6319e-002	1.6989e-006
			MSE 5.15 e – 013	

Table 6. Weight, Bias, Epoch, Time and Performance of the Network

Weight and Bias				Epoch, Time and Performance		
Net-IW{1,1}		Net-LW{1,2}	Net-B{1}	Epoch	Time	Performance
0.0249	0.4503	0.7269	0.9798			
0.6714	0.5825	0.3738	0.2848	1052	0:00:14	3.35e-14
0.8372	0.6866	0.5816	0.5950			
0.9715	0.7194	0.1161	0.9622			
0.0569	0.6500	0.0577	0.1858			

Conclusion:

In this work, it has been successfully designed feed forward neural network (FFNN) for solving (MVFIEs). This design includes fast and efficient algorithm (LM) with one hidden layer that has 5 neurons and one output layer. From the numerical examples, it can be seen that the design (FFNN) method is accurate and efficient to estimate the numerical solution of these equations, because the errors decrease to smaller values compared with the solution for the same examples solved by the other methods(8, 9) .

References

1. Babolian E, Maleknejad K, Mordad M. A numerical method for solving Fredholm– Volterra integral equations in two– dimensional space using block pluse functions and an operational matrix. *Journal of* Computational and Applied Mathematics.2011;235: 3965– 3971.
2. Nemati S, Lima P, Ordokhani, Y. Numerical Method for the Mixed Volterra – Fredholm Integral Equations using Hybrid Legendre Functions. *Conference Application of Mathematic 2015.* Institute of Mathematics AS CR, Prague 2015;85:184-193
3. Mohammed K, Numerical Solution for Mixed Volterra – Fredholm Integral Equations of the second kind by using Bernstein Polynomials Method. Mathematical Theory and Modeling ,2015;5(10): 154-162
4. Nili A, Laeli D. A Numerical Solution of Mixed Volterra – Fredholm Integral Equations of Urysohn type on non-rectangular regions using meshless methods. Journal of Linear and Topology Algebra, 2015;4(4): 289-304.
5. Hassan I, Francis A, On the Solution of Volterra – Fredholm and Mixed Volterra – Fredholm Integral Equations Using the New Iterative Method . *Applied Mathematics*,2016;6(1): 1-5.

6. Hristev R M. The ANN Book. GNU public license; 1998. 374 p.
7. Martin T H, Howard D. neural network design. 2nd ed. India: Thomson press;1996.736 p.
8. Laeli D H, Maalek Ghaini F M, Hadizadeh M.A meshless approximate solution of mixed Volterra – Fredholm integral equations. International Journal of Computer Mathematics,2012; 1-12.
9. Lechoslaw H. Computational Methods for Volteraa – Fredholm Integral Equations. Computational Methods in Science and Technology, 2002;8 (2): 13 – 26.

حل معادلة فولتيرا - فريدهولم التكاملية المختلطة باستخدام الشبكات العصبية

ناهض سليم محمد سعيد **امين شامان امين**

قسم الرياضيات التطبيقية، كلية العلوم، جامعة الانبار، الانبار، العراق.

الخلاصة :

الهدف الاساسي في هذا البحث هو تقديم طريقه عدديه جديده لحل هذا النوع من المعادلات باستخدام الشبكات العصبية (ANN). حيث تم تصميم شبة عصبيه ذات تغذية اماميه(FFNN) سريعة، هذا التصميم ذو الطبقات المتعددة والذي يحوي على طبقه واحده خفيه تحتوي على خمسة وحدات خفيه وتستخدم الدالة التحويل (log_sigmoid) وطبقة واحدة للإخراج، وتم تدريب الشبة باستخدام خوارزمية ليفن برك (Levenberg – Marquardt) . ولبيان دقة و كفاءة الطريقة المقدمة تم مقارنة نتائج الامثلة التوضيحية مع الحلول المضبوطة لهذه الأمثلة، و من خلال المقارنة تبين بان الطريقة ذات كفاءة و دقة عالية وذات خطاء قليل جدا.

الكلمات المفتاحية:التغذية الأمامية للشبكة العصبية ،خوارزمية ليفنبرك للتدريب، معادلة فولتيرا- فريدهولم التكاملية المختلطة .