International Journal of Partial Differential Equations and Applications, 2017, Vol. 5, No. 1, 1-9
Available online at http://pubs.sciepub.com/ijpdea/5/1/1
©Science and Education Publishing
DOI:10.12691/ijpdea-5-1-1

Artificial Neural Network for Solving Fuzzy Differential Equations under Generalized H – Derivation

Mazin H. Suhhiem*

Department of Statistics, University of Sumer, Alrifaee, Iraq
*Corresponding author: mazin.suhhiem@yahoo.com

Abstract The aim of this work is to present a novel approach based on the artificial neural network for finding the numerical solution of first order fuzzy differential equations under generalized H-derivation. The differentiability concept used in this paper is the generalized differentiability since a fuzzy differential equation under this differentiability can have two solutions. The fuzzy trial solution of fuzzy initial value problem is written as a sum of two parts. The first part satisfies the fuzzy condition, it contains no adjustable parameters. The second part involves feed-forward neural networks containing adjustable parameters. Under some conditions the proposed method provides numerical solutions with high accuracy.

Keywords: *fuzzy differential equation, artificial neural network, generalized H-derivation, error function, trial solution*

Cite This Article: Mazin H. Suhhiem, “Artificial Neural Network for Solving Fuzzy Differential Equations under Generalized H – Derivation.” *International Journal of Partial Differential Equations and Applications*, vol. 5, no. 1 (2017): 1-9. doi: 10.12691/ijpdea-5-1-1.

1. Introduction

Nowadays, fuzzy differential equations (FDEs) is a popular topic studied by many researchers since it is utilized widely for the purpose of modeling problems in science and engineering. Most of the practical problems require the solution of a FDE which satisfies fuzzy initial or fuzzy boundary conditions, therefore, a fuzzy initial or fuzzy boundary problem should be solved. However, many fuzzy initial or fuzzy boundary value problems could not be solved exactly, sometimes it is even impossible to find their analytical solutions. Thus, considering their approximate solutions is becoming more important [1].

The theory of FDE was first formulated by Kaleva and Seikkala. Kaleva had formulated FDE in terms of the Hukuhara derivative (H-derivative). Buckley and feuring have given a very general formulation of a first-order fuzzy initial value problem. They first find the crisp solution, make it fuzzy and then check if it satisfies the fuzzy differential equation [2].

In recent years artificial neural network (ANN) for estimation of the ordinary differential equation (ODE) and partial differential equation (PDE) has been used. We briefly review some articles in the literature concerning the differential equations. In (1990) lee, Kang [3] used parallel processor computers to solve a first order differential equation with Hopfield neural network models. In (1994) Meade, Fernandez [4,5] solved linear and non-linear ODEs by using feed-forward neural networks (FFNN) architecture and B-splines of degree one. In (1997) Lagaris, Likas, et al. [6,7] used ANN for solving ODEs and PDEs with the initial / boundary value problems. In (1999) Liu, Jammes [8] developed some properties of the trial solution to solve the ODEs by using ANN. In (2003) Ali, Ucar, et al. [9] solved the vibration control problems by using ANN. In (2004) Tawfiq [10] presented and developed supervised and unsupervised algorithms for solving ODE and PDE. In (2006) malek, shekari [11] presented numerical method based on ANN and optimization techniques which the higher-order ODE answers approximates by finding a package form analytical of specific functions. In (2008) Pattanaik, Mishra [12] applied and developed some properties of ANN for solution of PDE in RF Engineering**.** In (2010) Baymani, Kerayechian, et al. [13] proposed ANN approach for solving stokes problems. In (2011) Oraibi [14] designed FFNN for solving ordinary initial value problem. In (2012) Ali [15] designed fast FFNN to solve two point boundary value problems. In (2013) Hussein [16] designed fast FFNN to solve singular boundary value problems. In (2014) Tawfiq, Al-Abrahemee [17] designed ANN to solve singular perturbation problems, and other researchers.

Numerical solution of FDE by using ANN is the subject of a very modern because it only goes back to 2010. In (2010) Effati and pakdaman [18] used ANN for solving FDE, they used for the first time the ANN to approximate fuzzy initial value problems. In (2012) Mosleh, Otadi [19] used ANN for solving fuzzy Fredholm integro-differential equations. In (2013) Ezadi, Parandin, et al. [20] used ANN based on semi-Taylor series to solve first order FDE. In (2016) Suhhiem [21] developed and used fuzzy ANN for solving fuzzy and non-fuzzy differential equations.

In 2008, the concept of the generalized Hukuhara – differentiability is studied by Chalco-Cano and Roman

Flores [22,23] to solve FDE. In this work, for solving FDE Under Generalized H – Derivation, we present modified method which relies on the function approximation capabilities of FFNN and results in the construction of a solution written in a differentiable, closed analytic form. This form employs FFNN as the basic approximation element, whose parameters (weights and biases) are adjusted to minimize an appropriate error function. To train the ANN which we design, we employ optimization techniques, which in turn require the computation of the gradient of the error with respect to the network parameters. In this proposed approach the model function is expressed as the sum of the two terms: the first term satisfies the fuzzy initial / fuzzy boundary conditions and contains no adjustable parameters. The second term can be found by using FFNN, which is trained so as to satisfy the FDE. It is necessary to note that the solution of the FDE by using ANN based on conversion the FDE into a system of ODEs.

2. Basic Definitions

In this section, the basic notations which are used in fuzzy calculus are introduced

Definition (1), [19]: The r-level (or r-cut) set of a fuzzy set $\tilde{A}$ labeled by A_r, is the crisp set of all x in X (universal set) such that : $\mu_{\tilde{A}}(x) \geq r$; i. e.

$$A_r = \{x \in X : \mu_{\tilde{A}}(x) \geq r, r \in [0,1]\}. \quad (1)$$

Definition (2), [20]: Extension Principle

Let X be the Cartesian product of universes $X_1, X_2, \ldots, X_m$ and $\tilde{A}_1, \tilde{A}_2, \ldots, \tilde{A}_m$ be m - fuzzy subset in $X_1, X_2, \ldots, X_m$ respectively, with Cartesian product $\tilde{A} = \tilde{A}_1 \times \tilde{A}_2 \times \ldots \times \tilde{A}_m$ and f is a function from X to a universe Y, $(y = f(X_1, X_2, \ldots, X_m))$. Then, the extension principle allows to define a fuzzy subset $\tilde{B} = f(\tilde{A})$ in Y by $\tilde{B} = \{(y, \mu_{\tilde{B}}(y)) : y = f(X_1, X_2, \ldots, X_m),\ (X_1, X_2, \ldots, X_m) \in X\}$, where

$$\mu_{\tilde{B}}(y) = \begin{cases} \sup_{(x_1,\ldots,x_m) \in f^{-1}(y)} \mathrm{Min}\left\{\mu_{\tilde{A}_1}(x_1), \ldots, \mu_{\tilde{A}_m}(x_m)\right\}, \\ \qquad if\ f^{-1}(y) \neq \varnothing \\ 0, \qquad otherwise \end{cases} \quad (2)$$

and f^{-1} is the inverse image of f.

For m = 1, the extension principle will be:

$$\tilde{B} = f(\tilde{A}) = \{(y, \mu_{\tilde{B}}(y)) : y = f(x), x \in X\},$$

where

$$\mu_{\tilde{B}}(y) = \begin{cases} \sup_{x \in f^{-1}(y)} \mu_{\tilde{A}}(x), if\ f^{-1}(y) \neq \varnothing \\ 0, \qquad otherwise. \end{cases} \quad (3)$$

Definition (3), [1]: Fuzzy Number

A fuzzy number ũ is completely determined by an ordered pair of functions $(\underline{u}(r), \overline{u}(r))$, $0 \leq r \leq 1$, which satisfy the following requirements :

1) $\underline{u}(r)$ is a bounded left continuous and non decreasing function on [0,1].

2) $\overline{u}(r)$ is a bounded left continuous and non increasing function on [0,1].

3) $\underline{u}(r) \leq \overline{u}(r)$, $0 \leq r \leq 1$.

The crisp number a is simply represented by :

$$\underline{u}(r) = \overline{u}(r) = a, 0 \leq r \leq 1.$$

The set of all the fuzzy numbers is denoted by E^1.

Remark (1), [19]: For arbitrary $\tilde{u} = (\underline{u}, \overline{u})$, $\tilde{v} = (\underline{v}, \overline{v})$ and $K \in R$, the addition and multiplication by K can be defined as :

1)

$$\underline{(u+v)}(r) = \underline{u}(r) + \underline{v}(r) \quad (4)$$

2)

$$\overline{(u+v)}(r) = \overline{u}(r) + \overline{v}(r) \quad (5)$$

3)

$$\underline{(Ku)}(r) = K\underline{u}(r), \overline{(Ku)}(r) = K\overline{u}(r), if\ K \geq 0 \quad (6)$$

4)

$$\underline{(Ku)}(r) = K\overline{u}(r), \overline{(Ku)}(r) = K\underline{u}(r), if\ K < 0. \quad (7)$$

For all r ∈ [0,1].

Remark (2), [2]:

The distance between two arbitrary fuzzy numbers $\tilde{u} = (\underline{u}, \overline{u})$ and $\tilde{v} = (\underline{v}, \overline{v})$ is given as :

$$D(\tilde{u}, \tilde{v}) = \left[\int_0^1 (\underline{u}(r) - \underline{v}(r))^2 dr + \int_0^1 (\overline{u}(r) - \overline{v}(r))^2 dr\right]^{\frac{1}{2}}. \quad (8)$$

Remark (3), [2]: (E^1, D) is a complete metric space.

Remark (4), [1]: The operations of fuzzy numbers (in parametric form) can be generalized from that of crisp intervals. Let us have a look at the operations of intervals. $\forall a_1, b_1$, $a_2, b_2 \in R$, $A = [a_1, b_1]$ and $B = [a_2, b_2]$.

Assuming A and B numbers expressed as interval, main operations of intervals are:

1) Addition: $A + B = [a_1, b_1] + [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$.

2) Subtraction: $A - B = [a_1, b_1] - [a_2, b_2] = [a_1 - b_2, b_1 - a_2]$.

3) Multiplication:

$$A.B = \begin{bmatrix} \min\{a_1a_2, a_1b_2, b_1a_2, b_1b_2\}, \\ \max\{a_1a_2, a_1b_2, b_1a_2, b_1b_2\} \end{bmatrix}$$

4) Division: $A/B = [\min\{a_1/a_2, a_1/b_2, b_1/a_2, b_1/b_2\}, \max\{a_1/a_2, a_1/b_2, b_1/a_2, b_1/b_2\}]$ excluding the case $a_2 = 0$ or $b_2 = 0$.

5) Inverse: $A^{-1} = [a_1, b_1]^{-1} = \left[\min\left\{\frac{1}{a_1}, \frac{1}{b_1}\right\}, \max\left\{\frac{1}{a_1}, \frac{1}{b_1}\right\}\right]$

excluding the case $a_1 = 0$ or $b_1 = 0$.

In the case of $0 \le a_2 \le b_2$, multiplication operation can be simplified as:

$$A.B = \left[\min\{a_1a_2, a_1b_2\}, \max\{b_1a_2, b_1b_2\}\right]$$

when previous sets A and B is defined in the positive real number R^+, the operations of multiplication, division and inverse are written as :

3´) Multiplication: $A.B = [a_1, b_1]. [a_2, b_2] = [a_1 a_2, b_1 b_2]$

4´) Division : $A /B = [a_1, b_1] / [a_2, b_2] = \left[\frac{a_1}{b_2}, \frac{b_1}{a_2}\right]$.

5´) Inverse: $A^{-1} = [a_1, b_1]^{-1} = \left[\frac{1}{b_1}, \frac{1}{a_1}\right]$.

Definition (4), [20]: Triangular Fuzzy Number

Among the various shapes of fuzzy numbers, triangular fuzzy numbers is the most popular one. A triangular fuzzy number is a fuzzy number represented with three points as follows : $\tilde{A} = (a_1, a_2, a_3)$, where $a_1 \le a_2 \le a_3$.

This representation is interpreted as membership functions :

$$\mu_{\tilde{A}}(x) = \begin{cases} 0, & if\ x < a_1 \\ \frac{x-a_1}{a_2-a_1}, & if\ a_1 \le x \le a_2 \\ \frac{a_3-x}{a_3-a_2}, & if\ a_2 \le x \le a_3 \\ 0, & if\ x > a_3. \end{cases} \quad (9)$$

Now if you get crisp interval by r- cut operation, interval $[A]_r$ shall be obtained as follows $\forall$ r $\in$ [0,1] from:

$$\frac{\underline{A}-a_1}{a_2-a_1} = r, \frac{a_3-\overline{A}}{a_3-a_2} = r.$$

We get: $\underline{A} = (a_2 - a_1)r + a_1$, $\overline{A} = (a_2 - a_3)r + a_3$.

Thus:

$$[A]_r = \left[\underline{A}, \overline{A}\right] = \left[(a_2-a_1)r + a_1, (a_2-a_3)r + a_3\right] \quad (10)$$

which is the parametric form of triangular fuzzy number $\tilde{A}$.

Definition (5), [19]: Fuzzy Function

A classical function F: X $\to$ Y maps from a fuzzy domain $\tilde{A} \subseteq X$ into a fuzzy range $\tilde{B} \subseteq Y$ if and only if $\forall x \in X, \mu_{\tilde{B}}(F(x)) \ge \mu_{\tilde{A}}(x)$.

Remark (5), [18]:

(1) The function $F: R \to E^1$ is called a fuzzy function.

(2) We call every function defined in set $\tilde{A} \subseteq E^1$ to $\tilde{B} \subseteq E^1$ a fuzzy function.

Definition (6), [18]: The fuzzy function $F: R \to E^1$ is said to be continuous if :

For an arbitrary $t_1 \in R$ and $\epsilon > 0$ there exists a $\delta > 0$ such that :

$$|t-t_1| < \delta \Rightarrow D(F(t), F(t_1)) < \epsilon,$$

where D is the distance between two fuzzy numbers.

Definition (7), [18]: Let I be a real interval. The r-level set of the fuzzy function $y : I \to E^1$ can be denoted by :

$$[y(t)]^r = \left[y_1^r(t), y_2^r(t)\right] \quad t \in I, r \in [0,1] \quad (11)$$

The Seikkala derivative $y'(t)$ of the fuzzy function y(t) is defined by :

$$[y'(t)]^r = \left[(y_1^r)'(t), (y_2^r)'(t)\right] \quad t \in I, r \in [0,1]. \quad (12)$$

Definition (8), [18]: let $u, v \in E^1$. If there exist $w \in E^1$ such that u= v + w, then w is called the H-difference (Hukuhara-difference) of u, v and it is denoted by $w = u \ominus v$.

In this work the $\ominus$ sign stands always for H-difference, and let us remark that $u \ominus v \ne u + (-1) v$.

Definition (9), [22,23]: H – Differentiability

Let $F: (a,b) \to E^1$ and $t_0 \in (a,b)$. We say that F is H-differential (Hukuhara-differential) at t_0, if there exists an element $F'(t_0) \in E^1$ such that for all h > 0 (sufficiently small), $\exists$ $F(t_0+h) \ominus F(t_0)$, $F(t_0) \ominus F(t_0 - h)$ and the limits (in the metric D)

$$\lim_{h\to 0} \frac{F(t_0+h) \ominus F(t_0)}{h} = \lim_{h\to 0} \frac{F(t_0) \ominus F(t_0-h)}{h} = F'(t_0) \quad (13)$$

then $F'(t_0)$ is called fuzzy derivative (H-derivative) of F at t_0, where D is the distance between two fuzzy numbers.

It is necessary to note that the definition (9) is the classical definition of the H-derivative (or differentiability in the sense of Hukuhara).

Definition (10), [22,23]: Generalized H – Differentiability

Let $F : T \to E^1$ and $t_0 \in T \subseteq R$. F is differentiable at t_0, if

(1) there exist an element $F'(t_0) \in E^1$, such that for all h > 0 sufficiently small, there are $F(t_0+h) \ominus F(t_0)$, $F(t_0) \ominus F(t_0-h)$ and the limits (in the metric D)

$$\lim_{h\to 0} \frac{F(t_0+h) \ominus F(t_0)}{h} = \lim_{h\to 0} \frac{F(t_0) \ominus F(t_0-h)}{h} = F'(t_0) \quad (14)$$

(in this case, F is called (1)-differentiable)

or

(2) there exist an element $F'(t_0) \in E^1$, such that for all h > 0 sufficiently small, there are $F(t_0) \ominus F(t_0 + h)$, $F(t_0-h) \ominus F(t_0)$ and the limits (in the metric D)

$$\lim_{h\to 0} \frac{F(t_0) \ominus F(t_0+h)}{-h} = \lim_{h\to 0} \frac{F(t_0-h) \ominus F(t_0)}{-h} = F'(t_0) \quad (15)$$

(in this case, F is called (2)-differentiable)

Where the relation (1) is the classical definition of the H-derivative.

Theorem (1): Let $F : I \to E^1$ be a function and denote $[F(t)]^r = [f_r(t), g_r(t)]$, for each $r \in [0,1]$. Then

(i) If F is differentiable in the first form **(1)** of definition (10), then f_r and g_r are differentiable functions and

$$[F'(t)]^r = \left[f_r'(t), g_r'(t)\right]$$

(ii) If F is differentiable in the second form **(2)** of definition (10), then f_r and g_r are differentiable functions and

$$[F'(t)]^r = \left[g_r'(t), f_r'(t)\right]$$

Proof: see [22].

3. Artificial Neural Networks [10]

Artificial neural networks (ANNs) 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. ANN has been developed as generalizations of mathematical models of human cognition or neural biology, based on the assumptions:

1) Information processing occurs at many simple elements called neurons that is fundamental to the operation of ANNs.

2) Signals are passed between neurons over connection links.

3) Each connection link has an associated weight which, in a typical neural net, multiplies the signal transmitted.

4) Each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

Note: The units in a network are organized into a given topology by a set of connections, or weights, shown as lines in a diagram.

3.1. Characterize of Artificial Neural Network [10]

ANN is Characterized by:

1) Architecture: it is pattern of connections between the neurons.

2) Training Learning Algorithm: it is method of determining the weights on the connections.

3) Activation function: The output of a neuron depends on the neuron's input and on its activation function.

3.2. Typical Architecture of ANN [10]

ANNs are often classified as single layer or multilayer. In determining the number of layers, the input units are not counted as a layer, because they perform no computation. Equivalently, the number of layers in the net can be defined to be the number of layers of weighted interconnects links between the slabs of neurons. This view is motivated by the fact that the weights in a net contain extremely important information.

3.3. The Bias [21]

In sections (3.4) and (3.5), we describe the main implementation of the back-propagation algorithm for multi-layer feed forward neural network (FFNN). The most implementations of this algorithm employ an additional class of weights known as biases (Figure 1). Biases are values that are added to the sums calculated at each node(except input nodes) during the feed-forward phase. The negative of a bias is sometimes called a threshold. For simplicity, biases are commonly visualized simply as values associated with each node in the intermediate and output layers of a network, but in practice are treated in exactly the same manner as other weights, with all biases simply being weights associated with vectors that lead from a single node whose location is outside of the main network.

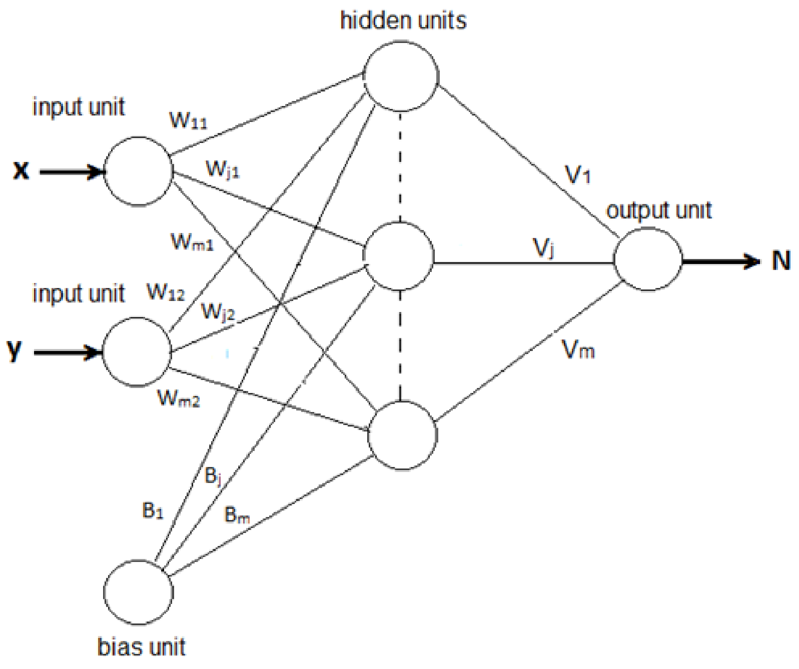

Figure 1. (2 × m × 1) Totally connected FFNN

3.4. Multilayer Feed Forward Architecture [21]

In a layered neural network the neurons are organized in the form of layers. We have at least two layers: an input and an output layer. The layers between the input and the output layer (if any) are called hidden layers, whose computation nodes are correspondingly called hidden neurons or hidden units. Extra hidden neurons raise the network's ability to extract higher-order statistics from (input) data. The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitute the input signals applied to the neurons (computation nodes) in the second layer (i. e., the first hidden layer). The output signals of the second layer are used as inputs to the third layer, and so on for the rest of the network. A layer of nodes projects onto the next layer of the neurons (computation nodes), but not vice versa. In other words, this network is a feed forward neural network (Figure 1). i.e., when any output of the neurons is input of neurons of the same level or preceding levels, the network is described as feed forward, if there is at least one connected exit as entrance of neurons of previous levels or of the same level, including themselves, the network is denominated of feedback. The feedback networks that have at least a closed loop of back propagation are called recurrent. The neurons in each layer of the network have as their inputs the output signals of the preceding layer only. The set of output signals of the neurons in the output (final) layer of the network constitutes the overall response of the network to the activation pattern supplied by the source nodes in the input (first) layer. The ANN is said to be totally connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer, otherwise the network is called partially connected. In this work, totally connected multilayer FFNN is used.

3.5. Back propagation Training Algorithm [21]

Training a network by back propagation involves three stages:

1)The feed forward of the input training pattern.

2) The back propagation of the associated error.

3) The adjustment of the weights.

The term back propagation refers to the process by which derivatives of the neural network error with respect to the neural network weights and biases can be computed. This process can be used with a number of different optimization strategies. In another word the standard back propagation is based on the gradient descent, back propagation also known as the Generalized Delta Rule. It is the most widely used supervised training algorithm for ANN. Back propagation is a well-known training method for the multilayer FFNN and it has many industrial applications in function approximation, pattern association, and pattern classification. Because of its importance, we will discuss it in some detail

3.6. Activation Function [21]

The activation function (sometimes called a transfer function) can be a linear or nonlinear function. There are many different types of activation functions. Selection of one type over another depends on the particular problem that the neuron (or ANN) is to solve. The activation function denoted by $S: R \to R$ defines the output of a neuron, which is bounded monotonically increasing, differentiable and satisfies: $\mathrm{Lim}_{x\to+\infty}$ s(x) =1 and $\mathrm{Lim}_{x\to-\infty}$ s(x) =0.

The sigmoid function, is by far the most common form of activation function used in construction of ANNs. An example of the sigmoid function is the logistic function defined the range from 0 to 1, an important feature of the sigmoid function that it is differentiable.

It is sometimes desirable to have the activation function range from -1 to 1 allowing an activation function of the sigmoid type to assume negative values, for example, the hyperbolic tangent function which is smooth function.

During this work, we take $s(x) = \tanh(x)$ as an activation function, depending on the results of [21] which evidence that an transfer function $\tanh(x)$ enables the training algorithm to learn faster.

Theorem (2): The World Approximation Builder

The multi-Layer perceptron (MLP) network with one hidden Layer with a sigmoid functions in the middle layer and linear transformation functions in output layer are able to approximate all functions in any degree of the integral of the square. (see [3]).

4. Technique of The Proposed Method

4.1. First Order Fuzzy Differential Equation

A fuzzy differential equation of the first order is in the form:

$$y'(t) = F(x, y(x)), x \in [a,b] \qquad (16)$$

with the fuzzy initial condition $y(a) = y_0$, where y is a fuzzy function of X and F $(x, y(x))$ is a fuzzy function of the crisp variable X and the fuzzy variable y while y' is the fuzzy derivative (If we consider $y'(x)$ in the second form (2) of definition (10) According to our proposed method) of y and y_0 is a fuzzy number

It is clear that the fuzzy function $F(x, y)$ is the mapping $F: R \times E^1 \to E^1$ [18].

Now it is possible to replace (16) by the following equivalent system:

$$\begin{aligned} \underline{y}'(x) &= \underline{F}(x,y) = H(x,\underline{y},\overline{y}),\ \underline{y}(a) = \underline{y}_0 \\ \overline{y}'(x) &= \overline{F}(x,y) = G(x,\underline{y},\overline{y}),\ \overline{y}(a) = \overline{y}_0 \end{aligned} \qquad (17)$$

where

$$\begin{aligned} H(x,\underline{y},\overline{y}) &= \min\{F(x,u) : u \in [\underline{y},\overline{y}]\} \\ G(x,\underline{y},\overline{y}) &= \max\{F(x,u) : u \in [\underline{y},\overline{y}]\}. \end{aligned} \qquad (18)$$

The parametric form of system (17) is given by:

$$\begin{aligned} \underline{y}'(x,r) &= H[x,\underline{y}(x,r),\overline{y}(x,r)],\ \underline{y}(a,r) = \underline{y}_0(r) \\ \overline{y}'(x,r) &= G[x,\underline{y}(x,r),\overline{y}(x,r)],\ \overline{y}(a,r) = \overline{y}_0(r) \end{aligned} \qquad (19)$$

where $X \in [a, b]$ and $r \in [0, 1]$. Now with a discretization of the interval $[a, b]$, a set of points x_i, $i = 1,2,3, \dots g$ are obtained. Thus for an arbitrary $x_i \in [a, b]$, the system (19) can be rewritten as:

$$\begin{aligned} &\underline{y}'(x_i, r) - H\left[x_i, \underline{y}(x_i, r), \overline{y}(x_i, r)\right] = 0 \\ &\overline{y}'(x_i, r) - G\left[x_i, \underline{y}(x_i, r), \overline{y}(x_i, r)\right] = 0 \end{aligned} \quad (20)$$

with the initial conditions: $\underline{y}(a, r) = \underline{y}_0(r)$, $\overline{y}(a, r) = \overline{y}_0(r)$, $r \in [0, 1]$.

In this work, the function approximation capabilities of feed-forward neural networks is used by expressing the trial solution for the system (19) as the sum of two terms (see eq 22). The first term satisfies the initial conditions /boundary conditions and contains no adjustable parameters. The second term involves a feed-forward neural network to be trained so as to satisfy the fuzzy differential equations. Since it is known that a multilayer perceptron with one hidden layer can approximate any function to arbitrary accuracy, the multilayer perceptron is used as the type of the network architecture.

If $\underline{y}_t(x, r, \underline{p})$ is a trial solution for the first equation in system (19) and $\overline{y}_t(x, r, \overline{p})$ is a trial solution for the second equation in system (19) where $\underline{p}$ and $\overline{p}$ are adjustable parameters.

Indeed, $\underline{y}_t(x, r, \underline{p})$ and $\overline{y}_t(x, r, \overline{p})$ are approximation of $\underline{y}(x, r)$ and $\overline{y}(x, r)$ respectively, then a discretize version of the system (19) can be converted to the following optimization problem:

$$\min_{\overline{p}} \sum_{i=1}^{g} \left(\begin{aligned} &\left(\underline{y}'_t(x_i, r, \underline{p}) - H\begin{bmatrix} x_i, \underline{y}_t(x_i, r, \underline{p}), \\ \overline{y}_t(x_i, r, \overline{p}) \end{bmatrix} \right)^2 \\ &+ \left(\overline{y}'_t(x_i, r, \overline{p}) - G\begin{bmatrix} x_i, \underline{y}_t(x_i, r, \underline{p}), \\ \overline{y}_t(x_i, r, \overline{p}) \end{bmatrix} \right)^2 \end{aligned} \right) \quad (21)$$

(Here $\vec{p} = (\underline{p}, \overline{p})$ contains all adjustable parameters)
subject to the initial conditions: $\underline{y}_t(a, r, \underline{p}) = \underline{y}_0(r)$, $\overline{y}_t(a, r, \overline{p}) = \overline{y}_0(r)$.

Each trial solution $\underline{y}_t(x, r, \underline{p})$ and $\overline{y}_t(x, r, \overline{p})$ employs one feed-forward neural network for which the corresponding networks are denoted by $\underline{N}(x, r, \underline{p})$ and $\overline{N}(x, r, \overline{p})$ with adjustable parameters $\underline{p}$ and $\overline{p}$ respectively. The trial solutions $\underline{y}_t$ and $\overline{y}_t$ should satisfy the initial conditions, and the networks must be trained to satisfy the differential equations. Thus $\underline{y}_t$ and $\overline{y}_t$ can be chosen as follows:

$$\begin{aligned} &\underline{y}_t(x, r, \underline{p}) = \underline{y}(a, r) + (x - a)\overline{N}(x, r, \overline{p}) \\ &\overline{y}_t(x, r, \overline{p}) = \overline{y}(a, r) + (x - a)\underline{N}(x, r, \underline{p}) \end{aligned} \quad (22)$$

where $\underline{N}(x, r, \underline{p})$ and $\overline{N}(x, r, \overline{p})$ are single-output feed-forward neural network with adjustable parameters $\underline{p}$ and $\overline{p}$ respectively. Here X and r are the network inputs. It is easy to see that in (22), $\underline{y}_t$ and $\overline{y}_t$ satisfy the initial conditions.

Thus the corresponding error function that must be minimized over all adjustable neural network parameters will be:

$$E = \sum_i \left(\begin{aligned} &\left[\frac{\partial \underline{y}_t(x_i, r, \underline{p})}{\partial x} - H\left(x_i, \underline{y}_t(x_i, r, \underline{p})\right) \right]^2 \\ &+ \left[\frac{\partial \overline{y}_t(x_i, r, \overline{p})}{\partial x} - G\left(x_i, \overline{y}_t(x_i, r, \overline{p})\right) \right]^2 \end{aligned} \right) \quad (23)$$

where x_i's are points in [a, b].

For solving FDE which described in this subsection we use two ANNs, each network is of dimension $2 \times m \times 1$: two input units x and r, one hidden layer with m units and one linear output unit.

For every entries x and r the input neurons makes no changes in its input, so the inputs to the hidden neurons are :

$$\begin{aligned} &\underline{net}_j = x\underline{w}_{j1} + r\underline{w}_{j2} + \underline{b}_j \\ &\overline{net}_j = x\overline{w}_{j1} + r\overline{w}_{j2} + \overline{b}_j, \quad j = 1, 2, \dots m \end{aligned} \quad (24)$$

$\underline{w}_{j1}$ and $\underline{w}_{j2}$ are the weight parameters from the input layer to the jth unit in the hidden layer in the first network, $\overline{w}_{j1}$ and $\overline{w}_{j2}$ are the weight parameters from the input layer to the jth unit in the hidden layer in the second network, $\underline{b}_j$ and $\overline{b}_j$ are the jth weight biases for the jth units in the hidden layers in the first and second network.

The outputs in the hidden neurons are:

$$\begin{aligned} &\underline{z}_j = s\left(\underline{net}_j\right) = s\left(x\underline{w}_{j1} + r\underline{w}_{j2} + \underline{b}_j\right) \\ &\overline{z}_j = s\left(\overline{net}_j\right) = s\left(x\overline{w}_{j1} + r\overline{w}_{j2} + \overline{b}_j\right). \end{aligned} \quad (25)$$

The output neurons makes no changes in its inputs, so the inputs to the output neurons are equal to outputs:

$$\begin{aligned} &\underline{N}(x, r, \underline{p}) = \sum_{j=1}^{m} \underline{v}_j \underline{z}_j = \sum_{j=1}^{m} \underline{v}_j s\left(x\underline{w}_{j1} + r\underline{w}_{j2} + \underline{b}_j\right) \\ &\overline{N}(x, r, \overline{p}) = \sum_{j=1}^{m} \overline{v}_j \overline{z}_j = \sum_{j=1}^{m} \overline{v}_j s\left(x\overline{w}_{j1} + r\overline{w}_{j2} + \overline{b}_j\right) \end{aligned} \quad (26)$$

where $\underline{v}_j$ and $\overline{v}_j$ are the weight parameters from the jth units in the hidden layers to the output layer in the first and second network.

4.2. Reducing a FDE to a System of ODEs [22,23]

The solution of the fuzzy differential equation (16) is depend on the choice of the derivative (in the first form or in the second form of definition (10).

Let us explain the proposed method, if we denote

$$[y(x)]^r = \left[y_1^r(x), y_2^r(x)\right], \quad [y_0]^r = [y_{01}^r, y_{02}^r]$$

and

$$[F(x, y(x))]^r = \begin{bmatrix} F_1^r\left(x, y_1^r(x), y_2^r(x)\right), \\ F_2^r\left(x, y_1^r(x), y_2^r(x)\right) \end{bmatrix} \quad (27)$$

we have the following results :

Case I. If we consider $y'(x)$ in the first form **(1)** of definition (10), then we have to solve the following system of ODEs

$$\frac{d}{dt}\left(y_1^r(x)\right)=F_1^r\left(x,y_1^r(x),y_2^r(x)\right)\quad y_1^r(a)=y_{01}^r$$
$$\frac{d}{dt}\left(y_2^r(x)\right)=F_2^r\left(x,y_1^r(x),y_2^r(x)\right)\quad y_2^r(a)=y_{02}^r.$$

Case II. If we consider $y'(t)$ in the second form **(2)** of definition (10) then we have to solve the following system of ODEs

$$\frac{d}{dt}\left(y_1^r(x)\right)=F_2^r\left(x,y_1^r(x),y_2^r(x)\right)\quad y_1^r(a)=y_{01}^r$$
$$\frac{d}{dt}\left(y_2^r(x)\right)=F_1^r\left(x,y_1^r(x),y_2^r(x)\right)\quad y_2^r(a)=y_{02}^r.$$

The existence and uniqueness of the two solutions (for problem (16)) which described above are given by the following theorem

Theorem (3): Let $F: I\times E^1\rightarrow E^1$ be a continuous fuzzy function such that there exists $k>0$ such that $D(F(x,w),F(x,z))\le k\,D(w,z)$ for all $t\in I$ and $w,z\in E^1$ then the problem (16) has two solutions (one (1)-differentiable and the other one (2)-differentiable) on I, where $I=[a,b]$.

Proof: see [23].

To illustrate how we can find the two solutions for a fuzzy differential equation under generalized H-derivation, we present the following example :

Consider the fuzzy initial value problem

$$y'=-y(x),\ \ y(0)=[0.96+0.04r,1.01-0.01r]$$

(1) According to subsection (4.2), **Case I.**, after reducing the above problem, we have the following system of ODEs

$$\frac{d}{dt}\left(y_1^r(x)\right)=-y_1^r(x),\ \ y_1^r(0)=0.96+0.04r$$
$$\frac{d}{dt}\left(y_2^r(x)\right)=-y_2^r(x),\ \ y_2^r(0)=1.01-0.01r$$

Which gives the following fuzzy analytical solution

$$y(x,\alpha)=[(0.96+0.04\alpha)e^{-x},(1.01-0.01\alpha)e^{-x}].$$

(2) According to subsection (4.2), **Case II.**, after reducing the above problem, we have the following system of ODEs

$$\frac{d}{dt}\left(y_1^r(x)\right)=-y_2^r(x),\ \ y_1^r(0)=0.96+0.04r$$
$$\frac{d}{dt}\left(y_2^r(x)\right)=-y_1^r(x),\ \ y_2^r(0)=1.01-0.01r$$

Which gives the following fuzzy analytical solution

$$y(x,\alpha)=\begin{bmatrix}(0.985+0.015r)e^{-x}-(1-r)0.025e^{x},\\(0.985+0.015r)e^{-x}+(1-r)0.025e^{x}\end{bmatrix}.$$

5. Numerical Example

To show the behavior and properties of the proposed method, one problem will be solved in this section. We have used a multilayer perceptron having one hidden layer with ten hidden units and one output unit. The activation function of each hidden unit is hyperbolic tangent activation function. The analytical solutions $\underline{y}_a(x,r)$ and $\overline{y}_a(x,r)$ have been known in advance. Therefore, we test the accuracy of the obtained solutions by computing the deviation (absolute error):

$$\overline{e}(x,r)=\left|\overline{y}_a(x,r)-\overline{y}_t(x,r)\right|,$$
$$\underline{e}(x,r)=\left|\underline{y}_a(x,r)-\underline{y}_t(x,r)\right|$$

Where $\overline{y}_t(x,r)$ and $\underline{y}_t(x,r)$ are the trial solutions.

In order to obtain better results, more hidden units or training points may be used. To minimize the error function we have used BFGS quasi-Newton method (For more details, see [21]).

Example (1): Consider the following fuzzy initial value problem:

$$y'=-y+x+1,\ \text{with}\ x\in[0,1]$$

$y(0)=[0.96+0.04r,1.01-0.01r]$, where $r\in[0,1]$.

The analytical solution (According to subsection(4.2), Case II.) for this problem are :

$$\underline{y}_a(x,r)=x+(0.985+0.015r)e^{-x}-(1-r)0.025e^{x}$$
$$\overline{y}_a(x,r)=x+(0.985+0.015r)e^{-x}+(1-r)0.025e^{x}.$$

The trial solution (According to the proposed method in this work) for this problem are :

$$\underline{y}_t(x,r)=(0.96+0.04r)+x\overline{N}(x,r,\overline{p})$$
$$\overline{y}_t(x,r)=(1.01-0.01r)+x\underline{N}(x,r,\underline{p}).$$

The ANN trained using a grid of ten equidistant points in [0, 1].

The error function that must be minimized for this problem will be :

$$\begin{aligned}E=\textstyle\sum_{i=1}^{11}(&[x_i\textstyle\sum_{j=1}^{10}\underline{v}_j\underline{w}_{j1}s'\left(x_i\underline{w}_{j1}+r\underline{w}_{j2}+\underline{b}_j\right)\\&+(1+x_i)\textstyle\sum_{j=1}^{10}\underline{v}_js\left(x_i\underline{w}_{j1}+r\underline{w}_{j2}+\underline{b}_j\right)\\&-x_i-0.01r+0.01]^2\\&+[x_i\textstyle\sum_{j=1}^{10}\overline{v}_j\overline{w}_{j1}s'\left(x_i\overline{w}_{j1}+r\overline{w}_{j2}+\overline{b}_j\right)\\&+(1+x_i)\textstyle\sum_{j=1}^{10}\overline{v}_js\left(x_i\overline{w}_{j1}+r\overline{w}_{j2}+\overline{b}_j\right)\\&-x_i+0.04r-0.04]^2).\end{aligned}\qquad(28)$$

Then we use (28) to update the weights and biases. analytical and trial solutions for this problem can be found in Table 1 and Table 2.

6. Conclusion

In this paper, we have presented numerical method based on artificial neural network for solving first order fuzzy initial value problem under generalized H-derivation. The method which we have used allows us to translate the FDE into system of ODEs and then solve this system. we demonstrate the ability of ANN to approximate the solution of FDEs. Therefore, we can conclude that the method which we proposed can handle effectively all types of FDEs and provide accurate approximate solution throughout the whole domain and not only at the training set. As well, one can use the interpolation techniques to find the approximate solution at points between the training points or at points outside the training set. Further research is in progress to apply and extend this method to solve higher order FDEs.

Table 1. Analytical and trial solutions for example (1), x = 2

r	$\underline{y}_a(x,r)$	$\underline{y}_t(x,r)$	$\underline{e}(x,r)$	$\overline{y}_a(x,r)$	$\overline{y}_t(x,r)$	$\overline{e}(x,r)$
0	1.948578852	1.948579274	0.000000422	2.318031656	2.318032417	0.000000761
0.1	1.967254495	1.967254967	0.000000472	2.299762019	2.299762662	0.000000643
0.2	1.985930138	1.985930525	0.000000387	2.281492381	2.281492466	0.000000085
0.3	2.004605781	2.004605986	0.000000205	2.263222744	2.263223455	0.000000711
0.4	2.023281424	2.023282053	0.000000629	2.244953107	2.244953513	0.000000406
0.5	2.041957068	2.041957703	0.000000635	2.226683470	2.226683566	0.000000096
0.6	2.060632711	2.060632805	0.000000094	2.208413832	2.208413946	0.000000114
0.7	2.079308354	2.079308402	0.000000048	2.190144195	2.190144557	0.000000362
0.8	2.097983997	2.097984071	0.000000074	2.171874558	2.171875271	0.000000713
0.9	2.116659640	2.116660365	0.000000725	2.153604920	2.153604974	0.000000054
1	2.135335283	2.135335791	0.000000508	2.135335283	2.135336111	0.000000828

Table 2. Analytical and trial solutions for example (1), r = 0.5

x	$\underline{y}_a(x,r)$	$\underline{y}_t(x,r)$	$\underline{e}(x,r)$	$\overline{y}_a(x,r)$	$\overline{y}_t(x,r)$	$\overline{e}(x,r)$
0	0.98	0.98	0	1.005	1.005	0
0.1	0.984236500	0.984236574	0.000000074	1.011865774	1.011865864	0.000000090
0.2	0.997322738	0.997322850	0.000000112	1.027857807	1.027858634	0.000000827
0.3	1.018388849	1.018388968	0.000000119	1.052135319	1.052136129	0.000000810
0.4	1.046644837	1.046645251	0.000000414	1.083940454	1.083941124	0.000000670
0.5	1.081372664	1.081372866	0.000000202	1.122590696	1.122591654	0.000000958
0.6	1.121919064	1.121919133	0.000000069	1.167472034	1.167472118	0.000000084
0.7	1.167689005	1.167689342	0.000000337	1.218032823	1.218032891	0.000000068
0.8	1.218139735	1.218140638	0.000000903	1.273778258	1.273778484	0.000000226
0.9	1.272775348	1.272775429	0.000000081	1.334265426	1.334265671	0.000000245
1	1.331141823	1.331142019	0.000000196	1.399098868	1.399099381	0.000000513

References

[1] Mosleh M., Otadi M, "Simulation and Evaluation of Fuzzy Differential Equations by Fuzzy Neural Network", Applied Soft Computing, 12, 2817-2827, 2012.

[2] Buckley J. J., Feuring T., "Fuzzy Differential Equations", Fuzzy Sets and Systems, 110, 69-77, 2000.

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

[4] 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.

[5] 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.

[6] 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.

[7] Lagaris I. E., Likas A., et al., "Artificial Neural Networks For Solving Ordinary and Partial Differential Equations", IEEE Transaction on Neural Networks, Vol. 9, No. 5, 987-1000, 1998.

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

[9] Alli H., Ucar A., et al., "The Solutions of Vibration Control Problems Using Artificial Neural Networks", Journal of the Franklin Institute, 340, 307-325, 2003.

[10] 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.

[11] 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.

[12] 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.

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

[14] Oraibi Y. A., "Design Feed-Forward Neural Networks For Solving Ordinary Initial Value Problem", M.Sc. Thesis, College of Education Ibn Al-Haitham, University of Baghdad, Iraq, 2011.

[15] Ali M. H., "Design Fast Feed-Forward Neural Networks to Solve Two Point Boundary Value Problems", M.Sc. Thesis, College of Education Ibn Al-Haitham, University of Baghdad, Iraq, 2012.

[16] Hussein A. A. T., "Design Fast Feed-Forward Neural Networks to Solve Singular Boundary Value Problems", M.Sc. Thesis, College of Education Ibn Al-Haitham, University of Baghdad, Iraq, 2013.

[17] Tawfiq L. N. M., Al-Abrahemee K. M. M., "Design Neural Network to Solve Singular Perturbation Problems", Applied and Computational Mathematics, Vol. 3, No. 3, 1-5, 2014.

[18] Effati S., Pakdaman M., "Artificial Neural Network Approach For Solving Fuzzy Differential Equations", Information Sciences, 180, 1434-1457, 2010.

[19] Mosleh M., Otadi M., "Fuzzy Fredholm Integro-Differential Equations with Artificial Neural Networks", Communications in Numerical Analysis, Article ID cna-00128, 1-13, 2012.

[20] Ezadi S., Parandin N., et al., "Numerical Solution of Fuzzy Differential Equations Based on Semi-Taylor by Using Neural Network", Journal of Basic and Applied Scientific Research, 3(1s), 477-482, 2013.

[21] 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.

[22] Cano Y. C., Flores H. R., "On New Solutions of Fuzzy Differential Equations", Chaos, Solitons and Fractals, 38, 112-119, 2008.

[23] Cano Y. C., Flores H. R., et al., "Fuzzy Differential Equations with Generalized Derivative", Fuzzy Sets and Systems, 160, 1517-1527, 2008.