Academic Science Journal

Using Fuzzy Feed-Forward Neural Network for Linguistic Processing

AbdulRahim K. Rahi[1] and Sozan S. Haydar Ali[2]

[1]Dijlah University Collage
Department of Statistics and Informatics – Collage of Administration and Economics - Sulamani University

[1] abdulrahim.khalaf@duc.edu.iq
[2] sozan.haider@univsul.edu.iq

Received: 24 November 2022 **Accepted: 30 January 2023**

DOI: https://dx.doi.org/10.24237/ASJ.01.02.723B

Abstract

Fuzzy sets have been implemented efficiently to manage unclear data, language terms, and vague notions. Recently, considerable work has been dedicated to merging neural-network techniques with fuzzy sets. In this study, present the structure of a fuzzy feed-forward neural network (FFFNN) with a trapezoidal fuzzy set. In addition to handling real input vectors, it is also capable of handling fuzzy input vectors. Generally, the output of a FNN is a fuzzy vector. According to the extension principle of Zadeh, each unit of a FNN has an input-output relationship. To determine the costs associated with fuzzy calculations and fuzzy objectives, developed a cost function. At that point, created a learning algorithm from the cost capacity to align the four variables of each trapezoidal fuzzy weight. In conclusion, demonstrate our methodology using numerical models.

This paper discusses fuzzy generalized delta rules with distinct back propagation techniques on trapezoidal fuzzy sets. The suggested architecture offers a more compelling and distinctive feature in that every node can process fuzzy sets or verbal terms, maintaining the simplicity of the back propagation algorithm. As a result, the resulting architecture can deal with issues in which input parameters and desired outcomes are expressed verbally. Furthermore, this methodology functions at a higher level of abstraction of data by operating at the verbal level rather than the numerical level. Our goal is to realize monitored learning in this type of supervised neural network, and the effort is based on a standard back propagation algorithm.

Keywords: Neural networks, learning algorithm, back-propagation algorithm, fuzzy sets.

استعمال الشبكة العصبية الضبابية ذات التغذية الأمامية للمعالجة اللغوية

عبد الرحيم خلف راهي[1] وسوزان صابر حيدر[2]

[1] كلية دجلة الجامعة

[2] قسم الاحصاء والمعلوماتية – كلية الادارة والاقتصاد - جامعة السليمانية

الخلاصة

لقد تم استخدام المجموعات الضبابية بنجاح من أجل التعامل مع البيانات غير الدقيقة أو المصطلحات اللغوية أو المفاهيم غير الواضحة. في الآونة الأخيرة، تم بذل جهد كبير في اتجاه الجمع بين الشبكة العصبية الاصطناعية والمجموعات الضبابية. في هذا البحث، أقترح بنية الشبكة العصبية الضبابية ذات التغذية الأمامية مع مجموعة ضبابية شبه منحرف، يمكن للشبكة العصبية المقترحة أن تتعامل مع متجهات الإدخال الضبابية بالإضافة الى متجه المدخلات الحقيقية. في كلتا الحالتين، تكون النواتج من الشبكة العصبية الضبابية هي مخرجات غامضة. يتم تحديد علاقة المدخلات والمخرجات لكل وحدة من الشبكة العصبية الضبابية من خلال مبدأ تمديد Zadeh. بعد ذلك نحدد دالة التكلفة لمجموعات المستويات (h -cut) للمخرجات الغامضة والأهداف الغامضة ثم نشتق خوارزمية تعلم من دالة التكلفة لتعديل أربع معلمات لكل وزن ضبابي شبه منحرف. أخيرًا نوضح نهجنا بأمثلة عددية. تم النظر في مجموعات ضبابية شبه منحرفة، وتمت مناقشة قاعدة دلتا المعممة الضبابية مع خوارزميات الانتشار العكسي المختلفة. الخاصية الأكثر إثارة للاهتمام والمميزة للبنية المقترحة هي قدرة كل عقدة على معالجة مجموعات ضبابية أو مصطلحات لغوية، مع الحفاظ على بساطة خوارزمية الانتشار العكسي. وبالتالي، فإن البنية الناتجة قادرة على التعامل مع المشكلات التي يتم فيها وصف معلمات الإدخال والأهداف المرغوبة بمصطلحات لغوية. تتميز هذه المنهجية بميزة أخرى مثيرة للاهتمام تتمثل في قدرتها على العمل على المستوى اللغوي بدلاً من المستوى العددي، أي أنها يمكن أن تعمل على مستوى أعلى من تجريد البيانات. هدفنا هو تحقيق عملية التعلم تحت الإشراف (supervised) في هذا النوع من الشبكات العصبية معتمدا على خوارزمية الانتشار العكسي التقليدية.

الكلمات المفتاحية: الشبكات العصبية، خوارزمية التعلم، خوارزمية الانتشار العكسي، المجموعات الضبابية.

Introduction

The definition of a FNN is almost the same as that of a back-propagation (BP) network. The distinction lies in using fuzzy figures instead of actual numbers and broadened operations rather than traditional operations. The objective of this connection is to blend the advantages of both areas to process inaccurate or hazy data [4].

Replacing FNNs with crisp networks has the following benefits:

• Data with uncertainty can be handled with fuzziness. As a result, it is possible to approximate a wider range of functions.

• Fuzzy input data can be considered as an extension of crisp input data, so a few training data can cover the entire input range. There are several points that can be taken into consideration in order to reduce the risk of overtraining that is always present when using the conventional back propagation learning law.

• Taking into account the input range it allows us to gain insight into how the network reacts to unfamiliar inputs.

• It can be viewed as a form of "relaxation" of the error surface due to the vagueness of the weights. Back-propagation learning laws may become trapped in local minimal when this impact is utilized.

There are, however, some disadvantages to using fuzzy neural systems. A proportional back-propagation net has a much lower computational cost than a FNN (storing a trapezoidal fuzzy number as of now requires four genuine values).

A general Architecture of FNNS

The primary goal of this study is to devise a standard structure for FNNs along with a suitable learning method where every component can manage fuzzy sets and verbal expressions. Simultaneously, the back-propagation algorithm's straight forwardness, efficacy, and features were safeguarded [2]. The basic architecture used is the feed-forward neural network, with layers ($l = 1,2,3,\ldots,L$) as:

$l = 1$: input layer with n_I nodes

$l = 2,3,\ldots,L-1$: hidden layers with n_H nodes

$l = L$: output layer with n_O nodes

An input linguistic term or fuzzy set will be given to the network (input n_I), which will produce an n_O output fuzzy set. An illustration of how each node processes and transforms fuzzy sets is shown in figure (2) [6],[8],[14].

Augmentation of standard BP networks suggested by D.E. Rumelhart *et al.* (1986)[12] encompassed fuzzified input vectors, objective vectors, connection weights, and offsets (e.g., broadened to fuzzy figures). For the fuzzy data, incorporated trapezoidal fuzzy figures. To arrive at a definite learning principle, circumscribe fuzzy weights and fuzzy inputs in trapezoidal fuzzy figures while allowing fuzzy inputs and objectives to be represented in any fuzzy figure [12].

Architecture of Trapezoidal FNNs

This section proposes a FNN with trapezoidal fuzzy figures for its weights, inputs, and outputs. This neural network can manage fuzzy and accurate inputs with fuzzy numbers as outcomes. Subsequently, a learning technique is derived from a cost function defined for fuzzy outputs and goals' level sets (i.e., $h-cuts$). Finally, the proficiency of the proposed FNN for executing fuzzy if-then rules was tested through a numerical example. The proposed architecture further utilized normalized trapezoidal fuzzy sets (NTFS) as its fundamental element is presented in figure (1). Moreover, this option has been settled because of the ease of explaining an NTFS and its competence from a computational perspective. As demonstrated in figure (1), the membership function of a trapezoidal fuzzy number can be outlined by its four components as follows: Therefore, a trapezoid fuzzy number can be identified by its four components as follows:

$$A = (a, b, \alpha, \beta)$$

where:

$$\mu_A(x) = \begin{cases} 0 & , if\ x \le a - \alpha \quad or \quad x \ge b + \beta \\ \frac{x-a+\alpha}{\alpha} & , if\ a - \alpha < x \le a \\ 1 & , if\ a \le x \le b \\ \frac{-x+b+\beta}{\beta} & , if\ b < x \le b + \beta \end{cases} \quad \text{...(1)}$$

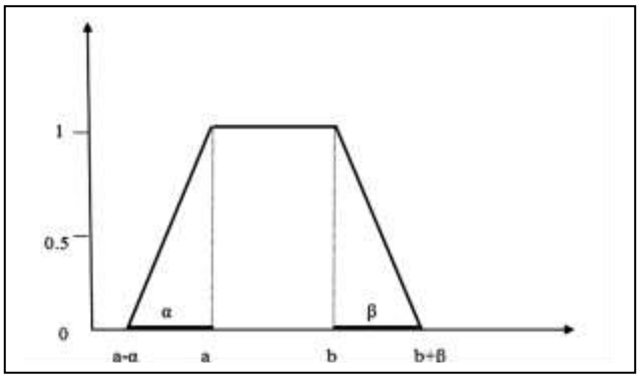

Figure 1: Membership Function of a Trapezoidal Fuzzy Number [2]

Here are a few specific cases that can be derived easily [2],[5]:

$a \ne b\ , \alpha = \beta$	Set of fuzzy symmetric trapezoids,
$a = b\ , \alpha \ne 0\ , \beta \ne 0$	Set of triangular fuzzy points,
$a = b\ , \alpha = \beta = 0$	Sets of real values (crisp),
$a \ne b\ , \alpha = \beta = 0$	Sets that are crisp (based on intervals).

Frequently, it will be written simply $A = (A^a, A^b, A^\alpha, A^\beta)$ instead of $A = (a, b, \alpha, \beta)$.

As far as NTFS is concerned, manipulating fuzzy sets is a straight forward process. A positive fuzzy set can be summarized as follows:

$$(A^a, A^b, A^\alpha, A^\beta) + (B^a, B^b, B^\alpha, B^\beta) = (A^a + B^a, A^b + B^b, A^\alpha + B^\alpha, A^\beta + B^\beta) \text{ ....(2)}$$

and the product of a fuzzy set by a positive constant is given by:

$$\boldsymbol{k} * (\boldsymbol{A^a}, \boldsymbol{A^b}, \boldsymbol{A^\alpha}, \boldsymbol{A^\beta}) = (\boldsymbol{k} * \boldsymbol{A^a}, \boldsymbol{k} * \boldsymbol{A^b}, \boldsymbol{k} * \boldsymbol{A^\alpha}, \boldsymbol{k} * \boldsymbol{A^\beta}) \quad \textbf{(3)}$$

A normalized fuzzy set can be approximated as the product of two positive NTFS:

$$(\boldsymbol{A^a}, \boldsymbol{A^b}, \boldsymbol{A^\alpha}, \boldsymbol{A^\beta}) * (\boldsymbol{B^a}, \boldsymbol{B^b}, \boldsymbol{B^\alpha}, \boldsymbol{B^\beta}) \approx (\boldsymbol{A^a B^a}, \boldsymbol{A^b B^b}, \boldsymbol{A^a B^\alpha} + \boldsymbol{A^\alpha B^a}, \boldsymbol{A^b B^\beta} + \boldsymbol{A^\beta B^b}) \quad \textbf{(4)}$$

Fuzzy sets with negative values increase computation time [1],[3],[9]. Each NTFS component represents one of four weights that characterize links between nodes,

$$\boldsymbol{W_{ij}} = (\boldsymbol{W_{ij}^a},\ \boldsymbol{W_{ij}^b},\ \boldsymbol{W_{ij}^\alpha},\ \boldsymbol{W_{ij}^\beta}) \quad \textbf{(5)}$$

A fuzzy network comprises nodes that process and transform fuzzy trapezoidal sets. NTFS are received at each node by weights composed of four terms, with the connectivity defined by connections between nodes. There are four terms that correspond to different components of the NTFS. As shown in figure (2), a single node has the following characteristics. An example of how this capability may be used is to produce fuzzy output based on linguistic terms or fuzzy sets received by the network [2],[5].

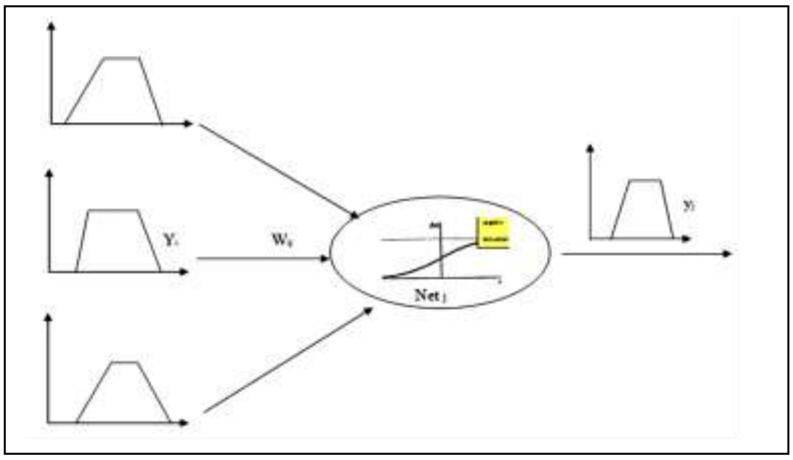

Figure 2: Fuzzy Neural Node [2]

As the outputs of connected nodes are added together, a generic node's state is determined

$$\boldsymbol{O_{i,l-1}} = (\boldsymbol{O_{i,l-1}^a},\ \boldsymbol{O_{i,l-1}^b},\ \boldsymbol{O_{i,l-1}^\alpha},\ \boldsymbol{O_{i,l-1}^\beta}) \quad \textbf{(6)}$$

In addition, the weights associated with such connections are important

$$W_{ij,l} = (W^{a}_{ij,l},\ W^{b}_{ij,l},\ W^{\alpha}_{ij,l},\ W^{\beta}_{ij}) \text{ ......(7)}$$

That is

$$net_{j,\,l} = \sum_{i=1}^{\#(l-1)} W_{ij,l} \cdot y_{i,l-1}$$

$$= \{ \sum_{i=1}^{\#(l-1)} W^{a}_{ij,l} \cdot y^{a}_{i,l-1} ,\ \sum_{i=1}^{\#(l-1)} W^{b}_{ij,l} \cdot y^{b}_{i,l-1} ,\ \sum_{i=1}^{\#(l-1)} W^{\alpha}_{ij,l} \cdot y^{\alpha}_{i,l-1} ,\ \sum_{i=1}^{\#(l-1)} W^{\beta}_{ij,l} \cdot y^{\beta}_{i,l-1}\}$$

$$= (net^{a}_{j,\,l},\ net^{b}_{j,\,l},\ net^{\alpha}_{j,\,l},\ net^{\beta}_{j,\,l}) \text{ ......(8)}$$

The quadruple,$(net^{a}_{j,\,l},\ net^{b}_{j,\,l},\ net^{\alpha}_{j,\,l},\ net^{\beta}_{j,\,l})$, Trapezoidal fuzzy sets must fulfill the following constraints to be meaningful:

$$net^{a}_{j,\,l} \le net^{b}_{j,\,l},\quad net^{\alpha}_{j,\,l} \ge 0,\quad net^{\beta}_{j,\,l} \ge 0 \text{ ......(9)}$$

As far as the permissible weight values are concerned, different constraints can be adopted during the initialization and updating phases.

In the following, six different possible strategies are got:

$$W^{a}_{ij,l} \le W^{b}_{ij,l},\quad W^{\alpha}_{ij,l} \ge 0,\quad W^{\beta}_{ij} \ge 0) \text{ ......(10)}$$

$$0 \le W^{a}_{ij,l} \le W^{b}_{ij,l},\ W^{\alpha}_{ij,l} \ge 0,\ W^{\beta}_{ij} \ge 0) \text{ ......(11)}$$

$$W^{a}_{ij,l} \ne W^{b}_{ij,l} \ne W^{\alpha}_{ij,l} = W^{\beta}_{ij} \text{ ......(12)}$$

$$W^{a}_{ij,l} = W^{b}_{ij,l} ,\quad W^{\alpha}_{ij,l} = W^{\beta}_{ij} \text{ ......(13)}$$

$$W^{a}_{ij,l} = W^{\alpha}_{ij,l} ,\quad W^{b}_{ij,l} = W^{\beta}_{ij} \text{ ......(14)}$$

$$W^{a}_{ij,l} = W^{b}_{ij,l} = W^{\alpha}_{ij,l} = W^{\beta}_{ij} \quad \text{.......(15)}$$

Determining the right weighting strategy for a neural network can be a challenge. One option is to use eq. (10), a simple way to satisfy eq. (9). All positive weights are provided by eq. (11) which can limit the capacity of the network. Alternatively, all four weights can be completely independent eq. (12), equal eq. (15), or somewhere between eq. (13) and eq. (14). With the last option eq. (12), explicit constraints are not imposed, allowing the network to learn eq. (9) with the correct datasets during the training phase. Neural networks can benefit from using this choice because it simplifies the process and optimizes performance [2], [7], [9], [12].

To standardize the process, sigmoidal activation functions used to create normalized fuzzy sets from fuzzy trapezoidal sets. In addition to producing a homogeneous environment, this nonlinear transformation has other advantages. The function adjusts the fuzzy set values, allowing for more precise output. A sigmoidal activation function is a powerful tool that can modify fuzzy set values to produce a more accurate data representation. This type of transformation is advantageous in applications where precise output is required. The output of the fuzzy set can be precisely calibrated and controlled by utilizing the sigmoidal activation function. Overall, using the sigmoidal activation function on fuzzy trapezoidal sets effectively produces normalized fuzzy sets. This nonlinear transformation is advantageous because it produces a homogeneous environment, allowing for a more precise:

$$O = f(net) = f(net^{a},\ net^{b},\ net^{\alpha},\ net^{\beta}) \quad \text{.......(16)}$$

As depicted in figure (3), a trapezoidal fuzzy set will approximate this value through linear interpolation $O = (O^{a},\ O^{b},\ O^{\alpha},\ O^{\beta})$. The membership function of whose members is as follows:

$$\mu(O) = \begin{cases} 0 & ,\ if\ O < f(n^{a} - n^{\alpha}) \quad or \quad O \geq f(n^{b} - n^{\beta}) \\ \frac{O - f(n^{a} - n^{\alpha})}{f(n^{a}) - f(n^{a} - n^{\alpha})} & ,\ if\ f(n^{a} - n^{\alpha}) \leq O \leq f(n^{a}) \\ 1 & ,\ if\ f(n^{a}) \leq O \leq f(n^{b}) \\ \frac{f(n^{b} - n^{\beta}) - O}{f(n^{b} - n^{\beta}) - f(n^{b})} & ,\ if\ f(n^{b}) \leq O \leq f(n^{b} - n^{\beta}) \end{cases} \quad \text{.......(17)}$$

that is

$$O_{j,l} = f\left(net^{a}_{j,l},\ net^{b}_{j,l},\ net^{\alpha}_{j,l},\ net^{\beta}_{j,l}\right)$$
$$\cong \{f(net^{a}_{j,l}),\ f(net^{b}_{j,l}),\ f(net^{a}_{j,l}) - f(net^{a}_{j,l} - net^{\alpha}_{j,l}), f(net^{b}_{j,l} + net^{\beta}_{j,l}) - f(net^{b}_{j,l})\}$$
$$= \{O^{a}_{j,l},\ O^{b}_{j,l},\ O^{\alpha}_{j,l},\ O^{\beta}_{j,l}\} \quad \text{.........(18)}$$

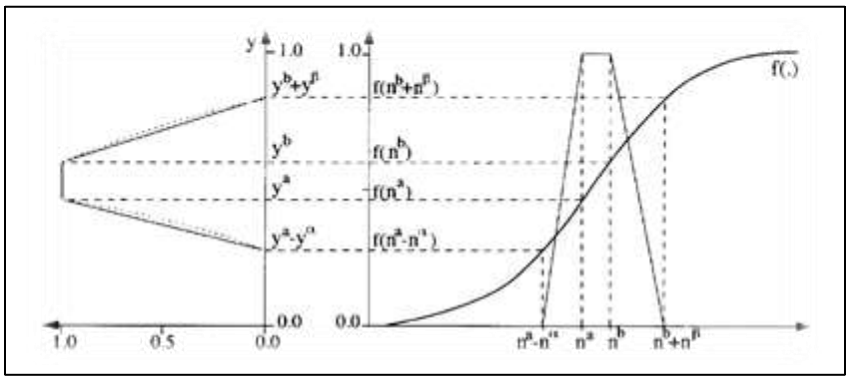

Figure 3: The transformation produced by the NTFS activation function [2].

When calculating h-level sets of fuzzy outputs from fuzzy inputs, fuzzy weights, and fuzzy biases, it is essential to restrict the fuzzy weights and fuzzy biases to trapezoidal fuzzy numbers. This enables the derivation of the crisp learning rule in the following section. However, any fuzzy number can be used for both fuzzy and fuzzy inputs. The input-output relation of each unit for h-level sets can be derived using this method. Thus, crisp learning rules can be generated, which is highly beneficial for a variety of applications:

Input units:

$$[O_{pi}]_h = [[O_{pi}]^L_h\ , [O_{pi}]^U_h] = [[X_{pi}]^L_h\ , [X_{pi}]^U_h] \quad \text{.........(19)}$$

where

$$[[X_{pi}]^L_h\ , [X_{pi}]^U_h] = [X^{a}_{pi} - (1-h)X^{\alpha}_{pi}\ ,\ X^{b}_{pi} + (1-h)X^{\beta}_{pi}] \quad \text{.........(20)}$$

Hidden units:

$$[O_{pj}]_h = [[O_{pj}]^L_h\ , [O_{pj}]^U_h = [f([Net_{pj}]^L_h)\ , f([Net_{pj}]^U_h)] \quad \text{.........(21)}$$

$$[\boldsymbol{Net_{pj}}]_h^L = \sum_{\substack{i=1\\ [W_{ij}]_h^L \ge 0}}^{n_I} [\boldsymbol{W_{ij}}]_h^L \cdot [\boldsymbol{O_{pi}}]_h^L + \sum_{\substack{i=1\\ [W_{ij}]_h^L < 0}}^{n_I} [\boldsymbol{W_{ij}}]_h^L \cdot [\boldsymbol{O_{pi}}]_h^U + [\boldsymbol{\vartheta_j}]_h^L \text{ ....(22)}$$

$$[\boldsymbol{Net_{pj}}]_h^U = \sum_{\substack{i=1\\ [W_{ij}]_h^U \ge 0}}^{n_I} [\boldsymbol{W_{ij}}]_h^U \cdot [\boldsymbol{O_{pi}}]_h^U + \sum_{\substack{i=1\\ [W_{ij}]_h^U < 0}}^{n_I} [\boldsymbol{W_{ij}}]_h^U \cdot [\boldsymbol{O_{pi}}]_h^L + [\boldsymbol{\vartheta_j}]_h^U \text{ ....(23)}$$

Such that:

$$[[\boldsymbol{W_{ij}}]_h^L, [\boldsymbol{W_{ij}}]_h^U] = [\boldsymbol{W_{ij}^a} - (\boldsymbol{1}-\boldsymbol{h})\boldsymbol{W_{ij}^\alpha},\ \boldsymbol{W_{ij}^b} + (\boldsymbol{1}-\boldsymbol{h})\boldsymbol{W_{ij}^\beta}] \text{ ....(24)}$$

Output units:

$$[\boldsymbol{O_{pk}}]_h = [[\boldsymbol{O_{pk}}]_h^L, [\boldsymbol{O_{pk}}]_h^U] = [\boldsymbol{f}([\boldsymbol{Net_{pk}}]_h^L), \boldsymbol{f}([\boldsymbol{Net_{pk}}]_h^U)] \text{ ....(25)}$$

$$[\boldsymbol{Net_{pk}}]_h^L = \sum_{\substack{j=1\\ [W_{jk}]_h^L \ge 0}}^{n_H} [\boldsymbol{W_{jk}}]_h^L \cdot [\boldsymbol{O_{pj}}]_h^L + \sum_{\substack{j=1\\ [W_{jk}]_h^L < 0}}^{n_H} [\boldsymbol{W_{jk}}]_h^L \cdot [\boldsymbol{O_{pj}}]_h^U + [\boldsymbol{\vartheta_k}]_h^L \text{ ....(26)}$$

$$[\boldsymbol{Net_{pk}}]_h^U = \sum_{\substack{j=1\\ [W_{jk}]_h^U \ge 0}}^{n_H} [\boldsymbol{W_{jk}}]_h^U \cdot [\boldsymbol{O_{pj}}]_h^U + \sum_{\substack{j=1\\ [W_{jk}]_h^U < 0}}^{n_H} [\boldsymbol{W_{jk}}]_h^U \cdot [\boldsymbol{O_{pj}}]_h^L + [\boldsymbol{\vartheta_k}]_h^U \text{ ....(27)}$$

A FNN's learning process

The learning process of trapezoidal FNN is also based on a suitable error function, which is minimized with respect to the weights and bias. Let us define a cost function to be minimized in learning of the FNN using the trapezoidal fuzzy output and the corresponding trapezoidal target output.

Cost Function

A major challenge is to choose an error function that minimizes the difference between the desired output and the actual output for every m output nodes of the L layer during the learning phase $t_k = \{t_{k,l}^a,\ t_{k,l}^b,\ t_{k,l}^\alpha,\ t_{k,l}^\beta\}$ and the output of the neural network $O_k =$

$\{O^{a}_{k,l},\ O^{b}_{k,l}, O^{\alpha}_{k,l}, O^{\beta}_{k,l}\}$ as a result of each example in the learning set. Through fuzzy arithmetic, it is possible to fuzzy the usual error function:

$$\boldsymbol{e_{ph} = \sum_{k=1}^{n_o} e_{pkh}} \quad \text{(28)}$$

where

$$\boldsymbol{e_{pkh} = e^{L}_{pkh} + e^{U}_{pkh}} \quad \text{(29)}$$

where e^{L}_{pkh} and e^{U}_{pkh} are the squared errors for the lower limits and the upper limits of the h-level sets, respectively:

$$\boldsymbol{e^{L}_{pkh} = h.\frac{([t_{pk}]^{L}_{h} - [O_{pk}]^{L}_{h})^2}{2}} \quad \text{(30)}$$

$$\boldsymbol{e^{U}_{pkh} = h.\frac{([t_{pk}]^{U} - [O_{pk}]^{U}_{h})^2}{2}} \quad \text{(31)}$$

In addition, it can be efficient to contemplate the differences between the 4 components of a trapezoidal fuzzy set as an alternative and efficient strategy (figure 4):

$$\boldsymbol{[t^{a}_{k}]_{h} - [O^{a}_{k,\,L}]_{h}\,, [t^{b}_{k}]_{h} - [O^{b}_{k,\,L}]_{h}\,, [t^{\alpha}_{k}]_{h} - [O^{\alpha}_{k,\,L}]_{h}\,, [t^{\beta}_{k}]_{h} - [O^{\beta}_{k,\,L}]_{h}} \quad \text{(32)}$$

Such components should be minimized independently by the cost function, *i.e.*

$$\boldsymbol{E_{h} = \{E^{a}_{h},\ E^{b}_{h},\ E^{\alpha}_{h},\ E^{\beta}_{h}\}} \quad \text{(33)}$$

$$\boldsymbol{= \{\sum_{k=1}^{m}([t^{a}_{k}]_{h} - [O^{a}_{k}]_{h})^2,\ \sum_{k=1}^{m}([t^{b}_{k}]_{h} - [O^{b}_{k}]_{h})^2,\ \sum_{k=1}^{m}([t^{\alpha}_{k}]_{h} - [O^{\alpha}_{k}]_{h})^2,\ \sum_{k=1}^{m}([t^{\beta}_{k}]_{h} - [O^{\beta}_{k}]_{h})^2\}} \quad \text{(34)}$$

As a result of this solution, delta-rule computations can be simplified. Since such error functions are critical in back-propagation algorithms, modifying only those components that differ from their target is the most appropriate strategy.

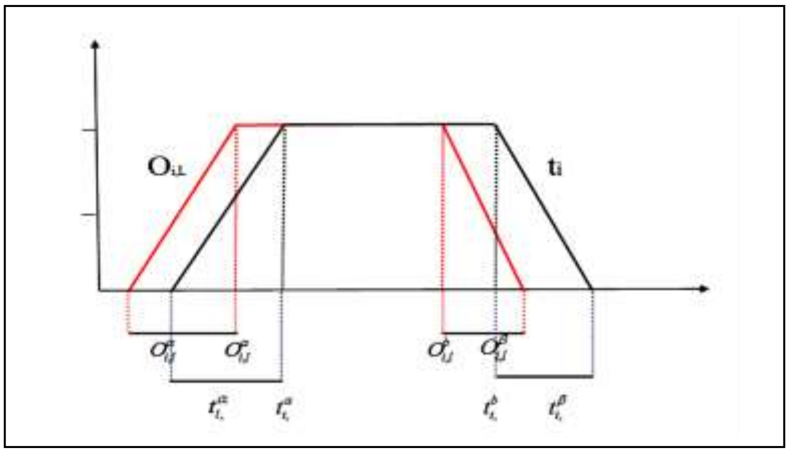

Figure 4: The Components of the Error Function

Learning Algorithm

The cost function is defined in the previous subsection. let us use this cost function e_{ph} to derive the learning algorithm. This algorithm specifies trapezoidal fuzzy weights and biases using their four parameters. First, consider the learning of the fuzzy weight $W_{jk,I} = (W^{a}_{jk,I}, W^{b}_{jk,I}, W^{\alpha}_{jk,I}, W^{\beta}_{jk,I})$ from the j^{th} hidden unit to the k^{th} output unit. The cost function provides the amount of adjustment required for each parameter. Using this information, an update rule was derived for each parameter e_{ph}

$$\boldsymbol{\Delta W^{a-\alpha}_{jk}(t+1) = -\eta \frac{\partial e_{ph}}{\partial W^{a-\alpha}_{jk}} + \alpha . \Delta w^{a-\alpha}_{jk}(t)} \quad \text{.......(35)}$$

$$\boldsymbol{\Delta W^{a}_{jk}(t+1) = -\eta \frac{\partial e_{ph}}{\partial W^{a}_{jk}} + \alpha . \Delta w^{a}_{jk}(t)} \quad \text{.......(36)}$$

$$\boldsymbol{\Delta W^{b}_{jk}(t+1) = -\eta \frac{\partial e_{ph}}{\partial W^{b}_{jk}} + \alpha . \Delta w^{b}_{jk}(t)} \quad \text{.......(37)}$$

$$\boldsymbol{\Delta W^{b+\beta}_{jk}(t+1) = -\eta \frac{\partial e_{ph}}{\partial W^{b+\beta}_{jk}} + \alpha . \Delta w^{b+\beta}_{jk}(t)} \quad \text{.......(38)}$$

where η is a learning constant, α is a momentum constant and t indexes the number of adjustments.

The explicit calculation of each derivative in equations (35) - (38) is shown as:

The h-level of W_{jk} can be calculated as:

$$[\boldsymbol{W_{jk}}]_{\boldsymbol{h}} = [[\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}}, [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{U}}] \quad \textbf{(A1)}$$

where:

$$[\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}} = (\boldsymbol{1} - \boldsymbol{h}) \cdot \boldsymbol{W_{jk}^{a-\alpha}} + \boldsymbol{h} \cdot \boldsymbol{W_{jk}^{a}} \quad \textbf{(A2)}$$

$$[\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{U}} = (\boldsymbol{1} - \boldsymbol{h}) \cdot \boldsymbol{W_{jk}^{b+\beta}} + \boldsymbol{h} \cdot \boldsymbol{W_{jk}^{b}} \quad \textbf{(A3)}$$

Therefore; obtained the following relations:

$$\frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}}}{\partial \boldsymbol{W_{jk}^{a-\alpha}}} = \frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{U}}}{\partial \boldsymbol{W_{jk}^{b+\beta}}} = (\boldsymbol{1} - \boldsymbol{h}) \quad \textbf{(A4)}$$

$$\frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{U}}}{\partial \boldsymbol{W_{jk}^{a-\alpha}}} = \frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{U}}}{\partial \boldsymbol{W_{jk}^{a}}} = \frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}}}{\partial \boldsymbol{W_{jk}^{b}}} = \frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}}}{\partial \boldsymbol{W_{jk}^{b+\beta}}} = \boldsymbol{0} \quad \textbf{(A5)}$$

$$\frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}}}{\partial \boldsymbol{W_{jk}^{a}}} = \frac{\partial [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{U}}}{\partial \boldsymbol{W_{jk}^{b}}} = \boldsymbol{h} \quad \textbf{(A6)}$$

The derivatives in each set of the FNN can be calculated using the input-output relation in equations (19)-(23) and the above relationships in (AI)-(A6), $\partial e_{ph}/\partial W_{jk}^{s}$ in the learning algorithm from the cost function e_{ph} as follows:

a). $$\frac{\partial e_{ph}}{\partial W_{jk}^{a-\alpha}} = \frac{\partial e_{ph}}{\partial [W_{jk}]_h^L} \cdot \frac{\partial [W_{jk}]_h^L}{\partial W_{jk}^{a-\alpha}} + \frac{\partial e_{ph}}{\partial [W_{jk}]_h^U} \cdot \frac{\partial [W_{jk}]_h^U}{\partial W_{jk}^{a-\alpha}}$$

$$= \partial e_{pjk}^{L}/\partial [W_{jk}]_h^L \, . (1 - h)$$

$$= \begin{cases} -\boldsymbol{h} \, . \, \boldsymbol{\delta_{pkh}^{L}} \, [\boldsymbol{O_{pj}}]_{\boldsymbol{h}}^{\boldsymbol{L}} \, . \, (\boldsymbol{1} - \boldsymbol{h}), \boldsymbol{if} \; [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}} \geq \boldsymbol{0} \\ -\boldsymbol{h} \, . \, \boldsymbol{\delta_{pkh}^{L}} \, [\boldsymbol{O_{pj}}]_{\boldsymbol{h}}^{\boldsymbol{U}} \, . \, (\boldsymbol{1} - \boldsymbol{h}), \boldsymbol{if} \; [\boldsymbol{W_{jk}}]_{\boldsymbol{h}}^{\boldsymbol{L}} < \boldsymbol{0} \end{cases} \quad \textbf{(39)}$$

where:

$$\delta^L_{pkh} = ([t_{pk}]^L_h - [O_{pk}]^L_h)[O_{pk}]^L_h(1 - [O_{pk}]^L_h)$$

b). $\frac{\partial e_{ph}}{\partial W^a_{jk}} = \begin{cases} -h \,.\, \delta^L_{pkh}\,[O_{pj}]^L_h \,.\,(h), if\ [W_{jk}]^L_h\ \geq 0 \\ -h \,.\, \delta^L_{pkh}\,[O_{pj}]^U_h \,.\,(h), if\ [W_{jk}]^L_h\ < 0 \end{cases}$ .... (40)

c). $\frac{\partial e_{ph}}{\partial W^b_{jk}} = \begin{cases} -h \,.\, \delta^U_{pkh}\,[O_{pj}]^U_h \,.\,(h), if\ [W_{jk}]^U_h\ \geq 0 \\ -h \,.\, \delta^U_{pkh}\,[O_{pj}]^L_h \,.\,(h), if\ [W_{jk}]^U_h\ < 0 \end{cases}$ ....(41)

where:

$$\delta^U_{pkh} = ([t_{pk}]^U_h - [O_{pk}]^U_h)[O_{pk}]^U_h(1 - [O_{pk}]^U_h)$$

d). $\frac{\partial e_{ph}}{\partial W^{b+\beta}_{jk}} = \begin{cases} -h \cdot \delta^U_{pkh}\,[O_{pj}]^U_h \cdot (1-h), & if[W_{jk}]^U_h \geq 0, \\ -h \cdot \delta^U_{pkh}\,[O_{pj}]^L_h \cdot (1-h), & if[W_{jk}]^U_h < 0, \end{cases}$ ....(42)

From above relations, obtained the following relations:

$\frac{\partial e_{ph}}{\partial W^{a-\alpha}_{jk}} = (1-h) \cdot \partial e^L_{pkh}/\partial[W_{jk}]^L_h$ ....(43)

$\frac{\partial e_{ph}}{\partial W^{a}_{jk}} = h \cdot \partial e^L_{pkh}/\partial[W_{jk}]^L_h$ .... (44)

$\frac{\partial e_{ph}}{\partial W^{b}_{jk}} = h \cdot \partial e^U_{pkh}/\partial[W_{jk}]^U_h$ ....(45)

$\frac{\partial e_{ph}}{\partial W^{b+\beta}_{jk}} = (1-h) \cdot \partial e^U_{pkh}/\partial[W_{jk}]^U_h$ ....(46)

Then the four parameters of the fuzzy weight W_{jk} are updated as:

$W^s_{jk}(t+1) = W^s_{jk}(t) + \Delta W^s_{jk}(t), \qquad s = a-\alpha,\ a,\ \ b,\ \ b+\beta$ ....(47)

The fuzzy weight W_{ij}, and the fuzzy biases $\theta_j$and θ_k are also adjusted in the same manner as the fuzzy weight W_{jk}.

In fact, for the computation of the delta rule:

$$[\Delta W_{ij,\,l}]_h^L(t) = -\eta \frac{\partial e_{ph}}{\partial W_{ij}^L} = ([\Delta W_{ij,\,l}^{a}]_h^L(t), [\Delta W_{ij,\,l}^{b}]_h^L(t), [\Delta W_{ij,\,l}^{\alpha}]_h^L(t), [\Delta W_{ij,\,l}^{\beta}]_h^L(t)) \text{ ....(48)}$$

$$[\Delta W_{ij,\,l}]_h^U(t) = -\eta \frac{\partial e_{ph}}{\partial W_{ij}^U} = ([\Delta W_{ij,\,l}^{a}]_h^U(t), [\Delta W_{ij,\,l}^{b}]_h^U(t), [\Delta W_{ij,\,l}^{\alpha}]_h^U(t), [\Delta W_{ij,\,l}^{\beta}]_h^U(t)) \text{ ....(49)}$$

The four components of $[\Delta W_{ij,\,l}]_h^L(t)$ and $[\Delta W_{ij,\,l}]_h^U(t)$ in the event that updates are made sequentially, they can be considered independent terms. Therefore, it is possible to derive a fuzzy generalized delta rule as follows:

$$[\Delta W_{ij,\,l}]_h^L(= (-\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{\alpha}]_h^L(t)}, -\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{b}]_h^L(t)}, -\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{\alpha}]_h^L(t)}, -\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{\beta}]_h^L(t)}) \text{ ....(50)}$$

$$[\Delta W_{ij,\,l}]_h^U(t) = (-\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{\alpha}]_h^U(t)}, -\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{b}]_h^U(t)}, -\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{\alpha}]_h^U(t)}, -\eta \frac{\partial e_{ph}}{\partial [W_{ij,\,l}^{\beta}]_h^U(t)}) \text{ ....(51)}$$

where $[\Delta W_{ij,\,l}]_h^L$ and $[\Delta W_{ij,\,l}]_h^U$ are obtained as:

$$[\Delta W_{ij,\,l}]_h^L = \eta \cdot [\delta_{j,l}]_h^L \cdot [O_{j,l-1}]_h^L \text{ ....(52)}$$

Each output unit (k=1, 2, .., n$_o$), compare its product $[O_{k,L}]_h$ with a correspondent desired
[
output (target value $[t_{k,L}]_h$) which was received from input training patterns, and compute its error information term by:

$$\begin{aligned}\delta_{k,L}]_h^L &= ([t_{k,L}]_h^L - [O_{k,L}]_h^L) \cdot f^{/}([net_{k,L}]_h^L) \\ &= ([t_{k,L}]_h^L - [O_{k,L}]_h^L) \cdot [O_{k,L}]_h^L \cdot (1 - [O_{k,L}]_h^L) \text{ ....(53)}\end{aligned}$$

and

$$\begin{aligned}[\delta_{k,L}]_h^U &= ([t_{k,L}]_h^U - [O_{k,L}]_h^U) \cdot f^{/}([net_{k,L}]_h^U) \\ &= ([t_{k,L}]_h^U - [O_{k,L}]_h^U) \cdot [O_{k,L}]_h^U \cdot (1 - [O_{k,L}]_h^U) \text{..(54)}\end{aligned}$$

got the following:

$$[\delta_{k,L}]_h = ([\delta_{k,L}]_h^L,\ [\delta_{k,L}]_h^U) \text{ ....(55)}$$

Also, each hidden unit (j) sums its delta inputs coming from units in the output layer for (l=1, 2, …, L-1) as:

$$[\boldsymbol{\delta}_{j,l}]_h^L = \{\sum_{\substack{t=1 \\ [W_{jt}]_h^L \geq 0}}^{n_H} [\boldsymbol{\delta}_{t,l+1}]_h^L \cdot [\boldsymbol{W}_{jt}]_h^L + \sum_{\substack{t=1 \\ [W_{jk}]_h^U < 0}}^{n_H} [\boldsymbol{\delta}_{t,l+1}]_h^U \cdot [\boldsymbol{W}_{jt}]_h^U\} \cdot [\boldsymbol{O}_{j,l}]_h^L \cdot (\mathbf{1} - [\boldsymbol{O}_{j,l}]_h^L) \text{....(56)}$$

and

$$[\boldsymbol{\delta}_{j,l}]_h^U = \{\sum_{\substack{t=1 \\ [W_{jt}]_h^L \geq 0}}^{n_H} [\boldsymbol{\delta}_{t,l+1}]_h^U \cdot [\boldsymbol{W}_{jt}]_h^U + \sum_{\substack{t=1 \\ [W_{jk}]_h^U < 0}}^{n_H} [\boldsymbol{\delta}_{t,l+1}]_h^L \cdot [\boldsymbol{W}_{jt}]_h^L\} \cdot [\boldsymbol{O}_{j,l}]_h^U \cdot (\mathbf{1} - [\boldsymbol{O}_{j,l}]_h^U) \text{...(57)}$$

Starting with a random weighting and with the constraint eq. (15), weights have been updated independently in the subsequent steps [13][14].

Numerical Example

By using FNN, let us approximate the implementation of fuzzy if-then rules in this example. In this mapping, we assume both the input and output spaces are unit intervals [0, 1]. A fuzzy output-input pair (X_p, T_p) can be depicted in the space of input-outputs, and the h-level sets with h = 0.2, 0.4, 0.6, 0.8 of $X_p \times T_p$, and $\eta = 0.5$.

The following three pairs of fuzzy input-output variables will be used as training data. Using the proposed learning algorithm, one output unit, two input units, and four hidden input units were used to train a FNN.

Table 1: Training Data Sets

x_1	x_2	target
{0.85, 0.95, 0.05, 0.05}	{0.45, 0.55, 0.05, 0.05}	{0.85, 0.90, 0.05, 0.05}
{0.05, 0.15, 0.05, 0.05}	{0.05, 0.15, 0.05, 0.05}	{0, 0, 0, 0}
{0.45, 0.55, 0.05, 0.05}	{0.05, 0.15, 0.05, 0.05}	{0.08, 0.12, 0.10, 0.10}

Based on a uniform interval [-0.6, +0.6], the following matrix contains the initial weights Following is a simplified breakdown of the general weight matrix into three matrices:

Initial Weights from first Input unit to hidden Layer $|W_{1j}| = \begin{array}{c} W_{11} \\ W_{12} \\ W_{13} \\ W_{14} \end{array} = \begin{array}{c} \{0.145,0.50,0.05,0.05\} \\ \{0.50,0.55,0.05,0.05\} \\ \{0.40,0.50,0.05,0.05\} \\ \{0.20,0.25,0.05,0.05\} \end{array}$

Initial Weights from second Input unit to hidden Layer $|W_{2j}| = \begin{array}{c} W_{21} \\ W_{22} \\ W_{23} \\ W_{24} \end{array} = \begin{array}{c} \{0.50,0.55,0.05,0.05\} \\ \{0.45,0.50,0.05,0.05\} \\ \{0.50,0.60,0.05,0.05\} \\ \{0.30,0.35,0.05,0.05\} \end{array}$

Initial Weights from hidden Layer to output Layer $|W_{i1}| = \begin{array}{c} W_{11} \\ W_{21} \\ W_{31} \\ W_{41} \end{array} = \begin{array}{c} \{0.20,0.25,0.05,0.05\} \\ \{0.35,0.40,0.05,0.05\} \\ \{0.50,0.60,0.05,0.05\} \\ \{0.20,0.30,0.05,0.05\} \end{array}$

We experimented to explore the effects of FFFNN, focusing on a network with one hidden layer and four nodes. Determined the minimum mean square error (MSE) during the training and testing the training and validation datasets. After training the selected network for 700 epochs, applied the proposed method and obtained an adjusted weight set, and the results of the experiments are discussed below:

Final Weights from first Input unit to hidden Layer $|W_{1j}| = \begin{array}{c} W_{11} \\ W_{12} \\ W_{13} \\ W_{14} \end{array} = \begin{bmatrix} W_{11}^{L} & W_{11}^{U} \\ W_{12}^{L} & W_{12}^{U} \\ W_{13}^{L} & W_{13}^{U} \\ W_{14}^{L} & W_{14}^{U} \end{bmatrix}$

$$= \begin{bmatrix} 0.169885899018162 & 0.601283205655338 \\ 0.535284832258353 & 0.676849246277717 \\ 0.447023992640629 & 0.660410319402502 \\ 0.224906638518036 & 0.365249342582516 \end{bmatrix}$$

Final Weights from second Input unit to hidden Layer $|W_{2j}| = \begin{array}{c} W_{21} \\ W_{22} \\ W_{23} \\ W_{24} \end{array} = \begin{bmatrix} W_{21}^{L} & W_{21}^{U} \\ W_{22}^{L} & W_{22}^{U} \\ W_{23}^{L} & W_{23}^{U} \\ W_{24}^{L} & W_{24}^{U} \end{bmatrix}$

$$= \begin{bmatrix} 0.513174887715498 & 0.6895292344931 \\ 0.468680205313246 & 0.577518983836384 \\ 0.524895054927393 & 0.698028528523752 \\ 0.313185867450725 & 0.420430153800426 \end{bmatrix}$$

From hidden layer to output layer, final weights $|W_{i1}| = \begin{matrix} W_{11} \\ W_{21} \\ W_{31} \\ W_{41} \end{matrix} = \begin{bmatrix} W_{11}^{L} & W_{11}^{U} \\ W_{21}^{L} & W_{21}^{U} \\ W_{31}^{L} & W_{31}^{U} \\ W_{41}^{L} & W_{41}^{U} \end{bmatrix}$

$$= \begin{bmatrix} 0.428214120839701 & 0.8939696346077 \\ 0.604092466391081 & 1.050576700048008 \\ 0.749136635633001 & 1.256616092942895 \\ 0.424145346094889 & 0.8752893843754 \end{bmatrix}$$

Optimizing neural network performance is a crucial factor in successful model development. In this study, the minimum value of the ((MSE) achieved was (9.9655511e-006) for the training set. Increasing the number of hidden nodes in the layers helps reduce the number of iterations necessary to reach an optimal weight and develop a good model. In this case, 700 iterations are sufficient to achieve the desired results. This approach can be applied to any neural network problem that deals with uncertain or vague information.

Conclusions

This study delves into the architecture of FFFNN, specifically focusing on trapezoidal fuzzy weights for h-level sets. As a result of its ability to process fuzzy sets and linguistic terms while utilizing the simplicity of the back propagation algorithm, the proposed solution is unique and valuable. By examining this architecture, researchers can better understand how to utilize FFFNNs effectively. Fuzzy logic has become a popular problem-solving tool because it works with linguistic terms rather than just numerical values. An approach such as this is helpful when the input parameters and desired targets are expressed in words. Through the use of NTFS, the network can provide a higher level of flexibility and accuracy by allowing higher data abstraction levels to be achieved. As a result of its ability to handle linguistic terms, fuzzy logic has become an effective solution to many problems. BP's convergence properties are maintained using a particular error function to simplify and make the learning algorithm more efficient. FNNs demonstrate good generalization properties, and a recent example has provided evidence that FFFNN have the potential to deliver satisfactory results in learning sets and three distinct test sets. Furthermore, FFFNN has been demonstrated to possess good generalization properties. This further confirms the potential of the newly proposed FNN architecture, in which each node can process fuzzy sets or linguistic terms.

References

1. George J. klir and Bo Yuan, Fuzzy sets and fuzzy logic, theory and applications, (1995).
2. G. Bortolan, An architecture of fuzzy neural networks for linguistic processing, Fuzzy Sets and Systems 100, pp. 197-215 (1998).
3. Hisao Ishibuchi, Hideo Tanaka, Fuzzy Neural Networks with Fuzzy Weights and Fuzzy Biases, IEEE, (1993).
4. Isa K., Mohamad S., and Tukiran Z., “Development of INPLANS: An Analysis on Students’ Performance using Neuro-Fuzzy, 2008.
5. J.J. Buckley and Y. Hayashi, Fuzzy neural networks: a survey, Fuzzy Sets and Systems (1991).
6. J.G. TAYLOR, Mathematical Approaches to Neural Networks, (1993).
7. Priti Srinivas Sajja, Type-2 Fuzzy User Interface for Artificial Neural Network based Decision Support System for Course Selection, International Journal of Computing and ICT Research (2008).
8. Ra´ul Rojas, Neural Networks, A Systematic Introduction, Springer, (1996).
9. Reddy.V.R,Reddy.V.V,& Jaganmohan.V.C, Fuzzy Logic Controller (FLC) Implementation for Space Vector Pulse Width Modulated Induction Motor Drive. International Journal, 6(2), pp. 916-926, (2018). \
10. Robert Full´er, Neural Fuzzy Systems, (1995).
11. Towfik, Z. S., & Jawad, S. F. Proposed method for optimizing fuzzy linear programming problems by using Two-phase technique. In 2010 1st International Conference on Energy, Power and Control (EPC-IQ) (pp. 89-96). IEEE.
12. Uthra, G., & Sattanathan, R, Confidence Analysis for Fuzzy Multi Criteria Ecision Making Using Trapezoidal Fuzzy Numbers, International Journal of Information Technology, 2(2), pp. 333-336. (2009).
13. Vasant, P., & Barsoum, N. N., Fuzzy optimization of unit’s products in mix-product selection problem using fuzzy linear programming approach. Soft Computing, 10(2), pp.144-151 (2006).
14. Wang, K., Computational intelligence in agile manufacturing engineering. Agile manufacturing: the 21st century competitive strategy (2001).