Study of Evolutionary Connectionism from the Perspective of Fuzzy Neural Network and Neuro-Fuzzy Inference Model

Rajesh S Prasad
Associate Professor
VIIT, Pune
India

ABSTRACT
Connectionism is a movement in cognitive science which hopes to explain human intellectual abilities using neural networks (also known as 'neural nets') [1]. Evolutionary Connectionist System is an adaptive, incremental learning and knowledge representation system that evolves its structure and functionality, where in the core of the system is a connectionist architecture that consists of neurons (information processing units) and connections between them. Neural networks are simplified models of the brain composed of large numbers of units (the analogs of neurons) together with weights that measure the strength of connections between the units. These weights model the effects of the synapses that link one neuron to another. Experiments on models of this kind have demonstrated an ability to understand information, adapt knowledge and evolve intelligence [2].

Fuzzy neural networks are connectionist architectures that are trained as neural networks, but their structure can be interpreted as a et of fuzzy rules. In contrast to them, neuro-fuzzy inference systems consist of a set of rules and an inference method that are embodied or combined with a connectionist structure for a better adaption[4].

This paper aims to explore the fuzzy neural approach and neuro-fuzzy inference system to amalgamate evolutionary connectionism and constitutes a challenge to classicism which has been a matter of hot debate in recent years.

Keywords
Evolving Connectionism; Connectionist systems; Evolving systems; Fuzzy Neural Networks (FNN); Neuro-Fuzzy system.

1. INTRODUCTION
1.1 Connectionist Method for Supervised learning
Connectionist systems for supervised learning learn from pairs of data(x, y), where the desired output vector **y** is known for an input vector **x**. If the model is incrementally adaptive, new data will be used to adapt the system's structure and function incrementally
If a system is trained incrementally, the generalization error of the system on the next new input vector from the input stream is called here local incrementally adaptive generation error. The local incrementally adaptive generalization error at the moment t, for example, when the input vector is x(t), and the output vector calculated by the system is y(t)', is expressed as $Err(t) = \|y(t)-y(t)'\|$.

dimensional error index (LNDEI) can be calculated at each time moment t as:

$$LRMSE(t) = \sqrt{(\sum i = 1,2,\ldots,t(Err(i)^2/t))} \quad \text{(i)}$$

$$LNDEI(t) = LRMSE(t)/std(y(1):y(t)) \quad \text{(ii)}$$

where std(y(1):y(*t*)) is the standard deviation of the output data points from time unit 1 to time unit t.

In a general case, the global generalization root mean square error (RMSE) and the non dimensional error index (NDEI) are evaluated on a set of p new (future) test examples from the problem space as follows:

$$RMSE = \sqrt{(\sum i = 1,2,\ldots,p[(y_i - y_i')^2] / p)} \quad \text{(iii)}$$

$$NDEI(t) = RMSE / std(y_1:y_p) \quad \text{(iv)}$$

where std(y_1:y_p), is the standard deviation of the data from 1 to *p* in the test set.

After a system is evolved on a sufficiently large and representative part of the whole problem space Z, its global generalization error is expected to become satisfactorily small.

Multilayer perceptrons (MLP) trained with a backpropogation algorithm use a global optimization function in incrementally adaptive pattern. In this method, pattern learning mode of the backpropogation algorithm, after each training example is presented to the system and propogated through it, an error is calculated and then all connections are modified in a backward manner. MLP can be trained in an incrementally adaptive mode, but they have limitations in this respect as they have fixed structure and the weight optimization is a global one if a gradient descent algorithm is used for this purpose.

Some connectionist systems that include MLP use a local objective (goal) function to optimize the structure during the learning process. In this case when a data pair (***x***, ***y***) is presented, the system optimizes its functioning always in a local vicinity of ***x*** from the input space **X**, and in the local vicinity of ***y*** from the output space **Y**.

1.2 Simple Evolving MLP
A simple evolving MLP (called as eMLP), is presented in figure 1as a simplified graphical representation. An eMLP consists of three layers of neurons, the input layer, with linear or other transfer functions, an evolving layer, and an output layer with a simple saturated linear activation function. It is a simplified version of the evolving fuzzy neural network (described in later section of the paper).

The evolving layer is the layer that will grow and adapt itself to the incoming data, and is the layer with which the learning algorithm is most concerned.

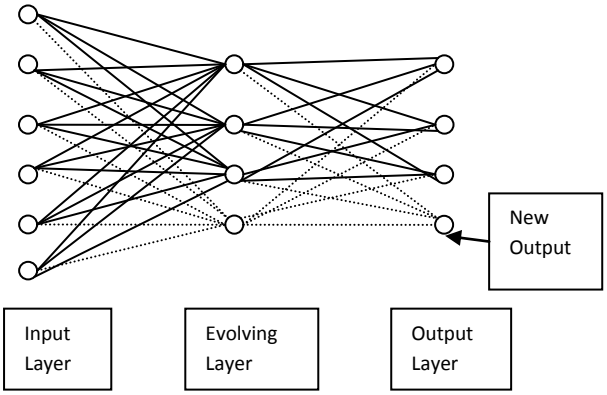

Figure 1. A simple evolving eMLP

If a linear activation function is used, the activation A of an evolving layer node n is determined by equation (v),

$$A_n = 1 - D_n \qquad (v)$$

Where A_n is the activation of the node **n** and D_n is the normalized distance between the input vector and the incoming weight vector for that node. Thus, examples which exactly match the exemplar stored within the neuron's incoming weights will result in an activation of 1 whereas examples that are entirely outside the exemplars region of input space will result in an activation of near 0.

The preferred form of learning algorithm is based on accommodating, within the evolving layer, new training examples by either modifying the connection weights of the evolving layer nodes, or by adding a new node.When a new node is added, its incoming connection weight vector is set to the input vector I, and its outgoing weight vector is set to the desired output vector O_d.The incoming weights to the winning node *j* are modified according to equation (vi), whereas the outgoing weights from node j are modified according to equation (vii).

$$W_{i,j}(t+1)=W_{i,j}(t)+\eta_1(I_i-W_{i,j}(t)) \qquad (vi)$$

Where :
$W_{i,j}(t)$ is the connection weight from input I to j at time t;
$W_{i,j}(t+1)$ is the connection weight from input I to j at time t+1;
η_1 is the learning rate one parameter;
I_i is the *i*th component of the input vector I.

$$W_{j,p}(t+1)=W_{j,p}(t)+\eta_2(A_j \text{ X } E_p) \qquad (vii)$$

Where :
$W_{j,p}(t)$ is the connection weight from j to output p at time t;
$W_{j,p}(t+1)$ is the connection weight from j to p at time t+1;
η_2 is the learning rate two parameter;
Aj is the activation of a node j, and

$$E_p = O_{d(p)} - O_{c(p)} \qquad (viii)$$

where E_p is the error at p; $O_{d(p)}$ is the desired output at p; and $O_{c(p)}$ is the calculated output at p.

The distance measure D_n in equation (v) above is preferably calculated as the normalized Hamming distance as shown in equation (ix):

$$D_n = (\sum|I_i - W_i|) / (\sum|I_i + W_i|), \text{ for } i = I \text{ to } K \qquad (ix)$$

Where,
K is the number of input nodes in the eMLP,
I is the input vector, and W is the input to the evolving layer weight matrix.

Aggregation of nodes in the evolving layer can be employed to control the size of the evolving layer during the learning process. The principle of aggregation is to merge those nodes which are especially close to each other. Aggregation can be applied for every (or after every n) training example.

2. CONNECTIONIST ARCHITECTURES

2.1 Evolving Fuzzy Neural Networks (EFuNN)

2.1.1 Architecture of EFuNN

EFuNN have a five layer structure, as shown in figure 2. Here nodes and connections are created / connected as data examples are presented. An optional short term memory layer can be used through a feedback connection from the rule (also called case) node layer. The layer of feedback connections could be used if temporal relationships of input data are to be memorized structurally.

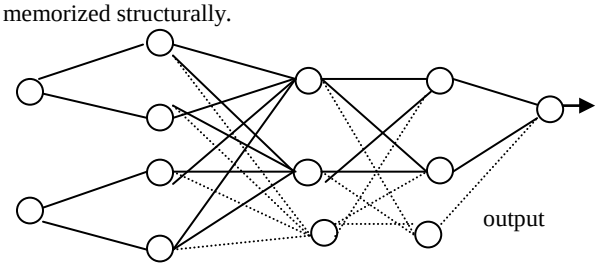

Figure 2. Evolving Fuzzy Neural Network

The input layer represents input variables. The second layer of nodes (fuzzy input neurons or fuzzy inputs) represents fuzzy quantization of each input variable space. For example, two fuzzy input neurons can be used to represent 'small' and 'large' fuzzy values. Different membership functions can be attached to these neurons (triangular, Gaussian etc.).

The third layer contains rule (case) nodes that evolve through supervised and / or unsupervised learning. The rule nodes represent prototypes (clusters, exemplars) of input-output data associations that can be graphically represented as associations of hyperspheres from the fuzzy input and the fuzzy output space. Each rule node ***r*** is defined by two vectors of connection weights, $W_1(\boldsymbol{r})$ and $W_2(\boldsymbol{r})$, the latter being adjusted through supervised learning based on the output error, and the former being adjusted through unsupervised learning based on similarity measure within a local area of the problem space.

The forth layer of neurons represents fuzzy quantization of the output variables, similar to the input fuzzy neuron representation. Here, a weighted sum input function and a saturated linear activation function is used for the neurons to calculate the membership degrees to which the output vector

associated with the presented input vector belongs to each of the output membership functions.
The fifth layer represents the values of the output variables. Here a linear activation function is used to calculate the defuzzified values for the output variables.

2.1.2 Adaptive learning in EFuNN

The learning in EFuNN is based on either of the following assumptions:
1. No rule nodes exist prior to learning and all of them are created (generated) during the evolving process; or
2. There is an initial set of rule nodes that are not connected to the input and output nodes and become connected through the learning (evolving) process.

2.1.3 Learning (evolving) algorithm for EFuNN

Each rule node, for example, $\boldsymbol{r}_j$, represents an association between a hypersphere from the fuzzy input space and a hypersphere from the fuzzy output space, the $W_1(\boldsymbol{r}_j)$ connection weights representing the co-ordinates of the center of the sphere in the fuzzy input space, and the $W_2(\boldsymbol{r}_j)$ the co-ordinates in the fuzzy output space. The radius of the input hypersphere of a rule node r_j is defined as $R_j = 1- S_j$, where S_j is the sensitivity threshold parameter defining the minimum activation of the rule node $\boldsymbol{r}_j$ to a new input vector $\mathbf{x}$ from a new example ($\mathbf{x}$, $\mathbf{y}$) in order for the example to be considered for association with this rule node.

The pair of fuzzy input-output data vectors ($\mathbf{x}_f$, $\mathbf{y}_f$) will be allocated to the rule node $\boldsymbol{r}_j$ if $\mathbf{x}_f$ falls into the $\boldsymbol{r}_j$ input receptive field (hypersphere), and $\mathbf{y}_f$ falls in the $\boldsymbol{r}_j$ output reactive field hypersphere. This is ensured through two conditions: that a local normalized fuzzy difference between $\mathbf{x}_f$ and $W_1(\boldsymbol{r}_j)$ is smaller than the radius R_j , and the normalized output error (**Err** = $\|\mathbf{y} - \mathbf{y}'\| / N_{out}$) is smaller than an error threshold **E**. N_{out} is the number of the out puts and $\mathbf{y}'$ is the output produced by EFuNN. The error parameter **E** sets the error tolerance of the system.

2.2 Evolving Neuro-Fuzzy Inference Models

2.2.1 Knowledge-Based Neural Networks (KBNN)

Knowledge is the essence of what a KBNN has accumulated during its operation. Manipulating rules in a KBNN can pursue the following objectives:
1. Knowledge Discovery
2. Improvement of the KBNN system

Different KBNNs are designed to represent different types of rules, some of them are listed below:
1. Simple propositional rules(e.g. IF x1 is A AND/OR x2 is B THEN y is C, where A, B, and C are constants, variables symbols of true/false type).
2. Propositional rules with certainty factors (e.g. IF x1 is A(CF1) AND x2 is B (CF2) THEN y is C (CFc)).

There are several methods for rule extraction from a KBNN. Three of them are explained below:
1. Rule extraction through activating a trained KBNN on input data and observing the patterns of activation ("the short term memory").
2. Rule extraction through analysis of the connections in trained KBNN ("the long term Memory").
3. Combined methods of (1) and (2).

In terms of applying the extracted from a KBNN rules to infer new information, there are three types of methods used in the KBNN:

1. The rule learning and rule inference modules constitute an integral structure where reasoning is part of the rule learning and vice-versa. This is the case in all fuzzy-neural networks and most of the neuro- fuzzy inference system.
2. The rules extracted from a KBNN are interpreted in another inference machine. The learning module is separated from the reasoning module. This is a main principle used in many AI and expert systems, where the rule base acquisition is separated from the inference machine.
3. The two options from above are possible within one intelligent system.

2.2.2 General Evolving Neuro –Fuzzy inference system

Evolving neuro-fuzzy inference systems are such systems, where both knowledge and the inference mechanism evolve and change in time, with more examples presented to the system. In the models here knowledge is represented as both fuzzy rules and statistical features that are learned in an offline or online, possibly, in a lifelong learning mode.

Figure 3 shows a general scheme of a fuzzy inference system. The decision making block is the fuzzy inference engine that performs inference over fuzzy rules and data from the database. The inference can be realized in a connectionist structure, thus making the system a neuro-fuzzy inference system.

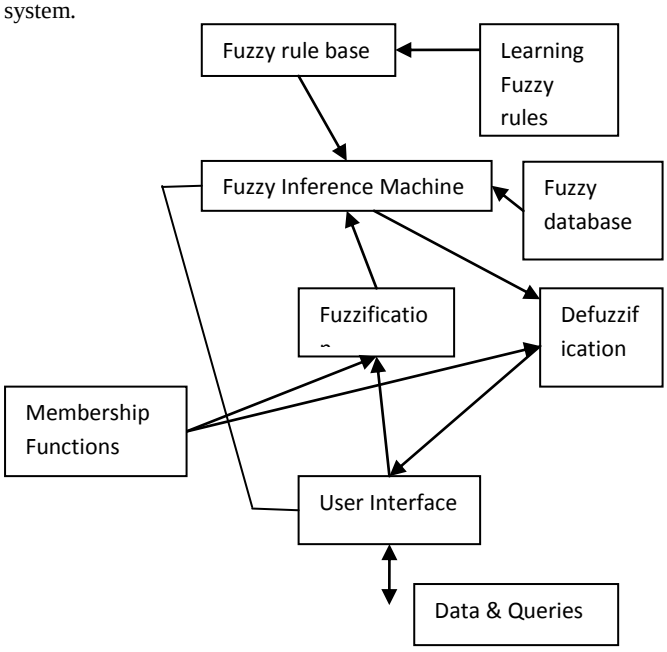

Figure 3. Fuzzy Inference System

2.2.3 Adaptive Neuro-Fuzzy Inference System (ANFIS)

ANFIS consists of five layers of MLP. The first layer represents fuzzy membership functions. The second layer and the third layers contains nodes that form the antecedent parts in each rule. The forth layer calculates the first order Takagi-Sugeno rules [5] for each fuzzy rule. The fifth layer, is the output layer, calculates the weighted global output of the system.

The backpropogtion algorithm I used to modify the initially chosen membership functions and the least mean square algorithm is used to determine the coefficients of linear output functions.

As many rules can be extracted from a trained ANFIS as there are a predefined number of rule nodes. By employing a hybrid learning procedure, the proposed architecture can refine fuzzy if-then rules obtained from human experts to describe the input-output behavior of a complex system. If human experts is not available, reasonable initial membership functions can still be set up intuitively and the learning process can begin to generate a set of fuzzy if-then rules to approximate a desired dataset.

2.2.4 Hybrid Neuro-Fuzzy Inference System(HyFIS)

HyFIS consists of two main parts, as shown in figure 4.
1. A fuzzy analysis module for fuzzy rule extraction from incoming data.
2. A connectionist module that implements and tunes the fuzzy rules through applying the backpropogation algorithm.

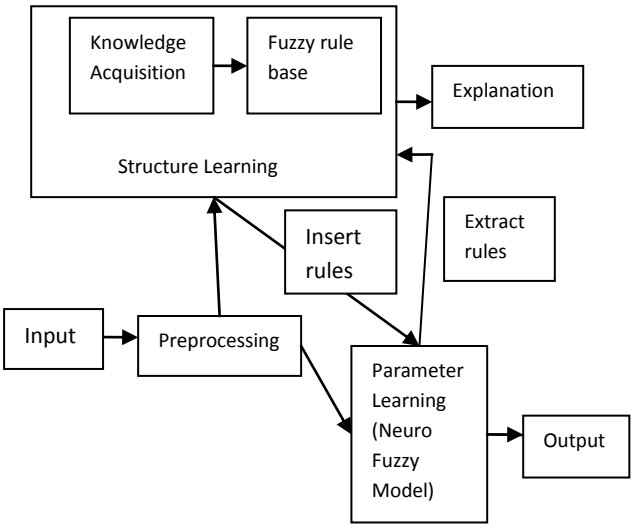

Figure 4. HyFIS

The system operates in the following mode:
1. Data examples (x, y) are assumed to arrive in chunks of m (as a partial case, let m =1).
2. For the current chunk K_i, consisting of m_i examples, n_i fuzzy rules are extracted as described below. They have a form illustrated with the following examples. IF x1 is Small AND x2 is Large THEN y is Medium.
3. The n_i fuzzy rules are inserted in the neuro-fuzzy module, thus updating the current structure of this module.
4. The updated neuro-fuzzy structure is trained with the backpropogation algorithm on the chunk of data K_i, or on a larger dataset if such is available.
5. New data x' that do no have known output vector, are propogated through the neuro-fuzzy module for recall.

2.2.5 Neuro-Fuzzy Inference Module

A block diagram of a hypothetical neuro-fuzzy module is given in figure 5. It consists of five layers: layer one is the input layer, layer two is the input membership function layer, layer three is the rule layer, layer four is the output membership function layer, and layer five is the output layer.

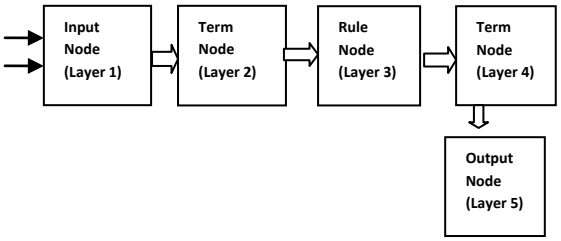

Figure 5. Hypothetical neuro-fuzzy module

Layer three performs the AND operation calculated as the min function on the incoming activation values of the membership function nodes. The membership functions are of a Gaussian type. Layer four performs the OR operation, calculated as the max function on the weighted activation values of the rule nodes connected to node in layer four:

$$O_j(4) = \max \{O_i(3) w_{i,j}\} \qquad (x)$$

Layer five performs a centre of area defuzzification:

$$O_l(5) = \sum O_j(4) C_j(4) \sigma_j(4) \,/\, \sum O_j(4)\, \sigma_j(4) \qquad (xi)$$

where,
$O_l(5)$ is the activation of the l^{th} output node;
$O_j(4)$ is the activation of the j^{th} node from layer 4 that represents a Gaussian output membership function with a center $C_j(4)$ and a width of $\sigma_j(4)$.

Through the backpropogation learning algorithm the connection weights $w_{i,j}$ as well as the centers and the width of the of the membership functions are adjusted to minimize the mean square error over the training dataset (or the current chunk of data).

3. CONCLUSION

Fuzzy neural networks are neural networks, with all neural network characteristics of training, recall, and adaption and so on, whereas Neuro-fuzzy inference systems are fuzzy rule-based systems and their associated fuzzy inference mechanisms that are implemented as neural networks for the purpose of learning and rule optimization [6].

EFuNN have features of knowledge based systems, logic systems, case-based reasoning system and adaptive connectionist-based systems all together. Through self-organization and self-adaption during the learning process, they allow for solving difficult engineering tasks as well as for simulation of emerging, evolving biological and cognitive processes to be attempted [3].

The EFuNN applications span across several application areas of information science, life sciences, and engineering, where systems learn from data and improve continuously.
ANFIS is not flexible in terms of changing the number of membership functions and rules over time, according to the incoming data. HyFIS can be used as both offline and online knowledge based learning system [5].

4. ACKNOWLEDGMENTS

Manuscript received on May 22 2010. This work is partially associated with a research project funded by the Board of College and University Development (BCUD), at University of Pune, India.

5. REFERENCES

[1] Amari, S. and Kasabov, N., "Brain-like Computing and Intelligent Information System, Springer Verlag, Singapore.

[2] Arbib M, "The Handbook of Brain Theory and Neural Networks", MIT Press, Cambridge, MA, (1995, 2002).

[3] Nikola Kasabov, "Evolving Connectionist Systems", Springer, Second Edition, 978-1-84628-345-1.

[4] Elman, Jeffrey L., "Distributed representations, Simple recurrent networks, and grammatical Structure", Machine Learning, 7, pp. 195-225.

[5] Kasabov N. AND Kozma R., " Neuro-Fuzzy Techniques for Intelligent Information System", Physical Verlag (Springer Verlag), Heidelberg, 1999.

[6] Rajesh Prasad, U.V. Kulkarni, "A Novel Evolutionary Connectionist Text Summarizer", International Conference on Anticounterfeiting, Security and Identification, ASID 2009, HongKong, by IEEE HongKong section.

[7] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine Learning*, vol. 7, pp. 195–225, 1991.

[8] Rajesh Prasad, U.V. Kulkarni, "Connectionist Approach to Generic Text Summarization", WASET, International Conference on Neural Networks, Oslo, Norway, July 2008.