NEUCOM 1207 PROD.TYPE: COM ED: CHANAKSHI
pp. 1–21 (col.fig.: NIL) PAGN: NVS – SCAN: MONICA/MOULI

ELSEVIER
Neurocomputing 00 (2001) 000–000
NEUROCOMPUTING
www.elsevier.com/locate/neucom

On-line learning, reasoning, rule extraction and aggregation in locally optimized evolving fuzzy neural networks

Nikola K. Kasabov*

Department of Information Science, University of Otago, P.O. Box 56, Dunedin, New Zealand

Received 5 April 1999; accepted 15 April 2000

Abstract

A fuzzy neural networks are connectionist systems that facilitate learning from data, reasoning over fuzzy rules, rule insertion, rule extraction, and rule adaptation. The concept of a particular class of fuzzy neural networks, called FuNNs, is further developed in this paper to a new concept of evolving neuro-fuzzy systems (EFuNNs), with respective algorithms for learning, aggregation, rule insertion, rule extraction. EFuNNs operate in an on-line mode and learn incrementally through locally tuned elements. They grow as data arrive, and regularly shrink through pruning of nodes, or through node aggregation. The aggregation procedure is functionally equivalent to knowledge abstraction. EFuNNs are several orders of magnitude faster than FuNNs and other traditional connectionist models. Their features are illustrated on a bench-mark data set. EFuNNs are suitable for fast learning of on-line incoming data (e.g., financial time series, biological process control), adaptive learning of speech and video data, incremental learning and knowledge discovery from large databases (e.g., in Bioinformatics), on-line tracing processes over time, life-long learning. The paper includes also a short review of the most common types of rules used in the knowledge-based neural networks. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Neurocomputing; Evolving fuzzy neural networks; On-line learning; Adaptive systems; Time series prediction; Rule extraction

1. Introduction: adaptive learning and knowledge processing in connectionist-based intelligent information systems

The complexity and the dynamics of many real-world problems, especially in engineering and manufacturing, requires sophisticated methods and tools for building

* Corresponding author. Tel.: + 64-3-479-8319; fax: + 64-3-479-8311.
E-mail address: nkasabov@otago.ac.nz (N.K. Kasabov).

intelligent information systems (IS). Such systems should be able to learn and deal with different types of data and knowledge through interaction with the environment in an incremental way. Seven major requirements to the IS are listed below as explained in [28]. An IS should be able to:

(1) learn fast from a large amount of data, e.g., through one-pass training;
(2) adapt in an on-line mode where new data is incrementally accommodated;
(3) have an "open" structure where new features (relevant to the task) can be introduced at any stage of the system's operation, e.g., the system creates "on the fly" new inputs, new outputs, new modules and connections;
(4) memorize data exemplars for a further refinement, or for information retrieval;
(5) learn and improve through active interaction with other IS and with the environment in a multi-modular, hierarchical fashion;
(6) adequately represent space and time in their different scales; have parameters that represent short- and long-term memory, age, forgetting, etc.;
(7) deal with knowledge in its different forms (e.g., rules; probabilities); analyze itself in terms of behavior, global error and success; "explain" what the system has learned and what it "knows" about the problem it is trained to solve; make decisions for a further improvement.

Some of the above seven issues have already been addressed in different connectionist systems. Such systems can successfully perform incremental learning [7–15], on-line learning [9,13,20]; can deal with rules [3,5,17,32,34,40,49,50,55]. The latter class of neural networks (NN) are also called knowledge-based neural networks (KBNN).

On-line learning is concerned with learning data as the system operates (usually in a real time) and data might exist only for a short time. NN models for on-line learning are introduced and studied in [1,9,13,20,34,55]. Several investigations proved that the most popular neural network models and algorithms are not suitable for adaptive, on-line learning, that include: multi-layer perceptrons trained with the backpropagation algorithm, radial basis function networks [13], self-organising maps (SOMs) [35,36], and fuzzy neural networks [22,40]. These models either operate on a fixed size connectionist structure, that limits its ability to accommodating new data; they may require both new data and the preciously used ones in order to adjust to the new data; they may require many iterations of passing data through the connectionist structure in order to learn it, which could be unacceptably time-consuming; they may be based on a global optimization algorithm, i.e., during the learning of each data item all the elements of the connectionist structure need to be adjusted. Problems of choosing the optimal initial structure, of arriving at a local minimum, of catastrophic forgetting, of lack of meaningful explanation of the stored in the connections information, and others, are often experienced (see [4]).

KBNN are pre-structured neural networks that allow for data and knowledge manipulation, including learning from data, rule insertion, rule extraction, adaptation and reasoning. KBNN have been developed either as a combination of symbolic AI systems and NN [3,25,49,50], or as a combination of fuzzy logic systems [56] and NN

[18,22–32,40,55], or as other hybrid systems [24,40,55]. Rule insertion and rule extraction operations are typical operations for a KBNN to accommodate existing knowledge along with data, and to produce an explanation on what the system has learned.

Many of the existing KBNN can capture rules but cannot operate in an on-line mode. They are usually trained in a batch, off-line mode, and then rules are extracted. A representative of this class KBNN is called fuzzy neural network FuNN [24–27,32] and is described in Section 3. FuNNs can be used to learn from data and for rule extraction and rule insertion. FuNN is further developed into a new class of KBNN called evolving fuzzy neural networks EFuNN (Section 4). EFuNNs have the advantages of the traditional NNs, KBNNs, and instance-based learning systems, but in addition they can operate in an on-line mode. The move from FuNNs to EFuNNs is a move from global optimization to local element tuning, from multiple-pass learning to one-pass learning, from a fixed connectionist structure to a fluctuating one, allowing for growing through insertion of nodes, and shrinking through node aggregation or pruning. This is also a move from one type of rules, that is dealt with in the FuNN structure, to another type, that is dealt with in the EFuNN structure, from the fuzzy neural network principles, where a NN structure is associated with linguistically meaningful fuzzy concepts, to neuro-fuzzy systems, where a connectionist-type learning mechanism is associated with a set of fuzzy rules. The two systems, FuNN and EFuNN, have a similar connectionist structure, but are based on different learning principles that define their functionality and applicability. A comparative analysis of FuNNs and EFuNNs is presented and illustrated on a bench-mark problem of dynamic time-series prediction.

Before the principles of FuNNs and EFuNNs, and the transition between the two, are described, different types of rules widely used in the KBNN, and some of them in the FuNN and the EFuNN architectures, are presented in the section below.

2. Rule extraction from KBNN — different types of rules

Knowledge (e.g., rules) is the essence of what a KBNN system has learned during its operation. Manipulating rules in a KBNN can pursue the following objectives:

(1) Understanding and explanation of the data used to train the KBNN; improvement of the KBNN system. The extracted rules can be analyzed either by experts, or by the system itself [18,22,24,32,40,55]. Different methods for reasoning can be subsequently applied to the extracted set of rules [24,56].

(2) Maintaining an optimal size of the KBNN that is adequate to the expected accuracy of the system. Reducing the structure of a KBNN can be achieved through regular pruning of nodes and connections thus allowing for knowledge to emerge in the structure, or through aggregating nodes into bigger rule clusters. The former approach is used in [19,21,38,39,41,44,46,47,53]. The latter one is explored in this paper. It is based on a regular aggregation of rule nodes in the KBNN structure, which is equivalent to aggregating rules into rule clusters before

new data and new knowledge is accommodated in the system. This is the case with the EFuNNs.

Different KBNNs are designed to represent different types of rules, some of them listed below [24]:

(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, or 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));
(3) Zadeh–Mamdani fuzzy rules [56] (e.g., IF $x1$ is A AND $x2$ is B THEN y is C, where A, B and C are fuzzy values represented by their membership functions);
(4) Takagi–Sugeno fuzzy rules [22,40] (e.g., IF $x1$ is A AND $x2$ is B THEN y is $a.x1 + b.x2 + c$, where A, B and C are fuzzy values and a, b and c are constants);
(5) Fuzzy rules of type (3) with degrees of importance and certainty degrees [18,24,32] (e.g., IF $x1$ is A (DI1) AND $x2$ is B (DI2) THEN y is C (CFc), where DI1 and DI2 represent the importance of each of the condition elements for the rule output, and the CFc represents the strength of this rule);
(6) Fuzzy rules that represent associations of clusters of data from the problem space (e.g., Rule j: IF [an input vector x is in the input cluster defined by its center ($x1$ is Aj, to a membership degree of MD1j, AND $x2$ is Bj, to a membership degree of MD2j) and by its radius Rj-in] THEN [y is in the output cluster defined by its center (y is C, to a membership degree of MDc) and by its radius Rj-out, with Nex(j) examples represented by this rule] (see [5,9,34,36]);
(7) Temporal rules [30] (e.g., IF $x1$ is present at a time moment $t1$ (with a certainty degree and/or importance factor of DI1) AND $x2$ is present at a time moment $t2$ (with a certainty degree/importance factor DI2) THEN y is C (CFc));
(8) Temporal, recurrent rules (e.g., IF $x1$ is A (DI1) AND $x2$ is B (DI2) AND y at the time moment $(t - k)$ is C THEN y at a time moment $(t + n)$ is D (CFc)).

FuNNs deal with rules of type (5), and EFuNNs deal with rules of type (6), (7) and (8). Here only EFuNN rules of type (6) are discussed.

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). The method is not practical for on-line, incremental learning in IS as past data may not be available for a consecutive activation of the trained KBNN. This method is widely used in brain-research (e.g., analyzing MRI, fMRI and EEG patterns and signals to detect rules of behavior [2,4]).
(2) Rule extraction through analysis of the connections in a trained KBNN [3,18,24,40,55] (the long-term memory). This approach allows for extracting knowledge without necessarily activating the connectionist system again on input data. It is appropriate for on-line learning and system improvement. This is the

case in the rule extraction procedures of FuNNs and EFuNNs. This approach is not used in brain study research as there are no methods known so far for processing information stored in neuronal synapses.
(3) Combined methods of (1) and (2).

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

(1) The rules extracted from a KBNN are interpreted in another inference machine. The learning module is separated from the reasoning module. This is also the main principle used in many AI and expert systems, where the rule base acquisition is separated from the inference machine [5,24,40].
(2) The rule base learning and reasoning modules constitute an integrated structure, so reasoning is part of the rule learning, and vice versa. This is the case of EFuNNs.
(3) The two options from above are possible within one intelligent system.

In terms of learning modes in a KBNN, we can differentiate the following cases: (1) off-line learning and rule extraction — first learning is performed and then rules are extracted which is one–off process, an example for such systems is FuNN; (2) on-line learning and rule extraction — rules can be extracted as part of the continuous on-line learning process, an example is EFuNN.

In terms of mode of optimization of the connections as part of the learning process, there are three cases: (1) globally optimized KBNN — an example is FuNN; (2) locally optimized KBNN — an example is EFuNN; (3) a mixture of both partial and global optimization. Globally optimized KBNN use global optimization algorithms, such as gradient-descent algorithms (e.g., [5,13,18,32,40,55]), and genetic algorithms (e.g., [16,48,52]). Usually these algorithms are computationally expensive, require the whole previous data in order the system to adjust to new data, and are applied in an off-line, batch mode. These methods often cause local minimum problems and catastrophic forgetting when used for on-line adaptation [24]. They require multiple iterations to train the KBNN. The KBNN optimized through a mixture of local and global optimization use a combination of locally optimized elements, and another part of the connectionist structure being globally optimized. This approach is used in radial basis function NN [13], the feature space mapping [9], the GAL method [1], growing and splitting elastic nets [14], melting octree network [11], CONSIDER [49]. The partially locally optimized KBNN still involve global optimization algorithms, that may cause a problem when applied to on-line learning tasks.

The method of local optimization of KBNN would allow for adjusting the KBNN to new data through tuning a small number of elements, and also for extracting locally meaningful rules. The rules can be tuned as the system works, can be optimized and used for a higher level processing. Some theoretical investigations on locally optimized systems have been conducted by Vapnik and Bottou [51]. One approach to building locally optimized KBNN for adaptive learning, rule extraction, rule aggregation, and rule reasoning for the purpose of a continuous adaptation in an on-line, life-long learning mode is based on the EFuNN structure and is presented in Sections 4 and 5.

On-line learning and local optimization in a KBNN would allow for tracing the process of knowledge emergence, for analyzing how rules change over time. That is illustrated in Section 5 on the bench mark data set with the use of the EFuNN.

3. Fuzzy neural networks FuNNs — global optimization in multiple steps, synergistic reasoning over synergistic rules

3.1. The FuNN architecture. learning, reasoning and rule extraction in FuNNs

Fuzzy neural networks are NN models that allow for fuzzy rules to be manipulated [5,18,22,24,32,34,55]. FuNN is a fuzzy neural network introduced in [24] and developed in [25–27,32] (see also WWW site: http://divcom.otago.ac.nz/infoscie/kel/CBIIS.htm – > FuzzyCope3). FuNN is a connectionist feed-forward architecture with five layers of neurons and four layers of connections (Fig. 1). The first layer of neurons receives the input information. The second layer (the condition element layer) calculates the fuzzy membership degrees to which the input values belong to predefined fuzzy membership functions, e.g., small, large. The third layer of neurons (rule layer) represents associations between the input and the output variables, fuzzy rules. The fourth layer (action element layer) calculates the degrees to which output membership functions are matched by the input data, and the fifth layer performs defuzzification and calculates exact values for the output variables. Each layer of neurons has a set of parameters that define, respectively, its summation function — to aggregate the incoming signals, its activation function — to calculate the activation value of the neurons, and its output function — to calculate the output values.

Learning in FuNNs is based on *global optimization* techniques and the output error is minimized through changing all the connection weights for multiple learning

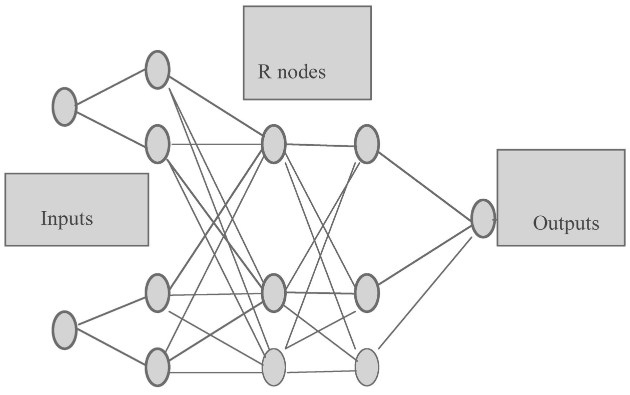

Fig. 1. An exemplar FuNN structure.

NEUCOM 1207

iterations [32]. Several global optimization training algorithms have been developed for FuNNs [24,32]. They include:

(a) A modified back-propagation (BP) algorithm that does not change the input and the output connections that represent the membership functions (MF).
(b) A modified BP algorithm that utilizes structural learning with forgetting, i.e., a small forgetting ingredient, e.g., 10^{-5}, is used when the connection weights are updated [21,37].
(c) A modified BP algorithm that updates both the inner connection layers and the membership layers. This is possible when the derivatives are calculated separately for the two parts of the triangular MF. These are also the non-monotonic activation functions of the neurons in the condition element layer [32].
(d) A genetic algorithm for training [52] — implemented in the FuzzyCOPE/3 simulator.
(e) Training with the so-called zeroing and pruning method [26].

The membership functions (MF), used in FuNN to represent fuzzy values, are either of a triangular, or of a Gaussian type, the centers of them being attached as weights to the corresponding connections. The MF can be modified during the learning procedure (e.g., altering the centers and the widths of the triangular MF).

When a trained FuNN is recalled on a new input data vector **x**', the FuNN performs a fuzzy reasoning procedure of a *synergistic type* [24,32], i.e., the output vector **y**' is calculated as a result of the activation of all rule nodes (each of them representing one fuzzy rule of type (5) as described in Section 1), every node contributing to certain degree to the final result. This is a typical fuzzy inference method, implemented in a FuNN structure with the weighted sum operation and a sigmoidal activation function.

The FuNN is designed to allow for fuzzy rules of type (5) to be inserted and extracted [24,27,32]. An example of a set of rules is given in Fig. 2 and will be explained later in this section. The rule extraction procedure consists of the following steps:

r1: if <Input1 is C 3.50> and <Input2 is A 3.85> then Output is <Output is A 0.98>

r2: if <Input1 is B 3.54> and <Input2 is B 2.73> then Output is <Output is B 2.28> and <Output is C 1.04>

r3: if <Input1 is C 0.84> and <Input2 is A 1.14> then Output is <Output is A 2.21>

r4: if <Input1 is B 1.52> and <Input2 is B 2.80> then Output is <Output is B 0.66>

r5: if <Input1 is C 0.65> and <Input2 is A 1.01> then Output is <Output is A 2.10>

r6: if <Input1 is A 0.84> and <Input2 is C 0.69> then Output is <Output is C 3.65>

Fig. 2. Modeling the gas-furnace data with FuNN — rules extracted after a FuNN was trained on the first half of the data set. *A*, *B*, and *C* denote "small", "medium", and "large" fuzzy values, respectively, represented as triangular membership functions.

NEUCOM 1207

RE1: Train a FuNN that was pre-structured according to certain fuzzy representation of the problem under consideration (inputs, outputs, fixed number of rule nodes, learning parameters, etc.)

RE2: Represent each rule node as a rule of type (5). For this purpose:

(a) The connection weights are thresholded with the use two thresholds, Thr1 and Thr2, for the condition-to-rule layer, and for the rule-to-action element layer, respectively.
(b) Each rule node is represented as a rule according to the FuNN structure shown in Fig. 1, where only the highest-value condition element for each input variable is represented in each rule with its neighboring condition elements all expressed in their corresponding linguistic fuzzy concepts.

RE3: Aggregate rules that have same condition and conclusion parts and differ only in the values of the degrees of importance and certainty degrees through calculating the *maximum* values for them.

3.2. *Using FuNNs for modeling and knowledge manipulation*

The gas-furnace data has been used by many researchers in the area of neuro-fuzzy engineering [22,40]. The data set consists of 292 consecutive values of methane at a time moment ($t-4$), and the carbon dioxide (CO_2) produced in a furnace at a time moment ($t-1$) as input variables, with the produced CO_2 at the moment (t) as an output variable.

The following steps illustrate just one method of using FuNNs for a time-series modelling: (1) A FuNN is trained on the first half of the data set until convergence; (2) The trained FuNN is tested on the other half and error is calculated; (3) Weighted fuzzy rules of type (5) are extracted from the trained FuNN, where 3 membership functions, uniformly distributed on each of the input and output variable domains, respectively, are used. Some of the extracted rules are shown in Fig. 2.

The structure and the training of FuNNs make it possible to interpret the incoming connection weights to the rule node as degrees of importance, and the outgoing connection weights from the rule nodes as certainty degrees.

FuNNs have several advantages when compared with the traditional connectionist systems, or with the fuzzy inference systems: (1) they are both statistical and knowledge engineering tools; (2) they are relatively resistant to catastrophic forgetting, i.e., when they are further trained on new data, they do not forget much about the previously used data; (3) they interpolate and extrapolate well in regions where data is sparse; (4) they accept both real input data and fuzzy input data.

At the same time there are difficulties when using FuNNs for incremental, on-line learning as FuNNs have a fixed structure and their training is based on global optimization, rather than on a local element tuning technique. The requirement for a global optimization and multiple iteration training would not be suitable for on-line training. The fixed number of rule nodes is opposite to the requirements listed in Section 1 for having an open structure where new data, new inputs and outputs can be

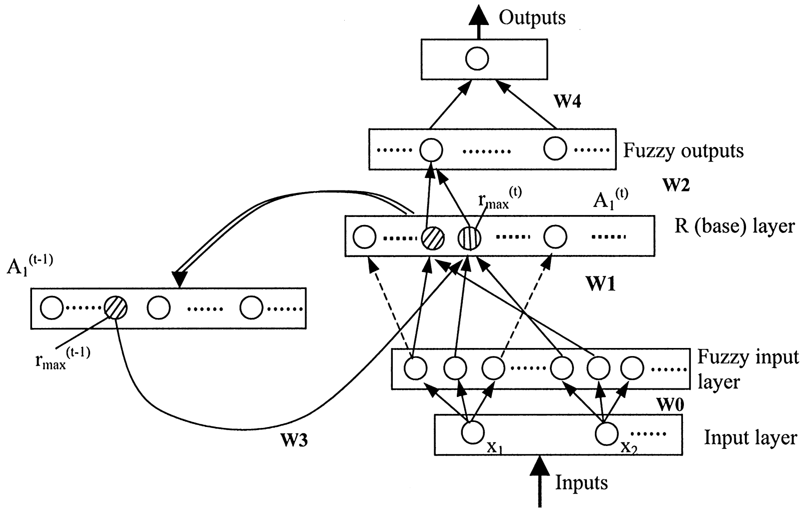

Fig. 3. An exemplar EFuNN structure.

accommodated at any time of the operation of the system. In order to overcome these drawbacks a new architecture was developed — the EFuNN, as presented in the next section.

4. Evolving fuzzy neural networks EFuNNs: local optimization, on-line learning, one-rule reasoning, structural growth, and structural aggregation

4.1. The architecture and the principles of EFuNNs

EFuNNs are introduced in [28–30] where preliminary results were given. Here EFuNNs are further developed, analyzed and applied to a bench-mark problem. An EFuNN has a five-layer structure, similar to the structure of FuNNs (Fig. 1), but 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 between input data are to be memorized structurally [30] — Fig. 3.

In contrast to the FuNNs, in EFuNNs the rule layer, the condition-to-rule connection layer and the rule-to-action connection layer have the meaning expressed in rules of type (6) (from Section 2). The third layer of neurons in EFuNN evolves through hybrid supervised/unsupervised learning. Each rule node r is defined by two vectors of connection weights — $W1(r)$ and $W2(r)$, 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 fourth layer of neurons represents fuzzy quantization for the output variables, similar to the input fuzzy neurons representation. The fifth layer represents the real values for the output variables.

The evolving process may be based on either of the following two 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 many rule nodes that are connected to the input and output nodes but their connections are defined and updated through the learning (evolving) process. The latter case is more biologically plausible as when a child is born its brain contains almost all neurons and connections but their structure and functionality evolve with the development of the person. The EFuNN evolving algorithm presented in the next section does not make a difference between these two cases.

Each rule node, e.g., r_j, represents an association between a hyper-sphere from the fuzzy input space and a hyper-sphere from the fuzzy output space, the $W1(r_j)$ connection weights representing the co-ordinates of the center of the sphere in the fuzzy input space, and the $W2(r_j)$ — the co-ordinates in the fuzzy output space. The radius of the input hyper-sphere 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 r_j to a new input vector $\mathbf{x}$ from a new example $(\boldsymbol{x},\boldsymbol{y})$ in order the example to be considered for association with this rule node. The pair of fuzzy input–output data vectors $(\boldsymbol{x}_f,\boldsymbol{y}_f)$ will be allocated to the rule node r_j if $\boldsymbol{x}_f$ falls into the r_j input receptive field (hyper-sphere), and $\boldsymbol{y}_f$ falls in the r_j output reactive field hyper-sphere. This is ensured through two conditions, that a local normalized fuzzy difference between $\boldsymbol{x}_f$ and $W1(r_j)$ is smaller than the radius r_j and the normalized output error $\mathrm{Err} = \|\boldsymbol{y} - \boldsymbol{y}'\|/N_{\mathrm{out}}$ is smaller than an error threshold E, N_{out} is the number of the outputs in the EFuNN and $\mathbf{y}'$ is the output vector produced by the EFuNN. The error threshold parameter E sets the error tolerance of the system. It also defines the radius of the output cluster for each rule node.

Definition. *A local normalised fuzzy distance* between two fuzzy membership vectors $\boldsymbol{d}_{1f}$ and $\boldsymbol{d}_{2f}$ that represent the membership degrees to which two real data vectors $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ belong to pre-defined MFs, is calculated as

$$D(\boldsymbol{d}_{1f},\boldsymbol{d}_{2f}) = \|\boldsymbol{d}_{1f} - \boldsymbol{d}_{2f}\|/\|\boldsymbol{d}_{1f} + d_{2f}\|, \tag{1}$$

where: $\|\boldsymbol{p} - \boldsymbol{q}\|$ denotes the sum of all the absolute values of a vector that is obtained after vector subtraction (or summation in case of $\|\boldsymbol{p} + \boldsymbol{q}\|$) of two vectors $\boldsymbol{p}$ and $\boldsymbol{q}$; " / " denotes division. For example, if $\boldsymbol{d}_{1f} = (0,0,1,0,0,0)$ and $\boldsymbol{d}_{2f} = (0,1,0,0,0,0)$, than $D(d_1, d_2) = (1 + 1)/2 = 1$ which is the maximum value for the local normalized fuzzy difference when uniformly distributed triangular membership functions are used. In EFuNNs, the local normalized fuzzy distance is used to measure the distance between a new input data vector and a rule node in the local vicinity of this rule node. In RBF networks Gaussian radial basis functions are allocated to the nodes and used as activation functions to calculate the distance between the node and the input vector across the whole input space.

Through the process of associating new data points to a rule node r_j, the center of this node and its radius adjust in the fuzzy input space depending on the distance between the new input vectors and the current rule node position, and on a learning rate l_j, and in the fuzzy output space depending on the output error through the Widrow–Hoff LMS algorithm (delta algorithm). This adjustment can be represented mathematically by the change of the connection weights of the rule node r_j from $W1(r_j^{(t)})$ and $W2(r_j^{(t)})$ to $W1(r_j^{(t+1)})$ and $W2(r_j^{(t+1)})$, respectively, according to the following vector operations:

$$W1(r_j^{(t+1)}) = W1(r_j^{(t)}) + l_j(W1(r_j^{(t)}) - \boldsymbol{x}_f) \tag{2}$$
$$W2(r_j^{(t+1)}) = W2(r_j^{(t)}) + l_j(A2 - \boldsymbol{y}_f)A1(r_j^{(t)})$$

where $A2 = f_2(W2A1)$ is the activation vector of the fuzzy output neurons in the EFuNN structure when $\boldsymbol{x}$ is presented; $A1(r_j^{(t)}) = f_1(D(W1(r_j^{(t)}),\boldsymbol{x}_f))$ is the activation of the rule node $r_j^{(t)}$; a simple linear function can be used for f_1 and f_2, e.g., $A1(r_j^{(t)}) = 1 - D(W1(r_j^{(t)}), \boldsymbol{x}_f))$; l_j is the current learning rate of the rule node r_j calculated as $l_j = 1/\mathrm{Nex}(r_j)$, where $\mathrm{Nex}(r_j)$ is the number of examples currently associated with rule node r_j. The statistical rationale behind this is that the more examples are currently associated with a rule node the less it will "move" when a new example has to be accommodated by this rule node. When a new example is associated with a rule node r_j not only its location in the input space, but also its receptive field expressed as its radius Rj, and its sensitivity threshold Sj, change as follows:

$$Rj^{(t+1)} = Rj^{(t)} + D(W1(r_j^{(t+1)}), W1(r_j^{(t)})), \tag{3}$$
$$\text{respectively: } Sj^{(t+1)} = Sj^{(t)} - D(W1(r_j^{(t+1)}), W1(r_j^{(t)}))$$

While the connection weights $W1$ and $W2$ capture fuzzy co-ordinates of the learned prototypes (exemplars) represented as centers of hyper-spheres, the temporal layer of connection weights $W3$ from Fig. 3 captures temporal dependencies between consecutive data examples. If the winning rule node at the moment $(t-1)$ (to which the input data vector at the moment $(t-1)$ was associated) was $r_{\max}^{(t-1)}$, and the winning node at the moment t is $r_{\max}^{(t)}$, then a link between the two nodes is established as follows:

$$W3(r_{\max}^{(t-1)}, r_{\max}^{(t)})^{\text{new}} = W3(r_{\max}^{(t-1)}, r_{\max}^{(t)})^{\text{old}} + l_3.A1(r_{\max}^{(t-1)})A1(r_{\max}^{(t)}) \tag{4}$$

where $A1(r^{(t)})$ denotes the activation of a rule node r at a time moment (t); l_3 defines the degree to which the EFuNN associates links between rule nodes (clusters, prototypes) that include consecutive data examples. If $l_3 = 0$, no temporal associations are learned.

The EFuNN system was explained so far with the use of one rule node activation (the winning rule node for the current input data). The same formulas as above are applicable when activation of m rule nodes $(m > 1)$ is propagated and used (the so-called "many-of-n" mode, or "m-of-n" for short). Usually $m = 3$.

The supervised learning in EFuNN is based on the above-explained principles, so when a new data example $\boldsymbol{d} = (\boldsymbol{x},\boldsymbol{y})$ is presented, the EFuNN either creates a new rule node r_n to memorize the two input and output fuzzy vectors $W1(r_n) = \boldsymbol{x}_f$ and $W2(r_n) = \boldsymbol{y}_f$, or the EFuNN adjusts the position and the radius of an existing rule node r_j to accommodate this example.

NEUCOM 1207

After a certain time (when certain number of examples have been presented) some neurons and connections may be pruned or aggregated. Aggregation techniques are explained in a later section of the paper.

Different pruning rules can be applied for a successful pruning of unnecessary nodes and connections. One of them is given below:

IF (Age(r_j) > OLD) AND (the total activation TA(r_j) is less than a pruning parameter Pr times Age (r_j)) THEN prune rule node r_j, where Age($\mathrm{r_j}$) is calculated as the number of examples that have been presented to the EFuNN after r_j had been fist created; OLD is a pre-defined age limit; Pr is a pruning parameter in the range [0,1], and the total activation TA(r_j) is calculated as the number of examples for which r_j has been a correct winning node (or among the m winning nodes in the m-of-n mode of operation).

The pruning rule and the way the values for the pruning parameters are defined, depend on the application task.

4.2. Local versus global generalization error. EFuNNs for on-line learning of dynamic time series

Modeling, tracing and predicting chaotic dynamic time series of continuously incoming data (with possibly changing dynamics) is an extremely difficult task. It can only be attempted with the use of on-line learning methods and the application of locally optimized structures as it is the case of EFuNNs. Here, the on-line learning ability of EFuNNs through one-pass local optimization is illustrated on the same gas-furnace bench-mark time-series data set as it was the case in Section 3 when FuNN was used.

In contrast to the experiment with the FuNN, here an EFuNN was trained on each data pair of input–output vectors as they become available in an on-line mode, and then tested immediately to predict the following data item, before the latter is accommodated (learned) in the system. Fig. 4a shows the real versus the predicted by an EFuNN values when the EFuNN was trained on the first half of the gas-furnace data (off-line, one-pass training). The evolved EFuNN is tested in an off-line mode on the second half of the data, similar to the training and testing of FuNN from Section 3 (Fig. 4b). The results are much better when EFuNN continues to evolve on each data example from the second half of the data set after it is tested on this example (on-line continuous evolving), as we can assume that the desired output values of the time series become available in the evolving process — Fig. 4c.

For an on-line learning mode, in which the EFuNN is adjusted incrementally to each example from the data stream, the generalization error on the next new input vector (for which the output vector is not known) is called *local generalization error.* The local generalization error at the moment t, for example, when the input vector is Xdt, and calculated by the evolved EFuNN output vector is Ydt', is expressed as Err_t. The cumulative local generalization error can be estimated as

$$\mathrm{TErr}_t = \mathrm{sum}\{\mathrm{Err_t}\},\ _{t=1,2,\ldots i}. \tag{5}$$

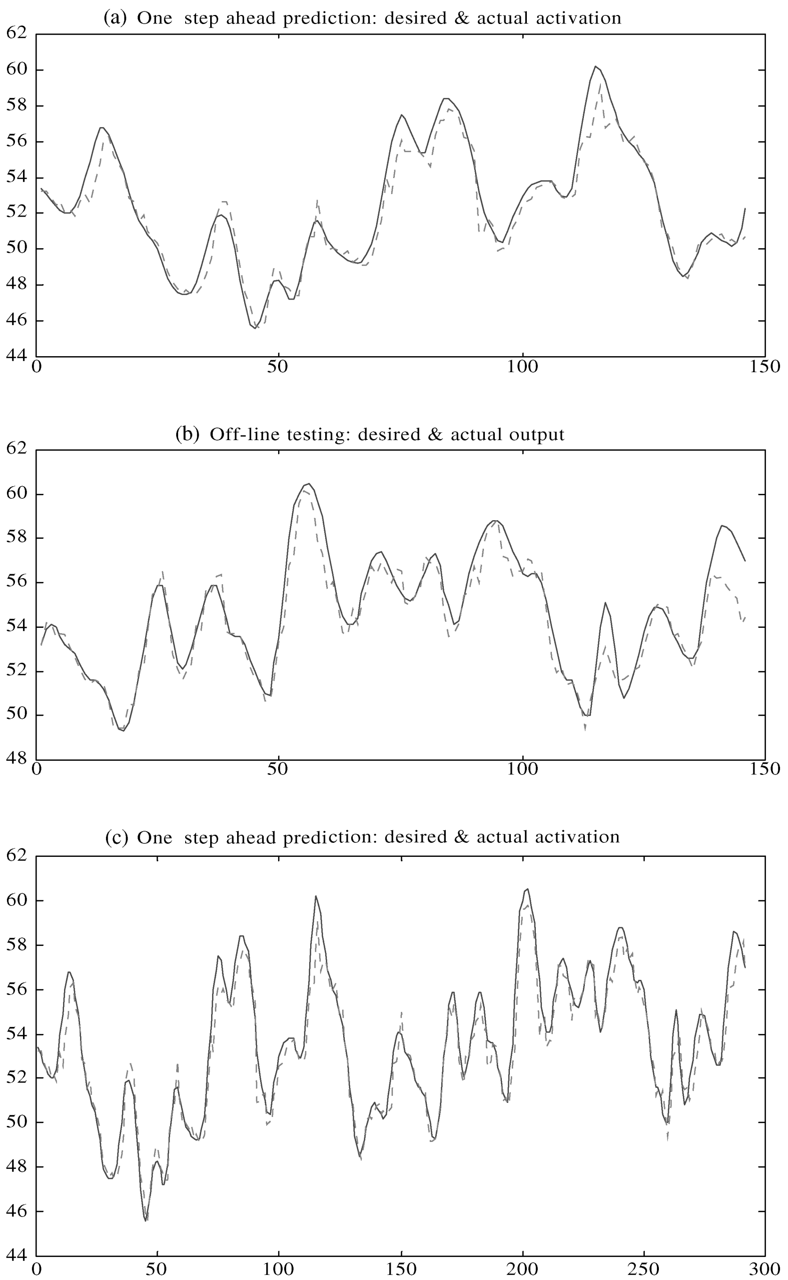

Fig. 4. The process of an EFuNN evolving on the gas-furnace data set. The desired versus the predicted one step ahead value by the EFuNN when: (a) EFuNN trained on the first half of the data; RMSE = 0.88; (b) the trained from (a) EFuNN is tested on the second half of the data set — RMSE = 4.70; (c) an EFuNN trained on the first half is continuously tested on each example from the second half and after that this example is added to the EFuNN — RMSE = 0.81. The following parameter values have been assigned [3.000, 0.100, – 1.000, 70.000, 60.000, 0.500, 2.000, 1.000, 1.000, 0.100, 0.100] to the respective parameters: number of *MF*, *E*, *Pr*, *Age*, N_{agg}, R_{max}, distance measure (in this case it is Euclidean); m; normalized values used; *T*1 and *T*2.

In contrast to the global generalization error calculated for FuNNs, which can also be calculated for EFuNNs, here the error Err_t is calculated after the EFuNN has learned the previous example ($Xd(t-1)$, $Yd(t-1)$). Each example is propagated only once through the EFuNN, both for testing the error and learning (after the output vector becomes known). The root mean square error can be calculated for each data point Di from the input data stream as

$$\mathrm{RMSE}(i) = \mathrm{sqrt}(\mathrm{sum}\{\mathrm{Err_t}\}_{t=1,2,\ldots,i})/i, \tag{6}$$

where $\mathrm{Err}_t = (d_t - o_t)^2$, d_t is the desired output value and o_t is the EFuNN output value produced for the t_{th} input vector Dt. The non-dimensional error index NDEI(i) can also be calculated as

$$\mathrm{NDEI}(i) = \mathrm{RMSE}(i)/\mathrm{std}(D(1:i)), \tag{7}$$

where $\mathrm{std}(D_1 : Di)$ is the standard deviation of the data points from D_1 to Di.

After an EFuNN is evolved on some examples from the problem space, its g*lobal generalization error* can be evaluated on a set of p new examples from the problem space as follows:

$$\mathrm{GErr} = \mathrm{sum}\{\mathrm{Err}_i\}_{i=1,2,\ldots p}, \tag{8}$$

where Err_i is the error for a vector x_i from the input space X, which vector has not been and will not be used for training the EFuNN before the value GErr is calculated. After having evolved an EFuNN on a small, but representative part of the whole problem space, its global generalization error may become sufficiently small.

When issues such as universality of the EFuNN mechanism, learning accuracy, generalization and convergence for different tasks are discussed, two cases must be distinguished:

(A) The incoming data is from a compact and bounded data space. In this case the more data vectors are used for evolving an EFuNN, the better its global generalization is on the whole problem space (or on an extraction of it).
(B) Open problem space, where the data dynamics and data probability distribution change over time in a continuous way. Here, only local generalization error can be evaluated.

5. Fuzzy rule insertion, on-line rule adaptation, and rule extraction in EFuNNs for on-line applications

5.1. Rule insertion, rule adaptation, rule aggregations and rule extraction in EFuNNs

EFuNNs are adaptive rule-based systems. Manipulating rules is essential for their operation. This includes rule insertion, rule extraction, and rule adaptation.

At any time (phase) of the evolving (learning) process, fuzzy or exact rules can be *inserted* and extracted from an EFuNN structure. Insertion of fuzzy rules is achieved through setting a new rule node r_j for each new rule, such that the connection weights $W1(r_j)$ and $W2(r_j)$ of the rule node represent this rule. For example, the fuzzy rule

(*IF x_1 is Small and x_2 is Small THEN y is Small*) can be inserted into an EFuNN structure by setting the connections of a new rule node to the fuzzy condition nodes x_1-Small and x_2-Small and to the fuzzy output node y-Small to a value of 1 each. The rest of the connections are set to a value of zero. Similarly, an exact rule, e.g., *IF x_1 is* 3.4 *and x_2 is* 6.7 *THEN y is* 9.5, can be inserted into an EFuNN structure. Here the membership degrees to which the input values $x_1 = 3.4$ and $x_2 = 6.7$, and the output value $y = 9.5$ belong to the corresponding fuzzy values are calculated and attached to the corresponding connection weights.

Rule extraction and rule aggregation are important operations as EFuNN is a knowledge-based connectionist model. Each rule node r_j can be expressed as a fuzzy rule, for example:

> r_j: IF x_1 is Small 0.85 and x_1 is Medium 0.15 and x_2 is Small 0.7 and x_2 is Medium 0.3 (Radius of the receptive field $Rj = 0.1$, maxRadius$j = 0.2$) THEN y is Small 0.2 and y is Large 0.8 (Radius of the reactive field E) [number of examples associated with this rule 20 out of 175],

where the numbers attached to the fuzzy labels denote the degrees to which the centers of the input and the output hyper-spheres belong to the respective MF. The degrees associated to the condition elements are the connection weights from the matrix $W1$. Only values that are greater than a threshold $T1$ are shown in the rules. The degrees associated with the conclusion part are the connection weights from $W2$ that are greater than a threshold $T2$. The other parameters associated with the rule represent the importance and the strength of the rule. An example of rules extracted from a bench-mark dynamic time-series data is given in the next sub-section. The two thresholds $T1$ and $T2$ are used to disregard the connections from $W1$ and $W2$ that represent small and insignificant membership degrees (e.g., less than 0.1).

Another knowledge-based technique applied to EFuNNs is *rule node aggregation*. Through this technique several rule nodes that are close to each other in the problem space are merged into one. The idea is illustrated in Fig. 5.

The aggregation procedure is presented here on a simple case of three rule nodes r_1, r_2, and r_3 . Either of the two aggregation rules can be used to calculate the $W1$

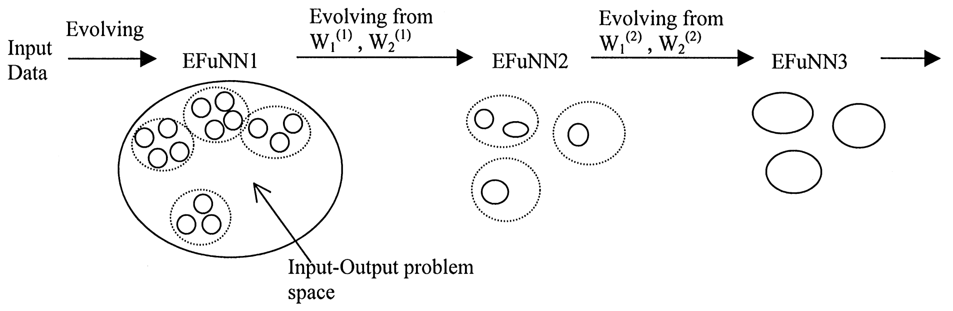
Input Data
Evolving
EFuNN1
Evolving from $W_1^{(1)}$, $W_2^{(1)}$
EFuNN2
Evolving from $W_1^{(2)}$, $W_2^{(2)}$
EFuNN3
Input-Output problem space

Fig. 5. The process of EFuNN aggregation and structure optimization can be viewed as a process of knowledge abstraction — associated clusters in the problems space emerge from the structure.

connections of a new aggregated rule node r_{agg} (the same formulas are used to calculate the $W2$ connections)

- *as a geometrical center of the three nodes:*

$$W1(r_{agg}) = (W1(r_1) + W1(r_2) + W1(r_3))/3. \tag{9}$$

- *as a weighted statistical center:*

$$W2(r_{agg}) = (W2(r_1)N_{ex}(r_1) + W2(r_2)N_{ex}(r_2) + W2(r_3)N_{ex}(r_3))/N_{sum}, \tag{10}$$

where $\mathrm{Nex}(r_{agg}) = N_{sum} = N_{ex}(r_1) + N_{ex}(r_2) + \mathrm{Nex}(r_3)$; $\mathrm{Rr}_{agg} = d(W1(r_{agg}),\ W1(r_j)) + Rj = R_{max}$, where r_j is the rule node among the three nodes that has a maximum distance from the new node r_{agg} and Rj is its current radius of the receptive field. The three rule nodes will aggregate only if the radius of the aggregated node is less than a pre-defined maximum radius R_{max}.

In order to chose for a given node r_j which other nodes it should aggregate with, two subsets of nodes are formed — the subset of nodes Nj-pos $= \{r_k\}$ that if activated to a degree of 1 will produce an output value $\mathbf{y}'(r_k)$ that is different from $y'(r_j)$ in less than the error threshold E, and the subset of nodes Nj-neg $= \{r_p\}$ so that nodes r_p cause output values to be different from $y'(r_j)$ and the difference is higher than E. Rule nodes r_k from the first subset that are closer to r_j in the input space than the closest to r_j node from the second subset $\{r_p\}$ in terms of $W1$ distance, are aggregated if the radius of the new node r_{agg} is less than the pre-defined limit R_{max} for a receptive field.

Instead of aggregating all the rule nodes from Nj-pos that are closer to the rule node r_j than the closest node from the other class Nj-pos, it is possible to keep the closest node from this aggregation pool to the other class as a separate node — a "guard", thus preventing miss-classification between the two classes in the bordering area.

Through node creation and their consecutive aggregation, an EFuNN system can adjust over time to changes in the data stream and at the same time — preserve its generalization capabilities.

Through analysis of the weights $W3$ of an evolved EFuNN, *temporal correlation* between time consecutive exemplars can be expressed in terms of rules and conditional probabilities, e.g.

$$\mathrm{IF} r_1^{(t-1)} \mathrm{THEN} r_2^{(t)}\ (0.3). \tag{11}$$

The meaning of the above rule is that some examples that belong to the rule (prototype) r_2 follow in time examples from the rule prototype r_1 with a relative conditional probability of 0.3.

5.2. Rule extraction and adaptation on the bench-mark time series data

In the experiment described below the following steps were taken:

(1) An EFuNN was evolved on half the data set for one pass learning and rules were extracted — Fig. 6a.

(a)

```
R1:if Input1 is (2 0.919) and Input2is (2 0.908) R=0.097
   then Output is  (2 0.907) Nex=29/146
R2:if Input1 is (2 0.725) (3 0.260) and Input2 is (1 0.180)(2 0.815) R= 0.132
   then Output is (1 0.198) (2 0.803) Nex=17/146
R3:if Input1 is(1 0.298)(2 0.702) and Input2 is (2 0.673)(3 0.327) R=0.097
   then Output is (2 0.599)(3 0.421) Nex=9/146
R4:if Input1 is (1 0.518)(2 0.482) and Input2 is (2 0.432)(3 0.568) R=0.120
   then Output is (2 0.428)(3 0.573) Nex=12/146
R5:if Input1 is (2 0.604)(3 0.396) and Input2 is (1 0.488)(2 0.512) R= 0.055
   then Output is (1 0.490)(2  0.517) Nex=16/146
R6:if Input1is (2 0.326)(3 0.674) and Input2 is (1 0.745)(2 0.255) R= 0.216
   then Output is (1 0.743)(2 0.258) Nex=11/146
R7:ifInput1 is (2 0.764)(3 0.236) and Input2 is (1 0.410)(2 0.590) R= 0.153
   then Output is (1 0.334)(2 0.671) Nex=10/146
R8:if Input1 is (3 0.955) and Input2 is (1  0.927) R= 0.088
   then Output is (1 0.973) Nex=3/146
...............
R33: ..........
```

(b)

```
R1:if Input1 is (2 0.882) andInput2 is (2 0.873)(3 0.108) R= 0.107
   then Output is  (2 0.906) Nex=54/292
R2:if Input1 is (2 0.792)(3 0.186) and Input2 is (1 0.245)(2 0.754) R= 0.105
   then Output is  (1 0.199)(2 0.804) Nex=42/292
R3:if Input1 is (1 0.346)(2 0.654) and Input2 is (2 0.699)(3 0.301) R= 0.127
   then Output is  (2 0.667)(3 0.341) Nex=35/292
R4:if Input1 is (1 0.608)(2 0.392) and Input2 is (2 0.482)(3 0.518) R= 0.190
   then Output is  (2 0.402)(3 0.610) Nex=37/292
R5:if Input1 is (2 0.604)(3 0.396) and Input2 is (1 0.446)(2 0.554) R= 0.274
   then Output is (1 0.460)(2 0.542) Nex=39/292
R6:if Input1 is (2 0.326)(3 0.674) and Input2 is (1 0.750)(2 0.250) R= 0.265
   then Output is  (1 0.749)(2 0.253) Nex=11/292
R7:if Input1 is (3 0.955) and Input2 is (1 0.928) R= 0.103
   then Output is  (1 0.973) Nex=3/292
R8:if Input1 is (1 0.113)(2 0.887) and Input2 is (2 0.595)(3 0.405) R= 0.247
   then Output is  (2 0.670)(3 0.339) Nex=10/292
R9:if Input1 is (1 0.803)(2 0.197) and Input2 is (3 0.901) R= 0.189
   then Output is   (3 0.923) Nex=9/292
.........................
R57:.....................
```

Denotation: The fuzzy values are denoted with numbers as follows: 1- small, 2- medium, 3 – large; the antecedent and the consequent weights are rounded to the third digit after the decimal point; smaller values than 0.1 are ignored as 0.1 is used as a threshold T1=T2 for rule extraction)

Fig. 6. Fuzzy rules (of type (6) as explained in Section 2) are extracted from: (a) an evolved on the first half of the gas-furnace data EFuNN — 33 rules; (b) an EFuNN that is evolved first on the first half on the data set, and afterwards — on the second half — 57 rules, 33 of which represent the first half of the data — some of them updated on examples from the second half.

(2) The evolved in (1) EFuNN was further evolved on the other half of the data and a next set of rules was extracted — Fig. 6b.

When compared the two sets they show a significant difference as the EFuNN trained on the fist set is further evolved on the second set which reflects in both existing rules update and the creation of new rules.

5.3. Comparing EFuNNs with FuNNs

In [34] an extensive comparative table that compares FuNNs and 15 other neuro-fuzzy techniques on the gas-furnace bench-mark data set is presented. FuNNs are shown to produce much less generalization error than the other techniques given there and compare well with the ANFIS system [22]. The clarity of the FuNNs rules though is much better than the Takagi–Sugeno rules used in ANFIS. FuNNs generalize well when the problem space is closed and bounded.

Both FuNN and EFuNN produce a similar global generalization error in an off-line mode when trained on the first half of the data and tested on the second half, with FuNNs outperforming EFuNNs slightly.

The EFuNN has a definite advantage to the FuNN as follows:

- EFuNN structure is open, i.e., it grows and shrinks according to the data distribution and the problem under consideration.
- EFuNNs can be further trained on new data as they do not forget previously used data, unless the pruning parameters allow for that.
- EFuNNs are much faster than FuNNs as they require one-pass learning.
- The explanation power of the EFuNN rules is better as they are local rules rather than synergistic rules as it is in FuNN.

Simulators of both FuNN (as part of the FuzzyCOPE/3 environment) and EFuNNs, that include learning, rule insertion and rule extraction, are available from the Web site: http://divcom.otago.ac.nz/infoscience/kel/projects/CBIIS.htm.

6. Conclusions

The paper compares two architectures of knowledge-based neural networks (KBNN) — FuNN, that is designed for off-line learning and knowledge manipulation, and EFuNN — designed for on-line continuous learning and knowledge manipulation. The EFuNNs are orders of magnitude faster than FuNNs used for the same tasks without compromising with the accuracy. The proposed method for structure optimization through rule aggregation can be used to make meaningful abstractions during the process of on-line, life long learning in the EFuNN connectionist systems.

Further applications include: adaptive speech and language processing [30]; Bioinformatics; intelligent agents on the WWW [54] for financial and economic analysis and prediction; adaptive mobile robot control; adaptive process control; adaptive expert systems; adaptive artificial life systems.

7. Uncited references

[6,33,42,43,45]

Acknowledgements

This research is part of a research programme funded by the New Zealand Foundation for Research Science and Technology, UOOX0016.

References

[1] E. Alpaydin, GAL: networks that grow when they learn and shrink when they forget, TR 91-032, International Computer Science Institute, Berkeley, CA, 1991.

[2] S. Amari, N. Kasabov (Eds.), Brain-Like Computing and Intelligent Information Systems, Springer, Berlin, 1997.

[3] R. Andrews, J. Diederich, A.B. Tickle, A survey and critique of techniques for extracting rules from trained artificial neural networks, Knowledge-Based Systems 8 (1995) 373–389.

[4] M. Arbib (Ed.), The Handbook of Brain Theory and Neural Networks, The MIT Press, Cambridege, MA, 1995.

[5] H. Berenji, P. Khedkar, Learning and tuning fuzzy logic controllers through, IEEE Trans. Neural Networks 3 (1992) 724–740.

[6] G. Carpenter, S. Grossberg, Pattern Recognition by Self-Organizing Neural Networks, The MIT Press, Cambridge, MA, 1991.

[7] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, D.B. Rosen, FuzzyARTMAP: a neural network architecture for incremental supervised learning of analog multi-dimensional maps, IEEE Trans. Neural Networks 3 (5) (1991) 698–713.

[8] H. DeGaris, Circuits of production rule — genNets — the genetic programming of nervous systems, in: R. Albrecht, C. Reeves, N. Steele (Eds.), Artificial Neural Networks and Genetic Algorithms, Springer, Berlin, 1993.

[9] W. Duch, G. Diercksen, Feature Space Mapping as a universal adaptive system, Compu. Phys. Commun. 87 (1995) 341–371.

[10] G. Edelman, Neuronal Darwinism: The Theory of Neuronal Group Selection, Basic Books, New York, 1992.

[11] L.M. Encarnacao, M.H. Gross, An adaptive classification scheme to approximate decision boundaries using local Bayes criterias — Melting Octree Networks, Report 92-047, International Computer Science Institute, Berkeley, CA, 1992.

[12] C. Fahlman, C. Lebiere, The cascade- correlation learning architecture, in: D. Turetzky (Ed.), Advances in Neural Information Processing Systems, Vol. 2, Morgan Kaufmann, Los Altos, 1990, pp. 524–532.

[13] J.A.S. Freeman, D. Saad, On-line learning in radial basis function networks, Neural Comput. 9 (7) (1997).

[14] B. Fritzke, Vector quantization with growing and splitting elastic net, in: ICANN'93: Proceedings of the International Conference on Artificial Neural Networks, Amsterdam, 1993.

[15] B. Fritzke, A growing neural gas network learns topologies, Adv. Neural Inform. Process. Systems 7 (1995).

[16] D.E. Goldberg, Genetic Algorithms in Search, Optimisation and Machine Learning, Addison-Wesley, Reading, MA, 1989.

[17] R.M. Goodman, C.M. Higgins, J.W. Miller, P. Smyth, Rule-based neural networks for classification and probability estimation, Neural Comput. 14 (1992) 781–804.

[18] T. Hashiyama, T. Furuhashi, Y. Uchikawa, A Decision Making Model Using a Fuzzy Neural Network, in: Proceedings of the second International Conference on Fuzzy Logic & Neural Networks, Iizuka, Japan, 1992, pp. 1057–1060.

[19] Hassibi, Stork, Second order derivatives for network pruning: optimal brain surgeon, in: Advances in Neural Information Processing Systems, Vol. 4, 1992, pp. 164–171.

[20] T.M. Heskes, B. Kappen, On-line learning processes in artificial neural networks, in: Math. Foundations of Neural Networks, Elsevier, Amsterdam, 1993, pp. 199–233.

[21] M. Ishikawa, Structural learning with forgetting, Neural Networks 9 (1996) 501–521.
[22] R. Jang, ANFIS: adaptive network-based fuzzy inference system, IEEE Trans. Systems Man Cybernet. 23 (3) (1993) 665–685.
[23] N. Kasabov, M. Watts, Spatio-temporal evolving fuzzy neural networks and their applications for on-line, adaptive phoneme recognition, TR 99/03, Department of Information Science, University of Otago, New Zealand.
[24] N. Kasabov, Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, The MIT Press, Cambridge, MA, 1996.
[25] N. Kasabov, Adaptable connectionist production systems, Neurocomputing 13 (2–4) (1996) 95–117.
[26] N. Kasabov, Investigating the adaptation and forgetting in fuzzy neural networks by using the method of training and zeroing, Proceedings of the International Conference on Neural Networks ICNN'96, Plenary, Panel and Special Sessions volume, 1996, pp. 118–123.
[27] N. Kasabov, Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems, Fuzzy Sets and Systems 82 (2) (1996) 2–20.
[28] N. Kasabov, ECOS: a framework for evolving connectionist systems and the eco learning paradigm, Proceedings of ICONIP'98, Kitakyushu, IOS Press, October 1998, pp. 1222–1235.
[29] N. Kasabov, The ECOS framework and the ECO learning method for evolving connectionist systems, J. of Adv. Comput. Intell. 2 (6) (1998) 195–202.
[30] N. Kasabov, Adaptive learning system and method, Patent Reg.No.503882, New Zealand, 2000.
[31] N. Kasabov, Evolving fuzzy neural networks – algorithms, applications and biological motivation, in Proceedings of Iizuka'98, Iizuka, Japan, World Scientific Singapore, October 1998, pp. 271–274.
[32] N. Kasabov, J.S. Kim, M. Watts, A. Gray, FuNN/2 – a fuzzy neural network architecture for adaptive learning and knowledge acquisition, Inform. Sci. - Appl. 101 (3–4) (1997) 155–175.
[33] S.B. Kater, N.P. Mattson, C. Cohan, J. Connor, Calcium regulation of the neuronal cone growth, Trends Neurosci. 11 (1988) 315–321.
[34] J. Kim, N. Kasabov, HyFIS: adaptive hybrid connectionist fuzzy inference systems, TR 99/05, Department of Information Science, University of Otago, New Zealand.
[35] T. Kohonen, The self-organizing map, Proceedings of the IEEE 78 (9) (1990) 1464–1497.
[36] T. Kohonen, Self-Organizing Maps, 2nd Edition, Springer, Berlin, 1997.
[37] R. Kozma, N. Kasabov, Rules of chaotic behaviour extracted from the fuzzy neural network FuNN, Proceedings of the WCCI'98 FUZZ-IEEE International Conference on Fuzzy Systems, Anchorage, May, 1998.
[38] A. Krogh, J.A. Hertz, A simple weight decay can improve generalisation, Adv. Neural Inform. Process. Systems 4 (1992) 951–957.
[39] Y. Le Cun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), Advances in Neural Information Processing Systems, Vol. 2, Morgan Kaufmann, Los Altos, 1990 pp. 598–605.
[40] C.T. Lin, C.S.G. Lee, Neuro Fuzzy Systems, Prentice-Hall, Englewood Cliffs, NJ, 1996.
[41] D. Miller, J. Zurada, J.H. Lilly, Pruning via dynamic adaptation of the forgetting rate in structural learning, Proceedings IEEE ICNN'96, Vol.1, 1996, p.448.
[42] M.T. Mitchell, Machine Learning, MacGraw-Hill, New York, 1997.
[43] X. Mitchell, X. Melanie, An Introduction to Genetic Algorithms, MIT Press, Cambridge, MA, 1996.
[44] M. Mozer, P. Smolensky, A technique for trimming the fat from a network via relevance assessment, in: D. Touretzky (Ed.), Advances in Neural Information Processing Systems, Vol. 2, Morgan Kaufmann, Los Altos, CA, 1998, pp. 598–605.
[45] S.R. Quartz, T.J. Sejnowski, The neural basis of cognitive development: a constructivist manifesto, Behav. Brain Sci., to appear.
[46] R. Reed, Pruning algorithms — a survey, IEEE Trans. Neural Networks 4 (5) (1993) 740–747.
[47] A. Sankar, R.J. Mammone, Growing and pruning neural tree networks, IEEE Trans. Comput. 42 (3) (1993) 291–299.
[48] W. Schiffman, M. Joost, R. Werner, Application of genetic algorithms to the construction of topologies for multilayer perceptrons, In: C.R. Reeves, R.F. Albrecht, N.C. Steele (Eds.), Artificial Neural Nets and Genetic Algorithms, Springer Wien, New York, 1993.

NEUCOM 1207

[49] R. Sun, A connectionist model for commonsense reasoning incorporating rules and similarities, in: Knowledge Acquisitions, Academic Press, Cambridge, 1992.
[50] G. Towel, J. Shavlik, M. Noordewier, Refinement of approximate domain theories by knowledge-based neural networks, Proceedings of the eight National Conference on Artificial Intelligence AAAI'90, Morgan Kaufmann, Los Altos, CA, 1990, pp. 861–866.
[51] V. Vapnik, L. Bottou, Neural Comput. 5 (1993) 893–909.
[52] M. Watts, N. Kasabov, Genetic algorithms for the design of fuzzy neural networks, in Proceedings of ICONIP'98, Kitakyushu, IOS Press, October 1998, pp. 793–795.
[53] L.X. Wang, Adaptive Fuzzy Systems and Control, Prentice Hall, Englewood Cliffs, NJ, 1994.
[54] M. Woldrige, N. Jennings, Intelligent agents: theory and practice, Knowledge Eng. Rev. (10) (1995).
[55] T. Yamakawa, H. Kusanagi, E. Uchino, T. Miki, A new effective algorithm for neo fuzzy neuron model, Proceedings of Fifth IFSA World Congress, 1993, pp. 1017–1020.
[56] L. Zadeh, Fuzzy Sets, Inform. and Control 8 (1965) 338–353.

Nikola K. Kasabov is Professor of Information Science in the Department of Information Science, University of Otago, Dunedin, New Zealand. He received his M.Sc. degree in Computer Science from the Technical University in Sofia, Bulgaria, in 1971. He obtained his Ph.D. degree in Mathematical Sciences in 1975 from the same university. Kasabov has published over 200 works, among them over 50 journal papers, 90 conference papers, 15 book chapters, 5 text books, 2 edited research books, 3 edited conference proceedings, 19 patents and authorship certificates in the area of intelligent systems, connectionist and hybrid connectionist systems, fuzzy systems, expert systems, speech recognition, and data analysis. He is Director of the research laboratory for Knowledge Engineering and Computational Intelligence in the Department of Information Science, University of Otago. Kasabov is the immediate past President of APNNA — Asia Pacific Neural Network Assembly. He is member of the TC12 group on Artificial Intelligence of IFIP and also member of the IEEE, INNS, NZCS, NZRS, ENNS, IEEE Computer Society. He was the general chairman of the First, the Second and the Third New Zealand International Conferences on Artificial Neural Networks and Expert Systems — ANNES'93, ANNES'95 and ANNES'97 (the latter jointly held with ICONIP'97 and ANZIIS'97).