Dynamic Evolving Fuzzy Neural Networks with 'm-out-of-n' Activation Nodes for On-line Adaptive Systems [1]

Nikola Kasabov, *Member of IEEE* and Qun Song

Department of Information Science

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

Phone: +64 3 479 8319, fax: +64 3 479 8311

nkasabov@otago.ac.nz, qsong@infoscience.otago.ac.nz

Abstract. The paper introduces a new type of evolving fuzzy neural networks (EFuNNs), denoted as mEFuNNs, for on-line learning and their applications for dynamic time series analysis and prediction. mEFuNNs evolve through incremental, hybrid (supervised/unsupervised), on-line learning, like the EFuNNs. They can accommodate new input data, including new features, new classes, etc. through local element tuning. New connections and new neurons are created during the operation of the system. At each time moment the output vector of a mEFuNN is calculated based on the m-most activated rule nodes. Two approaches are proposed: (1) using weighted fuzzy rules of Zadeh-Mamdani type; (2) using Takagi-Sugeno fuzzy rules that utilise dynamically changing and adapting values for the inference parameters. It is proved that the mEFuNNs can effectively learn complex temporal sequences in an adaptive

[1] Technical report TR99-04, Department of Information Science, University of Otago, PO Box, 56 Dunedin, New Zealand, submitted to IEEE Trans. on Fuzzy Systems, 20/05/99

way and outperform EFuNNs, ANFIS and other neural network and hybrid models. Rules can be inserted, extracted and adjusted continuously during the operation of the system. The characteristics of the mEFuNNs are illustrated on two bench-mark dynamic time series data, as well as on two real case studies for on-line adaptive control and decision making. Aggregation of rule nodes in evolved mEFuNNs can be achieved through fuzzy C-means clustering algorithm which is also illustrated on the bench mark data sets. The regularly trained and aggregated in an on-line, self-organised mode mEFuNNs perform as well, or better, than the mEFuNNs that use fuzzy C-means clustering algorithm for off-line rule node generation on the same data set.

Key words: dynamic evolving fuzzy neural networks; on-line learning; adaptive control; dynamic time series prediction; fuzzy clustering.

1. Introduction

The complexity and dynamics of real-world problems, especially in engineering and manufacturing, require sophisticated methods and tools for building on-line, adaptive intelligent systems (IS). Such systems should be able to grow as they operate, to update their knowledge and refine the model through interaction with the environment [2, 32, 33]. This is especially crucial when solving AI problems such as adaptive speech and image recognition, multi-modal information processing, adaptive prediction, adaptive on-line control, intelligent agents on the WWW [6, 57].

Seven major requirements of the present IS (that are addressed in the ECOS framework presented in [32, 36]) are discussed in [30, 32, 33, 36]. They are concerned with fast learning, on-line incremental adaptive learning, open structure organisation, memorising information, active interaction, knowledge acquisition and self-improvement, spatial and temporal learning.

On-line learning is concerned with learning data as the system operates (usually in a real time) and the data might exist only for a short time. Several investigations [16, 25, 52, 53, 54] proved that the most popular neural network models and algorithms that include multilayer perceptrons trained with the backpropagation algorithm, radial basis function networks, self-organising maps SOMs, fuzzy neural networks, and fuzzy rule-based inference systems are not suitable for adaptive, on-line learning.

At the same time several models for adaptive, on-line learning and for structure and knowledge adaptation have been developed, that include connectionist models [1, 2, 4, 8, 9, 10, 15, 17, 18, 20, 23, 27, 38, 39, 46, 54, 55], fuzzy logic models [5, 22, 28, 44, 58], models based on genetic algorithms [12], hybrid models [21, 28, 29, 31, 34, 37, 44, 47, 58].

One of the recently proposed models, called evolving fuzzy neural networks (EFuNN) [30, 32, 33] has features that make it promising for on-line adaptive systems. Here the EFuNN model is further developed with the idea that not just the winning rule node's activation is propagated (as it is the case in several models: the radial-basis function NN [48], the counter-propagation NN [24], the SOM [40, 41], the one-of-n version EFuNN [32, 33]), but a group of rule nodes is dynamically selected for every new input vector and their activation values are used to calculate the dynamical parameters of the output function. The output is calculated either with the use of : (1) weighted fuzzy rules of Zadeh-Mamdani type [59], or (2) Takagi-Sugeno fuzzy rules [56]. This is in contrast to the ANFIS fuzzy neural networks [28] and other NN models that use the activation of all hidden rule nodes where fixed parameter-values are calculated thus making the system not efficient for on-line adaptive learning.

The paper is organised as follows. Section 2 gives a brief description of EFuNNs and mEFuNNs that utilise Zadeh-Mamdani fuzzy rules, while section 3 introduces the dmEFuNN model using the Takagi-Sugeno fuzzy rules. In section 4 mEFuNNs and dmEFuNNs are applied to two bench-mark dynamic time series data, while in section 5 they are applied to two real problems for dynamic process control based on dynamic time series prediction and discrete event prediction. The results are compared with the results obtained with the use of EFuNNs, ANFIS, and multilayer perceptrons (MLP) trained with the BP algorithm. The comparative analysis indicates clearly the advantage of mEFuNNs and dmEFuNNs when used for both off-line, and especially on-line learning applications. The EFuNNs and the mEFuNNs perform well as on-line learning models and also as unsupervised, self-organised clustering models. The latter is demonstrated in the last section where on-line trained mEFuNNs are shown to perform better or similar to the fuzzy C-means clustering technique applied on the same data for rule node generation in an off-line mode.

2. Evolving Fuzzy Neural Networks EFuNNs and mEFuNNs

2.1 The EFuNN structure

Fuzzy neural networks are connectionist structures that implement fuzzy rules and fuzzy inference [22, 28, 29, 31, 34, 35, 44, 58]. FuNNs represent a class of them [31, 34, 35]. EFuNNs are FuNNs that evolve according to the ECOS principles [32, 33]. EFuNNs were introduced in [30, 32, 33]. Here the major principles of EFuNNs are briefly explained and the concept of mEFuNNs that utilise Zadeh-Mamdani fuzzy rules is presented.

EFuNNs have a five-layer structure (see Fig.1) similar to the structure of FuNNs [34]. But here nodes and connections are created/connected as data comes starting with no nodes in the beginning. An optional short-term memory layer can be used through a feedback connection from the rule (also called, case) node layer [30]. The layer of feedback connections could be used if temporal relationships between input data are to be memorised structurally.

[Figure 1]

The input layer represents input variables. The second layer of nodes (fuzzy input neurons, or fuzzy inputs) represents fuzzy quantification of each input variable space. For example, two fuzzy input neurons can be used to represent "small" and "large" fuzzy values. Different membership functions (MF) can be attached to these neurons (triangular, Gaussian, etc.). The number and the type of MF can be dynamically modified in an EFuNN [30, 32, 33]. New neurons can evolve in this layer if, for a given input vector, the corresponding variable value does not belong to any of the existing MF to a degree greater than a membership threshold. A new fuzzy input neuron, or an input neuron, can be created during the operation of the system. The task of the fuzzy input nodes is to transfer the input values into membership degrees to the MF.

The third layer contains rule (case) nodes that evolve through a hybrid supervised/unsupervised learning. The rule nodes represent prototypes (exemplars, clusters) of input-output data associations, graphically represented as an association of hyper-spheres from the fuzzy input and fuzzy output spaces [30]. Each rule node r is defined by two vectors of connection weights – $W_1(r)$ and $W_2(r)$, 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 input problem space. The fourth layer of neurons represents fuzzy quantization for the output variables. The fifth layer represents the real values for the output variables.

Each rule node represents an association between a hyper-sphere from the fuzzy input space and a hyper-sphere from the fuzzy output space, there $W_1(r_j)$ connection weights of a rule node r_j represent co-ordinates of a hyper sphere centre in the fuzzy input space, and the $W_2(r_j)$ – the corresponding co-ordinates in the fuzzy output space. The radius of an input hyper-sphere defining the minimum activation of a rule node to a new input vector in order for the new vector to be associated to this rule node, is (1 – Sthr), where Sthr is a sensitivity threshold parameter. For example, two pairs of fuzzy input-output data vectors $d_1 = (X_{d1},Y_{d1})$ and $d_2 = (X_{d2},Y_{d2})$ will be allocated to a rule node r_1 if they fall into the r_1 input and output hyper spheres, i.e. the local normalised fuzzy difference between X_{d1} and X_{d2} is smaller than the radius *r* and the local normalised fuzzy difference between Y_{d1} and Y_{d2} is smaller than an error threshold Errthr. The local normalised fuzzy difference between two fuzzy membership vectors d_{1f} and d_{2f} that represent the membership degrees to which two real values d_1 and d_2 data belong to a pre-defined membership functions (MF), are calculated as $D(d_{1f},d_{2f}) = sum(abs(d_{1f} - d_{2f}))/sum(d_{1f} + d_{2f}))$. For example, if $d_{1f} = (0,0.3,0.7,0,0,0)$ and $d_{2f} = (0,0.6,0.4,0,0,0)$, than $D(d_1,d_2) = (0.3+0.3)/2 = 0.3$. Through the process of associating (learning) of new data examples to a rule node, the two centres of this node hyper-spheres adjust in the fuzzy input space, depending on a learning rate lr_1, and in the fuzzy output space, depending on a learning rate lr_2. The adjustment of a centre r_1^1 to its new position r_1^2 can be represented mathematically by the change in the connection weights of the rule node r_1 from $W_1(r_1^1)$ and $W_2(r_1^1)$ to $W_1(r_1^2)$ and $W_2(r_1^2)$ as follows [30]:

$$W_2(r_1^2) = W_2(r_1^1) + lr_2 . Err(Y_{d1},Y_{d2}) . A_1(r_1^1),$$

$$W_1(r_1^2) = W_1(r_1^1) + lr_1 . Ds(X_{d1},X_{d2}),$$

where: $Err(Y_{d2},Y_{d2}) = Ds(Y_{d2},Y_{d2}) = Y_{d2} - Y_{d2}$, is the signed value rather than the absolute value difference vector; $A_1(r_1^1)$ is the activation of the rule node r_1^1 for the input vector X_{d2}.

The EFuNN algorithm, to evolve EFuNNs from incoming examples, is presented and illustrated in [30, 32, 33].

In spite of the similarity between the structure of the FuNNs and the EFuNNs, there is a significant difference between them, in terms of the learning and reasoning principles applied. EFuNNs are based on: (1) local-element tuning, similar to [7, 8, 9, 10], (2) single-rule inference, based on the winner-takes-all rule for the rule node activation, (3) one-pass training, (4) instance-based learning and reasoning, (5) dynamic structure, similar to [13, 23, 27, 42, 45, 49, 51, 55]. In contrast, the FuNNs are based on: (1) global optimisation technique [3], (2) synergistic, many-rules fuzzy inference, based on the activation of all rule nodes, (3) multiple iteration learning, with the use of a modified gradient descent, backpropagation algorithm, (4) hybrid localised - distributed connectionist learning and reasoning, (5) fixed structure.

Here EFuNNs are further developed into mEFuNNs through keeping the principles (1), (3), (4) and (5), and extending the principle (2) to using several (m) of the highest activated rule nodes instead of one. mEFuNNs are successfully applied in sections 4 and 5 to complex dynamic time series prediction tasks. They are shown in section 6 to perform in an on-line mode as well as the fuzzy C-means clustering algorithm in an off-line mode.

In the first version of mEFuNNs, the same evolving algorithm is used, but the activation of the m highest activation rule nodes is used instead of the winner only, as it is shown in the mEFuNN evolving algorithm below.

2.2 The mEFuNN evolving algorithm

Here the algorithm for on-line training of a mEFuNN is given. Vector operations are used to simplify the denotation:

1. Initialise mEFuNN connections which may be set through inserting fuzzy rules in the structure [36]. If there are no initial rule (case) nodes available and there are no rule nodes connected to the fuzzy input and fuzzy output neurons, then create the first node rn = 1 to represent the first example (X_{d1}, Y_{d1}) and set its input W_1(rn) and output W_2(rn) connection weight vectors as follows:

<Create a new rule node rn>: W_1(rn) = EX; W_2(rn) = TE, where TE = Y_{d1f} is the fuzzy output vector for the current fuzzy input vector EX = X_{d1f}.

2. WHILE <there are examples in the input stream> DO

Enter the current example (X_{di}, Y_{di}), EX denoting its fuzzy input vector. If new variables appear in this example, which are absent in the previous examples, create new input and/or output nodes with their corresponding membership functions.

3. Find the *normalised fuzzy local distance* between the fuzzy input vector EX and the already stored patterns (prototypes, exemplars) in the rule (case) nodes $r_j = r_1, r_2, \ldots, r_n$

$$D(EX, r_j) = \text{sum}(\text{abs}(EX - W_1(r_j))) / \text{sum}(W_1(r_j)+EX)$$

4. Find the activation A1 (r_j) of the rule (case) nodes r_j, $r_j = r_1 : r_n$. Here radial basis activation function, or a saturated linear one, can be used, i.e. A1 (r_j) = radbas (D(EX, r_j)), or A1(r_j) = satlin (1 – D(EX, r_j)). The former may be appropriate for function approximation tasks, while the latter may be preferred for classification tasks.

5. Update the pruning parameter values for the rule nodes, e.g. age, average activation.

6. Find *m* case nodes r_j with highest activation value A1(r_j) which is above a sensitivity threshold Msthr.

7. From the *m* case nodes, find one node r(inda1) which has the maximum activation value maxa1.

8. If maxa1 < Sthr, then <Create a new rule node> using the procedure from step 1

ELSE

9. Propagate the activation of the chosen set of *m* rule nodes (rj1,..,rjm) to the fuzzy output neurons:

$$A2 = \text{satlin}\ (A1(rj1:rjm)\ .\ W_2)$$

10. Calculate the fuzzy output error vector: Err = A2 - TE.

11. IF (D(A2,TE) > Errthr) <Create a new rule nod> using the procedure from step 1

12. Update: (a) the input, and (b) the output of the m-1 rule nodes *k = 2: jm* in case of a new node was created, or m rule nodes *k =j1 :jm* , in case of no new rule node was created:

(a) $Ds(EX,W_1(r_k)) = EX - W_1(r_k)$; $W_1(r_k) = W_1(r_k) + lr_1.Ds(EX, W_1(r_k))$, where lr_1 is the learning;

(b) $A2(r_k) = \text{satlin}(W_2(r_k).\ A1(r_k))$; $Err(r_k) = TE – A2(r_k)$;

$W_2(r_k) = W_2\ (r_k) + lr_2.\ Err(r_k).A1(r_k)$, where lr_2 is the learning rate;

13. Prune rule nodes rj and their connections that satisfy the following fuzzy pruning rule to a pre-defined level:

IF (a rule node r_j is OLD) AND (average activation $A1av(r_j)$ is LOW) and (the density of the neighbouring area of neurons is HIGH or MODERATE (i.e. there are other prototypical nodes that overlap with j in the input-output space; this condition apply only for some strategies of insetting rule nodes as explained in a sub-section below)

THEN the probability of pruning node (r_j) is HIGH

The above pruning rule is fuzzy and it requires that the fuzzy concepts of OLD, HIGH, etc., are defined in advance (as part of the EFuNN's chromosome). As a partial case, a fixed value can be used, e.g. a node is OLD if it has existed during the evolving of a FuNN from more than 1000 examples. The use of a pruning strategy and the way the values for the pruning parameters are defined depends on the application task.

14. Aggregate rule nodes, if necessary, into a smaller number of nodes. A C-means clustering algorithm can be used for this purpose as illustrated in the last section.

15. END of the while loop and the algorithm

3. Dynamic Evolving Fuzzy Neural Networks dmEFuNNs with "m-out-of–n" Activation Nodes and Takagi-Sugeno Fuzzy Rules

3.1 General principles

The introduced here dynamic evolving systems dmEFuNNs have a similar structure as the mEFuNNs, but they use the Takagi-Sugeno fuzzy inference rules to calculate the output vectors. Instead of the Zadeh-Mamdani fuzzy rules of the type, e.g.

IF x_1 is Small and x_2 is Small THEN y is Small, used in EFuNNs [59], here the first-order Takagi-Sugeno fuzzy rules [56] are used that are of the following type:

$$\text{IF } x_1 \text{ is Small and } x_2 \text{ is Small THEN } y = a_0 + a_1.x_1 + a_{2.}\, x_2 + \ldots + a_n.x_n ,$$

Where a_0 , a_1,…, a_n are dynamically calculated for any new input vector from a system of m linear equations:

$$y_1 = a_0 + a_1.x_{11} + a_{2.}\, x_{12} + \ldots + a_n.x_{1n} ,$$

$$y_2 = a_0 + a_1.x_{21} + a_{2.}\, x_{22} + \ldots + a_n.x_{2n} ,$$

……………………..

$$y_m = a_0 + a_1.x_{m1} + a_{2.}\, x_{m2} + \ldots + a_n.x_{mn} ,$$

where: y_1, y_2, …, y_m are the output values for the m highest activated rule nodes; (x_{i1}, x_{i2}, …, x_{in}) is the characterising vector of the input space cluster for the rule node r_i; this vector is subject to adjustment following the same rule for adjusting the fuzzy input hyper-sphere in EFuNN as explained in [30, 32, 33].

For solving the above system of equations the following method is used and applied in an on-line (or in an off- line) mode during a dmEFuNN recall (before its adaptation). The method is explained here on a simple example of n inputs and one output system:

1) For each input vector X_{di}, find m rule nodes r_{i1}, r_{i2}, …, r_{im} with the closest fuzzy normalised local distance Ds to the fuzzy input vector X_{dif}, calculated as:

$$Ds(X_{dif}, r_{ij}) = \text{sum}\ (\text{abs}(X_{dif} - W1(r_{ij})) \,/\, \text{sum}\ (X_{dif} + W1(r_{ij}))$$

where $W1(r_{ij})$ is the weight matrix of the connections from the fuzzy quantification layer to the rule nodes layer.

2) Use the formula, $\boldsymbol{A} = [a_0\ a_1\ a_2 \dots a_n]^T = (X^T Z X)^{-1} X^T Z Y$, to calculate the weighted least-square estimator (WLSE), where Z is weighting matrix for placing heavier emphasis on more important data. In this case, Z is a diagonal matrix and $z_{jj} = 1 - Ds(X_{dif}, r_{ij})$.

3) The output of dmEFuNN is calculated as

$$\mathbf{y} = X_{dif}\, \boldsymbol{A}$$

The structure of dmEFuNN is given in Fig.2

[Figure 2]

The first, second and third layers of dmEFuNNs have exactly the same structures and functions as the EFuNNs.

The fourth layer, the fuzzy inference layer, selects m rule nodes from the third layer which have the closest fuzzy normalised local distance to the fuzzy input vector, and then, a Takagi-Sugeno fuzzy rule will be formed using the weighted least-square estimator.

The last layer calculates the output of dmEFuNN.

3.2. Choosing the number of activated nodes *m* in dmEFuNNs

The number m of activated nodes used to calculate the output values for a dmEFuNN is not less than the number of the input nodes plus one.

The dmEFuNNs calculate the output using a Takagi-Sugeno fuzzy rule which is formed by weighted least-square estimator. A first-order Takagi-Sugeno fuzzy rule has a consequent part that is a linear function:

$$y = a_0 + a_1 x_1 + a_2 x_2 + \dots + a_n x_n.$$

where : $x_1, x_2, \dots, x_n$ are n elements of the input vector.

On the basis of the theory of Least-Square Estimator, the experiments have to be performed to obtain a training data set composed of m data pairs in order to get the n + 1 coefficients **a** = $[a_0, a_1, a_2, \dots, a_n]$. It is necessary that m >= n+1 to identify uniquely the unknown vector **a.** If m = n +1 and the matrices composed of m data pairs is a nonsingular, then **a** can be obtained by solving a linear equations. However, since the data might be contaminated by noise, or the model might not be appropriate for describing the target system, usually m is greater than n + 1, indicating that there are more data pairs than fitting parameters.

In the case of dmEFuNNs, these m data pairs come from the m activated nodes that means the number m is not less than the number of input elements n + 1.

4. Off-line versus on-line learning in mEFuNNs and dmEFuNNs – global versus local generalisation

The EFuNNs, the mEFuNNs and the dmEFuNNs can be used for both off-line learning and on-line learning thus optimising global generalisation error, or a local generalisation error. This is in contrast to the multi-layer perceptrons (MLP), or the adaptive neural-fuzzy inference systems (ANFIS and FuNN) that calculate global generalisation error only [44]. Let us assume that after certain training steps, several nodes have been created. In case of dmEFuNNs, for a new input vector (for which the output vector is not known), a subspace consisted of m rule nodes is found and a first-order Takagi-Sugeno fuzzy rule is formed using the Least-Square Estimator method. This rule is used to calculate the dmEFuNN output value. In this way a dmEFuNN acts as an universal function approximator using m linear functions in a small m-dimensional node subspace. The accuracy of approximation depends on the size of the node subspaces, the smaller the subspace is, the higher the accuracy. It means that if there are sufficient training data vectors and sufficient rule nodes are created, a satisfying accuracy can be obtained.

In an on-line learning a mEFuNN is evolved incrementally on different segments of data from the input stream (as a partial case this is just one data item). Off-line learning can also be applied on a mEFuNN, when the system is evolved on part of the data, and then tested on another part from the problem space, which completes the training and testing procedure as it is the case in many traditional NN models.

When issues such as universality of the mEFuNN mechanism, learning accuracy, generalisation and convergence for different tasks are discussed, two cases must be distinguished [30]:

(a) The incoming data is from a compact and bounded data space. In this case the more data vectors are used for evolving a mEFuNN, the better its generalisation is on the whole problem space (or an extraction of it). After a mEFuNN is evolved on some examples from the problem space, its *global generalisation error* can be evaluated on a set of p new examples from the problem space as follows:

$$GErr = \text{sum } \{Err_i\}_{i=1,2,\dots p},$$

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 mEFuNN before the value GErr is calculated. After having evolved an mEFuNN on a small, but representative part of the whole problem space, its global generalisation error can become sufficiently small. This is valid for both off-line learning mode and on-line learning (when an mEFuNN is evolved on k examples and then used to generalise on the next p examples).

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

$$TErr_t = \text{sum } \{Err_t\}_{,\ t=1,2,\dots i}.$$

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

$$RMSE(i) = \text{sqrt } (\text{sum}\{Err_t\}_{t=1,2,\dots,i}) / i),$$

where: $Err_t = (d_t - o_t)^2$, d_t is the desired output value and o_t is the DEFuNN output value produced for the t_{th} input vector. The non-dimensional error index NDEI(i) can also be calculated as follows:

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

where: std (D(1:i)) is the standard deviation of the data points from 1 to i.

(b) Open problem space, where the data dynamics and data probability distribution can change over time in a continuous way. Here, local generalisation error only can be evaluated.

5. mEFuNNs and dmEFuNNs for On-line Adaptive Learning of Bench-mark Data Sets: A Comparative Analysis with Other Techniques

The presented above mEFuNNs and dmEFuNNs can be used for both off-line and on-line learning tasks, such as classification, decision making, dynamic time-series approximation and prediction. Here the latter is illustrated on two bench-mark time-series data sets: the gas-furnace data set, also used in [14, 28, 34], and the Mackey Glass data set, also used in [26, 28, 43, 48]. The performances of the mEFuNN and dmEFuNN systems are compared with other connectionist and fuzzy connectionist models. In the off-line cases, each method took first half of the data set as training data and the following half as testing data. In the on-line cases, the mEFuNNs were trained on data step-by-step, and tested on the following data point.

5.1. mEFuNNs and dmEFuNNs for the gas-furnace bench-mark dynamic time series prediction

The gas-furnace data has been used by many researchers in the area of neural-fuzzy engineering for control, prediction and adaptive learning [14, 28, 30, 34, 37]. 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 were taken in two experiments illustrated in Fig.3a,b,c,d with the use of mEFuNNs:

1) Off-line learning: a dmEFuNNs is trained in an off-line mode on the first half of the data from the gas furnace data set, and tested on the whole data set for one step ahead prediction (Fig.3a) and three steps ahead prediction (Fig.3b);

2) On-line learning: a dmEFuNNs is trained on each data item in an on-line mode and tested immediately to predict one step ahead output data value (Fig.3c); and three steps ahead data value (Fig. 3d).

[Figure 3,a,b,c,d]

In the above experiments the dmEFuNN was set with 5 MF and the following parameter values: sensitivity threshold Sthr = 0.9; error threshold Errthr = 0.05; learning rate for the first and second layer lr1 = 0.01, lr2 = 0.001 respectively; number of nodes for WLSE m = 8.

To compare the performance of the dmEFuNN with the performances of other connectionist models for both off-line and on-line learning on the same data sets and same experimental settings, the following other models were experimented, and the corresponding results presented too in table 1 and table 2: mEFuNN, EFuNN, ANFIS, multi-layer perceptron MLP. While neither MLPs, nor ANFIS models are appropriate to be used in an on-line learning mode, the dmEFuNNs outperforms both of them in an off-line learning mode. Both mEFuNNs and dmEFuNNs outperform the EFuNNs in the on-line learning mode. They are also better than a linear regression method and the 'Random Walk' method. Other NN models, such as CMAC [1] show unsatisfactory performance on complex time series data.

The same superiority of mEFuNNs and dmEFuNNs is demonstrated on another bench-mark data set and on two real-world problems as shown below.

5.2 mEFuNNs and dmEFuNNs for on-line, adaptive learning of the Mackey Glass time series data

The Mackey Glass (MG) time delay differential equation:

$$\frac{d(x)}{d(t)} = \frac{0.2x(t - \tau)}{1 + x^{10} (t - \tau)} - 0.1x(t)$$

is a chaotic time series for some values of the parameters x(0) and τ. The fourth-order Runge-Kutta method was applied to find the numerical solution to the above MG equation. Here $x(0) = 1.2$, $\tau = 17$, and $x(t) = 0$ for $t < 0$ were assumed. The data set for training and testing mEFuNNs is in the following format:

$$[x(t-18)\ \ x(t-12)\ \ x(t-6)\ \ x(t)\ \ x(t+6)]$$

It intends to predict future value x(t+6) from 4 points spaced at six time intervals.

The following two experiments were conducted:

(1) Off-line training of dmEFuNNs: 1000 data points, from t = 118 to 1117, are extracted. The first half of the data set is taken as training data, and the other half as testing data; both are 500-by-5 matrices for the off-line mode. The following parameters are used in the experiment: 5 MF; Sthr = 0.95; errthr = 0.05; lr1 = 0.01; lr2 = 0.001; and WLSE = 8. The results are shown in fig.4a and in table 1.

(2) On-line training of dmEFuNNs: a dmEFuNN is trained on each input-output pair in an on-line mode and tested immediately to predict the following output value (Fig. 4b and table 2). In

this experiment the dmEFuNN was set up with 5 MF and the following parameter values: Sthr = 0.95; errthr = 0.05; lr1 = 0.01; lr2 = 0.001; and WLSE = 8.

[Figure 4a,b]

Table 1. shows the results of off-line training and testing when mEFuNNs, EFuNN, ANFIS and MLP-BP neural network were used on the same data sets and for the same tasks. The MLP-BP neural network applied here used an approximation of Newton's method called Levenberg-Marquardt which more powerful than the gradient descent method. Table 2. shows the results of on-line training and testing for some time series prediction when mEFuNNs, EFuNN Linear Regression and Random Walk were used on the same data sets. ANFIS or MLP network usually are not suited for these cases, in which training or learning process is only 'one-pass'.

The capabilities of mEFuNNs are attributed to the fact that mEFuNNs can achieve a complex non-linear mapping, similar to [11, 19]. As for the similar training and testing precision, mEFuNNs take less time than ANFIS and the MLP-BP network. It becomes more obvious when there are more inputs and more examples used. dmEFuNN uses the Weighted Least-Square Estimator method rather than a gradient descent method. This is an important characteristic that makes dmEFuNN well-suited for on-line, adaptive systems. As it is the case in EFuNNs, mEFuNNs can automatically create and adjust their structure. The complexity of the structure depends on the complexity of the training data and the chosen parameters.

[Table 1. & Table 2.]

6. Using mEFuNNs and dmEFuNNs For On-line Learning and Prediction in

Real Applications

6.1 Waste water flow prediction problem

The problem is to predict a waste water flow coming from three pumps into a sewage plant [34]. The flow is measured every hour. It is important to be able to predict the volume of the flow as the collecting tank has a limited capacity (in this case it is 650 cubic meters) and a sudden overflow will cause bacteria, that the clean water, to be thrown away. As there is very little data available before the control system is installed and put in operation, the control system has to be adaptive and learn the dynamics of the flow as it operates in an on-line mode.

The data set has the following format:

[F(t) F(t-1) MA12h(t) MA24h(t) F(t+1)],

where: F(t), F(t-1) and F(t+1) are the water flows at time t, t-1 and t+1 (hours) respectively. F(t+1) is the output. MA12h(t) and MA24h(t) are the moving average 12 hours and 24 hours data. The time series data consists of 475 data points.

A dmEFuNN is trained in both off-line (see Fig.5a and table 1), and in an on-line mode for one-step ahead prediction (Fig.5b and table 2). In the above experiments the dmEFuNN was set up with 5 MF and the following parameter values: Sthr = 0.9; errthr = 0.05; lr1 = 0.01; lr2 = 0.001; and WLSE = 12.

[Figure 5a,b]

6.2 Dissolved oxygen prediction in a chemical bio-reactor for sewage treatment

It is necessary to treat waste-water before it flows into the environment, removing harmful chemical compounds and pathogenic forms. Nitrogen found in all municipal and agricultural wastewater and in some industrial effluents have to be removed. This removal is usually achieved through a biological treatment split in to two distinct phases: nitrification and denitrification. In the sequencing batch reactors (SBR), a tank is used for nitrification and denitrification in distinct time phases. The problem is to identify the endpoint of a biological reaction through studying the pH-, ORP- (Oxidation Reduction Potential) and DO- (Dissolved Oxygen) profiles in the SBR. Here, a dmDEFuNN is used for identification and prediction of a DO-profile.

The data set for training dmEFuNN is in the following format: [DO(t-15) DO(t-10) DO(t-5) DO(t) DO(t+5)], where: DO(t) is the value of dissolved oxygen; DO(t+5) is the output value. The dmEFuNN is trained on an initial training data set in an off-line mode, and then, it is continuously trained in on-line mode on three sets of data and tested on the prediction of five steps ahead output values. In this experiment the dmEFuNN was set with 5 MF, and with the following parameter values: Sthr = 0.9; Errthr = 0.05; lr1 = 0.01; lr2 = 0.001; and WLSE = 12. The results are shown in Fig.6. In Fig.6(a), the initial off-line training results are shown, and in Fig. 6(b), (c), (d) show the results of dmEFuNN in an on-line training and prediction on the each of three consecutive batch data sets.

[Figure 6]

7. Aggregation of Rule Nodes Trough C-means Clustering Algorithm

Aggregation is the process of merging several rule nodes into one, thus producing smaller number of nodes (clusters). Reducing the number of rule nodes may be needed to keep the number of the evolved nodes in a predefined limit, to improve the generalisation of the system, and to speed-up the inference process. Too small number of rule nodes will not allow the system to perform well on new data when local generalisation applies. The task of finding the optimum number of rule nodes is an optimisation task, which can be applied regularly while the system operates in an on-line mode. As each rule node represents one cluster of data, the process can be viewed as a clustering process when a smaller number of clusters are being sought.

The number of the rule nodes in EFuNNs, mEFuNNs and dmEFuNNs depends on the system parameters, such as the sensitivity threshold Sthr, error threshold Errthr, etc. Table 3a shows the number of the rule nodes when different values for the parameters are used for an mEFuNN system evolved in an on-line mode on the gas-furnace data (146 examples). Fig.7a shows the training data (denoted by “o”) in a two-dimensional input space where the rule nodes are also shown (denoted by “+”). Here the number of the rule nodes is 39.

Two methods for on-line rule node aggregation based on using two thresholds, one for the input fuzzy space and the other – for output fuzzy space, are introduced in [37]. Here the C-means clustering algorithm is used over the rule nodes when a pre-defined number of clusters are sought which in tabl.3b is chosen to be the same number as from tabl.3a. The comparative study shows that trough applying the C-means clustering algorithm the same aggregation effect can be achieved with similar error obtained. Fig.7b shows the result of the aggregation of an mEFuNN which had initially 134 nodes to 39 clusters.

In the experiments above the mEFuNN system worked in an on-line, self-organised mode. To compare the performance of the system in this mode to the performance of the system evolved in an off-line mode, the C-means clustering algorithm was applied to generate rule nodes (cluster centers) on the initial data set of 146 data items for a different number of clusters (set to be the same as the number of the clusters from table 3a and 3b). After the cluster centers were found, their co-ordinates in the fuzzy input and fuzzy output spaces are used as connection weights attached to the rule nodes in an mEFuNN systems and the system was tested for a global generalisation error – table 3c. In contrast to tables 1a and 2a, here both RMSE and NDEI errors are very similar or lower than the case when the rule nodes of an mEFuNN were evolved through the C-means clustering algorithm in an off-line mode. This experiment demonstrates the power and the accuracy of the evolving and aggregating techniques for on-line, self-organised learning of complex dynamic time series with the use of mEFuNN structures.

8. Conclusions and directions for further research

This paper presents the principles of two types of dynamic evolving fuzzy neural networks, mEFuNNs and dmEFuNNs for building on-line, knowledge-based, adaptive learning systems. Both mEFuNNs and dmEFuNNs are based on the EFuNN architecture [30, 32, 33], but use a dynamically selected set of the *m* highly activated rule nodes to evaluate the parameters of the output functions, which in the former case are fuzzy rules of Zadeh-Mamdani type, and in the latter case – fuzzy rules of of Takagi-Sugeno type. The value of m depends on the number of the input variables n, but it is not less than n + 1.

The proposed systems demonstrate superiority when compared with the EFuNNs, the fuzzy neural network ANFIS, the MLP and statistical methods. For just one pass of adaptive learning, mEFuNNs and dmEFuNNs achieve a local generalisation error that is close or smaller than the global generalisation error achieved in a MLP, or ANFIS, or with the use of fuzzy C-means clustering algorithm to evolve the rule nodes.

A solution to the problem of rule node aggregation is proposed with the use of the C-means clustering algorithm. The clustering algorithm is applied regularly on on-line evolved mEFuNNs. A significant reduction of the number of the evolved rule nodes is achieved without sacrificing the accuracy for the prediction.

Further directions for research include: application of mEFuNNs and dmEFuNNs for adaptive mobile robot control; mathematical investigations of some convergence properties of mEFuNNs and dmEFuNNs; application of mEFuNNs and dmEFuNNs for building adaptive learning agents for embedded systems and for systems on the Internet.

Acknowledgements

This research is part of a research programme funded by the New Zealand Foundation for Research Science and Technology, contract UOO808.

Reference

1. Albus, J.S., A new approach to manipulator control: The cerebellar model articulation controller (CMAC), Tarns. of the ASME: Journal of Dynamic Systems, Measurement, and Control, pp.220:227, Sept. (1975)

2. Amari, S. and Kasabov, N. eds, “Brain-like Computing and Intelligent Information Systems”, Springer Verlag,1997.

3. Amari, S., Mathematical foundations of neuro-computing, Proc. of IEEE, 78 (9), Sept. (1990)

4. Arbib, M. (ed) The Handbook of Brain Theory and Neural Networks,The MIT Press, 1995.

5. Bezdek, J (ed). Analysis of Fuzzy Information, 3 vols. Boca Raton,Fla, CRC Press, 1987.

6. Bollacker, K., S.Lawrence and L.Giles, CiteSeer: An autonomous Web agent for automatic retrieval and identification of interesting publications, 2nd International ACM conference on autonomous agents, ACM Press, 1998, 116-123

7. Bottu and Vapnik, "Local learning computation", Neural Computation, 4, 888-900 (1992)

8. Carpenter, G. and Grossberg S., Pattern recognition by self-organizing neural networks , The MIT Press, Cambridge, Massachusetts (1991)

9. Carpenter, G. and S. Grossberg, "ART3: Hierarchical search using chemical transmitters in self-organising pattern-recognition architectures", Neural Networks, 3(2) 129-152(1990).

10. Carpenter, G. 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 Transactions of Neural Networks , vol.3, No.5, 698-713 (1991).

11. Cybenko, G., Approximation by super-positions of sigmoidal function, Mathematics of Control, Signals and Systems, 2, 303-314 (1989)

12. DeGaris, H., "Circuits of Production Rule - GenNets – The genetic programming of nervous systems", in: Albrecht, R., Reeves, C. and Steele, N. (eds) Artifical Neural Networks and Genetic Algorithms, Springer Verlag (1993)

13. Fahlman, C., and C. Lebiere, "The Cascade- Correlation Learning Architecture", in: Turetzky, D (ed) Advances in Neural Information Processing Systems, vol.2, Morgan Kaufmann, 524-532 (1990).

14. Farmer, J.D., and Sidorowitch, Predicting chaotic time series, Physical Review Letters, 59, 845

(1987)

15. Freeman, J., D. Saad, “On-line learning in radial basis function networks”, Neural Computation vol. 9, No.7 (1997).

16. French, “Semi-destructive representations and catastrophic forgetting in connectionist networks, Connection Science, 1, 365-377 (1992)

17. Fritzke, B. “A growing neural gas network learns topologies”, Advances in Neural Information Processing Systems, vol.7 (1995).

18. Fukuda, T., Y. Komata, and T. Arakawa, "Recurrent Neural Networks with Self-Adaptive GAs for Biped Locomotion Robot", In: Proceedings of the International Conference on Neural Networks ICNN'97, IEEE Press (1997)

19. Funihashi, K., On the approximate realization of continuous mappings by neural networks, Neural Networks, 2, 183-192 (1989)

20. Gaussier, T., and S. Zrehen, “A topological neural map for on-line learning: Emergence of obstacle avoidance in a mobile robot”, In: From Animals to Animats No.3, 282-290, (1994).

21. Goodman, R., C.M. Higgins, J.W. Miller, P.Smyth, "Rule-based neural networks for classification and probability estimation", Neural Computation, 14, 781-804 (1992).

22. Hashiyama, T., T. Furuhashi, Y Uchikawa,. “A Decision Making Model Using a Fuzzy Neural Network”, in: Proceedings of the 2nd International Conference on Fuzzy Logic & Neural Networks, Iizuka, Japan, 1057-1060, (1992).

23. Hassibi and Stork, “Second order derivatives for network pruning: Optimal Brain Surgeon,” in: Advances in Neural Information Processing Systems, 4, 164-171, (1992).

24. Hech-Nielsen, R. “Counter-propagation networks”, IEEE First int. conference on neural networks, San Diego, vol.2, pp.19-31 (1987)

25. Heskes, T.M., B. Kappen, “On-line learning processes in artificial neural networks”, in: Math. foundations of neural networks, Elsevier, Amsterdam, 199-233, (1993).

26. Crowder, R.S., 'Predicting the Mackey-Glass timeseries with cascade-correlation learning.' In D. Touretzky, G, Hinton, and T. Sejnowski, editors, Proc. of the 1990 Connectionist Models summer School, page 117-123, Carnegic Mellon University, 1990

27. Ishikawa, M., "Structural Learning with Forgetting", Neural Networks 9, 501-521, (1996).

28. R. Jang, “ANFIS: adaptive network-based fuzzy inference system”, IEEE Trans. on Syst.,Man, Cybernetics, 23(3), May-June, 665-685, (1993).

29. Kasabov, N. "Adaptable connectionist production systems”. Neurocomputing, 13 (2-4) 95-117, (1996).

30. Kasabov, N. “Evolving Connectionist Systems for On-line, Knowledge-based Learning: Principles and Applications, TR 99/02, Department of Information Science, University of Otago IEEE Transactions on Man, Cybernetics and Systems, submitted (1999)

31. Kasabov, N., "Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems", Fuzzy Sets and Systems 82 (2) 2-20 (1996).

32. Kasabov, N., “ECOS: A framework for evolving connectionist systems and the eco learning paradigm”, Proc. of ICONIP'98, Kitakyushu, Oct. 1998, IOS Press, 1222-1235.

33. Kasabov, N., “Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation”, in: Yamakawa, T. and G.Matsumoto (eds) Methodologies for the conception,

design and application of soft computing, World Scientific, 1998, 271-274

34. Kasabov, N., Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering, The MIT Press, CA, MA, (1996).

35. Kasabov, N., J. S Kim, M. Watts, A. Gray, “FuNN/2- A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition”, Information Sciences - Applications, 101(3-4): 155-175 (1997)

36. Kasabov, N., Woodford, B. Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for Building Adaptive, Intelligent Expert Systems, in Proc. of FUZZ-IEEE, Seoul, August 1999 (1999)

37. Kasabov,N., Watts, M "Spatial-temporal evolving fuzzy neural networks STEFuNNs and applications for adaptive phoneme recognition, TR 99/03 Department of Information Science, University of Otago (1999)

38. Kawahara, S., Saito, T. “On a novel adaptive self-organising network”, Cellular Neural Networks and Their Applications, 41-46 (1996)

39. Kim J. and Kasabov N. “HyFIS: hybrid connectionist fuzzy inference for adaptive dynamicsystems, Neural Networks, submitted (1999)

40. Kohonen, T., “The Self-Organizing Map”, Processdings of the IEEE, vol.78, N-9, pp.1464-1497, (1990)

41. Kohonen, T., Self-Organizing Maps, second edition, Springer Verlag, 1997

42. Krogh, A., and J.A. Hertz, “A simple weight decay can improve generalisation”, Advances in Neural Information Processing Systems, 4 951-957, (1992).

43. Lapedes, A.S. and R. Farber. Nonlinear signal processing using neural networks: prediction and system modeling. Technical Report LA-UR-87-2662, Los Alamos National Laboratory, Los Alamos, New Mexico, 87545, 1987.

44. Lin, C.T. and C.S. G. Lee, Neuro Fuzzy Systems, Prentice Hall (1996).

45. Maeda, M., Miyajima, H. and Murashima, S., “A self organizing neural network with creating and deleting methods, Nonlinear theory and its applications, 1, 397-400 (1996)

46. Mandziuk, J., Shastri, L. Incremental class learning approach and its applications to hand-written digit recognition, TR-98-015, International Computer Science Institute, California (1998)

47. Mitchell, M.T., "Machine Learning", MacGraw-Hill (1997)

48. Moody, J., Darken, C., Fast learning in networks of locally-tuned processing units, Neural Computation, 1, 281-294 (1989)

49. Mozer, M., and 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, 598-605 (1989).

50. Kasabov,N. "Learning fuzzy rules and approximate reasoning in fuzzy neural networks and hybrid systems", Fuzzy Sets and Systems 82 (2) 2-20 (1996).

51. Reed, R., “Pruning algorithms - a survey”, IEEE Trans. Neural Networks, 4 (5) 740-747, (1993).

52. Robins, A. and Frean, M. “Local learning algorithms for sequential learning tasks in neural networks, Journal of Advanced Computational Intelligence, vol.2, 6 (1998)

53. Rummery, G.A., and M. Niranjan, “On-line Q-learning using connectionist systems”, Cambridge University Engineering Department, CUED/F-INENG/TR 166 (1994)

54. Saad, D. (ed) On-line learning in neural networks, Cambridge University Press, 1999

55. Sankar, A., and R.J. Mammone, “Growing and Pruning Neural Tree Networks”, IEEE Trans. Comput. 42(3) 291-299 (1993).

56. Takagi, T. and Sugeno, M., Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. on Systems, Man, and Cybernetics, 15: 116-132, 1985.

57. Woldrige, M., and N. Jennings, “Intelligent agents: Theory and practice”, The Knowledge Engineering review (10) 1995.

58. Yamakawa, T., H. Kusanagi, E. Uchino and T. Miki, "A new Effective Algorithm for Neo Fuzzy Neuron Model", in: Proceedings of Fifth IFSA World Congress, 1017-1020, (1993).

59. Zadeh, L.A. Fuzzy sets. Information and control, 8: 338-353, 1965

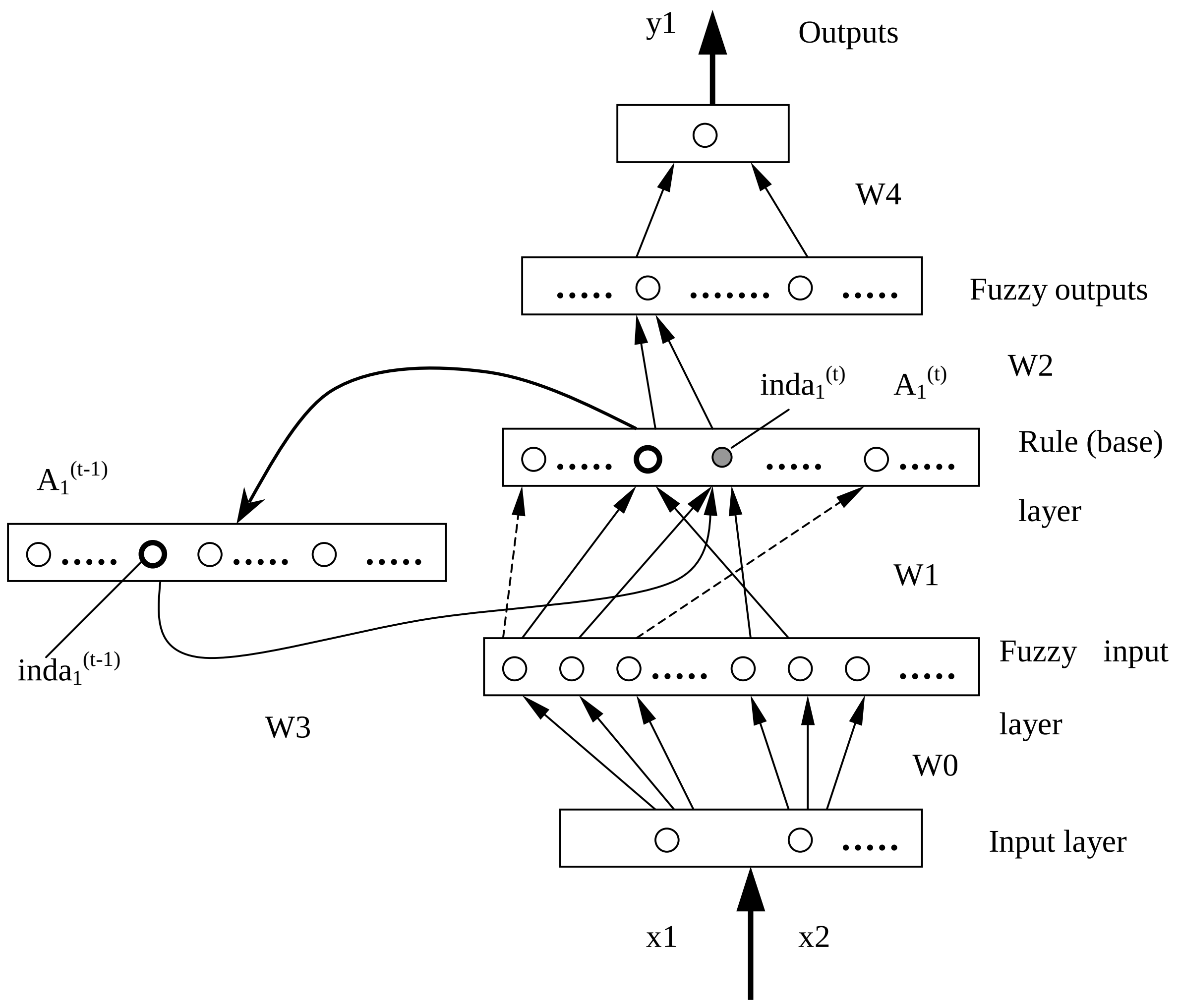


Fig. 1. The structure of EFuNNs

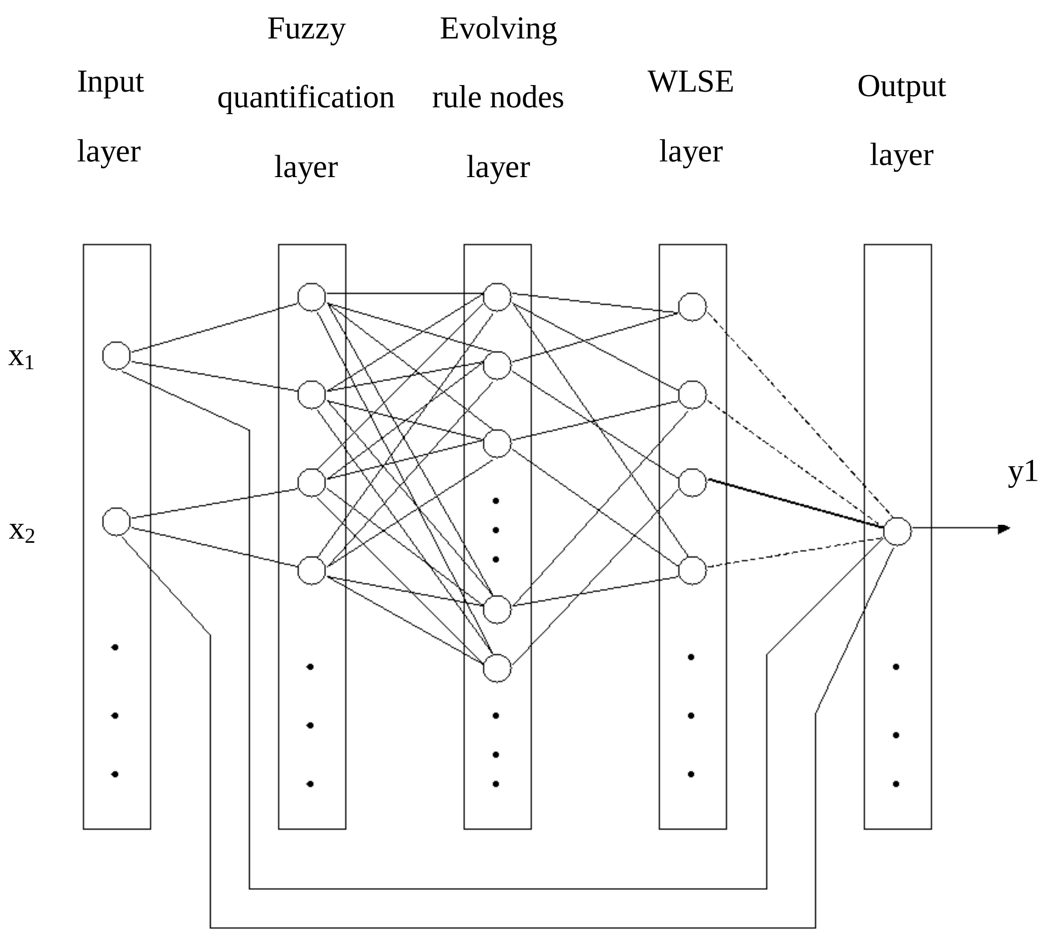


Fig. 2. The structure of mEFuNNs

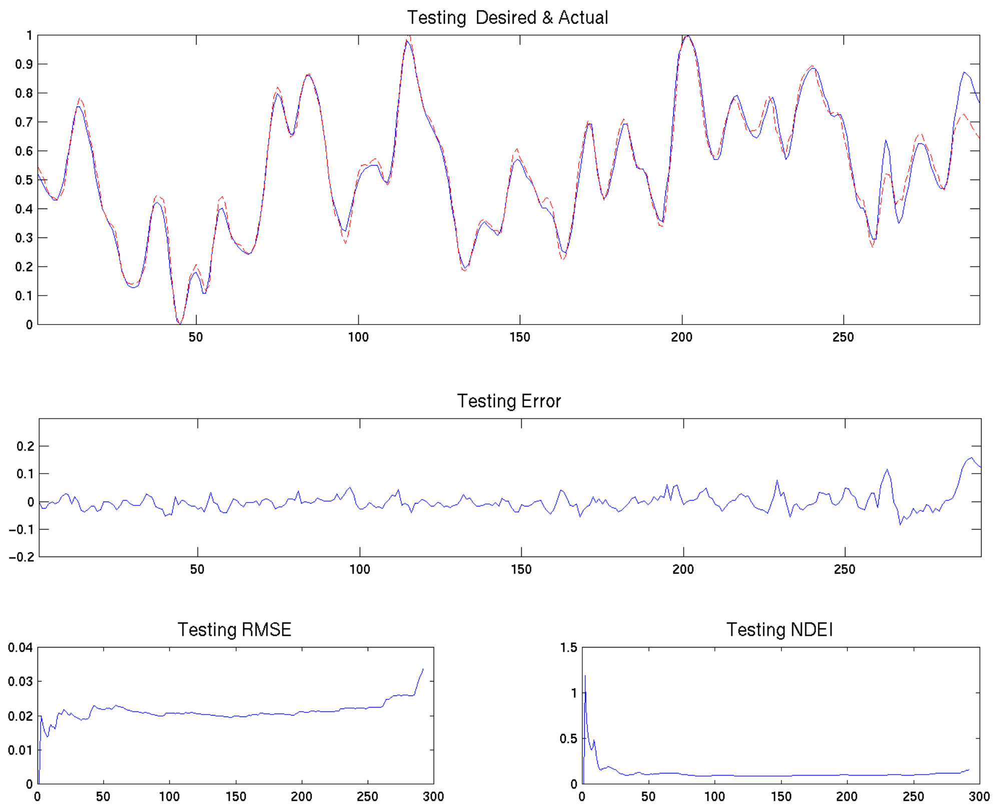


Fig. 3a. dmEFuNN off-line mode for Gas-Furnace time series prediction (t+1)

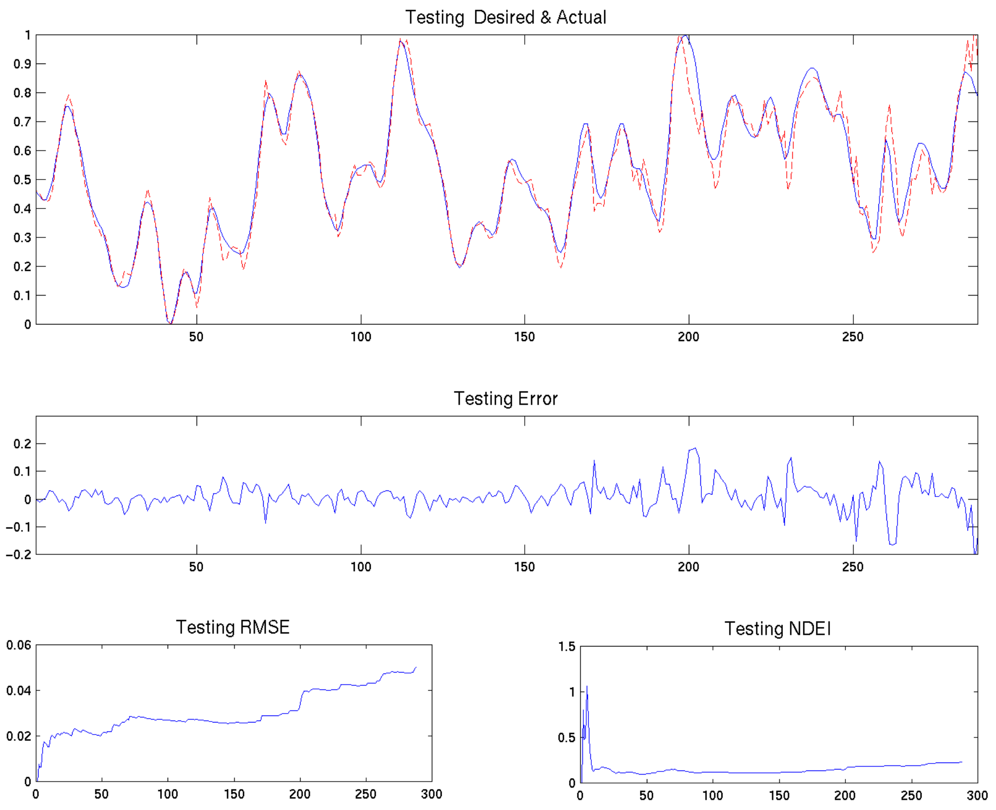


Fig. 3b. dmEFuNN off-line mode for Gas-Furnace time series prediction (t+3)

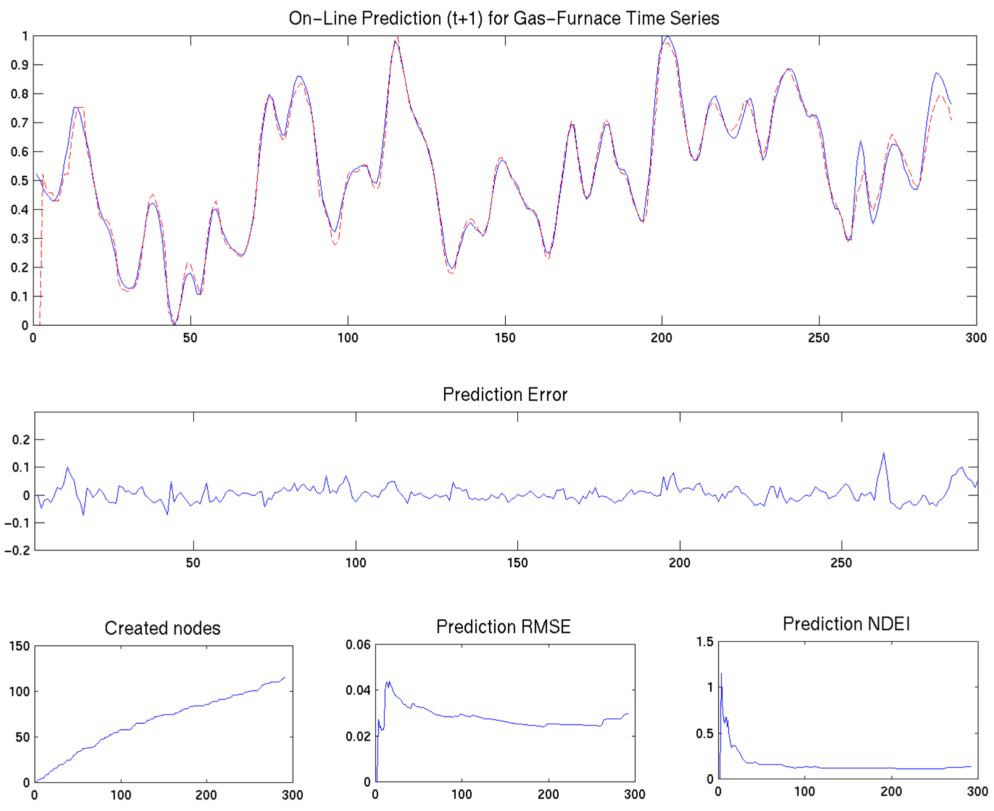


Fig. 3c. dmEFuNN on-line mode for Gas-Furnace time series prediction (t+1)

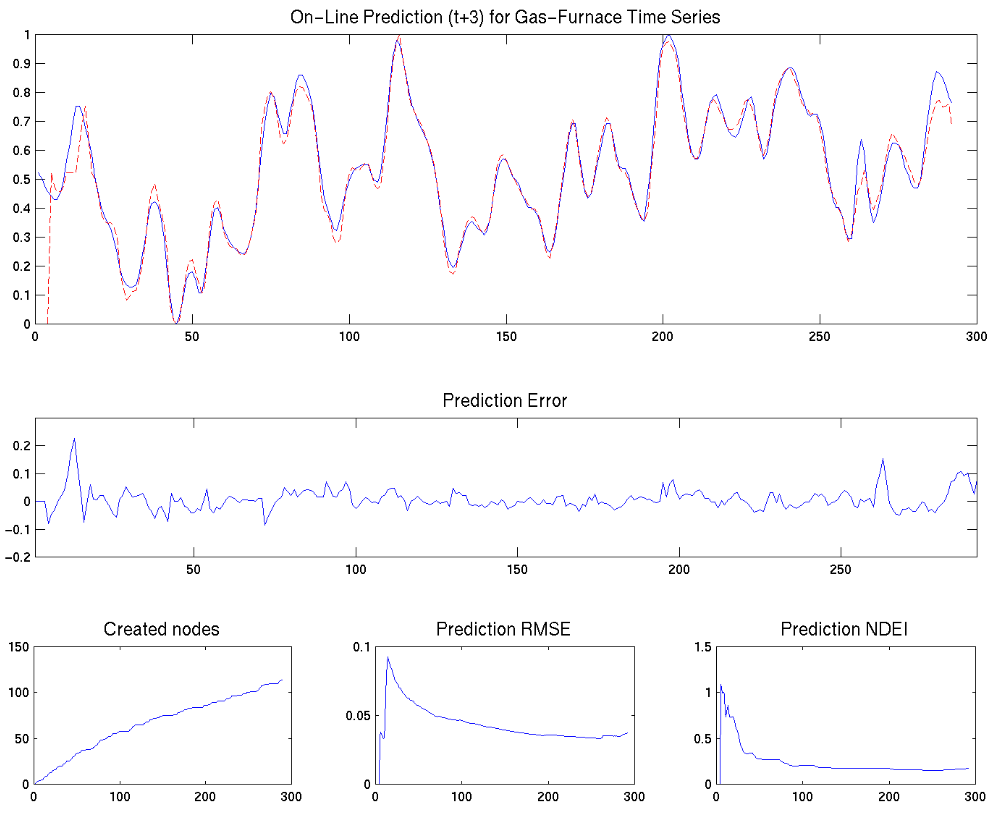


Fig. 3d. dmEFuNN on-line mode for Gas-Furnace time series prediction (t+3)

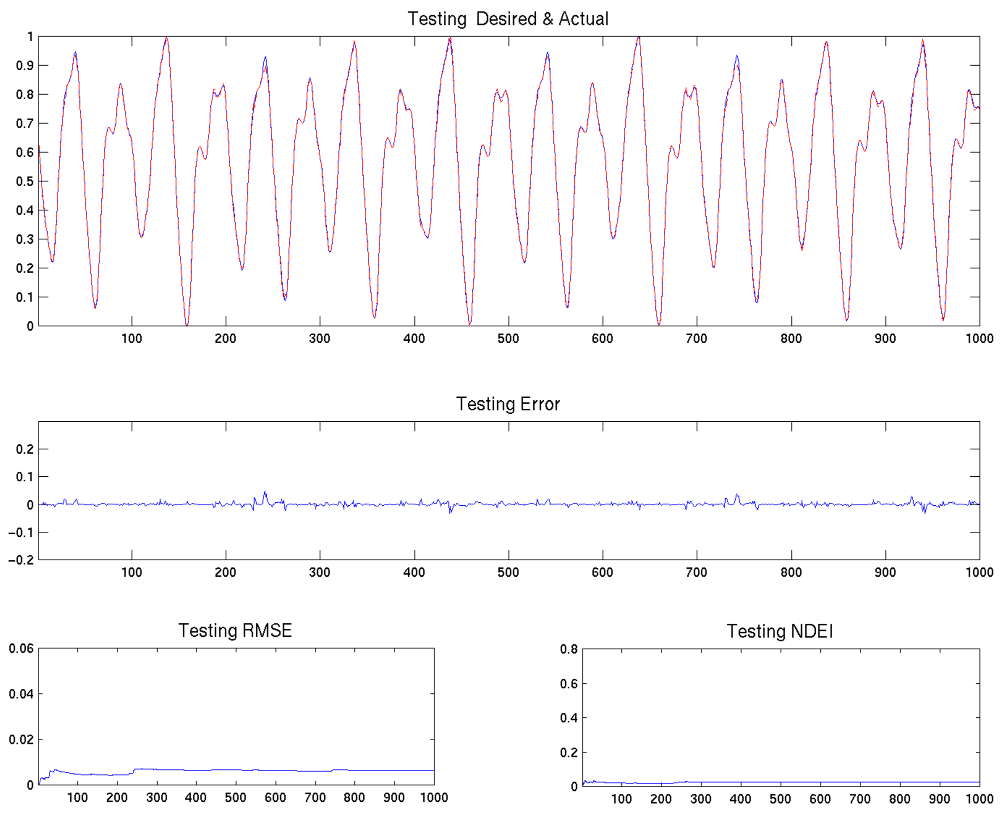


Fig.4a. dmEFuNN off-line mode for Mackey-Glass time series prediction (t+6)

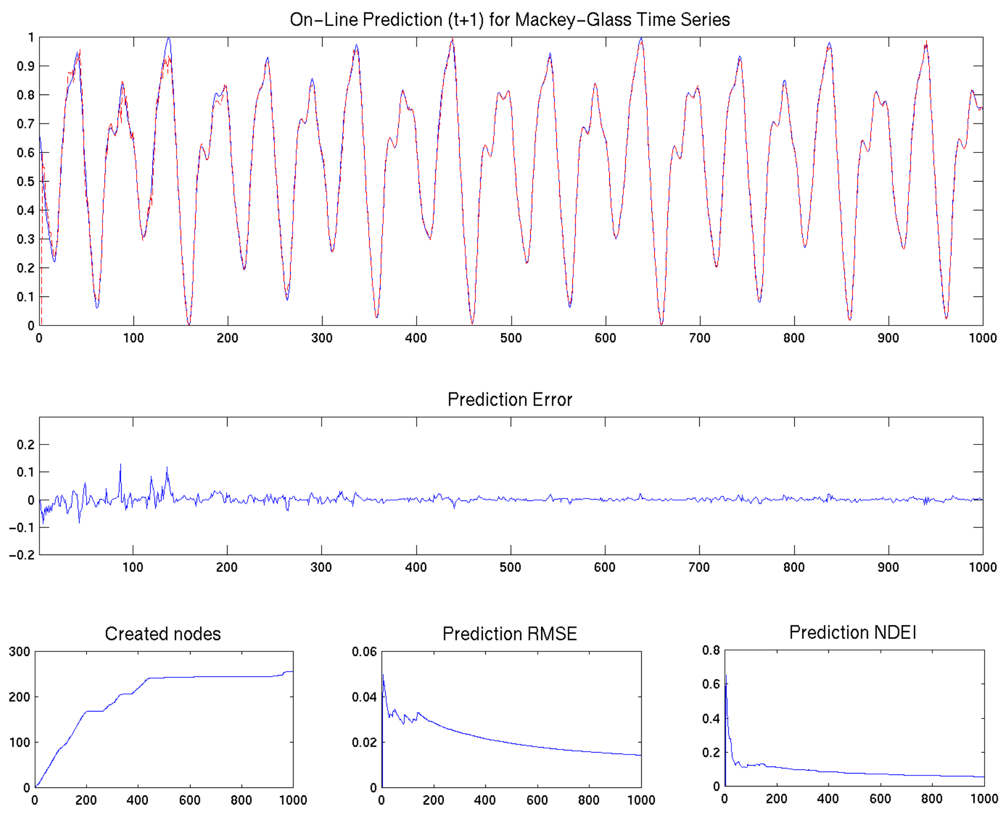


Fig. 4b. dmEFuNN on-line mode for Mackey-Glass time series prediction (t+6)

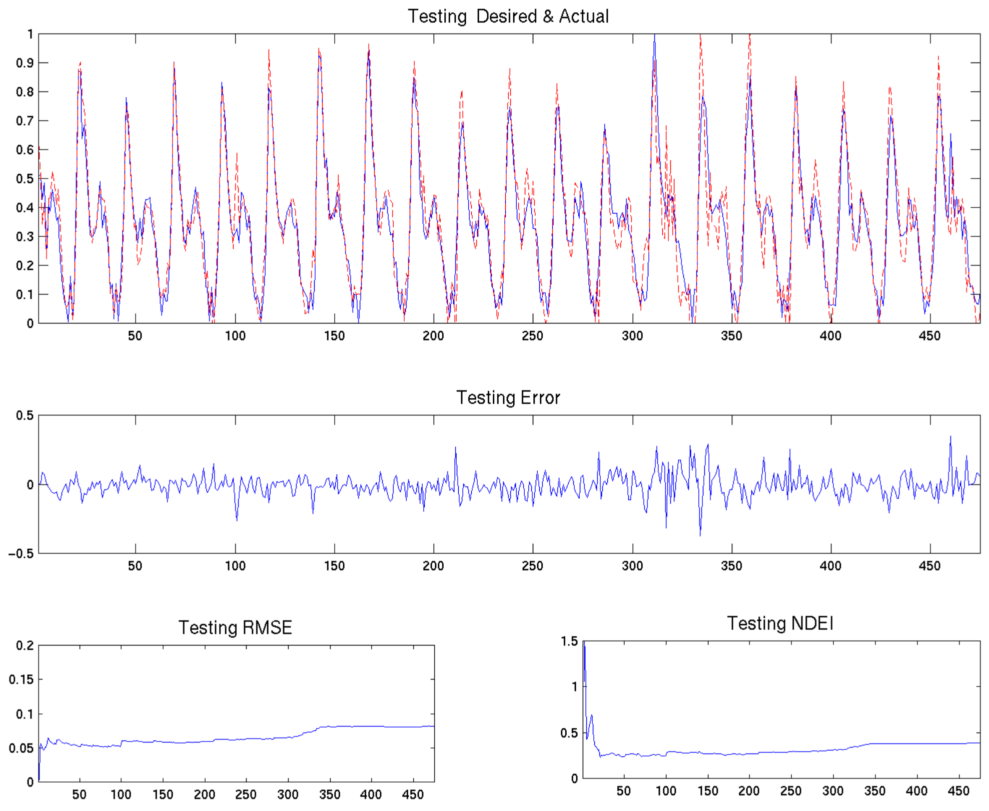


Fig. 5a. dmEFuNN off-line mode for Waste water flow time series prediction (t+1)

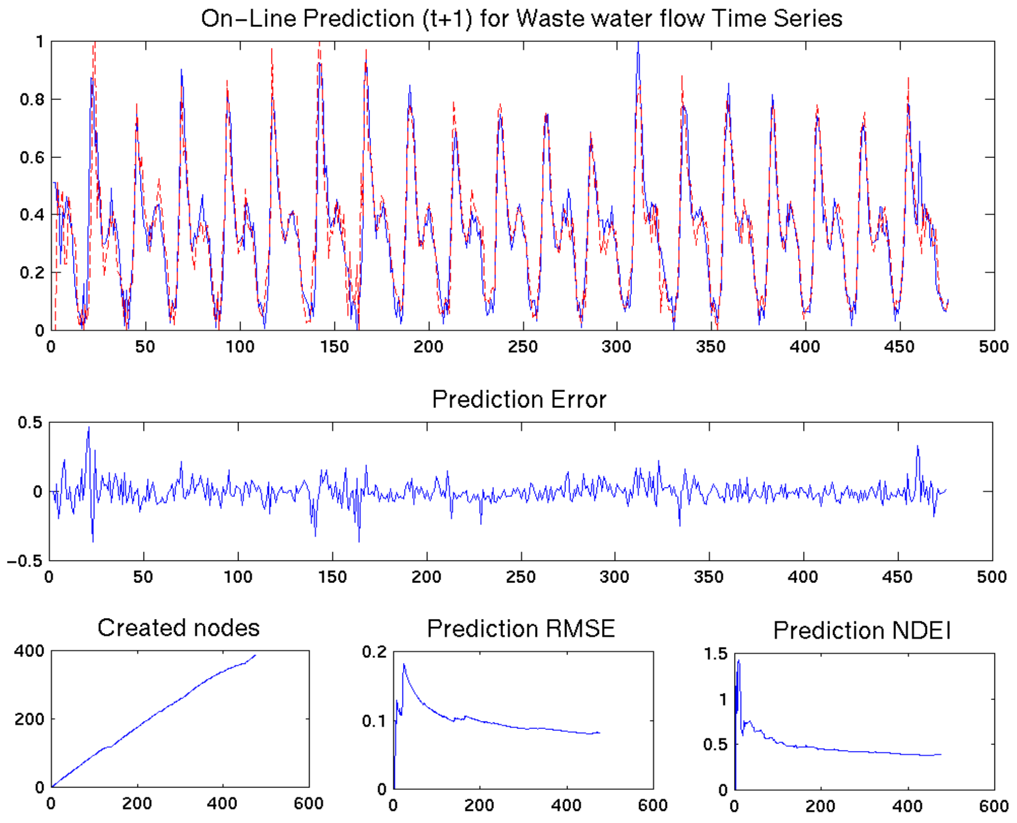
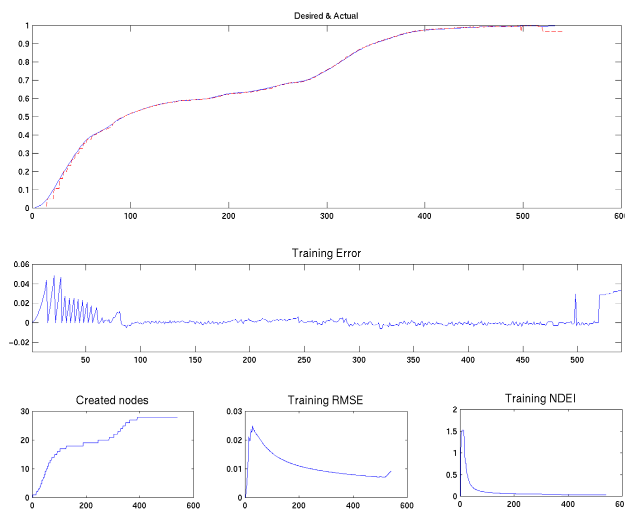
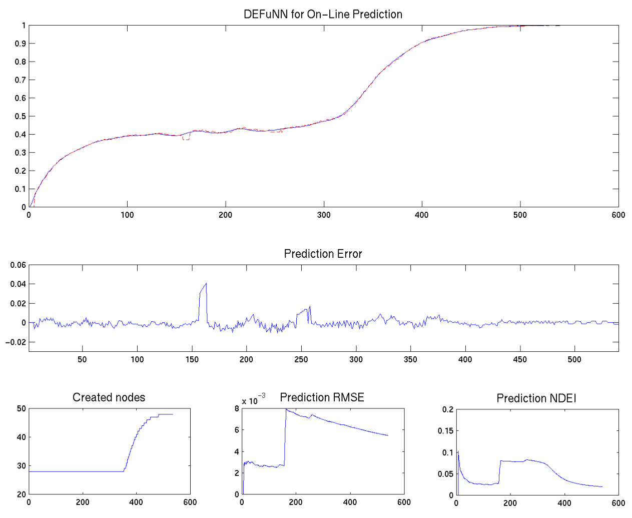


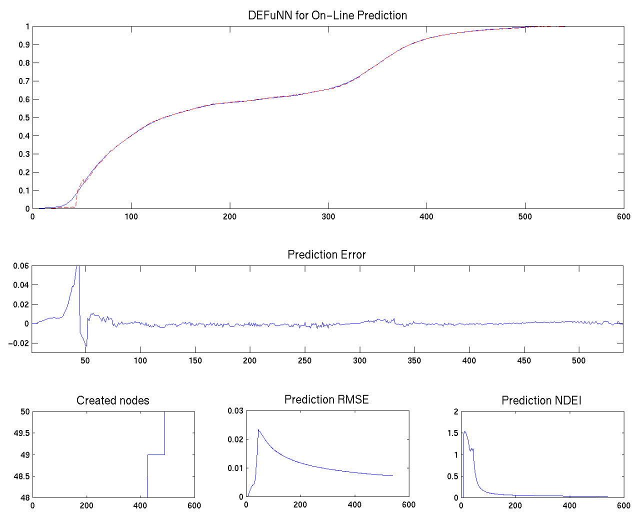
Fig. 5b. dmEFuNN on-line mode for Waste water flow time series prediction (t+1)



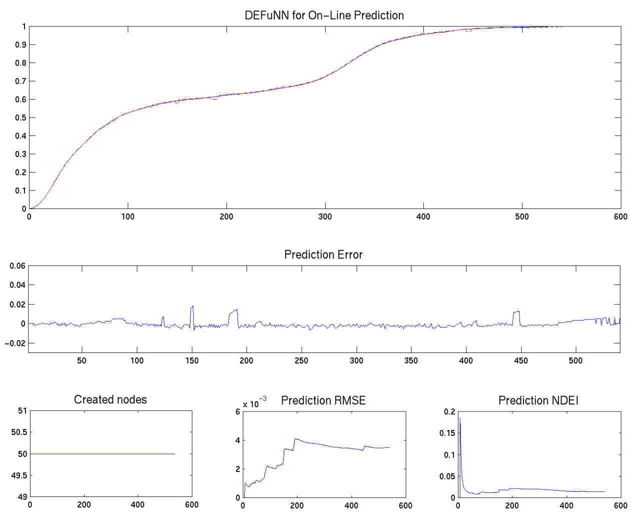
(a)



(b)



(c)



(d)

Fig. 6. dmEFuNN for DO (dissolved oxygen) on-line training and prediction

(a): initial, off-line training; (b), (c), (d): additional on-line training and testing on three consecutive data set

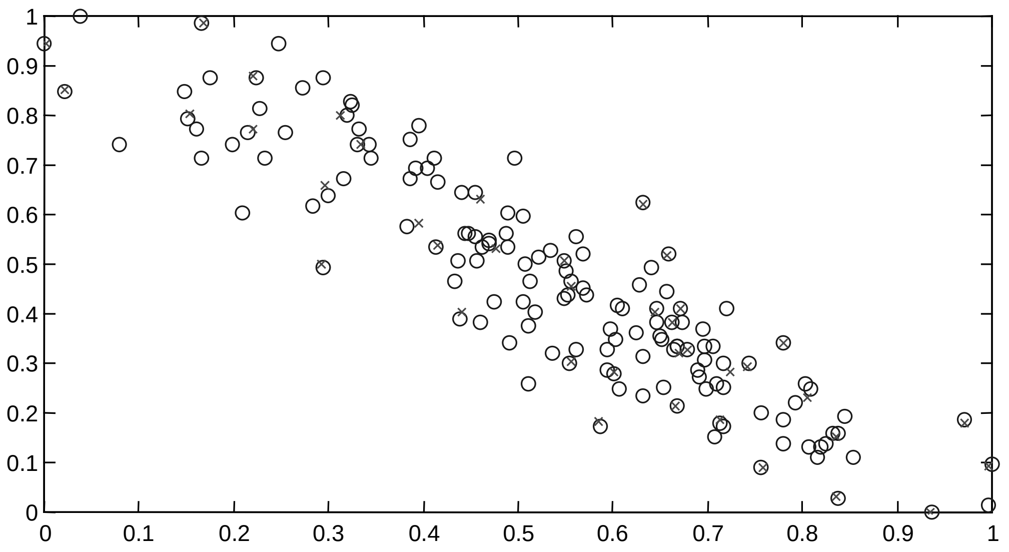

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Fig. 7 (a)

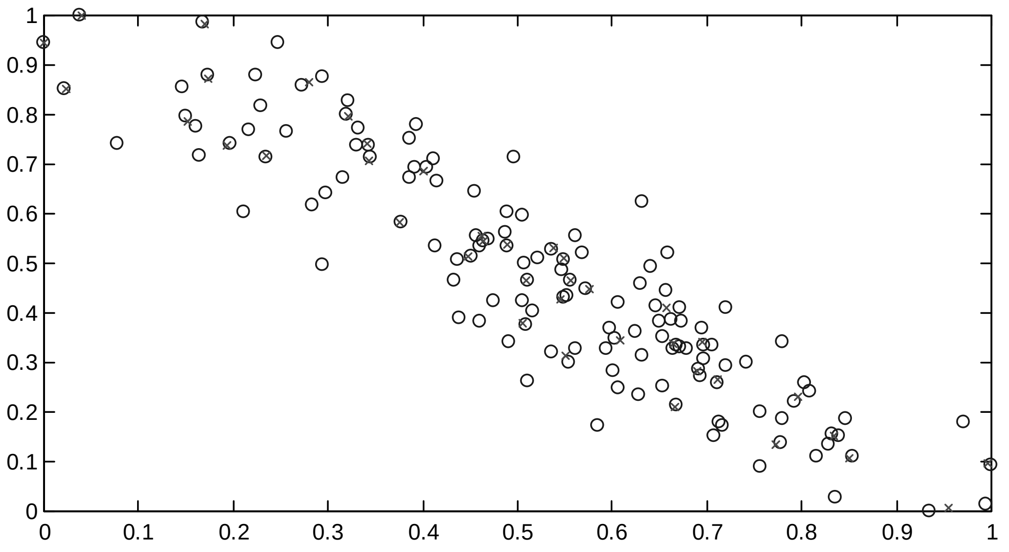

1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Fig. 7. (b)

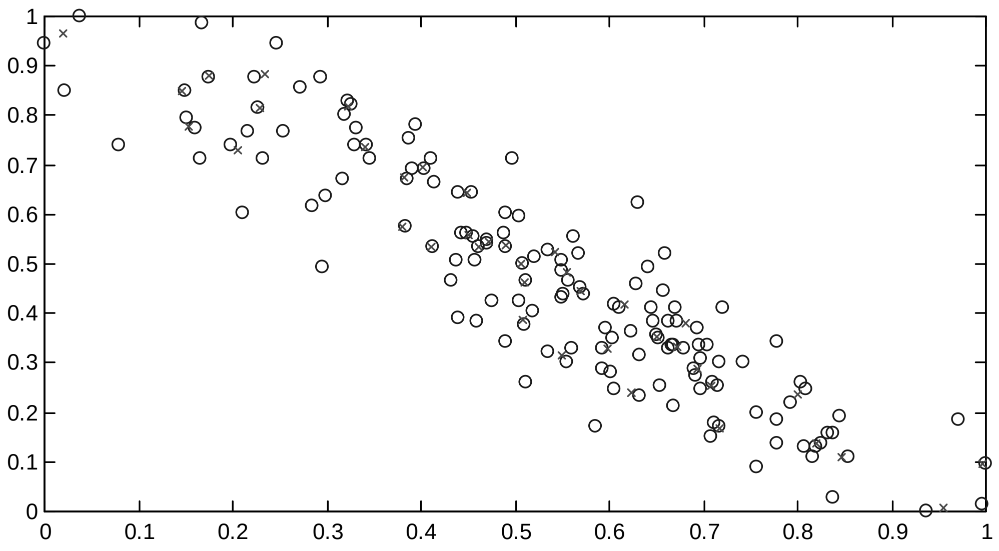


Fig. 7. (c)

Fig.7. (a) : For a chosen set of values for the parameters Sthr and Errthr 39 rule nodes are evolved from146 training examples. Mark ‘o’ denotes training examples and ‘x’ denotes created rule nodes by mEFuNN; Fig.7. (b) : Using FCMC to aggregate the rule nodes in an evolved mEFuNN from 134 to 39. Mark ‘o’ denotes rule nodes before aggregating and ‘x’ denotes rule nodes after aggregating; Fig.7. (c) : Using FCMC to create 39 rule nodes from 146 training examples. Mark ‘o’ denotes training examples and ‘x’ denotes the rule nodes.

Table. 1a. Off-line training and testing comparisons for gas-furnace data

Methods	dmEFuNN	mEFuNN	EFuNN	ANFIS	MLP
Epochs	1	1	1	150	100
CUPtime	4	4	4	5	6
Training RMSE	0.014	0.007	0.0023	0.018	0.016
Training NDEI	0.06	0.03	0.01	0.078	0.071
Testing RMSE	0.049	0.053	0.062	0.04	0.051
Testing NDEI	0.021	0.23	0.27	0.174	0.22

Table. 1b. Off-line training and testing comparisons for Mackey-Glass data

Methods	dmEFuNN	mEFuNN	EFuNN	ANFIS	MLP
Epochs	1	1	1	150	300
CPUtime	52	48	46	247	74
Training RMSE	0.0023	0.0064	0.0026	0.0054	0.0032
Training NDEI	0.0091	0.025	0.01	0.021	0.013
Testing RMSE	0.0042	0.012	0.0143	0.0046	0.0043
Testing NDEI	0.016	0.046	0.056	0.018	0.017

Table. 1c. Off-line training and testing comparisons for wastewater flow data

Methods	dmEFuNN	mEFuNN	EFuNN	ANFIS	MLP
Epochs	1	1	1	150	200
CUPtime	14	12	12	121	25
Training RMSE	0.019	0.011	0.0015	0.053	0.047
Training NDEI	0.083	0.045	0.0067	0.23	0.2
Testing RMSE	0.075	0.071	0.084	0.081	0.077
Testing NDEI	0.37	0.35	0.41	0.396	0.38

Table. 2a. On-line prediction comparisons for gas-furnace data

Methods	dmEFuNN	mEFuNN	EFuNN	Regression	Random Walk
Epochs	1	1	1	1	1
Nodes	198	233	233	N/A	N/A
RMSE	0.03	0.038	0.041	0.045	0.05
NDEI	0.138	0.177	0.191	0.211	0.232

Table. 2b. On-line prediction comparisons for Mackey-Glass data

Methods	dmEFuNN	mEFuNN	EFuNN	Regression	Random Walk
Epochs	1	1	1	1	1
Nodes	521	553	554	N/A	N/A
RMSE	0.0087	0.028	0.028	0.019	0.037
NDEI	0.034	0.111	0.108	0.073	0.146

Table. 2c. On-line prediction comparisons for wastewater flow data

Methods	dmEFuNN	mEFuNN	EFuNN	Regression	Random Walk
Epochs	1	1	1	1	1
Nodes	386	475	475	N/A	N/A
RMSE	0.076	0.09	0.095	0.103	0.117
NDEI	0.36	0.427	0.45	0.489	0.551

Table 3. (a) Changing the parameters to reduce the rule nodes

Nodes	RMSE	NDEI	Sthr	Errthr
134	0.0411	0.1898	0.95	0.04
116	0.0428	0.1978	0.92	0.08
101	0.0444	0.2054	0.9	0.1
76	0.0428	0.1976	0.85	0.12
56	0.0433	0.2003	0.8	0.15
39	0.044	0.2034	0.7	0.2

Table 3. (b) Using FCMC to aggregate the rule nodes

Nodes	RMSE	NDEI
116	0.0418	0.1933
101	0.0425	0.1965
76	0.044	0.2034
56	0.0419	0.1935
39	0.0429	0.1984

Table 3.(c) Using FCMC to create the rule nodes

Nodes	RMSE	NDEI
134	0.0425	0.1965
116	0.044	0.2036
101	0.046	0.2126
76	0.0449	0.2074
56	0.0422	0.1952
39	0.0432	0.1998