"On-Line" Time Series Prediction System------EFuNN-T

Xin Wang

Department of Information Science, University of Otago, Dunedin, New Zealand

(Email: xinw@infoscience.otago.ac.nz)

Abstract

An "on-line" time series prediction system EFuNN-T based on a model of evolving fuzzy neural network------EFuNN is presented in this paper. EFuNN, as a particular type of evolving fuzzy neural network, evolves both its structure and parameters to accommodate new coming data. EFuNN-T, as an application of EFuNN in the field of time series prediction, performs "one-step" ahead prediction in an "on-line" mode and has the functions of network refining both on structure and parameters, such as node pruning, node aggregation, parameters self-tuning, and knowledge insertion/extraction while it is performing the "on-line" operation.

1.Introduction

In the past two decades, neural networks have been extensively used for a wide range of applications in the domain of time series prediction, and they have gained great success in these fields. Among them the multi-layer perceptron (MLP), is probably the most commonly used network models in time series prediction, especially ones with back-propagation (BP) training algorithm

However most classical neural network models in time series prediction, such as multi-layer perceptron with BP training [1] and Radial Basis Function Network [2], are based on the belief that there are linear or non-linear relationships between historical data and future values in a time series system. What they usually do is regard a time series as a linear or non-linear system, and trying to identify it, or forming a function between parent and historical states and future state of the system, i.e. for a time series *Y(t),(t=1,2,...,* enumerates the elements of the sequence), there may be a relationship between future value and relative factors:

$$Y(t+1) = F[X(t), X(t-1),..., X(t-n)] \quad (1)$$

Where *Y(t+1)* is output of a system in next time step *t+1* (is also the value to be predicted in a time series prediction issue). *X(t),X(t-1)...,X(t-n)* are factors of the system that are relative to *Y*.

Some problems in these prediction models are that many real world problems cannot be expressed or described by linear or non-linear functions, and sometimes they cannot be identified at all. Another problem is that many prediction systems that take MLP or RBF Network as model have to re-train the network after a period of prediction in order to make the network fit in with current situation. In addition as time series system may have different features in different time period, i.e. they are time changing. In the process of the prediction the former network's structure and parameters may be not the best for next step prediction. Hence it is required that prediction network should modify its structure and parameters continuously in the process of prediction.

EFuNN (Evolving Fuzzy Neural Network, proposed by Kasabov [4] [5]) is a fuzzy neural network model. It uses feed forward neural network to process fuzzified data (or knowledge), then defuzzifies the fuzzy data as the output. The whole training process in EFuNN is trying to form a fuzzy hyper-reflection between input hyper-plane and output hyper-plane instead of simulating the function between the network inputs and outputs. Comparing with other models, EFuNN has several novel characteristics. First the data processed in the neural network of EFuNN is fuzzy data. Second EFuNN evolves its structure in time. All nodes on the third layer of EFuNN and the connections are created during learning. Third EFuNN could modify its parameters, such as sensitivity thresholds (which is a threshold for determining whether the outputs of rule layer is allowed to pass forward or not) and learning rates for every rule nodes. Fourth EFuNN could refine its network architecture by rule node pruning and aggregation to make it accommodate new coming data.

In this paper, a time series prediction system EFuNN-T is created according to the principle of EFuNN with features of "on-line" prediction and life-long learning. Instead of training before prediction or once a period as other neural networks usually do in order to suit new data model, EFuNN-T performs learning and prediction simultaneously without repeated training and separated training and predicting.

2.Evolving Fuzzy Neural Network (EFuNN)

Generally the EFuNN is a connectionist feed forward networks with five layers of neurons and four layers of connections as shown in Figure1 [3]

[4] [5]. The input layer represents input variables of the network as crisp value. The second layer implements fuzzification for the inputs. It represents fuzzy quantisation of input variables. The third layer contains rule nodes that evolve through learning. Evolving means all nodes on the third layer are created during learning. These nodes represent prototypes of data mapping between the fuzzy input and fuzzy output space. The fourth layer is fuzzy output layer where each node represents fuzzy quantisation of the output variables. The output variable layer makes the defuzzification for fuzzy output variables.

The training process is the process of inserting/ creating nodes in rule layer and forming or updating connections to fuzzy input layer W_1 and fuzzy output layer W_2. The rule nodes represent prototypes of input-output data associations that can be graphically represented as associations of hyper-spheres from the fuzzy input and the fuzzy output space. W_1 is adjusted via unsupervised learning based on the similarity between the fuzzy input vector and the prototypes already stored. The updating of W_2 is according LMS algorithm that minimise the fuzzy output error.

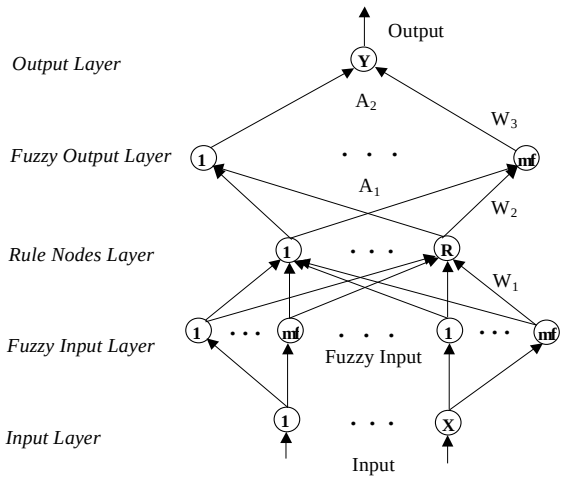

Figure 1: the structure of EFuNN

3.The Algorithm of EFuNN-T

The algorithm mainly consists of following steps: network initiation, inputs feeding forward, connections updating and parameters tuning, node aggregation and pruning, and rule extraction [6].

EFuNN-T performs prediction in an "on-line", "one pass", "one-step ahead" mode. That means after each input vector that has been fed into the network, by the error of last step prediction the network does connection updating (or node creating, if necessary), parameters tuning, node aggregating, and node pruning. Then the network propagates the signals forward, and makes the prediction. In this "on-line" mode, the network could get the "best recognition" of the current situation of the time series process and could made itself 'most appropriate' for the new coming data, then give the prediction just one-step ahead.

3.1.Network Initiation

EFuNN-T adjusts its parameters and creates all its rule nodes during training. To initiate the network, the parameters should be set by initial values, then the first training example will be used to create first node, and set the connections W_1, W_2. The way to do that is creating a new rule node 1 and set the connections for the new node as follows:

$$W_1 = E(1) \tag{2}$$

$$W_2 = T(1) \tag{3}$$

W_1 are the connections between second layer and third layer, and W_2 are the connections between third layer and fourth layer. $E(1)$, $T(1)$ are fuzzy value of first input example and its desired output. During the process of operation, $W_1=[w_1(1), w_1(2),...,w_1(n)]^T$, $W_2=[w_2(1), w_2(2),...,w_2(n)]^T$, where n is the number of nodes on third layer. $w_1(i)$ is a matrix of the connection from all nodes on second layer to the node i on third layer. $w_2(i)$ is a matrix of the connection from node i on third layer to all nodes on fourth layer. If a new rule node j is needed to accommodate data sample i, connections for the node should be created in following way:

$$w_1(j) = E(i) \tag{2'}$$

$$w_2(j) = T(i) \tag{3'}$$

3.2.Feed Forward Inputs

When an example numbered i is fed into as input, the first step of the system operation is fuzzifying the crisp value according to the number of the fuzzy membership function. Then either Euclidian distance or Fuzzy distance between the fuzzy input $E(i)$ from the second layer and the connections from all nodes on second layer to every nodes on third layer should be calculated to determine whether creating a node or not. Where Euclidian distance between number i fuzzy input $E(i)$ and $w_1(j)$ is defined as:

$$D_{E,i} = \left\| E(i) - w_1(j) \right\| / \sqrt{N} \tag{4}$$

$\|\cdot\|$ is Euclidian Normal, $j=1,2,...,n$ for every rule nodes on third layer. N is the number of column in $E(i)$. Corresponding Fuzzy distance is

$$D_{F,i} = \sum^{N} \left| E(i) - w_1(j) \right| \Big/ \sum^{N} \left[E(i) + w_1(j) \right] \tag{5}$$

Then get the activation A_1 for every nodes in rule layer:

$$A_{1,i} = 1 - D_{E/F,i} \quad (6)$$

i=1,2,...,n. Find the one or *M* rule node(s) R_j (or R_k, R_l,...) with highest activation A_{1j} (or A_{1k}, A_{ll},...) . If $A_{1j} > SThr_j$ ($SThr_j$ is Sensitivity Threshold set for activation of rule node *j*), put A_1 forward to next layer. Else create a new node and insert connection as function (2'), (3') into the connection matrix for this example. If $A_{1j} > SThr_j$, the output of fuzzy output layer (fourth layer) is:

$$A_2 = satlin\ (W_2 \cdot A_1) \quad (7)$$

$$Err = \left| A_2 - T \right| \quad (8)$$

Err is the fuzzy output error between predicted and desired value. If *Err>Ethr*, create a new rule node (*Ethr* is error threshold for fuzzy output). The crisp output can be get by:

$$O = W_3 \cdot A_2 \quad (9)$$

W_3 is the connection between fuzzy output layer and output layer that performs the defuzzification for the fuzzy value.

3.3. Updating and Tuning

After each example is processed, the connections should be updated and parameters should then be tuned according to following functions.

$$W_1^{t+1} = W_1^t + Lr \cdot (E - W_1^t) \quad (10)$$

$$W_2^{t+1} = W_2^t + Lr \cdot Err \cdot A_1 \quad (11)$$

Where *Lr* is learning rate (a matrix with different values for every node's connection updating), *Err* is the fuzzy output error of last step.

The parameters to be tuned include: sensitivity thresholds *SThr* and learning rates *Lr* for every rule nodes.

$$SThr^{t+1} = SThr^t - D_{E/F}(W_1^{t+1}, W_1^t) \quad (12)$$

$$Lr^{t+1} = 1/Ac \quad (13)$$

Where $D_{E/F}$ is either Fuzzy distance or Euclidian distance (function 4, 5) between W_1^{t+1} and W_1^t, *Ac* is the accumulated number of accommodated examples for every rule node.

3.4.Node Aggregation and Pruning

After certain step predictions have implemented (certain number of examples have been processed), node aggregation and pruning are needed. Aggregation is combining several rule nodes that are belonging to one 'class' into one node. Here 'belonging to one class' means these nodes are close each other enough in the hyperspace according to either Euclidian distance or Fuzzy distance, and they can be presented by one node.

If there are *m* rule nodes that the distances (either Fuzzy or Euclidian distance defined by function 4,5) between each other both on W_1 side ($D1_{E/F}$) and W_2 sides ($D2_{E/F}$) are all less than the thresholds ($ThrW_1$ for W_1 side and $ThrW_2$ for W_2 side) set for them.

$$D1_{E/F} < ThrW_1 \quad (14)$$

$$D2_{E/F} < ThrW_2 \quad (15)$$

The *m* nodes could be aggregated into one node with the connections W_1^{agg} and sensitivity threshold $SThr^{agg}$ as follows:

$$W_1^{agg} = \sum^m W_1 \Big/ m \quad (16)$$

$$W_2^{agg} = \sum^m W_2 \Big/ m \quad (17)$$

$$SThr^{agg} = 1 - \max\{ D(W_1^{agg}, W_1)\} \quad (18)$$

Where $max\{D(W_1^{agg},W_1)\}$ means the maximum of the distances between the node that created by aggregation and every nodes involving in the aggregation.

3.5.Rule Extraction.

Every rule node in EFuNN-T can generate a fuzzy rule from the connections between fuzzy input layer and rule layer and the connections between rule layer and fuzzy output layer. However only the connections that are over the thresholds specified for the connections are taken for extracting rules. In EFuNN-T, these Rules can be extracted into a document file (.doc file) as follows.

```
...
Rule 4:
if   [1] (2  0.482) (3  0.518)
     [2] (2  0.111) (3  0.889)
     [3] (2  0.359) (3  0.641)
     [4] (2  0.400) (3  0.600)
     [5] (2  0.444) (3  0.556)

then [1] (2  0.187) (3  0.813)
...
```

Here just takes No. 4 Rule for example, it means that there are 5 inputs and 1 output with 3 (fuzzy) membership functions (small(1), medium(2), and large(3)) in the system. The rule No.4 that extracted from the learning is:

if [1]. No.1 input is medium (2) to degree 0.482, and is great (3) to degree 0.518
[2]. No.2 input is medium (2) to degree 0.111, and is great (3) to degree 0.889
[3]. No.3 input is medium (2) to degree 0.359, and is great (3) to degree 0.641
[4]. ....

Then [1]. Output is medium (2) to degree 0.187, and is great (3) to degree 0.813.

4.GUI of EFuNN-T

Figure 2 shows the Graphical User Interface of EFuNN-T that consists of six sections:

1. **File Operation** Section. It manages the input file and output files.
2. **Parameters Input** Section. It is used for choosing parameters.
3. **Operation** Section This section controls the process of the prediction and display.
4. **Chart Display** Section. It displays a chart for plotting the results of predicted.
5. **Information** Section. All the information about the prediction and the result are given in this section.
6. **Figure Processing** Section. Some operations about the GUI figure can be done in this section.

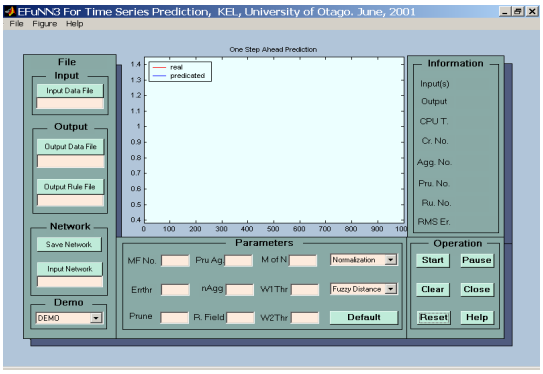

Figure 2 GUI of EFuNN-T

5.Case Studies

Here are two real world time series problems. One is the Mackey-Glass problem that is a benchmark time series task defined by the differential equation defined below:

$$\dot{x}(t) = \frac{0.2\,x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \qquad (19)$$

By the database derived from above equation, we could use EFuNN-T to simulate it. Here the method we take is predicting future values *x(t+6)* from 4 points spaced at six time interval in the past. That is taking *x(t), x(t-6), x(t-12), x(t-18)* to predict future value *x(t+6)*. Some key parameters set for the prediction are shown in Table 1. Three experiments are carried out for this dataset with both pruning and aggregation, only pruning, and only aggregation. The results about network scale, training time, and prediction RMS Error (Root Mean Square Error) also displayed in the Table1 and Figure3. It is evident that with pruning and aggregation, EFuNN-T reduces its network size that could shorten the training time comparing with the one without pruning, at the same time an increase of prediction accuracy is resulted in.

The other case is SE40 index prediction. SE40 is a time series of everyday closing index of a New Zealand share market. In this case, past 5 day's index are taken as inputs to predict just next day's index. Table2 and Figure 4 show the all the parameters chosen for the prediction and the experiment results.

6.Conclusion and Further Work

This paper introduces an evolving fuzzy neural network system EFuNN-T for time series prediction. EFuNN-T is a local learning model, which allows for fast adaptive learning and is robust to catastrophic forgetting. In each step only few connections and nodes are changed, or created after the entry of a new data item. This is in contrast to the global learning algorithm where, for each input vector, all connection weights change, thus making the system prone to catastrophic forgetting when applied for adaptive, "on-line" learning tasks. The experiments show the training parameters such as number of fuzzy membership functions, pruning or not, aggregation or not, are most relevant to the network behaviour and results of prediction.

From these real world examples, we can see EFuNN-T performs significantly in time series prediction. But it seems to be more promising for EFuNN-T that if it could deal with the current pattern

Parameters								Results		
No.	MF. No.	Errthr	Pruning	Pru. A.	nAgg.	R. Field	Mof N	Node	CPU.T	RMSE
1	5	0.1	NO		100	0.2	1	55	118s	0.03657
2	5	0.1	YES	50		0.2	1	52	32.6s	0.03508
3	5	0.1	YES	50	100	0.2	1	22	33.9	0.03356

Table 1: Parameters and results of Mackey-Glass problem prediction.

MF. No.: the fuzzy membership function number for fuzzification. **Errthr**: error threshold; **Pruning**: prune or not; **Pru. A**: the age of a node that increases one after the network processes an example; **nAgg.**: The number of example, after that an aggregation will be done; **R.Field**: maximum receptive radius; **M of N**: m greatest outputs of rule layer are taken into next layer; **Node**: number of node in rule layer of the network; **CPU. T**: CPU time taken for the training; **RMSE**: Root Mean Square Error

Parameters							Results		
MF.No	Errthr	Pruning	Pru. A.	nAgg.	R. Field	M of N	Node	CPU.T	RMSE
2	0.03	YES	50	50	0.3	1	7	26.4s	29.98

Table 2: Parameters and results of SE40 problem prediction

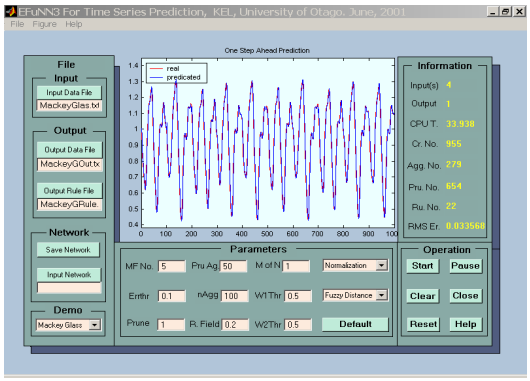

Figure 3: The prediction of Mackey-Glass problem

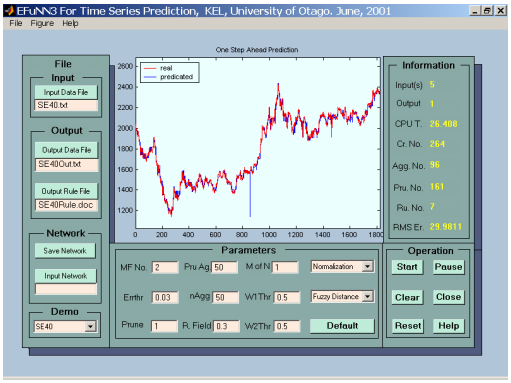

Figure 4: The prediction of SE40 problem

(next step prediction) with the memory or knowledge of history prediction experience (historical prediction errors). One measure for this object could be inserting an automaton in the network.

Automata are the machines that receive inputs take one of a number of different states that determined wholly or partly by these inputs and the states of the machine. The way to represent an automaton in EFuNN-T is adding a recurrent network in EFuNN because it has shown that recurrent network is capable of representing automaton. Quite amount research has been done about representing an automaton by first order and second order recurrent network [7] [8] [9]. If the automaton could be embedded in EFuNN, EFuNN would be more powerful in such a way that it could not only recognise current changing pattern (this is due to the function of automata), but also take EFuNN's past few steps' behaviours into account for next step prediction

Acknowledgements

This work is a part a research programme "Connectionist-based Intelligent Information System", which is funded by the New Zealand Foundation for Research Science and Technology, contract UOOX0016.

Reference

[1] D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning Internal Representation By Error Propagation," *Parallel Distributed Processing*, Cambridge, Mass, MIT Press, 1986.

[2] J. E. Moody, C. J. Darken, "Fast Learning in Networks of Locally Tuned Processing Units," *Neural Computation* Vol.1, pp. 281-294,1989.

[3] N. Kasabov, *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*, The MIT Press, 1996.

[4] N. Kasabov, "Evolving connectionist and fuzzy connectionist systems theory and applications for adaptive, on-line intelligent systems," *Neuro-Fuzzy Techniques for Intelligent Information Processing*, Heidelberg Physica Verlag, pp. 111-146, 1999.

[5] N. Kasabov, "ECOS: A Framework for Evolving Connectionist System and the ECO Learning Paradigm," *Proc. Of Int. Conference of Neural Information Processing (ICONIP'98)*, pp. 1222-1235, 1998.

[6] N. Kasabov, "Evolving Fuzzy Neural Network for Supervised/Unsupervised On-line, Knowledge-based Learning," *IEEE Trans. on Man, Machine and Cybernetics*, (in press), 2001.

[7] R. Alquezar, A. Sanfeliu, "An algebraic Framework to Represent Finite State Machines in Single-layer Recurrent Neural Networks," *Neural Computation*, Vol. 7(5): pp. 931-949, 1995.

[8] C. W. Omlin, C. L. Giles, "Constructing Deterministic Finite-state Automata in Recurrent Neural Networks," *Journal of the ACM*, Vol. 43(6), pp. 937-972, 1996.

[9] C. W. Omlin, K. K. Thornber, C. L. Giles, "Fuzzy Finite-state Automata can be Deterministically Encoded into Recurrent Neural Networks," *IEEE Transactions on Fuzzy Systems*, Vol. 6(1): pp. 76-89, 1998.