Neuro-Fuzzy Kolmogorov's Network with a Modified Perceptron Learning Rule for Classification Problems

Vitaliy Kolodyazhniy, Yevgeniy Bodyanskiy, Valeriya Poyedyntseva, and Andreas Stephan

Summary. A novel *Neuro-Fuzzy Kolmogorov's Network* (NFKN) is considered. The NFKN is based on the famous Kolmogorov's superposition theorem (KST) and is the development of the previously proposed Fuzzy Kolmogorov's Network (FKN). Modifications of the FKN architecture include multiple outputs as required for classification problems with more than two classes, as well as the possibility of defining different number of membership functions at each input. A new learning algorithm, based on the modified perceptron learning rule and designed for classification problems, is proposed. The validity of theoretical results and the advantages of the new NFKN are confirmed by experiments in data classification and visualization.

Key words: Kolmogorov's superposition theorem, Neo-fuzzy neuron, Membership function, Fuzzy inference, Classification, Perception learning rule, Batch training.

1 Introduction

A universal approximator called *Fuzzy Kolmogorov's Network* (FKN) with simple structure based on the Kolmogorov's Superposition Theorem [5], and its training procedure with high rate of convergence were proposed in [6, 7]. It was demonstrated that the FKN can be successfully used for time series prediction problems, such as the Mackey–Glass time series prediction and electric load forecasting, as well as for data classification like separation of two intertwined spirals and solving the XOR problem. However, the FKN training algorithm may require a large number of computations in the problems of high dimension, because it is based on the least squares technique, which requires inversion of matrices. This problem is alleviated in the *Neuro-Fuzzy Kolmogorov's Network* (NFKN) [1, 8], which is trained with a hybrid algorithm, where the least squares method is used only for the output layer, and the hidden layer is trained with a gradient descent-based procedure.

Although the FKN [6, 7] and especially the NFKN with the hybrid algorithm [1,8] demonstrated very promising results in classification problems (e.g. the NFKN was demonstrated to solve the N-parity problem for N=18 after only two training epochs in [8]), their training algorithms are based on the quadratic error function and thus are better suited for regression rather than for classification problems, because minimization of the sum of squared errors does not necessarily lead to the reduction in the number of misclassifications.

In this paper, we propose an efficient and computationally simple learning algorithm, whose complexity depends linearly on both the dimension of the input space and the number of neurons. The proposed algorithm is a batch multioutput modification of the perceptron learning rule [9] with improved convergence, and is designed for classification problems. We use the modified NFKN architecture [8], which is suitable for classification problems with large number of inputs and two or more classes. The efficiency of the new algorithm is confirmed by experiments, in which improved accuracy and the reduction in network size are achieved.

2 Network Architecture

The original FKN architecture [6, 7] is comprised of two layers of neo-fuzzy neurons (NFNs) [11] and is described by the following equations:

$$\hat{f}(x_1, \ldots, x_d) = \sum_{l=1}^{n} f_l^{[2]}(o^{[1,l]}), \quad o^{[1,l]} = \sum_{i=1}^{d} f_i^{[1,l]}(x_i), \quad l = 1, \ldots, n, \tag{1}$$

where n is the number of hidden layer neurons, $f_l^{[2]}(o^{[1,l]})$ is the lth nonlinear synapse in the output layer, $o^{[1,l]}$ is the output of the lth NFN in the hidden layer, $f_i^{[1,l]}(x_i)$ is the ith nonlinear synapse of the lth NFN in the hidden layer.

The universality of internal functions in the KST [4] suggests that we can introduce an extended version of the FKN, called NFKN, with Q outputs [8], having the same hidden layer for all the output neurons:

$$\hat{f}_q(x_1, \ldots, x_d) = \sum_{l=1}^{n} f_l^{[2,q]}(o^{[1,l]}), \quad o^{[1,l]} = \sum_{i=1}^{d} f_i^{[1,l]}(x_i), \tag{2}$$
$$l = 1, \ldots, n, q = 1, \ldots, Q,$$

where Q is the number of output layer neurons, $f_l^{[2,q]}(o^{[1,l]})$ is the lth nonlinear synapse of the qth NFN in the output layer.

The equations for the hidden and output layer synapses are

$$f_i^{[1,l]}(x_i) = \sum_{h=1}^{m_{1,i}} \mu_{i,h}^{[1]}(x_i) w_{i,h}^{[1,l]}, \quad f_l^{[2,q]}(o^{[1,l]}) = \sum_{j=1}^{m_{2,l}} \mu_{l,j}^{[2]}(o^{[1,l]}) w_{l,j}^{[2,q]}, \tag{3}$$
$$l = 1, \ldots, n, i = 1, \ldots, d, q = 1, \ldots, Q,$$

where $m_{1,i}$ and $m_{2,l}$ are the number of membership functions (MFs) per input in the hidden and output layers, respectively, $\mu_{i,h}^{[1]}(x_i)$ and $\mu_{l,j}^{[2]}(o^{[1,l]})$ are the

MFs, $w_{i,h}^{[1,l]}$ and $w_{l,j}^{[2,q]}$ are the tunable weights. We assume that the MFs are fixed, triangular (piecewise-linear), and equidistantly spaced over the range of each NFN input. The parameters (centers) of the MFs are not tuned.

As in the FKN, the MFs in the NFKN at each input in the hidden and output layers are shared between all neurons. However, in the NFKN architecture we allow for different number of MFs at each input. This property is essential for the processing of data sets with mixed numerical and categorical inputs, such that each category value of a categorical input corresponds to one MF and is encoded with a numerical value corresponding to the center of that MF. This is a more parsimonious and convenient approach than conventional binary coding of categories, because we do not have to introduce additional inputs to the classifier. When a missing input is encountered, no membership function for that input is activated, and the corresponding input synapse produces zero value. The NFKN architecture is shown in Fig. 1.

The outputs of the NFKN are computed via the following two-stage fuzzy inference procedure:

$$\hat{y}_q = \sum_{l=1}^{n} \sum_{j=1}^{m_{2,l}} \mu_{l,j}^{[2]} \left[\sum_{i=1}^{d} \sum_{h=1}^{m_{1,i}} \mu_{i,h}^{[1]}(x_i) w_{i,h}^{[1,l]} \right] w_{l,j}^{[2,q]}, \quad q = 1, \ldots, Q. \tag{4}$$

The description 4 corresponds to the following two-level fuzzy rule base:

$$\begin{gathered} \text{IF} x_i \text{ IS } X_{i,h} \text{ THEN } o^{[1,1]} = w_{i,h}^{[1,1]} d \text{ AND} \ldots \text{AND } o^{1,n} = w_{i,h}^{[1,n]} d, \\ i = 1, \ldots, d, h = 1, \ldots, m_{1,i}, \end{gathered} \tag{5}$$

$$\begin{gathered} \text{IF } o^{[1,l]} \text{ IS } O_{l,j} \text{ THEN } \hat{y}_1 = w_{l,j}^{[2,1]} n \text{ AND} \ldots \text{AND } \hat{y}_Q = w_{l,j}^{[2,Q]} n, \\ l = 1, \ldots, n, j = 1, \ldots, m_{2,l}, \end{gathered} \tag{6}$$

where $X_{i,h}$ and $O_{l,j}$ are the antecedent fuzzy sets in the first and second level rules, respectively.

Total number of rules is

$$N_R^{NFKN} = \sum_{i=1}^{d} m_{1,i} + \sum_{l=1}^{n} m_{2,l}, \tag{7}$$

i.e., it depends *linearly* on the number of inputs d. Straightforward grid-partitioning approach would produce $\prod_{i=1}^{d} m_{1,i}$ fuzzy rules, leading to combinatorial explosion and being practically not feasible for $d > 4$.

3 Learning Algorithm

The number of tunable weights in an NFKN is $S = S_1 + S_2$, where $S_1 = \sum_{i=1}^{d} m_{1,i} \cdot n$ is the number of parameters in the hidden layer, and $S_2 = \sum_{l=1}^{n} m_{2,l} \cdot Q$

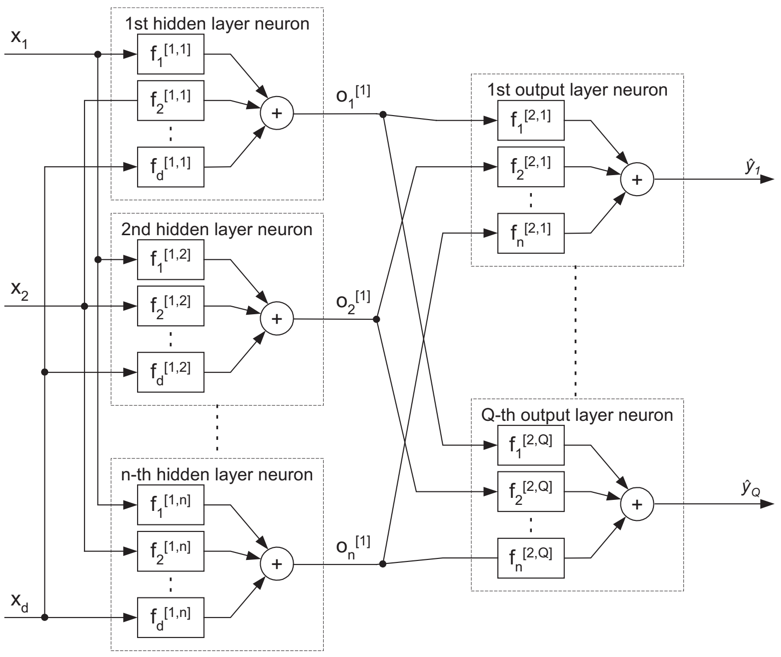

Fig. 1. NFKN with d inputs, n hidden layer neurons, and Q output layer neurons

is the number of parameters in the output layer. The weights of the NFKN here are determined by means of a batch-training algorithm as described below.

A training set containing N samples is used. The minimized error function is a batch Q-output modification of the function investigated in [9]:

$$E(t) = \sum_{q=1}^{Q}\sum_{k=1}^{N}\left[\,|\hat{y}_q(t,k)| - y_q(k)\hat{y}_q(t,k)\right] \tag{8}$$

$$= \sum_{q=1}^{Q}\sum_{k=1}^{N}\left[(sign\,\hat{y}_q(t,k) - y_q(k))\hat{y}_q(t,k)\right] = Tr\left[(sign\hat{Y}(t) - Y)^T\hat{Y}(t)\right]\,,$$

$$Y = (Y_1, Y_2, \ldots, Y_Q), \quad \hat{Y}(t) = (\hat{Y}_1(t), \hat{Y}_2(t), \ldots, \hat{Y}_Q(t)),$$

$$Y_q = \left[y_q(1), \ldots, y_q(N)\right]^T, \quad \hat{Y}_q(t) = \left[\hat{y}_q(t,1), \ldots, \hat{y}_q(t,N)\right]^T, \quad q = 1, \ldots, Q,$$

where Y is the matrix $(N \times Q)$ of target values, $\hat{Y}(t)$ is the matrix $(N \times Q)$ of network outputs at epoch t. Target values in Y are encoded as -1 or $+1$.

It is only the sign of the output rather than the output value itself that is important for the determination of classification error, and this is explicitly reflected in (8). So the error function (8) would be a potentially better choice for classification problems compared to the sum of squared errors as in [6, 7] or [1, 8].

The error function (8) can be minimized through recursive gradient-based optimization of the synaptic weights. To derive such a procedure for the output layer in a compact matrix–vector notation, re–write (4) as follows:

$$\hat{y} = W^{[2]^T} \varphi^{[2]}(o^{[1]}), \quad W^{[2]} = \begin{bmatrix} w_{1,1}^{[2,1]}, w_{1,2}^{[2,1]}, \ldots, w_{n,m_{2,n}}^{[2,1]} \\ \vdots \\ w_{1,1}^{[2,Q]}, w_{1,2}^{[2,Q]}, \ldots, w_{n,m_{2,n}}^{[2,Q]} \end{bmatrix}^T, \tag{9}$$
$$\varphi^{[2]}(o^{[1]}) = \left[\mu_{1,1}^{[2]}(o^{[1,1]}), \mu_{1,2}^{[2]}(o^{[1,1]}), \ldots, \mu_{n,m_{2,n}}^{[2]}(o^{[1,n]})\right]^T,$$

where $W^{[2]}$ is the matrix $(\sum_{l=1}^{n} m_{2,l} \times Q)$ of the output layer weights.

The update procedure for the output layer weights will be

$$W^{[2]}(t+1) = W^{[2]}(t) - \gamma \frac{\partial E(t)/\partial W^{[2]}}{\left\|\Phi^{[2]}\right\|} = W^{[2]}(t) + \gamma \frac{\Phi^{[2]^T}(Y - sign\hat{Y}(t))}{\left\|\Phi^{[2]}\right\|},$$
$$\Phi^{[2]} = \left[\varphi^{[2]}(o^{[1]}(1)), \ldots, \varphi^{[2]}(o^{[1]}(N))\right]^T, \left\|\Phi^{[2]}\right\| = \sqrt{\sum_{i=1}^{N} \sum_{j=1}^{\sum_{l=1}^{n} m_{2,l}} (\Phi^{[2]})_{i,j}^2}, \tag{10}$$

where γ is the learning rate, and $\Phi^{[2]}$ is the regressor matrix $(N \times \sum_{l=1}^{n} m_{2,l})$ for the linear output layer. The norm $\left\|\Phi^{[2]}\right\|$ in the denominator of (10) is present to speed up convergence.

We can derive a similar gradient descent-based learning rule for the hidden layer as well. To do this, let us introduce the vector $(S_1 \times 1)$ of the hidden layer weights $W^{[1]} = \left[w_{1,1}^{[1,1]}, w_{1,2}^{[1,1]}, \ldots, w_{d,m_{1,d}}^{[1,1]}, \ldots, w_{d,m_{1,d}}^{[1,n]}\right]^T$ and the Jacobian matrix $\Phi^{[1]} = \left[\varphi^{[1]}(x(1),1), \ldots, \varphi^{[1]}(x(N),1), \ldots, \varphi^{[1]}(x(1),Q), \ldots, \varphi^{[1]}(x(N),Q)\right]^T$ of size $(NQ \times S_1)$, where

$$\Phi^{[1]} = \left[\varphi^{[1]}(x(1),1), \ldots, \varphi^{[1]}(x(N),1), \ldots, \varphi^{[1]}(x(1),Q), \ldots, \varphi^{[1]}(x(N),Q)\right]^T,$$
$$\varphi^{[1]}(x,q) = \left[\varphi_{1,1}^{[1,1]}(x_1,q), \varphi_{1,2}^{[1,1]}(x_1,q), \ldots, \varphi_{d,m_{1,d}}^{[1,1]}(x_d,q), \ldots, \varphi_{d,m_{1,d}}^{[1,n]}(x_d,q)\right]^T,$$
$$\varphi_{i,h}^{[1,l]}(x_i,q) = a_l^{[2,q]}(o^{[1,l]}) \cdot \mu_{i,h}^{[1]}(x_i), i = 1, \ldots, d,$$
$$h = 1, \ldots, m_{1,i}, \ l = 1, \ldots, n, \ q = 1, \ldots, Q, \tag{11}$$

and $a_l^{[2,q]}(o^{[1,l]})$ are determined as in [8]:

$$a_l^{[2,q]}(o^{[1,l]}) = \frac{w_{l,p+1}^{[2,q]} - w_{l,p}^{[2,q]}}{c_{l,p+1}^{[2]} - c_{l,p}^{[2]}}, \tag{12}$$

where $w_{l,p}^{[2,q]}$ and $c_{l,p}^{[2]}$ are the weight and center of the pth MF in the lth synapse of the qth output layer neuron, respectively. The MFs in an NFN are always chosen such that only two adjacent MFs p and p+1 fire at a time [11].

Now we can obtain the gradient-based update procedure for the hidden layer weights:

$$\begin{gathered} W^{[1]}(t+1) = W^{[1]}(t) - \gamma \frac{\nabla_{W^{[1]}} E(t)}{\left\|\Phi^{[1]}\right\|} = W^{[1]}(t) + \gamma \frac{\Phi^{[1]^T}(\overrightarrow{Y} - sign\overrightarrow{\hat{Y}}(t))}{\left\|\Phi^{[1]}\right\|}, \\ \overrightarrow{Y} = (Y_1^T, Y_2^T, \ldots, Y_Q^T)^T, \quad \overrightarrow{\hat{Y}}(t) = (\hat{Y}_1^T(t), \hat{Y}_2^T(t), \ldots, \hat{Y}_Q^T(t))^T, \\ \left\|\Phi^{[1]}\right\| = \sqrt{\sum_{i=1}^{NQ} \sum_{j=1}^{S_1} (\Phi^{[1]})_{i,j}^2}, \end{gathered} \tag{13}$$

where $\overrightarrow{Y}$ and $\overrightarrow{\hat{Y}}(t)$ are vectors $(NQ \times 1)$.

Since no matrix inversions are involved, the considered training algorithm is less computationally intensive that both the original algorithm for the FKN [6, 7] and its hybrid modifications [1, 8]. The number of computations in (10) for both layers at each epoch depends linearly on both the dimension of the input space d and the number of neurons n.

For the processing of very large data sets when the storage of matrices for the complete data set is impossible because of memory limitations, the most memory-consuming calculations of the gradient $\nabla_{W^{[1]}} E(t) = -\Phi^{[1]} T(\overrightarrow{Y} - sign\overrightarrow{\hat{Y}}(t))$ and the norm $\left\|\Phi^{[1]}\right\|$ can be performed cumulatively sample by sample.

4 Experiments

To verify the theoretical results and compare the performance of the proposed network to the known approaches, we carried out experiments using the data from the well-known UCI repository [10]: Iris, Wisconsin Breast Cancer, Australian Credit, and German Credit. The parameters of the data sets are listed in Table 1. Note that two data sets, "Australian" and "German", have several categorical inputs, and the "Wisconsin" data set has 16 samples with missing values.

Table 1. Data sets used in experiments

Data set	Number of samples	Samples with missing values	Numerical attributes	Categorical attributes	Classes
Iris	150	0	4	0	3
Wisconsin	699	16	9	0	2
Australian	690	0	6	8	2
German	1000	0	7	13	2

The results of the experiments with the NFKN and the proposed training algorithm are summarized in Table 2. The column "neurons" describes the NFKN architectures: the numbers separated by "+" indicate the number of the hidden and output neurons, respectively. The column "weights" shows the number of tunable parameters. The next column shows the average number of epochs required for the learning algorithm to converge. The last two columns show the classification error rates. All the results in Table 2 were obtained with the learning rate $\gamma = 0.05$ in the procedures (10) and (13).

The most important advantages of the NFKN classifier are its simple architecture, which is not affected by the curse of dimensionality, and fast training procedures providing at the same time high accuracy of classification. All the results are at the level of accuracy achieved with the best classification techniques, e.g. the support vector machines [2]. For comparison, the best results obtained with the NFKN and the hybrid training algorithm from [8] are shown in Table 3.

Note that for the "Wisconsin" and "Iris" data the accuracy on the checking set in Tables 2 and 3 is the same, and for the "Australian" and "German" data sets the new training algorithm yields better results, which can be seen from the comparison of checking set errors in Tables 2 and 3. In addition, for the "Wisconsin" and "German" data best results are achieved with fewer neurons and weights.

Since all the best results from Table 2 are achieved with only two neurons in the hidden layer, visualization of multidimensional data in the two-dimensional space, formed by the outputs of the hidden layer neurons, is possible (an example is shown in Fig. 2).

Table 2. Results of experiments for NFKN with the proposed modified perceptron learning rule (tenfold cross-validation)

Data set	Neurons	Weights	Epochs	Training set errors(%)	Checking set errors(%)
Iris	2+3	54	9.9	1.33	4
Wisconsin	2+1	78	9.1	2.23	3.01
Australian	2+1	116	10.8	11.18	13.48
German	2+1	160	11.6	21.04	24.1

Table 3. Results of experiments for NFKN with the hybrid training algorithm (tenfold cross-validation)

Data set	Neurons	Weights	Epochs	Training set errors(%)	Checking set errors(%)
Iris	2+3	54	4	1.04	4
Wisconsin	4+1	176	11.9	0.54	3.01
Australian	2+1	116	11.7	10.42	14.2
German	3+1	240	29	14.32	24.8

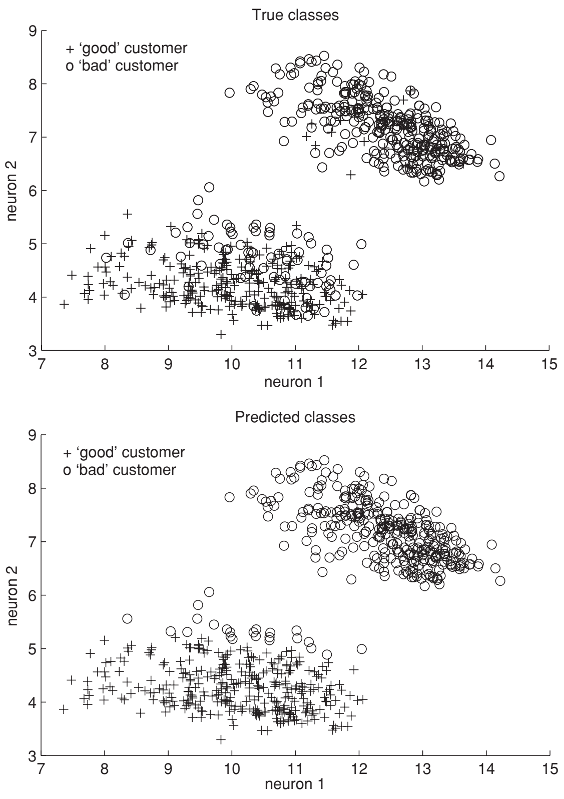

Fig. 2. Australian credit data projected by the hidden layer of the NFKN trained with the proposed algorithm (10), (13)

5 Conclusion

A simple and practical approach to the construction of neuro-fuzzy classifiers was considered. The described NFKN architecture is not affected by the curse of dimensionality because of its two-level structure according to the KST, and is suitable for classification problems with multiple classes and both continuous and discrete (categorical) input variables.

The use of the neo-fuzzy neurons enabled us to develop fast and simple training procedures for both the hidden and output layer parameters, based on the perceptron learning rule. A new batch multioutput modification of this learning rule was proposed.

We expect that the NFKN with the new learning algorithm can find applications in decision support and data mining [3], where classification and visualization of high-dimensional data are the key problems.

An important issue that needs to be further investigated is the interpretability improvement of the fuzzy rules in the NFKN, because the two-level rule base (5), (6) lacks transparency and differs from the rule format used in most of fuzzy modeling approaches.

References

1. Bodyanskiy, Ye., Kolodyazhniy, V., and Otto, P.: Neuro-Fuzzy Kolmogorov's Network for Time Series Prediction and Pattern Classification. In: U. Furbach (Ed.): Lecture Notes in Computer Science Series: Lecture Notes in Artificial Intelligence, vol. 3698, Springer, Berlin Heidelberg NewYork (2005) 191–202
2. Datasets used for classification: comparison of results, http://www.phys.uni.torun.pl/kmk/projects/datasets.html
3. Hastie, T., Tibshirani, R., and Friedman, J.: The Elements of Statistical Learning; Data Mining, Inference, and Prediction. Springer Series in Statistics (2001)
4. Katsuura, H. and Sprecher, D.: Computational aspects of Kolmogorov's superposition theorem. Neural Networks **7** (1994) 455–461
5. Kolmogorov, A.N.: On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. Doklady Akademii Nauk SSSR **114** (1957) 953–956
6. Kolodyazhniy, V. and Bodyanskiy, Ye.: Fuzzy Kolmogorov's Network. Proc. 8th Int. Conf. on Knowledge-Based Intelligent Information and Engineering Systems (KES 2004), Wellington, New Zealand, September 20–25, Part II (2004) 764–771
7. Kolodyazhniy, V., Bodyanskiy, Ye., and Otto, P.: Universal Approximator Employing Neo-Fuzzy Neurons. In: B. Reusch (Ed.): Computational Intelligence. Theory and Applications. Proc. 8th Fuzzy Days, Dortmund, Germany, Sep. 29–Oct. 1 2004. Advances in Soft Computing **2**, Springer, Berlin Heidelberg New York (2005) 631–640
8. Kolodyazhniy, V., Poyedyntseva, V., and Stephan, A.: Modified Fuzzy Kolmogorov's Network for Data Classification. Proc. 12th East West Fuzzy Colloquium, Zittau (2005) 210–217
9. Schynk, J.: Performance Surfaces of a Single-Layer Perceptron. IEEE Transactions on Neural Networks **1** (1990) 268–274
10. UCI Repository of Machine Learning Databases and Domain Theories, http://www.ics.uci.edu/∼mlearn/MLRepository.html
11. Yamakawa, T., Uchino, E., Miki, T., and Kusanagi, H.: A neo fuzzy neuron and its applications to system identification and prediction of the system behavior. Proc. 2nd International Conference on Fuzzy Logic and Neural Networks "IIZUKA-92", Iizuka, Japan (1992) 477–483