mance improves substantially as the degree of overlap is increased to nine and five for training and testing data, respectively. Further increase in the overlap did not offer any substantial improvements in the performance. We also experimented with different combinations of overlap in the training and testing data such as (11, 5), (13, 7), and (13, 5). These combinations yielded performances lower than those presented in the table for the same degree of training sample overlaps (refer to the last two rows in the table). Hence, the results clearly show the advantages and a consistent trend in performance variations with the degree of training and testing sample overlap.

IV. Conclusion

In this paper, we proposed a structure-adaptive multilayer overlapped SOM networks for labeled pattern classification. The proposed approach possesses several attractive features such as the presence of multiple hierarchical overlapped SOM's, and an ability to avoid overfitting by merging and pruning operations. The overlap allows us to employ a decision fusion scheme by a majority voting mechanism to improve the final classification performance. Further, different higher-level maps are trained using a different subset of the training data. We tested the developed network, HOSOM, on handwritten numerals and obtained the best results ever for any SOM-based classifier to our knowledge.

References

[1] H. U. Bauer and T. Villmann, "Growing a hypercubical output space in a self-organizing feature map," *IEEE Trans. Neural Networks*, vol. 8, pp. 218–226, 1997.
[2] S. B. Cho, "Neural-network classifiers for recognising totally unconstrained handwritten numerals," *IEEE Trans. Neural Networks*, vol. 8, pp. 43–53, 1997.
[3] B. Fritzke, "Growing cell structures—A self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441–1460, 1994.
[4] T. M. Ha and H. Bunke, "Off-line, handwritten numeral recognition by perturbation method," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 535–539, 1997.
[5] M. Herrmann, R. Der, and G. Balzuweit, "Hierarchical feature maps and nonlinear component analysis," in *Proc. ICNN'96*, Texas, vol. 2, pp. 1390–1394.
[6] T. Kohonen, *Self-Organizing Maps*, 2nd ed. Berlin, Germany: Springer-Verlag, 1997; also "The self-organizing maps," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480.
[7] J. Lampinen and E. Oja, "Clustering properties of hierarchical self-organizing maps," *J. Math. Imaging Vision*, vol. 2, no. 3, pp. 261–272, 1992.
[8] T. C. Lee and A. M. Peterson, "Adaptive vector quantization using a self-development neural network," *IEEE J. Select. Areas Commun.*, vol. 8, pp. 1458–1471, 1990.
[9] J. Wu, H. Yan, and A. Chalmers, "Handwritten digit recognition using two-layer self-organizing maps," *Int. J. Neural Syst.*, vol. 5, no. 4, pp. 357–362, 1994.

A "Mutual Update" Training Algorithm for Fuzzy Adaptive Logic Control/Decision Network (FALCON)

Sinan Altug, H. Joel Trussell, and Mo-Yuen Chow

***Abstract*— The conventional two-stage training algorithm of the fuzzy/neural architecture called FALCON may not provide accurate results for certain type of problems, due to the implicit assumption of independence that this training makes about parameters of the underlying fuzzy inference system. In this correspondence, a training scheme is proposed for this fuzzy/neural architecture, which is based on line search methods that have long been used in iterative optimization problems. This scheme involves synchronous update of the parameters of the architecture corresponding to input and output space partitions and rules defining the underlying mapping; the magnitude and direction of the update at each iteration is determined using the Armijo rule. In our motor fault detection study case, the mutual update algorithm arrived at the steady-state error of the conventional FALCON training algorithm as twice as fast and produced a lower steady-state error by an order of magnitude.**

***Index Terms*— Adaptation, artificial neural networks, fault detection and diagnosis, mutual update, training.**

I. Introduction

The fuzzy/neural (FZ/NN) architecture proposed by Lin and Lee [1] has been popular with numerous applications in the literature [1], [2]. In this architecture, two sets of parameters are of interest: *membership function parameters and rule parameters* (or *weights*). These correspond to the fuzzy partitions of the input and output spaces, and the relative importance of the rules underlying the architecture, respectively.

The training of the FZ/NN architecture involves modification of these parameters. In one stage, the rule parameters are updated using a modified competitive learning algorithm while the membership function parameters are fixed. In the other stage, the membership function parameters are updated by gradient descent while the rule parameters are fixed to the values found in the first stage.

This two-stage training method may not provide satisfactory results for certain problems, as the training implicitly assumes that the two sets of parameters are almost independent. Usually, the two sets are highly interdependent [1], [3], [4], and this training may not provide accurate results for the cases where this implicit assumption does not hold.

In this correspondence, a method for training this FZ/NN architecture is proposed, which involves synchronous update of the two sets of parameters, and eliminates the requirement for a two-stage training scheme. The proposed training scheme is the *mutual update algorithm*, where at each step, an update is performed in a way to alter either the membership function parameters, or the rule parameters, or both. The magnitude and the direction at each iteration are computed by examining the *sufficient decrease* condition in the objective function.

Manuscript received March 10, 1998; revised September 29, 1998. This work was supported by National Science Foundation under Grant ECS-9 521 609.

The authors are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695-7911 USA.

Publisher Item Identifier S 1045-9227(99)00638-4.

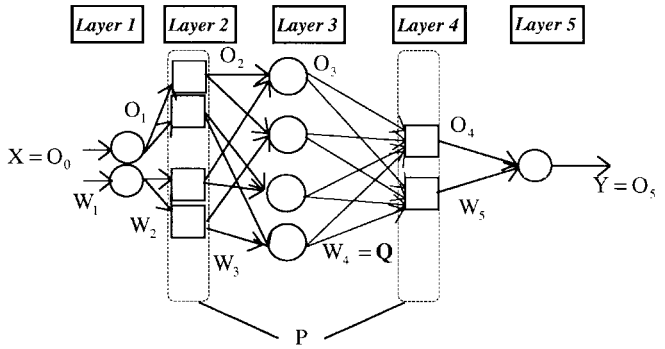

Fig. 1. The FALCON architecture.

II. The Mutual Update Algorithm

The FZ/NN architecture is analogous to a neural network, and is composed of five layers corresponding to the fuzzy inference system operations [1]. For the details of this layered architecture the reader should refer to [1], [2]. In this architecture, the rules are represented by a matrix of connection weights, which will be called rule parameters and denoted by Q. Gaussian membership functions are utilized with the form given in [1], and the parameters (standard deviation and mean values) corresponding to these functions defined in the input and output spaces are contained in P. The FZ/NN is a mapping F that maps the input vector X to the output vector Y, using P and Q

$$Y = F(X, P, Q). \tag{1}$$

The FZ/NN architecture is depicted in Fig. 1.

Let $u_i^{(j)}$ represent ith input of the jth layer, and $o_i^{(j)}$ the ith output of the jth layer. Layer 1 is a direct transmission of the inputs to the next layer. This layer is added for the purpose of having a single entry point for each input to the system

$$o_i^{(1)} = u_i^{(1)}. \tag{2}$$

In the second layer, the membership grade of each input to each fuzzy set is calculated, using the membership functional forms defined by their current parameter values. If Gaussian membership functional forms are used, each membership grade is characterized by the mean m_i and variance σ_i^2

$$o_i^{(2)} = e^{-(u_i^{(2)} - m_i)^2/2\sigma_i^2)}. \tag{3}$$

The choice of Gaussian functions is not a requirement; other functional forms can be chosen provided that the functions are parametrizable with a finite number of parameters.

With O_i and W_i defined as the output vector and connection weights matrix at the ith layer of the FZ/NN respectively, the output of layers one to five can be represented as

$$O_i = W_i^T O_{i-1}; \qquad 1 \le i \le 5 \tag{4}$$

with $O_0 = X$. Using $O_3 = (o_i^{(3)})$, the individual outputs at layer three can be represented as

$$o_i^{(3)} = f_i^{(3)}(P, X) \tag{5}$$

where $f_i^{(3)}$ are nonlinear functions that compute the intersection of the antecedent sets using the fuzzy partitions in the input space.

With the above definitions, W_4 corresponds to rule parameters matrix Q. Using W_4

$$O_4 = W_4^T O_3 = Q^T O_3. \tag{6}$$

The ith component of the FZ/NN output can be computed as

$$(Y)_i = F_i(X, P, Q) = \frac{(W_5^T)_i O_4}{(\Sigma_o^T)_i O_4} = \frac{(W_5^T)_i Q^T O_3}{(\Sigma_o^T)_i Q^T O_3} \tag{7}$$

with $(W_5)_{ij} = (\Sigma_o)_{ij}(M_o)_{ij}$; where Σ_o and M_o are partitions of P containing standard deviations and means of the membership functions defined on outputs, respectively, and $(W_5^T)_i$ and $(\sigma_o^T)_i$ represent the ith column partition of W_5^T and Σ_o^T, respectively.

For any individual layer four weight $w_{ij}^{(4)}$ vectors, where $W_4 = (w_{ij}^{(4)})$, the output F is conditioned by $w_{l;}^{(4)}, i \neq k, j \neq l$. Therefore, the value of $w_{ij}^{(4)}$ to which the training is expected to converge also depends on $w_{kl}^{(4)}, i \neq k, j \neq l$. The connection weight update stage in [1] does not take the value of $w_{kl}^{(4)}, i \neq k, j \neq l$, into consideration during the update of $w_{ij}^{(4)}$. In the connection weight update portion of the proposed mutual update algorithm however, the update of $w_{ij}^{(4)}$ is a function of W_4 rather than only $w_{ij}^{(4)}$. For detailed information on the FZ/NN architecture, the reader should consult [1].

A. The Concept of Sufficient Decrease and Armijo Rule

The proposed method of training in this correspondence for the fuzzy neural architecture is based on line-search methods that have been long used in iterative optimization literature. One of the widely used methods for minimizing a function is the method of steepest descent, which is also an integral part of backpropagation of output error via delta rule in the training of feedforward neural networks. The *step size* is chosen such that *sufficient decrease* in the value of the objective function is achieved [5]. Achieving sufficient decrease in the value of a function f at a point x_o provided by the step size α is equivalent to the satisfaction of the following condition:

$$f(x_o - \alpha \nabla f_x(x_o)) - f(x_o) \le -\alpha \varepsilon \|\nabla f_x(x_o)\|^2 \tag{8}$$

with $x_1 = x_o - \alpha \nabla f_x(x_o)$ and $0 < \varepsilon < 1$, which is selected in accordance with [6]. Utilizing the Armijo Rule [7] in an iterative optimization process, α is chosen at each iteration such that sufficient decrease condition in (8) is satisfied. The Armijo rule is an example of a line search that is carried out on a ray from the current point in a direction in which the function is locally decreasing. For this experiment, the value of the step size α is determined by letting $\alpha = \beta^m$, where $\beta \in (0, 1)$, and m is the smallest positive integer that provides sufficient decrease in the function [5]–[7]. An upper limit for m should be chosen so that the algorithm terminates if sufficient decrease is not achieved.

There is usually a tradeoff between simplicity and efficiency. This correspondence focuses on the FZ/NN system rather on the searching technique in the Armijo rule, thus, the determination of the step sizes α_p and α_q was done by a simple linear search technique. Other more efficient techniques and discussion on the convergence property and local minimum problems of different searching algorithms can be found in [6].

B. Mutual Update Schemes Utilizing the Armijo Rule

Utilizing Armijo rule to train the FZ/NN architecture, step sizes for the updates of P and Q are selected exclusively at each iteration of the algorithm. The objective function $E(P, Q)$ is defined as the mean square output error. The sufficient condition for the FZ/NN objective function is defined as

$$\begin{aligned} E(P_o - \alpha_p \nabla_p E(P_o, Q_o), -\alpha_q \nabla_q E(P_o, Q_o)) - E(P_o, Q_o) \\ -\varepsilon\{\alpha_p \|\nabla_p E(P_o, Q_o)\|^2 + \alpha_q \|\nabla_q E(P_o, Q_o)\|^2\} \end{aligned} \tag{9}$$

where P_o and Q_o are the current membership function and rule parameter matrices. The sufficient decrease examination is carried

out by checking whether step sizes $\alpha_p > 0$ and $\alpha_q > 0$ that satisfy the condition in (9) exist. If such α_p and α_q exists, then P and Q are updated according to

$$P_+ = P_o - \alpha_p \nabla_p E(P_o, Q_o) \tag{10}$$

$$Q_+ = Q_o - \alpha_q \nabla_q E(P_o, Q_o). \tag{11}$$

Where P_+ and Q_+ are the next parameter and rule weights matrices, respectively. In the case where such α_p and α_q cannot be simultaneously found, individual α_p and α_q that provide sufficient decrease in one direction are calculated individually. This is done by substituting $\alpha_q = 0$ in (9) and computing step size α_p that satisfies (9), and updating P according to (10); or substituting $\alpha_p = 0$ in (9) and computing step size α_q that satisfies (9); and then updating Q according to (11) for the rule connection weights. These two cases are referred to as exclusive membership function parameter update and exclusive rule connection weights update, respectively. When such α_p and α_q can be individually found yet they do not satisfy the condition put forth in (9), the direction that provides the larger increase is chosen. The iteration terminates if sufficient decrease in none of the above three cases can be achieved.

Comparing the updates of P and Q with the training of [1], the membership function parameter update strategy is analogous to the supervised membership function training portion of [1]. However, in [1], P is updated without examining Q. Furthermore, as mentioned previously, the rule weight training in [1] is carried out prior to membership function parameter training via a modified competitive learning strategy, with P kept constant throughout the training.

C. Training Results with Mutual Update Algorithm

The mutual update training algorithm was compared with the training scheme of [1] for a data set acquired from an induction motor simulation. In this paper, the *MotorSIM* software [3], based on the MATLAB-SIMULINK platform is used to generate the appropriate motor data needed, in a cost effective and time efficient manner, to demonstrate the effectiveness of the proposed training algorithm. The *MotorSIM* tool provides motor performance simulations for different fault and operating conditions. The components in the MotorSIM software include friction, winding, saturation, thermal, vibration, noise, and disturbance models, in addition to the basic electrical and mechanical models. For the sake of simplicity and ease of discussion, the types of motor faults in this paper are classified into two major categories: electrical and mechanical. The electrical incipient fault, related to the *winding* conditions, is assumed to be reflected in the equivalent winding turns N, while the mechanical incipient fault is assumed to be a friction-related fault as reflected in the friction coefficient B (also known as damping coefficient). The details of which can be found in [3]. The data consists of motor system variables such as rotor speed, stator current and load torque, and the corresponding motor friction [3]. The FZ/NN is constructed in a way to determine the motor friction using the other motor system variables mentioned above.

For a fair comparison of the mutual update algorithm, the original algorithm and a modified version of the conventional algorithm were implemented. The original algorithm is a two-step method where after initialization, the weights are found using a competitive learning scheme in the first step. The second step is the adaptation of the membership function parameters via gradient descent. In the modified version, instead of employing the two-stage training scheme, at every iteration, the rule connection weights were updated along with the membership function parameters, similar to the mutual update algorithm. However, the step sizes were kept constant.

The output error versus training epochs for the mutual update algorithm, the original training algorithm of [1], and the modified version of [1] mentioned above are presented in Fig. 2. Comparing the output error of the FZ/NN for the above three update schemes, it is observed from Fig. 2 that the mutual update algorithm yields lower output error than either the method utilized in [1] or the modified version of [1] mentioned above. The modification of the connection weights according to the mutual update algorithm guarantees monotonic decrease of the output error at each iteration, as opposed to the modified case with constant step sizes. Also different from [1], employing the mutual update algorithm, membership function parameters as well as rule weights are updated, eliminating the independence assumption of the two stage algorithm about P and Q. It is noted though, that computer time per iteration for the mutual update algorithm is expected to be longer as the algorithm requires examination of the sufficient decrease conditions for the membership function and rule weight matrices.

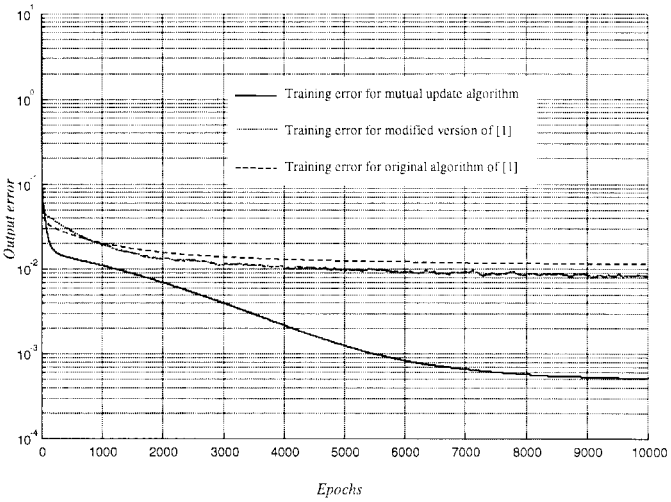

Fig. 2. Output error during training for mutual update algorithm, modified version of [1], and the original training algorithm of [1].

III. Summary

In this correspondence, a training scheme is proposed for the fuzzy neural architecture in [1], based on line search methods that have long been used in iterative optimization problems. The algorithm that the training is based on involves synchronous update of the parameters of the architecture corresponding to input and output space partitions and the rules defining the underlying mapping; the magnitude and direction of the update term at each iteration is determined using the Armijo rule. In our motor fault detection study case, the mutual update algorithm arrived at the steady-state error of the conventional FALCON training algorithm [1] as twice as fast and produced a lower steady-state error by an order of magnitude.

References

[1] C. T. Lin and C. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Computers*, vol. 40, pp. 1320–1336, 1991.

[2] ——, *Neural Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[3] M. Y. Chow, *Methodologies of Using Artificial Neural Network and Fuzzy Logic Technologies for Motor Incipient Fault Detection*. Singapore: World Scientific, 1997.

[4] P. Goode and M. Y. Chow, "Using a neural/fuzzy system to extract knowledge of incipient fault in induction motors part I—Methodology," *IEEE Trans. Ind. Electron.*, vol. 42, pp. 131–138, 1995.

[5] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*. Philadalphia, PA: SIAM, 1995.

[6] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific J. Math.*, pp. 1–3, 1966.
[7] D. G. Luenberger, *Linear and Nonlinear Programming*. New York: Addison-Wesley, 1989.

Automated Labeling for Unsupervised Neural Networks: A Hierarchical Approach

Roberto Tagliaferri, Nicola Capuano, and Giorgio Gargiulo

***Abstract*— In this paper a hybrid system and a hierarchical neural-net approaches are proposed to solve the automatic labeling problem for unsupervised clustering. The first method consists in the application of nonneural clustering algorithms directly to the output of a neural net; the second one is based on a multilayer organization of neural units. Both methods are a substantial improvement with respect to the most important unsupervised neural algorithms existing in literature. Experimental results are shown to illustrate clustering performance of the systems.**

***Index Terms*—Automated labeling, clustering, hierarchical neural networks, hybrid neural networks.**

I. Introduction

Unsupervised neural nets (NN's) divide the patterns belonging to a training set into subsets called clusters by using specific similarity measures in such a way that patterns of the same clusters are very similar while patterns belonging to different clusters are very dissimilar. After the learning process each neuron represents a single cluster composed by all the patterns enclosed in the Voronoi region identified by the neuron itself. Several times the so obtained partition of the input space is more detailed than required because the net, to correctly learn, need more neurons than the class number. In this case we have some neurons representing the same class and some neurons positioned between the classes without representing any specific one.

These facts lead us to a next step, where, by using the *a priori* knowledge of an expert, we label each cluster with the name of a class in such a way that we can use the NN as a classifier. On the other way, if the net has as many outputs as the class number then it is very easy to label the output units, by using just an element for each class. Unfortunately, the neuron number for the classical one layer unsupervised NN's need to be much greater than the number of output classes to avoid a great error during the learning. In fact, during the learning, in many models, when a neuron wins the net adapts not only its weights but also the weights of its neighbors (soft-max adaptation) to avoid the fall in local minima of the error function. With a low number of neurons, when presenting in input a new pattern, we strongly modify also the weights related to other classes. This implies that classes with a greater number of input patterns have more than one neuron for representing them, while other classes have not even one neuron. In this case we say that the NN is under-dimensioned and it has an insufficient generalization capacity.

As a consequence, we need to use many neurons with respect to the class number and to have a complex labeling phase with several class representatives distributed over the output layer. To label every neuron, we need to use a trial and error technique, by giving in input to the net more patterns for each class and then by seeing the activated units. The main risk is to activate only a subset of the available neurons. It is easy to see that such a method is boring and not reliable and need many well classified patterns. In this case it is better to use supervised NN's which work better than unsupervised NN's. Aim of this paper is to solve in an automatic manner and without the help of an expert the labeling phase of any unsupervised NN, by using two different approaches: the former, called hybrid, is based on the application of a classical clustering algorithm to the neuron weights; the latter, called hierarchical neural net, is based on the multilayer organization of the net on ever smaller layers from the input to the output. In the next section we illustrate the neural models, i.e., unsupervised NN's, Hybrid models and hierarchical unsupervised NN's. In Section III we show the behavior of the nets on some data sets used as example.

Manuscript received April 23, 1998; revised October 16, 1998.

R. Tagliaferri is with DMI, Università di Salerno and INFM unità di Salerno, 84081 Baronissi, Salerno, Italy.
N. Capuano is with Facoltà di Scienze, Università di Salerno, 84081 Baronissi, Salerno, Italy.
G. Gargiulo is with IIASS "E. R. Caianiello," 84019 Vietri s/m, Salerno, Italy.

Publisher Item Identifier S 1045-9227(99)01159-5.

II. The Neural-Network Models

A. Unsupervised Neural Nets

Kohonen self organizing maps (SOM's) [4], [5] are composed by a neuron layer structured in a rectangular grid. When a pattern x is presented to the net each neuron i receives the components and computes the distance from its weight vector. The unit k which has the minimum distance from the input pattern will be the winner. The adaptation step consists in the modification of the weights of the neurons in the following way:

$$w_i^{(t+1)} = w_i^{(t)} + \varepsilon^{(t)} \cdot h_{\sigma(t)}(d(i,k)) \cdot (x - w_i^{(t)})$$

where $\varepsilon^{(t)}$ is a gain term $(0 \square \varepsilon^{(t)} \square 1)$ decreasing in time, $h_{\Sigma(t)}(x)$ is a unimodal function with variance $\Sigma^{(t)}$ decreasing with x and $d(i,k)$ is the distance in the grid between the i and the k neurons.

The neural-gas NN's have a learning algorithm [7] which works better than the preceding one: in fact, it is quicker and it reaches a lower average distortion value.[1] It uses a soft-max adaptation of the weights and it classifies the neurons in an ordered list $(i_1, i_2, \cdots, i_n)$ following their distance form the input pattern. The weight adaptation depends on the position *rank*(i) of the i neuron in the list in the following manner:

$$w_i^{(t+1)} = w_i^{(t)} + \varepsilon^{(t)} \cdot h_\sigma(\mathrm{rank}(i)) \cdot (x - w_i^{(t)}).$$

The algorithm applies the gradient descent technique to the error function

$$E_{ng} = \frac{1}{2C(\sigma)} \sum_{i=1}^{m} \int P(x) \cdot h_\sigma(\mathrm{rank}(x)) \cdot (x - w_i)^2 \, d^n x,$$

$$\text{with } C(\sigma) = \sum_{k=0}^{m-1} h_\sigma(k).$$

[1] Let $P(x)$ be the pattern probability distribution over the set $V \subseteq \Re^n$ and let $w_{i(x)}$ be the weight vector of the neuron which classifies the pattern x, therefore we define average distortion as

$$E = \int P(x)(x - w_{i(x)})^2 \, d^n x.$$