PERGAMON

Pattern Recognition 36 (2003) 853–867

PATTERN
RECOGNITION
THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

www.elsevier.com/locate/patcog

Constructing and training feed-forward neural networks for pattern classification

Xudong Jiang *, Alvin Harvey Kam Siew Wah

Laboratories for Information Technology, 21 Heng Mui Keng Terrace, Singapore 119613, Singapore

Received 18 January 2001; accepted 18 April 2002

Abstract

A new approach of constructing and training neural networks for pattern classification is proposed. Data clusters are generated and trained sequentially based on distinct local subsets of the training data. Obtained clusters are then used to construct a feed-forward network, which is further trained using standard algorithms operating on the global training set. The network obtained using this approach effectively inherits the knowledge from the local training procedure before improving on its generalization ability through the subsequent global training. Various experiments demonstrate the superiority of this approach over competing methods. © 2002 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved.

Keywords: Classification; Neural networks; Clustering; Local and global training; Generalization

1. Introduction

Neural networks are well known as powerful tools in the area of pattern classification. In principle, multi-layer feed-forward networks with just a single hidden layer are universal approximators for arbitrary finite-input environment measures. This however does not imply that a neural network can easily learn the underlying functional mapping between the input data and the desired output. In fact, the main drawbacks of neural networks are problems associated with local minima and the slow convergence of the learning process. To tackle these problems, neural networks are increasingly using clustering techniques to reduce learning complexity.

A cluster can be modeled by a hypersphere represented by its center, i.e., the "prototype" and its radius, which determines the "region of influence" of the cluster. Prototypes used in initializing the weights of back propagation networks are known to yield reductions in training time [1]. Various clustering techniques have also been used to reduce the number of neurons in radial basis function and probabilistic neural networks [2–7]. In Ref. [8], a multi-objective genetic algorithm was employed to partition the pattern space into hyperspheres for mapping onto a hierarchical neural network to facilitate subsequent learning. In addition to these approaches devoted to reduce learning complexity and the number of hidden nodes, several existing neural network learning algorithms can themselves be regarded as some kind of clustering methodology, for example competitive learning [9], self-organizing map and learning vector quantization (LVQ) [10]. It is therefore quite evident that clustering techniques form the core learning procedure for numerous neural network models.

Traditional clustering methods [11–14] placed adjacent samples of the training set in a single cluster without taking into account their class assignments. The obvious drawback of these approaches is the missing class membership information, which is crucial for any classification learning procedure. Duda presented a training algorithm for cluster prototypes [15] while the LVQ approach [10] modifies the position of the prototypes through a training procedure on all input data. Both these algorithms require the number of prototypes to be defined and good initialization of the prototype values before training. Kong proposed a variant of these basic paradigms [16]. All these methods result in piecewise linear classifiers. The generalization capability is thus

* Corresponding author. Tel.: +65-6874-7588; fax: +65-6776-8109.
E-mail address: xdjiang@lit.org.sg (X. Jiang).

PII: S0031-3203(02)00087-0

relatively poor. The distribution of the prototypes approximates the distribution of the learning data. This results in that each prototype of one class represents a cluster that encloses roughly equal number of learning samples. To form a complicated decision boundary, a large number of prototypes are therefore needed to precisely approximate the training data distribution. In RCE approach [17], prototypes can neither be modified nor removed, but any misclassification reduces the scope of the prototype vector. Thus, the RCE clustering technique does not minimize the number of clusters that a particular classification task requires. Musavi meanwhile presented a clustering algorithm [5] to reduce the number of nodes of RBF networks. As clusters are restricted from enclosing any samples of the other classes, this technique usually leads to a large number of clusters.

In view of the drawbacks of methods discussed above, we present an efficient iterative locally trained clustering (LTC) technique, which generates clusters sequentially, moves cluster's centers generally away while enlarging their scope towards the Bayes decision surfaces by using local subsets of the training data. Clusters therefore grow to contain more and more samples within the Bayes decision region during training. The procedure for training a cluster is based on a local subset of training data. Thus, the complex training data-structures that are far from a cluster will not increase the complexity of the training procedure for this cluster. The ground ideas of this training procedure are decomposing a complicated data structure of the whole training set into a number of simple subsets, modifying and enlarging clusters to let them capture training samples as more as possible so long as the clusters are within the Bayes decision region. Therefore, this procedure simplifies the training and minimizes the number of clusters. The obtained clusters are then used to construct a feed-forward neural network, which contains piecemeal knowledge obtained from the individual local learning procedures. Finally, all the individual pieces of knowledge are integrated and generalized by training the constructed network using the entire set of training data. The resulting locally and globally trained (LGT) network is then applied on several benchmark classification problems to access its performance and compare it with other representative clustering approaches and neural networks.

2. Local training clusters (LTC) algorithm

Consider a training data set with M classes $C = C^1 \cup C^2 \cup \cdots \cup C^M$, where C is a set of N-dimensional pattern feature vectors $X = (x_1, x_2, \ldots, x_N)^{\mathrm{T}}$ whose class memberships are known. For each class C^m, we define its inverse class as $C^{mo} = C - C^m$. Let $p(X|C^m)$ and $p(X|C^{mo})$ denote the conditional probability density functions of X belonging to class C^m and C^{mo}, respectively. The ith hypersphere cluster C_i^m of the mth class C^m is represented by its center or prototype V_i^m and its radius r_i^m. The simplest measure of similarity between samples X and cluster's center is the Euclidean distance $\| \bullet \|$. A sample X is a member of C_i^m, i.e. $X \in C_i^m$, if and only if

$$\|X - V_i^m\| \leqslant r_i^m. \tag{1}$$

Suppose the number of samples in cluster C_i^m is n_i^m. It should be noted that a cluster C_i^m could enclose samples from class C^m as well as its inverse class C^{mo}. Let R_i^m and F_i^m be the two subsets that enclose the "correct" and "wrong" samples of cluster C_i^m, respectively, i.e. $R_i^m = \{X | X \in C_i^m \cap C^m\}$, $F_i^m = \{X | X \in C_i^m \cap C^{mo}\}$. The mean value and the number of learning samples in subsets R_i^m and F_i^m are denoted by Mr_i^m, nr_i^m and Mf_i^m, nf_i^m, respectively.

The training procedure for clusters should move their centers V_i^m away while expanding their radius r_i^m towards the Bayes decision surfaces in the hope that clusters can grow to enclose more samples of its own class within the Bayes decision region. For most practical problems, we can assume that in the decision region of class C^m there are a higher probability density for class C^m and a lower probability density for class C^{mo} in the region far away from the Bayes decision surfaces than that near the Bayes decision surfaces. It is therefore a right way to modify a cluster C_i^m by moving its center to where there are a higher probability density of its own class and a lower probability density of its inverse class. It is easy to see that V_i^m is the geometric center of cluster C_i^m and Mr_i^m and Mf_i^m are the centers of gravity of the subset R_i^m and F_i^m, respectively. Thus in most cases:

$$p(X|C^m)|_{X=Mr_i^m} \geqslant p(X|C^m)|_{X=V_i^m} \tag{2}$$

and

$$p(X|C^{mo})|_{X=Mf_i^m} \geqslant p(X|C^{mo})|_{X=V_i^m}. \tag{3}$$

Formulas (2) and (3) are definitely true at least if the two conditional probability densities vary monotonously in the region of influence of the cluster. Therefore, cluster centers V_i^m should be drawn towards Mr_i^m and away from Mf_i^m in the hope that the cluster can enclose more samples of the correct class C^m and less samples of the inverse class C^{mo}. The cluster radius r_i^m could then be enlarged somewhat since the cluster center might be drawn away from the Bayes decision surface.

The training data set C^a for training the first cluster of a class is the entire training set C. We first assign an arbitrary sample X_r of class C^m to be a cluster center V_i^m; then search the nearest neighboring sample X_f of the inverse class and let the initial cluster radius $r_i^m = \|X_f - V_i^m\|$. In this way, a cluster C_i^m is generated with $nr_i^m \geqslant 1$ and $nf_i^m = 1$. Initially, let $V_i^m(t) = V_i^m$, $r_i^m(t) = r_i^m$ and a parameter $j(t) = 0$. The proposed iterative LTC algorithm for training one cluster is as follows:

$$\textit{Step } 1\text{: } V_i^m(t+1) = V_i^m(t) + lr1\left[\frac{nr_i^m(t)}{n_i^m(t)}[Mr_i^m(t) - V_i^m(t)] - \frac{nf_i^m(t)}{n_i^m(t)}[Mf_i^m(t) - V_i^m(t)]\right]. \tag{4}$$

$$\textit{Step } 2\text{: } r_i^m(t+1) = r_i^m(t) + lr2(t)\lfloor \|V_i^m(t+1) - V_i^m(t)\| + \varepsilon \rfloor. \tag{5}$$

$$\textit{Step } 3\text{: If} \quad n_i^m(t+1) \leqslant n_i^m(t) \quad \text{then} \quad lr2(t+1) = 1.2lr2(t). \tag{6}$$

$$\text{if} \quad [nr_i^m(t+1) - nr_i^m(t)] < [nf_i^m(t+1) - nf_i^m(t)] \quad \text{then} \quad \begin{cases} lr2(t+1) = 0.9lr2(t) \\ r_i^m(t+1) = r_i^m(t) \\ j(t+1) = j(t) + 1. \end{cases} \tag{7}$$

$$\textit{Step } 4\text{: if} \quad [nr_i^m(t+1) - nr_i^m)] \geqslant [nf_i^m(t+1) - nf_i^m)] \quad \text{then} \quad \begin{cases} V_i^m = V_i^m(t+1) \\ r_i^m = r_i^m(t+1). \end{cases} \tag{8}$$

Step 5: if $j(t+1) \leqslant Ts$ and $(C^a - C_i^m) \neq \Phi$

then $(t+1) \Rightarrow t$ and goto step 1,

else stop training this cluster.

$lr1$ and $lr2(t)$ are the learning rates and generally $0 < lr1,\ lr2(t=0) \leqslant 1$. ε is a very small positive constant that is used to ensure $r_i^m(t+1) > r_i^m(t)$ after step 2 even if $V_i^m(t+1) = V_i^m(t)$. The cluster is iteratively trained until it either encloses all samples in the training set C^a or cannot be enlarged for Ts times of modification.

The detailed rationales for each processing step are as follows. Step 1 modifies the cluster center toward the center of mass of its own class and away from the center of mass of its inverse class with learning rates weighted by the corresponding numbers of samples $nr_i^m(t)$ and $nf_i^m(t)$, respectively. In most cases, especially if the cluster is near the Bayes decision surface, the cluster center will be moved away from the Bayes decision surfaces. Although this is not always true, step 1 at least attempts, only with the knowledge of the data distribution within the cluster, to enclose more samples of its own class and less samples of its inverse class. Step 2 enlarges the cluster in the hope of that the cluster can be enlarged somewhat without exceeding the Bayes decision surfaces since step 1 may move the cluster center away from the Bayes decision surfaces. Since this is not always successful, step 3 controls the learning parameters and the cluster growing process. Specifically, the learning rate $lr2(t)$ will increase if the modified cluster does not enclose more samples after step 2. This is important since the training samples are discretely distributed and the scale-space of training data might be quite different for different learning tasks. If there exist training samples outside the cluster, Eq. (6) ensures the cluster after step 2 enclosing more samples after some numbers of iterations. If the enlarged cluster increases its inverse class samples more than the correct class ones, the learning rate $lr2$ will be decreased, the cluster not enlarged and the counter $j(t)$ incremented by one. The cluster will however still be modified to reflect its new data composition in the hope that this will lead to successful enlargement of the cluster in the next or following iterations. However, this is not definitely true in every case. Therefore, step 4 records successful modifications of the cluster during the training procedure and the final trained cluster parameters are V_i^m and r_i^m instead of $V_i^m(t)$ and $r_i^m(t)$. Step 4 is important because the training procedure does not guarantee for each training iteration to get a better cluster than before. It ensures the best cluster parameters, in the sense of the largest value of $(nr_i^m - nf_i^m)$, to be recorded during the training procedure. Finally, step 5 provides the stopping criterion for the procedure of training a cluster, i.e. training will terminate if the cluster encloses all samples of the training set C^a or there is no cluster enlargement for Ts times. Otherwise, the procedure will repeat from step 1 through 5. Thus, given a limited value of Ts, the training procedure is going to converge for a finite training set. It is worth noting that the number of learning iterations is adaptive and typically different for each cluster.

We wish to point out that this training algorithm may not necessarily be the best method for every type of data distribution. It however provides a simple, fast and rational procedure to capture as many correct training samples as possible to form a local decision region. It may be possible to devise more sophisticated cluster models but this will lead to a significant increase in training time with no guarantee of performance improvements. Furthermore, the learning rates $lr1$ and $lr2$ provide the mechanism to adjust the trade-off between the amount of training data incorporated into the cluster for each iteration and the total training time involved. For our experiment, we utilized a predetermined constant value for $lr1$ while $lr2$ increases and decreases based solely on step 3 of the training procedure.

Fig. 1 illustrates a one-dimensional example showing the operation of the LTC algorithm in different regions with dissimilar data distribution. $p(C^m)$ and $p(C^{mo})$ are the a priori probabilities of occurrence of classes C^m and C^{mo}, respectively while DR is the Bayes decision region of class C^m. The feature space is divided into 5 regions, R1 to R5, as shown in Fig. 1.

If a small cluster for class C^m is initially generated within the region R2 or R4, or around the boundaries between R1

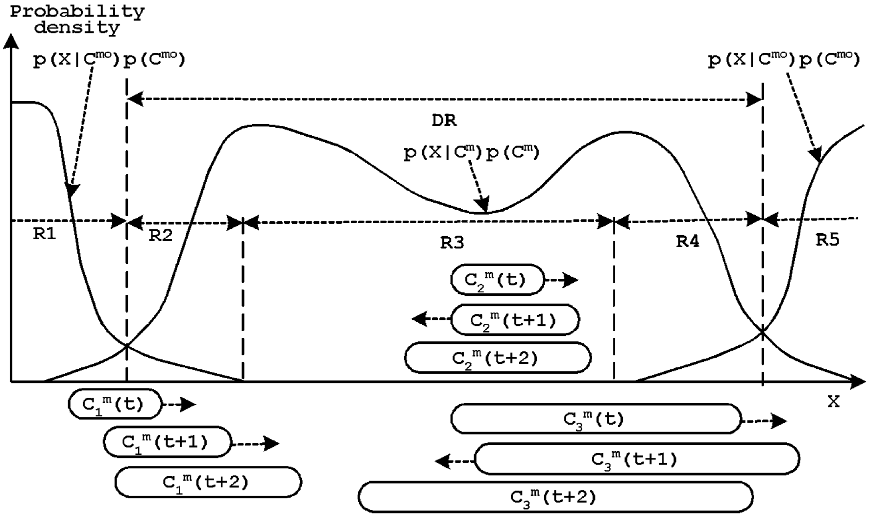


Fig. 1. Examples of the LTC training procedure in different regions.

and R2 or R4 and R5, Eqs. (4) and (5) will drag the cluster center away from the Bayes decision surfaces while enlarging the cluster size (as illustrated by $C_1^m(t)$, $C_1^m(t+1)$ and $C_1^m(t+2)$ in Fig. 1). Conversely, if a small cluster for class C^m is generated within the region R3, or between R2 and R3 or R3 and R4, the cluster will still be enlarged by the training procedure even though the cluster center may not necessarily be dragged away from the Bayes decision surfaces (as illustrated by $C_2^m(t)$, $C_2^m(t+1)$ and $C_2^m(t+2)$ in Fig. 1). When a cluster grows to a certain size, modifications of the cluster may cause its region to exceed the Bayes decision surfaces even if it is not enlarged. However, the next modification of the cluster may draw it away from the decision boundaries while further enlarging the cluster. Examples $C_3^m(t)$, $C_3^m(t+1)$ and $C_3^m(t+2)$ in Fig. 1 illustrate this case. After a number of iterations, the region of influence of a cluster may approximate the Bayes decision region DR. Even if a cluster C_i^m could not grow large enough to cover the decision region when its training procedure terminates, training samples of C^m outside the region of influence of C_i^m but within DR will generate other clusters, which together with C_i^m will approximate the whole decision region.

If an undesired extremely small cluster for class C^m is generated within the region R1 or R5, it will not be enlarged. Every updating iteration will likely result in the condition of Eq. (7) to be true. The training procedure will therefore terminate very fast and the cluster will remain very small if the cluster is not moved out of the inverse class regions. These extremely small clusters will be discarded from the final cluster set. This is an important step especially in the cases where samples of different classes are heavily overlapping. For example in Fig. 1, a large number of extremely small clusters for class C^m could be generated in regions R1 and R5. This will not only unnecessarily increase the number of hidden units of the neural network constructed later on but also lead to over-fitting and consequently, poor generalization. Unfortunately, there is no general satisfactory method to determine the minimum acceptable cluster size if one has only the training set and no other knowledge about the data. A similar problem is that one cannot determine the optimal k for k-nearest neighbor classifier to perform the best generalization if only a training data set is available. For our experiments, clusters will be discarded if their number of correct samples nr_i^m is smaller than a predetermined minimal value nr_{min}.

Having trained C_i^m, the set of correct samples within the cluster C_i^m (all X, $X \in R_i^m$) is removed from the training set C^a ($C^a - R_i^m \Rightarrow C^a$). The algorithm then generates and trains another cluster C_{i+1}^m for class C^m, using the same procedure as before but now using the reduced training set. This procedure will be continued until there are no samples of class C^m left in the remaining training set. Then, clusters of other classes are generated and trained based on the whole training set $C^a = C$, using the same procedure. Fig. 2 illustrates the flowchart of the proposed LTC training procedure where q_m is the number of clusters generated for class C^m.

The above-proposed LTC algorithm generates clusters sequentially, trains each cluster iteratively using local data within the cluster. It decomposes a complex learning task represented by the entire training data set into a number of simple sub-problems operating on small local training sets. Each cluster captures a piece of knowledge efficiently and rapidly without being sidetracked or affected by training data beyond the local vicinity of the cluster. Therefore, this training procedure overcomes problems such as Moving-target or Herd-effect [18], Crosstalk [19,20] and Catastrophic interference [8,21], which often occur in monolithic network learning. The LTC training algorithm generally minimizes

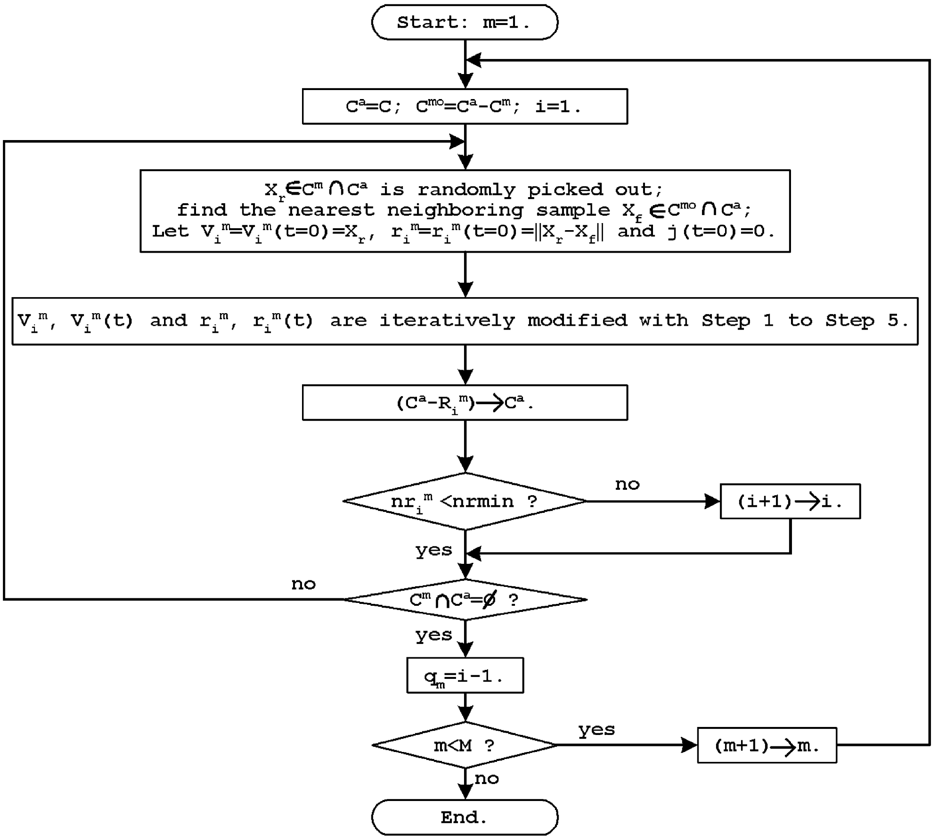

Start: m=1.
$C^a=C$; $C^{mo}=C^a-C^m$; i=1.
$X_r \in C^m \cap C^a$ is randomly picked out;
find the nearest neighboring sample $X_f \in C^{mo} \cap C^a$;
Let $V_i^m=V_i^m(t=0)=X_r$, $r_i^m=r_i^m(t=0)=\|X_r-X_f\|$ and j(t=0)=0.
V_i^m, $V_i^m(t)$ and r_i^m, $r_i^m(t)$ are iteratively modified with Step 1 to Step 5.
$(C^a-R_i^m) \rightarrow C^a$.
nr_i^m <nrmin ?
no
$(i+1) \rightarrow i$.
yes
no
$C^m \cap C^a=\emptyset$?
yes
$q_m=i-1$.
m<M ?
yes
$(m+1) \rightarrow m$.
no
End.

Fig. 2. Flowchart of the LTC training procedure.

the number of clusters for a typical classification task in comparison to some other clustering-based approaches. This will reduce the resultant neural network size as well as increase its generalization ability. However, as the algorithm uses hypersphere clusters, sufficiently anisotropic decision regions might result in larger number of clusters with subsequent adverse effects on network size and generalization ability.

The LTC algorithm produces a series of clusters C_i^m represented by their centers V_i^m, regions of influence r_i^m and class membership m. Based on the trained clusters, a two-layer feed-forward neural network (called the LTC network) can be constructed with threshold transfer functions as shown in Fig. 3. The first layer classifies input samples to various clusters by the weights V_i^m and the thresholds r_i^m. Each hidden node that represents a cluster becomes activated if and only if

$$-\|X - V_i^m\| + r_i^m > 0 \tag{9}$$

as the chosen threshold transfer function is given by

$$thd(\text{arg}) = \begin{cases} 1 & \text{if arg} > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

The second layer combines all clusters with the same class membership into one class. The weights w_{ms}^2 between the hidden nodes s and output nodes m are assigned to be 1 if the cluster s has class membership m; otherwise, they take the value of -1. The thresholds b_{ms}^2 are set to be -0.5. This assignment of network structure facilitates further global training of the network as detailed in the next section.

The LTC algorithm and network has several limitations that need to be addressed. Firstly, the local training procedure may produce sub-optimal placements of cluster centers. Secondly, due to its dependence on hypersphere clusters, the LTC network can only form piecewise spherical decision regions, which may leave holes (regions not covered by any cluster) or produce some conflict regions (regions covered by clusters of different classes). These limitations degrade the generalization performance of the network. Fortunately, the drawbacks could be overcome by fine-tuning the network using standard algorithms used in feed-forward neural networks and operating on the global set of training data. At this training stage, each training sample will contribute to the modification of each neuron parameter of the constructed network, thereby enhancing its overall generalization ability.

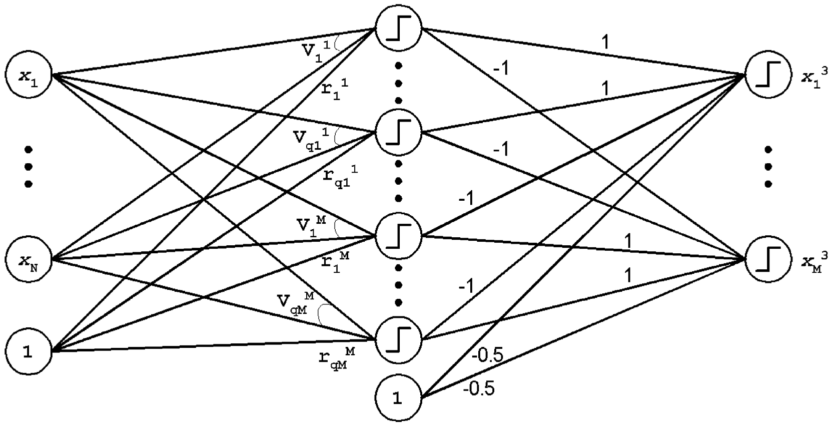


Fig. 3. LTC neural network constructed with clusters.

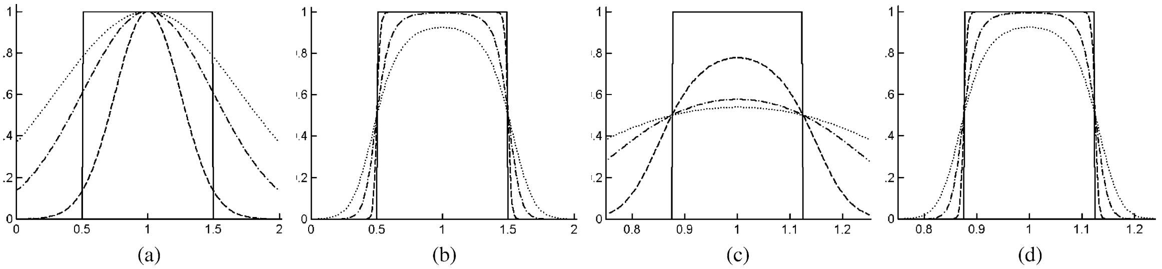


Fig. 4. Clusters represented by the threshold functions (—) and (a) Gaussian basis functions with $h_i^m = 1(\cdots)$; 0.5(-·-·); 0.125(- -) for $r_i^m = 0.5$, (b) and (c) sigmoid functions with $t_i^m = 10(\cdots)$; 20(-·-·); 80(- -) for $r_i^m = 0.5$ and 0.125, respectively and (d) $t_i^m = 160(\cdots)$; 320(-·-·); 1280(- -) for $r_i^m = 0.125$.

3. Network fine-tuning with the global training set

At this training stage, it is necessary to use “soft” nonlinear transfer functions such as the Gaussian function used in RBF networks or the sigmoid function used in multi-layer perceptrons (MLPs). In the LTC network, a cluster C_i^m is represented by the output of a hidden unit with the hard threshold function

$$o_i^m = thd(-\|X - V_i^m\| + r_i^m). \tag{11}$$

If a Gaussian basis function is used in the network, a cluster will be represented by

$$o_i^m = \exp(-\|X - V_i^m\|^2/h_i^m), \tag{12}$$

where h_i^m is a smoothing parameter, which controls both the radius of the cluster and the steepness of the transfer function. This implies that an excessively flat Gaussian function must be used to represent a large cluster. Similar criticisms and analysis of related basis functions for RBF networks were presented in Ref. [22]. Fig. 4a illustrates a cluster ($V_i^m = 1$, $r_i^m = 0.5$) represented by a threshold function and Gaussian functions with different smoothing parameter values.

If a sigmoid function is used, the cluster can be then represented by

$$o_i^m = \frac{1}{1 + \exp(-t_i^m(-\|X - V_i^m\|^2 + (r_i^m)^2))}, \tag{13}$$

where t_i^m, the temperature parameter, controls the steepness of the function. The cluster radius and function steepness can thus be independently controlled using the parameters r_i^m and t_i^m, respectively. Fig. 4b compares sigmoid functions with different smoothing parameter values. It is easy to see that the sigmoid function can arbitrarily accurately approximate the threshold function if the smoothing parameter is sufficiently large. This is but not the case for Gaussian function. Comparing Fig. 4a and b, it is obvious that the sigmoid function does a better job of representing a cluster than the Gaussian function. This provides the essential motivation for selecting the sigmoid function as the basis for the global training.

Although large values of t_i^m has the network well inheriting the characteristics of the LTC network, too large values of t_i^m would result in a network almost the same as the LTC network with hard threshold functions. Consequently, the network would lack the flexibility needed to finely tune the weights to produce the final enhanced LGT network. Conversely, too small values of t_i^m results in the network deviating too far from the LTC network leading to a slow rate of global learning. Furthermore, the values of the smoothing parameter t_i^m should be adaptive for each hidden unit due to the different sizes of clusters involved. Fig. 4c depicts sigmoid functions with the same values of t_i^m as in Fig. 4b but now used to represent a cluster with r_i^m of 0.125 (1/4 times as large as that in Fig. 4b). It is obvious that the functions in Fig. 4b and c appear starkly different despite possessing the same values of t_i^m. We thus need to normalize the temperature parameter t_i^m with respect to the cluster size. This is achieved using the following heuristic formula:

$$t_i^m = \frac{sc}{(r_i^m)^2}, \tag{14}$$

where sc is a constant and empirical studies conclude that $1 \leqslant sc \leqslant 5$ produces good results.

Fig. 4d illustrates sigmoid functions with values of t_i^m 16 times as large as that used in Fig. 4c for a cluster with size of 1/4 times as large as that in Fig. 4b. We believe that if we select a smoothing parameter for a large cluster as in Fig. 4b, the choice of the smoothing parameter in Fig. 4d is more reasonable than in Fig. 4c for a small cluster. Eq. (14) provides the means of effectively using the nonlinear sigmoid transfer function for every hidden node. For the output layer we simply assign the smoothing parameters to be ones.

In order to use standard algorithms (for example, BP and its various variants, Quickprop and Rprop) to train the network, it is convenient to use dot products instead of calculating Euclidean distances. This can be achieved without any loss of information by adding an additional component to the input vector of the network. We first convert the N-dimensional input vector X to an $(N+1)$-dimensional vector X^1 such that

$$X^1 = [X^{\mathrm{T}}, \|X\|^2]^{\mathrm{T}}. \tag{15}$$

Then define

$$V_i'^m = [(V_i^m)^{\mathrm{T}}, -1/2]^{\mathrm{T}} \tag{16}$$

and

$$g_i^m = \tfrac{1}{2}[(r_i^m)^2 - \|V_i^m\|^2]. \tag{17}$$

Since

$$\begin{aligned} -\|X - V_i^m\|^2 + (r_i^m)^2 &= 2(V_i^m)^{\mathrm{T}}X - \|X\|^2 \\ &\quad -\|V_i^m\|^2 + (r_i^m)^2 \end{aligned} \tag{18}$$

we have samples $X \in C_i^m$ if and only if

$$(V_i'^m)^{\mathrm{T}} X^1 + g_i^m > 0. \tag{19}$$

Although (19) with dot product conducts the same function as (9) with Euclidean distance operation, the weights modification in the later global training may have (19) deviating the Euclidean distance operation. This implies that the introduction of the dot product may give the LGT network more freedom for further training than keeping the Euclidean distance operation in the LGT network. This will be illustrated by examples of weights modification and non-spherical decision regions of the LGT network in the next section.

In practice, we multiply the weights and thresholds of the first layer by the smoothing parameter obtained in Eq. (14) and use the standard sigmoid transfer function with the unit smoothing parameter, which simplifies the global training procedure. The weight vectors W_s^1 and the thresholds b_s^1 of the first layer are therefore initialized as follows:

$$W_s^1 = t_i^m \cdot (V_i'^m)^{\mathrm{T}}, \tag{20}$$

$$b_s^1 = t_i^m \cdot g_i^m. \tag{21}$$

with

$$s = i + \sum_{j=0}^{m-1} q_j, \quad i \leqslant q_m, \quad q_0 = 0. \tag{22}$$

The weights w_{ms}^2 of the second layer between the hidden nodes s and output nodes m are given by

$$w_{ms}^2 = \begin{cases} 1 & \text{if } \sum_{j=0}^{m-1} q_j < s \leqslant \sum_{j=0}^{m} q_j \\ -1 & \text{others} \end{cases} \tag{23}$$

while the thresholds b_{ms}^2 are set to be -0.5.

Now our LGT network is constructed (initialized) with the dot product and the sigmoid transfer functions. The constructing procedure uses all information of the clusters obtained from the LTC algorithm (V_i^m, r_i^m, q_m and the class membership of the cluster). After this procedure, the network can be further trained with various standard methods (such as the BP, Quickprop or Rprop algorithms) by using the global set of training data. The final result is an enhanced LGT network. As the global training is only used to improve the generalization performance of the network, only few training epochs are needed while the effective initialization procedure reduces the problem of local minima.

If the cluster radius is controlled during LTC training such that each cluster contains no false learning samples and all clusters are used to construct the network, the least mean squared error of the LGT network can be made arbitrarily small with a sufficiently large value of sc. This means that the LGT network could overcome the problem of local minima. However, a small least mean squared error does not always equate to a small classification error on an independent test data set. In summary, to achieve optimal generalization performance, one should allow clusters to enclose samples of their inverse class, discard extremely small clusters and select a reasonable value for the constant sc.

4. Experimental results

This section illustrates the properties of the proposed LTC and LGT networks through several numerical experiments. It shows the placements and sizes of the clusters trained by LTC algorithm and the improvement of decision boundaries with global training and comparing training and classification performance with the K-means and LVQ1 [10] clustering algorithms as well as networks trained with the Rprop algorithm [23]. Rprop algorithm was selected for the global training of the LGT network since it is a gradient based method that has fast convergence properties compared to many other gradient based algorithm. Learning procedures were simulated by using MATLAB [1] and MATLAB's Neural Network Tool [24] on a Pentium IV 1.5 GHz PC.

For the LTC algorithm, the following parameters were fixed for all experiments: $lr1 = lr2(0) = 0.5$; $Ts = 20$. Only the parameter *nrmin* for the LTC algorithm and the constant *sc* for the LGT network were individually chosen for each experiment. We used the Rprop algorithm for global training of the LGT network with default training parameter values provided by MATLAB: learning rate $lr = 0.01$, increment to weight change $delt\_inc = 1.2$, decrement to weight change $delt\_dec = 0.5$, initial weight change $delta0 = 0.07$ and maximum weight change $deltamax = 50.0$ [24].

For a fair comparison, the Rprop algorithm was also used to train the MLP using sigmoid transfer functions and a single hidden layer, which were initialized with the Nguyen–Widrow initialization algorithm [25]. For terminology simplicity, we will call this the *Rprop network*. Training parameters are the same as those used for the LGT network except the number of the hidden nodes and training epochs. To compare the proposed LTC clustering algorithm with other representative clustering methods, K-means and LVQ clustering algorithm were also implemented in the experiments. We selected K-means and LVQ algorithms for comparison because they are well-known clustering methods and few parameters are required to be set by the user. For K-means and LVQ algorithm, the number of clusters for each class was assigned in proportion with the number of training samples for the class. Cluster centers for each class were initialized with Gaussian distributed random vectors with the same mean and variance as the training data composition for the class. For the K-means algorithm, each class was trained using only samples of their own class. The training will stop if the update of the clusters does not change cluster centers any more. After training with K-means algorithm, the 1-NNK (nearest neighbor classifier) with all trained clusters as its prototypes was used for classification. For LVQ training, training samples were presented to the network in random as this seems to produce better results.

To compare the proposed LGT network with networks constructed by LVQ and K-means clusters, each K-means and LVQ cluster was assigned a radius equal to half the distance between its center and the center of the nearest cluster that has different class membership. Clusters with these assigned radii were then used to construct a network for global training in the same way used for the proposed LGT network described in Section 3. For all training results, we report the mean values obtained from 50 repetitions. [2] For reference, the test data were classified by the 1-NNK with all training samples as prototypes.

4.1. The double spiral problem

The double spiral problem is a common benchmark for connectionist learning algorithms. According to Baum and Lang [26], a 2-50-1 BP network seems unable to find a correct solution to this problem. A 2-5-5-5-1 network with shortcut connections solved this problem but only after 20,000 training epochs using the BP algorithm [27], the resultant decision region is depicted in Fig. 5a. Fahlman's Cascade-Correlation meanwhile yielded correct solutions using between 12 and 19 hidden nodes after 1700 training epochs [28] with the decision region shown in Fig. 5b. Denoeux and Lengelle initialized the back propagation network with prototypes and solved this problem with a 3-20-1 network [1]. Perfect classification has been achieved after 1200 epochs of the accelerated back propagation algorithm with the associated decision region illustrated in Fig. 5c.

The LTC network was applied to the double spiral problem of 194 samples. For this two-class problem, we let the LTC algorithm generate clusters for only one class and create a network with one output node. Since the task is to achieve a perfect classification, the cluster radii were reset after training that they enclosed only correct samples and all clusters were reserved (i.e. $nr_{min} = 1$). An example of cluster placement by the LTC algorithm is illustrated in Fig. 6a. For this LTC network, regions inside the clusters are classified as one class and regions outside as the other class.

LGT networks with $sc = 2$ were trained using the Rprop algorithm. All networks in our 50 trials achieved perfect classification only after 50 epochs. An example of the decision region produced by the LGT network is depicted in Fig. 6b. The improvement of the decision region comparing to Fig. 6a is evident. By setting $sc = 1.5$, we get an almost perfect spiral decision region as shown in Fig. 6c after 80 epochs of training.

Table 1 reports the mean values of the training results of 50 trials of running for the LTC, LGT, Rprop, K-means and LVQ algorithms. While no training trial of Rprop(1) and Rprop(4) yielded the correct solution, 23 trials of Rprop(2) and 36 trials of Rprop(3) achieved perfect classification. Two examples of decision regions of Rprop networks that

[1] MATLAB is a registered trademark of The MathWorks, Inc.

[2] Examples of decision regions illustrated in the paper are obtained on a particular random run.

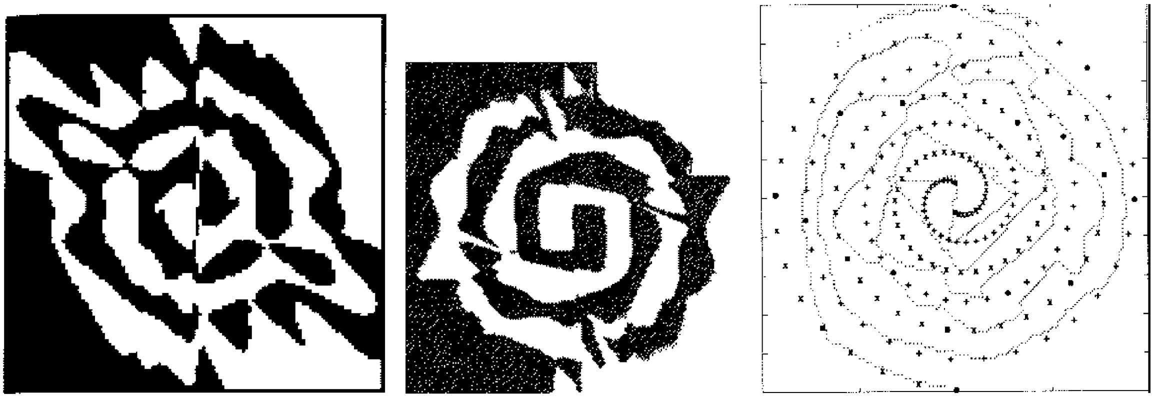


Fig. 5. Decision regions of (a) 2-5-5-5-1 network with shortcut connections and 20 000 epochs of BP training [27], (b) Cascade-Correlation with1700 epochs of training [28] and (c) 3-20-1 initialized network with 1200 epochs of accelerated BP training [13].

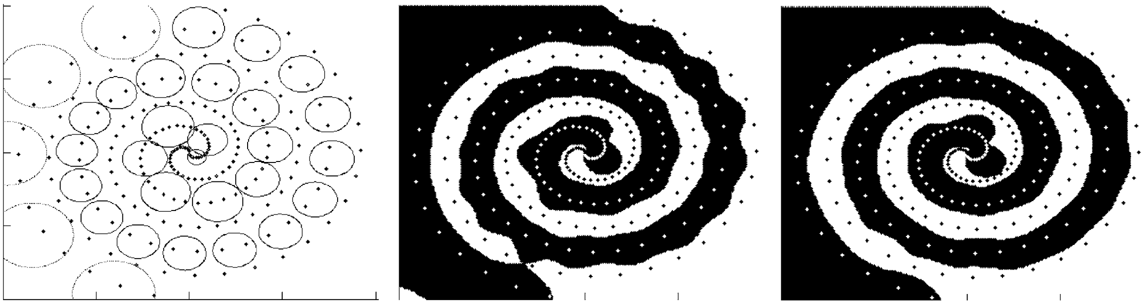


Fig. 6. (a) Cluster placement of the LTC algorithm for the double spiral problem with 194 points; decision regions of LGT network of 31 hidden nodes with (b) $sc = 2$ and 50 epochs of Rprop training and (c) $sc = 1.5$ and 80 epochs of Rprop training.

Table 1
Training records of the double spiral problem

Task	Networks/Algorithms	Numbers of hidden nodes	Training time		Classification errors (%)	Examples of generalization
			Epochs	Time (s)		
Spiral of 194 points	LTC	31.17	—	0.26	0	See Fig. 6a
	LGT($sc = 2/1.5$)	31.17	50/80	0.79/1.05	0/0	See Figs. 6b/c
	Rprop(1)/(2)	31	4000/5000	44.2 /54.8	4.15/1.03	—/See Fig. 7a
	Rprop(3)/(4)	40	4000/3000	60.4/ 46.2	0.42/1.96	See Fig. 7b/—
	K-means	32/64/96	10/12/6	0.33/0.44/0.36	29.9/24.7/13.4	—
	LVQ	32/64/96	300	192/245/304	26.3/18.6/5.78	—
	K-means-LVQ	32/64/96	10/11/9-300	193/246/305	20.6/6.86/5.27	—
	K-means-LVQ-Rprop	64	10-300-100	248	0.52	See Fig. 7c
Spiral of 776 points	LTC/LGT($sc = 1.5$)	32.36	—/80	0.41/2.58	0/0	See Figs. 8a/b
	Rprop(5)/(6)	32/40	5000/5000	132/157	1.97/1.35	—/See Fig. 8c
	K-means-LVQ	34/68/102	41/28/34-300	769/1023/1292	15.2/2.45/1.93	—
	K-means-LVQ-Rprop	68	31-300-100	1034	0.87	See Fig. 8d

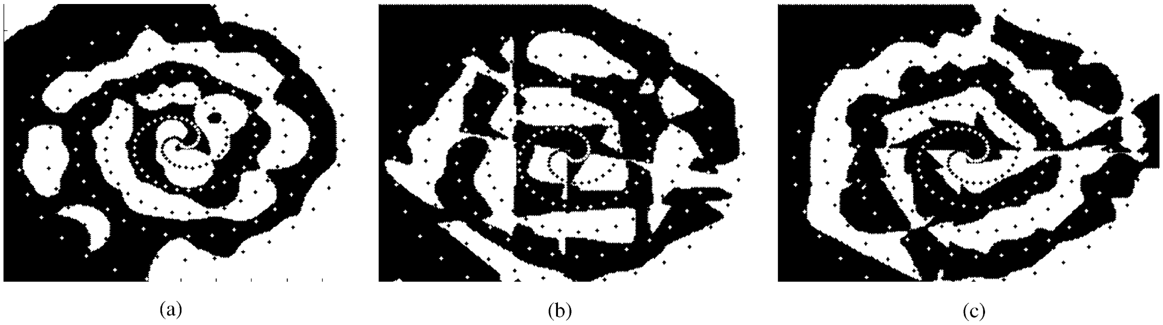


Fig. 7. Decision regions of Rprop networks with (a) 31 hidden nodes and 5000 epochs of training, (b) 40 hidden nodes and 4000 epochs of training and (c) of K-means-LVQ-Rprop network with 64 hidden nodes, $sc = 1.5$ and 10-300-100 epochs of training.

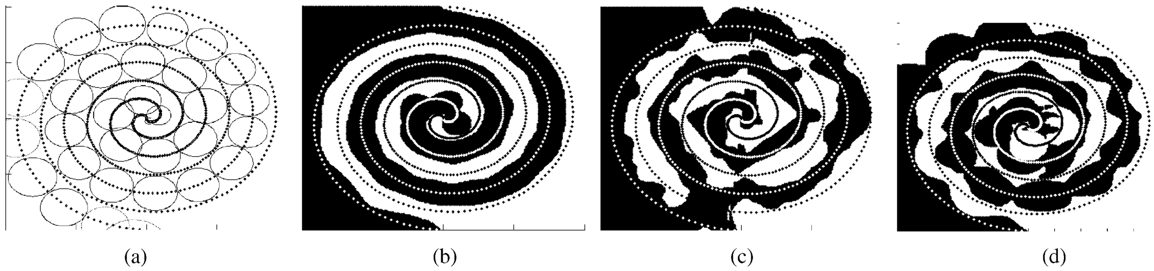


Fig. 8. (a) Cluster placement of the LTC algorithm for the spiral problem with 776 points; decision regions of (b) LGT network with 32 hidden nodes, $sc = 1.5$ and 80 epochs of Rprop training, (c) Rprop network with 40 hidden nodes and 5000 epochs of training and (d) K-means-LVQ-Rprop network with 64 hidden nodes, $sc = 1.5$ and 31-300-100 epochs of training.

produced perfect classification are shown in Fig. 7a and b, which are far from a perfect spiral shape. In our experiments, we found that much better results could be achieved if the clusters were trained first using K-means and subsequently LVQ, compared to K-means or LVQ alone, as shown by the K-means-LVQ algorithm in Table 1. Therefore, clusters trained by K-means-LVQ algorithm were used to construct a network that was further trained by Rprop algorithm in the way same as the LGT network. The training results of such network with $sc = 1.5$ (called K-means-LVQ-Rprop network) are given in Table 1 with an example of the decision region produced depicted in Fig. 7c.

To test the effect of increased number of training data on the algorithms, these experiments were repeated with 776 training samples with results being reported in Table 1. Fig. 8a and b illustrate an example of the LTC cluster placement and the LGT decision region, respectively. It is obvious that the significant increase in the training data did not upset the performance of the LTC and LGT networks. On average, only about one additional cluster was generated even though the number of training samples increased four fold. The Rprop network however seems to have much more difficulty with the increased training data. None of 50 training trials of either Rprop(5) or Rprop(6) achieved perfect classification. Examples of the decision regions of Rprop(6) and K-means-LVQ-Rprop with $sc = 1.5$ shown in Fig. 8c and d are clearly poorer than that produced by our LGT network.

These experiments clear demonstrate that the LGT network requires significantly shorter training time while producing better generalization results compared with various competing approaches for the double spiral problem. Although there are no test data for this problem, we can visually compare the generalization performance of different methods through the decision regions produced. Some recently developed neural network learning algorithms have been benchmarked using the double spiral problem [29,30] but the decision regions obtained are still inferior to the LGT network's performance obtained in this work.

4.2. A multi-model classification problem

Data for this multi-model classification problem were generated using 16 independent two-dimensional Gaussian random vectors $X_1, X_2, \ldots, X_{16}$ with different means, variances and a priori probabilities as shown in Table 2. Samples generated by X_1–X_8 were labeled as class one while those generated by X_9–X_{16} were labeled as class two. For this problem, the theoretical optimal Bayes classification error is calculated to be about 0.99% with the Bayes decision region illustrated in Fig. 9a. The nature of the training data enables us to test if small data clusters are ignored in the training

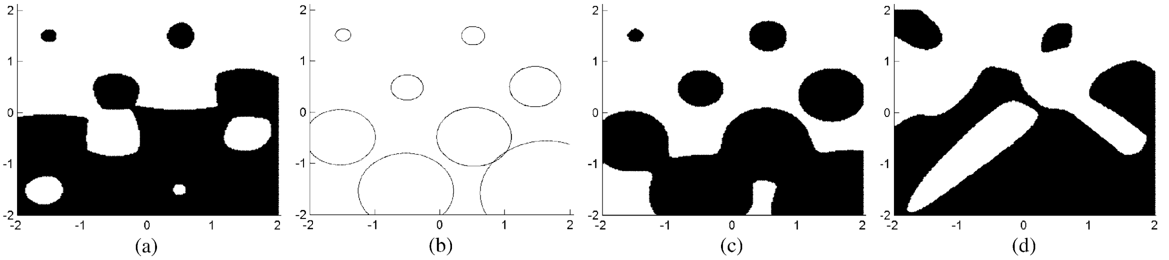


Fig. 9. (a) Bayes decision region, (b) cluster placement of the LTC algorithm with $nr_{min} = 7$ for clusters of one class; decision regions of (c) LGT network with 8 hidden nodes, $sc = 4$ and 30 epochs of Rprop training and (d) Rprop network with 10 hidden nodes and 2000 epochs of training.

Table 2
Parameters of Gaussian random vectors

Random vectors	Mean vectors	Square roots of variance	A priori probabilities
X_1/X_9	$(-1.5, 1.5)/(0.5, -1.5)$	0.0375	0.01389 (=1/72)
X_2/X_{10}	$(0.5, 1.5)/(-1.5, -1.5)$	2 × 0.0375	2 × 0.01389
X_3/X_{11}	$(-0.5, 0.5)/(1.5, -0.5)$	3 × 0.0375	3 × 0.01389
X_4/X_{12}	$(1.5, 0.5)/(-0.5, -0.5)$	4 × 0.0375	4 × 0.01389
X_5/X_{13}	$(-1.5, -0.5)/(0.5, 0.5)$	5 × 0.0375	5 × 0.01389
X_6/X_{14}	$(0.5, -0.5)/(-1.5, 0.5)$	6 × 0.0375	6 × 0.01389
X_7/X_{15}	$(-0.5, -1.5)/(1.5, 1.5)$	7 × 0.0375	7 × 0.01389
X_8/X_{16}	$(1.5, -1.5)/(-0.5, 1.5)$	8 × 0.0375	8 × 0.01389

procedure in favor of dominant clusters. 1440 samples were generated for training and after each training process, an independent data set of 144,000 samples was generated for testing. For such training and test data, the 1-NNK with all training data as prototype achieved the classification error on test sets of 1.53%.

The LTC algorithm with $nr_{min} = 7$ was used to generate clusters for a single class with an example of cluster placements produced illustrated in Fig. 9b. LGT networks were trained with $sc = 1.5$. However, after 50 epochs of training, the average classification errors on both the training and test set were found to be worse than that of the corresponding LTC networks (see Table 3). Increasing training epochs only reduced classification errors marginally. It seems that the LGT networks could not effectively inherit the knowledge acquired from prior LTC training because the value set for the smoothing constant sc was too small. This hypothesis was further strengthen by the fact that LGT networks with $sc = 4$ obtained better results only after 30 epochs of global training (see Table 3) with an example of the decision region produced depicted in Fig. 9c. The boundary improvements are however only marginal when compared with the LTC clusters generated. The Rprop(2) network meanwhile performed quite poorly as shown by its decision region in Fig. 9d. It could be seen that knowledge about small clusters of training data could not be learned even though the network had undergone 2000 epochs of training. Table 3 records various training and test results of Rprop networks with different number of hidden nodes and training epochs.

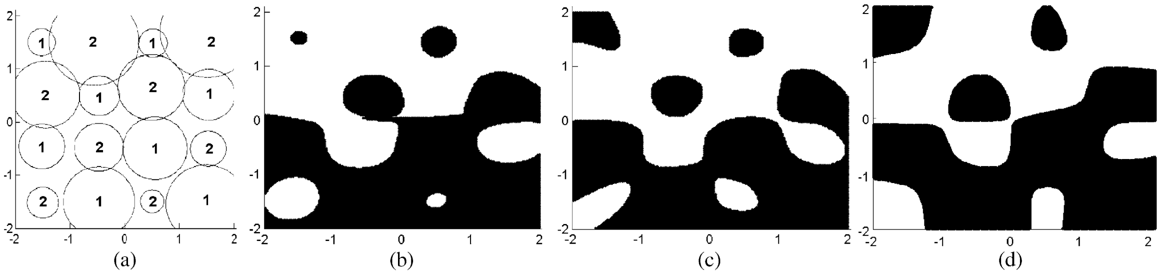
To make the LTC algorithm contribute more knowledge towards the LGT network, we used it to generate clusters for both classes. An example of cluster placements produced is shown in Fig. 10a with labeled class membership for each cluster. As could be seen, there are quite a number of overlapping regions where data are covered by clusters of belonging to different classes as well as regions not covered by any cluster at all. The LGT network successfully resolved these problems and produced decision region very similar to that of the optimal Bayes classifier as shown in Fig. 10b. For comparison, an example of the decision region produced by the Rprop(5) network is shown in Fig. 10c. Furthermore, we found that by using clusters of both classes, the LGT networks showed no problems in achieving good classification results with different values of sc (see Table 3). For the Rprop(5–7) networks, Rprop(7) showed effects of over-training with the lowest error rate on the training set but highest error rate on the independent test set (see Table 3). Training and test results for the K-means, LVQ, K-means-LVQ and K-means-LVQ-Rprop algorithms with different numbers of clusters are also listed in Table 3 while an example of the decision region using the K-means-LVQ-Rprop approach is shown in Fig. 10d.

In order to test the effects of using different values of nr_{min} on the performance of the network, we retrained the LTC network with $nr_{min} = 2$. For this case, 21.31 clusters were generated on average with a cluster placement example shown in Fig. 11a. The resultant LTC networks achieved an average classification error of 0.86% on the training set and 1.57% on the test set. Meanwhile, LGT networks with $sc = 2.5$ and trained with 50 epochs of Rprop, achieved an average classification error of 0.28% on the training set and 1.12% on the test set with a decision region example shown in Fig. 11b. One sees that the LGT network also well performed (classification error on the test set of 1.12%) although it was over-trained (classification error on the training set of 0.28%).

These empirical results above clearly show a significantly lower test set classification error rate for LGT networks

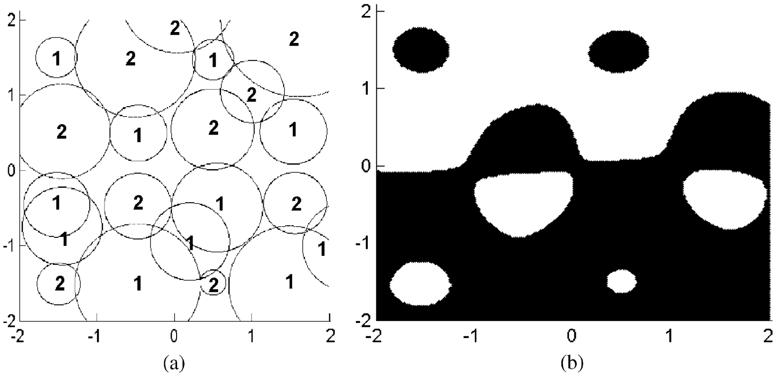
Table 3
Performance of networks for the multi-model classification problem

Networks	Numbers of hidden nodes	Training time		Classifi. errors on training sets (%)	Classifi. errors on test sets (%)
		Epochs	Time (s)		
LTC(1)/ LTC(2)	8.36/16.82	—	0.21/0.39	1.21/1.02	1.72/1.64
LGT(1)($sc = 1.5/4$)	8.36	50/30	1.18/0.89	1.94/0.71	2.26/1.12
LGT(2)($sc = 1.5/4$)	16.82	50/30	1.93/1.34	0.56/0.62	1.05/1.09
Rprop(1)/(2)	10	1000/2000	20.5/40.38	3.31/2.22	4.12/3.06
Rprop(3)/(4)	15	500/1000	13.8/26.7	2.02/0.81	2.37/1.63
Rprop(5)/(6)/(7)	20	500/300/600	16.7/10.2/20.0	0.67/0.76/0.47	1.23/1.48/1.63
K-means	16/20/24	13/13/15	1.02/1.08/1.13	8.41/7.78/6.79	8.93/7.94/7.14
LVQ	16/20/24	50	197/206/216	9.19/7.22/5.02	10.26/7.28/5.16
K-means-LVQ	16/20/24	12/14/14/−50	198/207/217	8.11/6.71/4.14	8.66/6.85/4.27
K-means-LVQ-Rprop	24	14-50-100	225	0.63	1.52



(a) (b) (c) (d)

Fig. 10. (a) Cluster placement of the LTC algorithm with $nr_{min} = 7$ for clusters of both classes; decision regions of (b) LGT network with 16 hidden nodes, $sc = 1.5$ and 50 epochs of Rprop training, (c) Rprop network with 20 hidden nodes and 500 epochs of training and (d) K-means-LVQ-Rprop network with 24 hidden nodes, $sc = 1.5$ and 13-50-100 training epochs.



(a) (b)

Fig. 11. (a) Cluster placement of the LTC algorithm with $nr_{min} = 2$ for clusters of both classes and (b) decision region of LGT network with 21 hidden nodes, $sc = 2.5$ and 50 epochs of Rprop training.

achieved with substantially reduced training time compared with competing Rprop, K-means and LVQ approaches. Furthermore, experiments also demonstrate the robustness of LGT networks to different values of nr_{min} and smoothing constant sc as long as clusters of the both classes are used to construct the networks.

Table 4 lists the weights and biases of sample LTC and LGT networks for this learning task. Although the difference of weight magnitudes between the LTC and LGT networks is significant, knowledge transfer from the LTC network is evident from the similar signs of corresponding weights for both networks (exception being the biases of the first layer).

Table 4
Weights and biases of the LTC and the corresponding LGT networks

LTC network trained with $nr_{min} = 7$				LGT network ($sc = 1.5$, after 50 epochs of Rprop training)				
$W_1^1, \ldots, W_{16}^1$		$b_1^1, \ldots, b_{16}^1$	W_1^2, b_1^2	$W_1^1, \ldots, W_{16}^1$			$b_1^1, \ldots, b_{16}^1$	W_1^2, b_1^2
1.4885	−1.5375	0.7344	1	4.0595	−4.4564	−1.2826	−5.4293	2.3702
0.5851	−0.4952	0.6510	1	3.8401	−4.6233	−0.5018	−0.2478	2.1476
1.5412	0.5152	0.4300	1	12.6614	3.7982	−4.0975	−9.9557	8.9640
−0.4645	−1.5103	0.7812	1	0.5786	−21.2692	−0.5338	1.8863	4.7073
−1.5233	−0.4553	0.4060	1	−13.7766	−5.3244	−4.9919	−11.2169	3.5769
−0.4684	0.4956	0.3734	1	−5.8937	5.3699	−7.5615	−0.4369	13.2057
0.5076	1.4872	0.2697	1	10.1727	30.7147	−10.5127	−24.5525	15.8206
−1.5249	1.4993	0.2551	1	−35.3146	33.7207	−11.3452	−52.0506	19.2293
−0.5103	1.4867	0.7811	−1	−1.2808	4.6301	−0.3497	−1.1028	−2.0747
1.5270	−0.4998	0.3290	−1	21.4047	−5.7079	−6.9260	−17.1750	−19.5038
1.5556	1.5563	0.7352	−1	4.0272	4.4272	−1.1551	−6.3004	−4.4206
−0.4680	−0.4601	0.2621	−1	−11.2833	−9.2719	−11.3995	−2.7146	−18.7471
−1.4542	0.5229	0.6298	−1	−5.5045	6.0064	−1.5145	−3.9867	−3.6655
0.4701	0.5528	0.5524	−1	3.6045	4.4388	−4.1826	−0.8931	−7.7200
0.4986	−1.4888	0.1664	−1	26.9165	−80.7624	−27.1137	−65.8550	−15.4203
−1.4955	−1.5054	0.2757	−1	−29.7446	−29.2777	−9.6711	−44.1128	−11.6775
			−0.5					−0.8172

4.3. Glass problem in Poben1 data sets

Proben1 is a collection of real world problems with actual data [31] to benchmark neural network learning algorithms. We particularly selected the "glass problem" from the Proben1 data sets because it was reported to show complex decision boundaries and heavy overlapping of classes, which severely test the ability of a classifier in achieving good generalization [32]. In addition, the glass problem is to classify the nine-dimensional data into six classes, which is different from the two previously examined problems in Sections 4.1 and 4.2 (classifying two-dimensional data into two classes). The problem essentially lies in classifying glass based upon the description of its splinters. For algorithm evaluation, the dataset of 214 samples was divided into three sets, i.e., the test, training and validation sets, which are built from three different partitions of the dataset and contain 50%, 25% and 25% of the total data samples, respectively. This partition is applied to three different orders of the whole dataset, leading to three different sequences for training, glass1, glass2 and glass3 [31]. The 1-NNK with all training samples as prototypes achieves test data classification errors of 37.74%, 35.85% and 22.64% for the glass1, glass2 and glass3.

In training the LTC network, we set $nr_{min} = 2$. Only the training data sets (exclusive validation sets) were used for cluster training and clusters of all classes were used to construct the network. Meanwhile for LGT training, we set $sc = 3$. The LGT network was trained with Rprop algorithm until the least mean squared error on the validation set increased for 20 epochs. The network weights at the minimum validation error rate were then used for the classification of the test data. This procedure was similarly applied in the Rprop training of networks initialized with the Nguyen–Widrow and K-means algorithm. Table 5 records the training and test results for glass1, glass2 and glass3.

For glass3 problem, the 1-NNK achieves surprisingly much better classification error on test data than all other classifiers implemented in this work. Nevertheless, it could be seen from Table 5 that our LGT approach performed significantly better for all the three glass problems than the K-means, LVQ, Rprop and K-means-Rprop networks in terms of classification errors on both the training and test sets. In addition, classification errors on both the training and test sets of glass1, glass2 and glass3 achieved by the LGT network in this work are much lower than the results reported in Refs. [31,32]. Although the global training epochs of LGT networks were less than that of Rprop networks initialized with the Nguyen–Widrow algorithm, the absolute training time of the LGT network was slightly longer for glass2. This is due to the early stopping of the Rprop training procedure for this data set with the application of the validation method.

5. Conclusion

A new method of constructing and training feed-forward neural networks for difficult pattern recognition problems is developed. Network construction is based on using clusters, which are generated and trained sequentially using the local subsets of the training data set. The proposed local training algorithm can rapidly learn about distinct local data subsets without being affected by the complicated global

Table 5
Performance of networks for the glass problems (glass1, glass2 and glass3)

Data sets/networks	Hidden nodes	Training times		Classification errors on the training set (%)			Classification errors on the test set (%)		
		Epochs	Times	Min	Mean	Max	Min	Mean	Max
1/LTC	12.17	—	0.17	15.36	21.24	31.46	33.56	36.74	42.72
1/LGT	12.17	46.75	0.61	8.41	17.61	24.30	22.64	28.04	32.71
1/Rprop	12	72.36	0.78	17.36	23.41	42.63	24.42	32.56	47.06
1/K-means	12	5.12	0.24	28.97	38.75	53.27	32.30	44.98	60.38
1/LVQ	12	50	14.02	37.38	42.52	53.27	33.19	44.38	55.73
1/K-means-Rprop	12	5.1-45.4	0.94	11.44	18.85	25.76	22.76	35.79	49.06
2/LTC	12.34	—	0.18	18.33	25.56	43.56	32.25	36.37	48.46
2/LGT	12.34	48.70	0.64	6.78	15.33	21.34	26.30	34.68	41.36
2/Rprop	12	51.83	0.62	18.84	28.54	54.47	35.96	48.74	58.15
2/K-means	12	4.74	0.21	29.31	42.61	49.25	35.85	49.17	58.49
2/LVQ	12	50	14.11	34.58	48.75	55.80	49.06	52.27	56.61
2/K-means-Rprop	12	4.7-51.2	1.02	7.55	16.56	21.98	30.19	39.44	49.17
3/LTC	12.75	—	0.17	12.34	20.05	26.53	26.42	32.43	41.51
3/LGT	12.75	46.56	0.65	10.28	14.72	21.23	25.43	31.54	40.17
3/Rprop	13	72.38	0.87	18.20	26.28	45.23	29.47	43.32	64.31
3/K-means	13	4.72	0.23	28.04	39.44	54.21	35.85	48.98	60.38
3/LVQ	13	50	14.09	37.38	43.27	57.94	52.83	60.76	79.24
3/K-means-Rprop	13	4.7-47.9	0.98	11.55	15.79	26.42	28.31	37.21	56.60

training data structure. It therefore overcomes the drawbacks of monolithic network training methods, which depends on the average characteristics of the entire training data set. Given the values of its learning parameters, the algorithm automatically determines the number of clusters, which correspond to the number of hidden units of the resultant LGT network. The number of LTC clusters tends to be minimal since clusters are permitted to enclose samples of their own as well as their inverse classes and the training algorithm always attempts to have a cluster growing to enclose more and more samples within the Bayes decision region. This effectively reduces the network size besides increasing its generalization ability.

However, since hypersphere clusters are used, the LTC network can only form piecewise spherical decision regions that limit the generalization ability. To overcome this drawback, the LTC network is converted to a network that both inherits the knowledge of the LTC network and is capable of further training using established training methods operating on the global training set. The resulting LGT network converges rapidly due to its inherited knowledge with good generalization ability from the global training.

We believe our proposed algorithm mimics the knowledge discovery approach of the human brain, which typically decomposes a complicated concept or idea into simpler subsets and learning the details of each subset sequentially before integrating and generalizing all the individual pieces of knowledge acquired. The effectiveness of the LGT network has been amply demonstrated through its superior results in terms of accuracy and learning speed (compared with other representative clustering and learning approaches implemented in this work) on three benchmark problems operating on both synthetic and real data sets.

References

[1] T. Denoeux, R. Lenglle, Initializing back propagation networks with prototypes, Neural Networks 6 (1993) 351–363.

[2] P. Burrascano, Learning vector quantization for the probabilistic neural network, IEEE Trans. Neural Networks 2 (1991) 458–461.

[3] H.G.C. Traven, A neural network approach to statistical pattern classification by ‘semi-parametric’ estimation of probability density functions, IEEE Trans. Neural Networks 2 (1991) 366–377.

[4] C.L. Scofield, D.L. Reilly, Into silicon: real time learning in a high density RBF neural network, IJCNN’91, Seattle, Washington, 1991, pp. 551–556.

[5] M.T. Musavi, W. Ahmed, K.H. Chan, K.B. Faris, D.M. Hummels, On the training of radial basis function classifiers, Neural Networks 5 (1992) 595–603.

[6] D.F. Specht, Enhancements to probabilistic neural networks, IJCNN’92, 1992, pp. 761–768.

[7] W. Pedrycz, Conditional fuzzy clustering in the design of radial basis function neural networks, IEEE Trans. Neural Networks 9 (1998) 601–612.

[8] R. Kumar, P. Rockett, Multiobjective genetic algorithm partitioning for hierarchical learning of high-dimensional pattern spaces: a learning-follows-decomposition strategy IEEE Trans. Neural Networks 9 (1998) 822–830.

[9] S. Grossberg, Studies of Mind and Brain, Reidel, Boston, 1982.

[10] T. Kohonen, The self-organizing map, Proc. IEEE 78 (1990) 1464–1480.

[11] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973.

[12] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.

[13] H. Spath, Cluster Analysis Algorithms for Data Reduction and Classification of Objects, Elis Horwood Publishers, New York, 1980.

[14] J. Moody, C.J. Darken, Fast learning in networks of locally tuned proceeding units, Neural Comput. 1 (1989) 281–294.

[15] R.O. Duda, H. Fassum, Pattern classification by iteratively determined linear and piecewise linear discriminate functions, IEEE Trans. Electron. Comput. 15 (1966) 220–232.

[16] Y.H. Kong, A.S. Noetzel, The piecewise linear neural network: training and recognition, IJCNN90, Vol. III, San Diego, CA, 1990, pp. 245–250.

[17] D.L. Reilly, et al., A neural model for category learning, Biol. Cybern. 45 (1982) 35–41.

[18] S.E. Fahlman, Ch. Lebiere, The cascade-correlation learning architecture, Advances in Neural Information Processing System, Vol. 2, Morgan Kaufmann Publishers, Los Aleos, CA, 1990, pp. 524–532.

[19] R.S. Sutton, Two problems with backpropagation and other steepest descent learning procedures for networks, Proceedings of the Eighth Annual Conference on Cognitive Science Society, 1986, pp. 823–831.

[20] D.C. Plaut, G.E. Hinton, Learning sets of filters using backpropagation, Computer Speech Languages 2 (1987) 35–61.

[21] N.E. Sharkey, A.J.C. Sharkey, An analysis of catastrophic interference, Connection Sci. 7 (3) (1995) 310–329.

[22] A.R. Webb, S. Shannon, Shape-adaptive radial basis functions, IEEE Trans. Neural Networks 9 (1998) 1155–1166.

[23] M. Riedmiller, H.A. Braun, A direct adaptive method for faster backpropagation learning: The RPROP algorithm, IEEE ICNN-93, San Francisco, CA, 1993, pp. 586–591.

[24] The Math Works, Inc., Neural Network Toolbox User's Guide: For Use with MATLAB, 1998.

[25] D. Nguyen, B. Widrow, Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights, IJCNN'90, Vol. 3, 1990, pp. 21–26.

[26] E.B. Baum, K.J. Lang, Constructing hidden units using examples and queries, In: P. Lippman, J.E. Moody, D.S. Touretzky (Eds.), Advances in Neural Information Processing Systems, Vol. 3, Morgan Kaufman, San Mateo, CA, 1991, pp. 904–910.

[27] K.J. Lang, M.J. Witbrock. Learning to tell two spirals apart, Proceedings of the Connectionist Models Summer School, 1988, pp. 52–59.

[28] S.E. Fahlman, An empirical study of learning speed in back-propagation networks, Proceedings of the 1988 Connectionist Models Summer School, 1988, pp. 17–26.

[29] S. Young, T. Downs, CARVE—A constructive algorithm for real-valued examples, IEEE Trans. Neural Networks 9 (1998) 1180–1190.

[30] S. Behnke, N.B. Karayiannis, Competitive neural trees for pattern classification, IEEE Trans. Neural Networks 9 (1998) 1352–1369.

[31] L. Prechelt, PROBEN 1—A set of neural network benchmark problems and bench marking rules, Technical Report 21/94 Fakultät für Informatik, Universität Karlsruhe, Germany, 1994.

[32] D. Heinke, F.H. Hamker, Comparing neural networks: a benchmark on growing neural gas, growing cell structure, and fuzzy ARTMAP IEEE Trans. Neural Networks 9 (1998) 1279–1291.

About the Author—XUDONG JIANG received the B.E. and M.E. degree from the University of Electronic Science and Technology of China in 1983 and 1986, respectively, and received the Ph.D. degree from the University of German Federal Armed Forces Hamburg, Germany in 1997, all in Electrical and Electronic Engineering. From 1986 to 1993, he was a Teaching Assistant and then a Lecturer at the University of Electronic Science and Technology of China where he received two Science and Technology Awards from the Ministry for Electronic Industry of China. He was a recipient of the German Konrad-Adenauer Foundation young scientist scholarship. From 1993 to 1997, he was with the University of German Federal Armed Forces Hamburg, Germany as a scientific assistant. From 1998 to 2002, He was with the Centre for Signal Processing, Nanyang Technological University, Singapore, first as Research Fellow and then as Senior Research Fellow. Currently he is a Senior Member of Research Staff and appointed as research leader for project Fingerprint Recognition at the Laboratories for Information Technology, Singapore. His research interest includes pattern recognition, neural networks, image processing, computer vision, biometrics, adaptive signal processing and spectral analysis.

About the Author—ALVIN HARVEY KAM SIEW WAH received his B.E. (Electrical) degree with 1st Class Honours from the University of Malaya, Malaysia, in 1996, graduating top of his class. He later obtained his Ph.D. degree in the area of computer vision in 2000, from the University of Cambridge, United Kingdom where his studies was fully funded by the Cambridge Commonwealth Trust, Trinity College and the Kuok Foundation. He is currently an associate research member and research leader at the Laboratories of Information Technology, Singapore. His research interest includes computer vision, machine learning, intelligent systems and artificial intelligence.