Multicolumn RBF Network

Ammar O. Hoori, *Student Member, IEEE*, and Yuichi Motai, *Senior Member, IEEE*

***Abstract*— This paper proposes the multicolumn RBF network (MCRN) as a method to improve the accuracy and speed of a traditional radial basis function network (RBFN). The RBFN, as a fully connected artificial neural network (ANN), suffers from costly kernel inner-product calculations due to the use of many instances as the centers of hidden units. This issue is not critical for small datasets, as adding more hidden units will not burden the computation time. However, for larger datasets, the RBFN requires many hidden units with several kernel computations to generalize the problem. The MCRN mechanism is constructed based on dividing a dataset into smaller subsets using the *k-d tree* algorithm. *N* resultant subsets are considered as separate training datasets to train *N* individual RBFNs. Those small RBFNs are stacked in parallel and bulged into the MCRN structure during testing. The MCRN is considered as a well-developed and easy-to-use parallel structure, because each individual ANN has been trained on its own subsets and is completely separate from the other ANNs. This parallelized structure reduces the testing time compared with that of a single but larger RBFN, which cannot be easily parallelized due to its fully connected structure. Small informative subsets provide the MCRN with a regional experience to specify the problem instead of generalizing it. The MCRN has been tested on many benchmark datasets and has shown better accuracy and great improvements in training and testing times compared with a single RBFN. The MCRN also shows good results compared with those of some machine learning techniques, such as the support vector machine and k-nearest neighbors.**

***Index Terms*— Deep neural network, k-d tree, k-nearest neighbors (KNNs), kernel, radial basis function networks (RBFNs).**

I. Introduction

WHILE the radial basis function network (RBFN) shows good performance for many complex problems in classification, it still suffers from an excessive amount of computations and slow convergence, particularly for large datasets [1]. Selecting the best centers for Gaussian hidden units will be more difficult and time-consuming due to kernel inner products. Yu *et al.* [2] suggested the incremental selection of hidden units through their error correction (ErrCor) algorithm as a way to reduce the computation time. ErrCor is a good method of selecting the most violating input vector as a new hidden unit center incrementally until convergence. This method has a fast and good start when designing an RBFN, but the process is delayed exponentially when the number of selected hidden units is increased.

This paper presents the multicolumn RBF network (MCRN) as a method to improve the accuracy and timing results of the RBFN using a multicolumn deep technique. Small pretrained RBFNs are deployed in a parallel structure. Those RBFNs are pretrained on portions of a dataset individually. During testing, a small number of individual artificial neural networks (ANNs) are selected using the k-nearest neighbors (KNNs) technique. Only the selected ANNs contribute to an averaged output. The MCRN has shorter training and testing time requirements compared with those of the RBFN and shows improved accuracy compared with the RBFN, support vector machine (SVM), and KNN.

Although the ErrCor algorithm [2] shows good results in speeding up the training convergence of the RBFN compared with other training algorithms, an excessive amount of computations are required. The ErrCor continuously inserts one hidden unit into the hidden layer each time until convergence while using the entire dataset in each step of the training phase to calculate the root mean square error (RMSE). The violated vector is chosen as the new hidden unit center and is removed from the training dataset. Inner product calculations become more difficult for the next round of selecting a new violated vector due to the increased number of hidden units. The process is initially fast and efficient, but the training time increases exponentially as the number of hidden units increases. In contrast, ErrCor shows good performance compared with other fast algorithms with respect to testing time. This superior performance is obvious for the RBFN, because selecting the most violating input vectors as hidden units gives the ANN a wide distribution to generalize the solution instead of randomly selecting hidden units. Moreover, ErrCor guarantees fewer hidden units by intelligently selecting important Gaussian centers. Having fewer units decreases the inner-kernel products and thus decreases the testing time. However, the ErrCor structure is difficult to deploy in a parallel environment. However, parallelism, if it could be applied, the system performance would improve dramatically. Our approach solves the critical problem of delays, primarily through the use of a parallel structure to increase the training and testing speed and improve the accuracy. It also decreases the use of hidden units at each RBFN by decreasing the number of training dataset instances. This will decrease the inner products as well.

Using a parallel-structured ANN as a deep technique shows superior results compared with the traditional ANN [3], [4]. The use of a smaller subset of data speeds up the process of selecting hidden units and decreases the number of hidden

Manuscript received April 21, 2016; revised October 27, 2016 and December 15, 2016; accepted January 2, 2017. This work was supported in part by the Higher Committee of Education Development of Iraq and in part by the National Science Foundation CAREER under Grant 1054333.

A. Hoori is with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284 USA, and also with the Department of Computer Engineering, University of Baghdad, Baghdad, Iraq (e-mail: hooriao@vcu.edu).

Y. Motai is with the Department of Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, VA 23284 USA (e-mail: ymotai@vcu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2017.2650865

units. While this superiority is true during training, this technique is more feasible during the testing. Fewer computations and a parallelized structure speeds up the entire process [5]. Each individual ANN can be considered as a specialist ANN for any input vector that belongs to its subset. The results are considerably more accurate when regional training is performed.

This paper is novel in that the MCRN introduces a solution to RBFN processing delays and large computational problems by: 1) using a well-developed and easy-to-use parallel structure and 2) using fewer hidden units per ANN and less dataset instances to train ANNs. The MCRN divides the entire dataset into small but informative subsets individually. The MCRN uses those subsets to train N individual ANNs. Those individual ANNs are stacked in parallel. They consider test instances that are under their responsibility, i.e., instances under their subset space. The MCRN is considered an ideal way to transition from a single-structured learning to a deep and parallel learning mechanism due to the separation of individual ANNs. The MCRN consists of three consecutive stages: the *input subset selector*, N–individual ANNs, and the *output combiner*. During testing, when a new instance is presented, the *input subset selector* box calculates the KNNs to that input. Each k neighbor belongs to a subset of the entire dataset, which is used to train individual ANNs. Only those k ANNs are executed. The *output combiner* averages their results to obtain the single elected output.

The remainder of this paper is organized as follows. The relevant studies are discussed in Section II. Section III presents the proposed deep kernel neural network in detail. Section IV presents and discusses the experimental results of the proposed methods. The conclusion of this paper is presented in Section V.

II. Relevant Studies on the RBFN and Deep Neural Networks

Due to the good performance exhibited by the RBFN and the greater problem solving involved with deep ANNs, researchers have attempted to enhance the RBFN and find different ways to apply it to deep learning.

A. RBFNs With Hidden Layer Kernels

Many researchers have shown that the RBFN can yield good results compared with other ANN techniques. They manipulated the RBFN in different ways to improve its performance. Their works were based on modifying the parameters of hidden units' radial basis functions, such as the centers or width, or even suggesting different Gaussian functions. Some other researchers focused on keeping parameters unchanged while speeding up the training process by finding ways to minimize the number of hidden units, which means minimizing the inner kernel products.

Kaminski and Strumillo [1] worked on optimizing the computations for training the RBFN. They used standard Gram–Schmidt orthonormalization to calculate weights by transforming the RBF kernel functions into orthonormal functions. The method showed good accuracy compared with traditional RBFNs.

TABLE I
Improving RBFN Results

Modification method	Purpose	Studies
Using Gram-Schmidt orthonormalization.	To optimize computations	[1]
Modifying RBF center and width.	To conduct hidden unit selection	[6]–[9]
Updating using LM algorithm.	To improve centers' position, width and weight	[10]
Compact two-step extension procedure.	To optimize computations	[11]
Incremental RBF training.	To decrease the number of hidden units	[2]

Panchapakesan *et al.* [6] studied the effect of moving centers of the RBFN and how to obtain smaller errors while fine-tuning the positions of the center. Bruzzone and Prieto [7] used a supervised technique for RBFN classifiers. Their technique considers the class memberships of training samples to select the centers and widths of the kernel functions associated with the hidden units of an RBFN. Mao and Huang [8] selected the hidden layer units of the RBFN based on the data structure preservation criterion, and Bors and Pitas [9] proposed what is the median radial basis function (MRBF) algorithm. The MRBF employs the marginal median for kernel location estimation and the median of the absolute deviations for scale parameter estimation.

Xie *et al.* [10] proposed an improved second-order algorithm (ISO) to train the RBFN. The ISO is used for adjusting centers, width, and weights. They updated the parameters of the RBF using the Levenberg–Marquardt algorithm.

Arif and Vela [11] noted the computational problem that arises in certain applications after training due to executing through a kernel of the size of the training set. Thus, they proposed a compact two-step extension procedure to resolve this issue. The extension exploits the universal approximation capabilities of generalized RBFNs to approximate and replace the projections onto the empirical kernel map used during execution.

Yu *et al.* [2] proposed an offline algorithm for incrementally constructing and training RBFN. In their work, the maximum violating vector from training instances is added as a new hidden unit to the RBFN structure at each iteration of the ErrCor algorithm. This vector, which represents the highest peak of the error surface, is eliminated from the training dataset. This process is repeated until convergence. The results demonstrate that the ErrCor algorithm can design a compact RBFN compared with other RBF algorithms.

Table I shows different ways of improving the results of the RBFN from different studies.

This paper uses the ErrCor algorithm suggested in [2] to train individual RBFNs by incrementally inserting a single hidden unit at a time and calculating the RMSE until the required tolerance is reached.

B. Deep Learning With ANN Techniques

Deep ANNs have been proven to be able to implement functions of higher complexity, which are able to address more difficult problems than shallow ANNs [12]. Szymanski and

McCane [13] compared a shallow ANN architecture with a deep ANN architecture. They showed that depth is an effective encoder of repeating patterns in the data and that deep ANNs can generalize and perform better than shallow ANNs.

Different implementations have been suggested to define the word "deep" in the ANN world. Implementations are performed using different ANN structures and connection ideas. Some studies considered using the convolutional neural network (CNN). Many researchers used the deep belief ANN (DBN) based on the restricted Boltzmann machine (RBM), whereas some considered a deep ANN to be one large ANN with many hidden layers and/or many hidden units, and others used a multicolumn structure as their deep structure.

Krizhevsky *et al.* [14] trained a large deep CNN to classify high-resolution images. Shuiwang *et al.* [15] developed a 3-D CNN to recognize human actions in real-world environment videos.

In 2006, Hinton and Salakhutdinov [16] proposed a deep learning architecture called the DBN for autoencoder neural networks using the RBM structure. In the same year, Hinton *et al.* [17] used a network with three hidden layers to implement the DBN using the RBM structure. Wong and Sun [18] proposed a new feature extraction method called regularized deep fisher mapping (RDFM), which learns an explicit mapping from a sample space to a feature space using a deep ANN. Stuhlsatz *et al.* [19] suggested an approach for the feature extraction method called generalized discriminant analysis using a deep learning ANN (GerDA DNN). In their work, they used the RBM as an unsupervised preoptimization for the ANN structure. Salakhutdinov *et al.* [20] introduced a hierarchal-deep model as a new deep learning ANN model. They demonstrated how a hierarchical Dirichlet process can learn prior over the activities of the top-level features in a deep RBM. Bu *et al.* [21] proposed a multilevel 3-D shape feature extraction framework using deep learning. Their deep learning structure is based on a DBN, which consists of multistage RBM models.

Chen and Salman [22] proposed a novel deep neural architecture for learning speaker-specific characteristics. They used an unsupervised multilayer feed forward ANN based on a deep autoencoder architecture.

Van De Steeg *et al.* [23] used a multilayer perceptron ANN with enlarged hidden layers and hidden units as their deep structure to solve the Tic-Tac-Toe 3-D game problem. Their deep structure with integrated pattern detectors outperforms smaller ANN structures.

Shao *et al.* [3] proposed multispectral neural networks to learn features from multicolumn deep ANNs. Their results indicated that spectrally embedding deep ANNs exhibit a lower error rate compared with a single deep neural network. Ciresan *et al.* [4] used the concept of a multicolumn deep ANN to improve image classification. Their work is based on gathering and averaging many parallel ANN outputs. Mall *et al.* [24] also used the concept of dividing a large dataset into sparse subsets using KNN. They used the fast and unique representative subset selection technique to obtain the points from different dense subsets. Those selected points are mapped into the original dataset to capture the intrinsic cluster structure present in the data. Those clusters contribute to the overall classifier output.

TABLE II
SUMMARY OF DEEP LEARNING TECHNIQUES

Method	Deep Structure	Studies
CNN	Multi-stage feed forward ANN.	[14], [15]
Deep belief ANN	RBM ANN.	[16]–[21]
DNA	Multi feed forward hidden Layers.	[22]
Single ANN	Enlarge hidden layers and hidden units.	[23]
Multi-column ANN	Parallel ANNs.	[3], [4], [24]

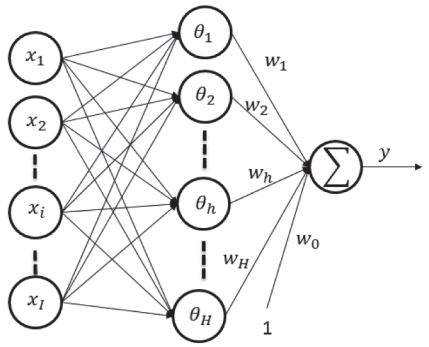

Fig. 1. ANN with I inputs, H RBF units, and a single output.

Table II provides a summary of research that used deep learning as a technique through different methods of implementation.

Some studies have used ensemble learning to divide the dataset into smaller subsets to reduce the difficulty of large-scale data [24], [25]. The consideration of a small subset may either speed up the overall learning process or yield better results. In this paper, the general concept of deep learning is based on the multicolumn method of distributing parallel ANNs, and the datasets are divided using the *k-d tree* algorithm.

III. MULTICOLUMN RBF NETWORK

This section is organized as follows. In Section III-A, a brief discussion of the RBFN is provided. Section III-B presents how the dataset is divided into subsets using the *k-d tree* algorithm. The MCRN structure and mechanism are described in detail in Section II.

A. Radial Basis Function ANN

Fig. 1 shows the standard internal structure of the RBFN. It consists of three layers: input, hidden, and output layers. The input layer has I units, denoted as $\mathbf{x} = [x_1, x_2 \ldots, x_i, \ldots x_I]$. The hidden layer has H RBF units, represented by $\theta = [\theta_1, \theta_2, \ldots, \theta_h, \ldots \theta_H]$. The output layer has a single unit, y. Each hidden unit h is calculated using the kernel function of RBF units [1], [2] as follows:

$$\theta_h(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_h\|^2}{\sigma_h}\right) \quad (1)$$

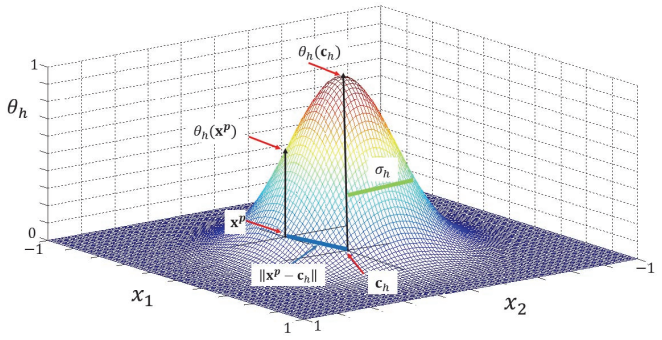

Fig. 2. 2-D radial basis function with center $\mathbf{c}_h$ and width σ_h. $\theta_h(\mathbf{x^p})$ is the output of neuron h when input vector $\mathbf{x^p}$ is applied.

where $\mathbf{c}_h$ and σ_h are the center and width of unit h, respectively, and the operation $\|\cdot\|$ is the Euclidean norm.

Fig. 2 shows a 2-D radial basis function that represents how each hidden unit h calculates θ_h for a new input $\mathbf{x}^p$, where p is the pattern number.

The ANN output is calculated by solving the following linear equation:

$$y = \sum_{h=0}^{H} w_h \theta_h(\mathbf{x}) \tag{2}$$

where w_h represents the weight between the h hidden unit and the single output unit; w_0 represents the bias weight between an input $\theta_0 = 1$ and the output unit for simplicity. The output equation shown in (2) has a linear function that sums all of the products from the previous layer and displays it as a single output [2].

The RBFN is trained offline with a training set $\{\mathbf{x}^p, y_d^p\}$, $p = 1, \ldots P$, where P is the number of training set pairs and y_d is the desired output. $e_p(\mathbf{w})$ in (3) is the error between the desired output y_d^p and network output $y(\mathbf{x}^p)$ when applying the input $\mathbf{x}^p$ to the ANN as follows:

$$e_p(\mathbf{w}) = y_d^p - y(\mathbf{x}^p) \tag{3}$$

where $\mathbf{w}$ is the matrix of all RBFN weights.

The performance criterion used to measure the convergence of the network is the RMSE $E(\mathbf{w})$ expressed as

$$E(\mathbf{w}) = \sqrt{\frac{1}{P}\sum_{p=1}^{P}[e_p(\mathbf{w})]^2}. \tag{4}$$

The network hidden weights are calculated by solving a set of P linear equations

$$y_d^p = \sum_{h=0}^{H} w_h \theta_h(\mathbf{x}_p). \tag{5}$$

Using matrix notation, the solution of (5) is

$$\mathbf{w} = \mathbf{Q}^+ \mathbf{y_d} \tag{6}$$

where $\mathbf{y_d}$ is the vector of the desired output, and $\mathbf{Q}^+ \in \mathbb{R}^{H \times P}$ is the pseudoinverse matrix of all H hidden unit functions θ throughout all P desired outputs.

The center $\mathbf{c}_h$ of the h hidden RBF unit is selected from input vectors $\mathbf{x}$ such that $\mathbf{c}_h \subset \mathbf{x}$. An efficient goal is to select the fewest vectors that perfectly generalize the problem.

It is not easy to determine which center should be selected or how many hidden units are needed to obtain the best RMSE. Considerable research has been conducted to improve RBF training [2], [1], [6]–[10]. However, the method used in this paper is to select the maximum violating input vector to be the center of the new RBF hidden unit [2], as shown in (7)

$$\mathbf{c}_{h+1} = \mathbf{x}^p, \forall \mathbf{x}^p = \arg\max_{\mathbf{x}^p}(|e_p(\mathbf{w})|) \quad . \tag{7}$$

Those new centers are incrementally inserted into the RBFN structure at each epoch using the incremental insertion method [2]. In this method, the error cost function, $E(\mathbf{w})$, is calculated at each epoch, and the maximum violating input vector will be inserted into the RBFN hidden layer as the new hidden unit center. This vector will be removed from the training dataset for the next epoch. The hidden layer of the RBFN increases by one unit upon an epoch. The process of selecting and inserting new units continues until the RMSE value converges toward a tolerance goal.

B. Subsets and the k-d Tree Algorithm

Better RBFN performance might be achieved by increasing the number of hidden units and/or hidden layers. The number of hidden units in an RBFN depends on the number of training dataset vectors selected. Increasing the number of hidden units will improve the network performance. More kernel functions will increase the smoothness of the separation surface, which will make instances more separable. Unfortunately, computation of those inserted kernels will be more difficult and time-consuming. Excessive computations and wasteful memory use occur repeatedly each time a new hidden unit is inserted until a satisfactory tolerance is achieved. Even with such compelling results, the resultant RBFN may become a large-structured ANN, which leads to high computational operations during testing.

In this paper, the number of selected hidden units is reduced by dividing the input space of the training dataset into subsets based on their overall dataset density. By overall density, we refer to the ratio of certain class instances to all instances in a specific region. Each subset will be a stand-alone training dataset for individual ANNs, as shown in Section III-B.

The concept of chopping a multidimensional training set is based on the *k-d tree* algorithm [26], which is used to divide a large dataset into small subsets. The *k-d tree* algorithm prevents zero-data subsets and ensures a well-distributed training set for each subset. Only a few features are used in the *k-d tree* chopping process. Those chopping features are selected such that the resultant subsets will have a sufficient number of instances to train an ANN and have the same density as the original dataset density to ensure consistent behavior. For simplicity and explanation purposes, consider the input space $\mathbb{R}^I$ to be $\mathbb{R}^3$ with a two-class classification problem. Thus, $\mathbf{x}$ consists of only three features x_1, x_2, and x_3, and each instance has a single output y. A random example of an entire space

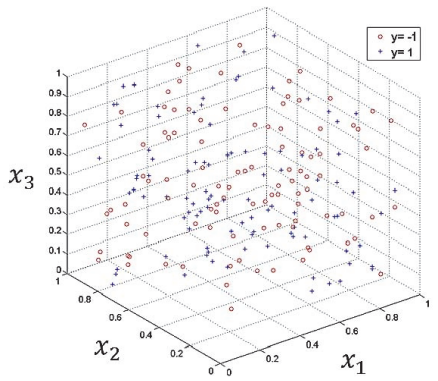

Fig. 3. Example of a dataset with three features x_1, x_2, and x_3. Each instance has a single output y.

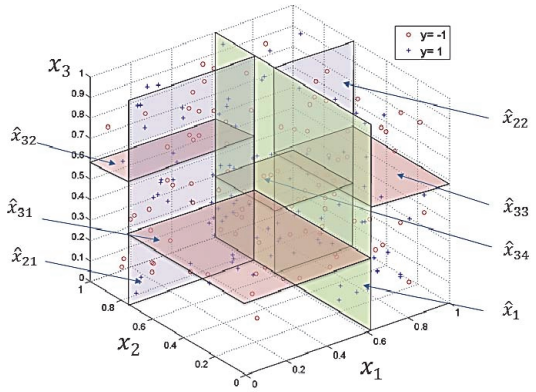

Fig. 4. Example of dividing the entire dataset based on the median of each dimension using *k-d tree* algorithm.

with 3-D features is shown in Fig. 3 with a two-label state (1 is represented as crosses, and -1 is represented as circles).

The *k-d tree* algorithm [26] considers all data as unlabeled data. It uses the median of each dimension to chop the training dataset into two subsets with approximately equal density. Fig. 4 shows the steps of chopping the entire dataset into small regions.

First, the *k-d tree* algorithm measures the median $\hat{x}_1$ of x_1 values of all data $\mathbf{x}$ in the entire space R. Then, the *k-d tree* chops R into two regions R_1 and R_2 based on $\hat{x}_1$, as shown in

$$\begin{aligned} R_1 &= \{\forall \mathbf{x} \in R : x_1 \leq \hat{x}_1\} \\ R_2 &= \{\forall \mathbf{x} \in R : x_1 > \hat{x}_1\}. \end{aligned} \tag{8}$$

The next step is to individually divide each region (R_1 and R_2) into two smaller regions based on the medians ($\hat{x}_{21}$ and $\hat{x}_{22}$) of the second dimension x_2. The result is four smaller regions R_{11} and R_{12} and R_{21} and R_{22}, as shown in (9)

$$\begin{aligned} R_{11} &= \{\forall \mathbf{x} \in R_1 | x_2 \leq \hat{x}_{21}\} \\ R_{12} &= \{\forall \mathbf{x} \in R_1 | x_2 > \hat{x}_{21}\} \\ R_{21} &= \{\forall \mathbf{x} \in R_2 | x_2 \leq \hat{x}_{22}\} \\ R_{22} &= \{\forall \mathbf{x} \in R_2 | x_2 > \hat{x}_{22}\}. \end{aligned} \tag{9}$$

In the same manner, the generated regions are divided by their own medians $\hat{x}_{31}$, $\hat{x}_{32}$, $\hat{x}_{33}$, and $\hat{x}_{34}$ based on the third parameter values x_3. Division can start over with the x_1 dimension until N regions are generated. Those N regions define N subsets as $S = \{s_1, s_2, \ldots, s_n, \ldots, s_N\}$, which are divided with nearly equal density, as shown in Fig. 5.

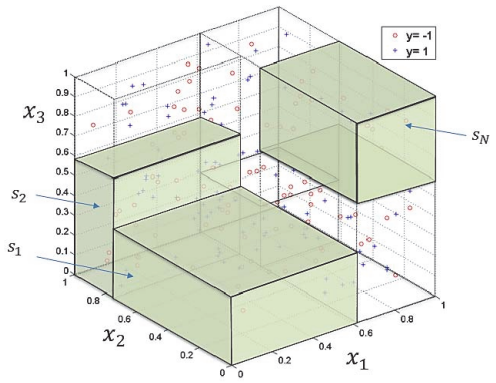

Fig. 5. Example of dividing an entire space into N subsets $\{s_1, s_2, \ldots, s_n, \ldots, s_N\}$.

Generally, each subset is bounded by inequalities based on the surrounding medians, as shown in (10)

$$\begin{aligned} s_1 &= \{\forall \mathbf{x} \in R | (x_1 \leq \hat{x}_1) \wedge (x_2 \leq \hat{x}_{21}) \wedge (x_3 \leq \hat{x}_{31})\} \\ s_2 &= \{\forall \mathbf{x} \in R | (x_1 \leq \hat{x}_1) \wedge (x_2 > \hat{x}_{21}) \wedge (x_3 \leq \hat{x}_{32})\} \\ &\ldots \\ s_N &= \{\forall \mathbf{x} \in R | (x_1 > \hat{x}_1) \wedge (x_2 \leq \hat{x}_{22}) \wedge (x_3 > \hat{x}_{33})\}. \end{aligned} \tag{10}$$

Each chopping action in any dimension duplicates the number of subsets. The number of total subsets is $N = 2^{\xi}$, where ξ represents how many divisions are made in all dimensions.

The *k-d tree* algorithm ensures that there are sufficient and well-distributed data at each subset. Division is also limited by the size of the training dataset. A small-sized dataset may result in scarce data at each subset S. A less informative dataset produces a poorly trained ANN. Therefore, this issue must be considered, and the number of subsets N should be selected carefully. In other words, the *k-d tree* algorithm chops the dataset into subsets with no information regarding the labels of the instances, and the density condition keeps the distribution of the subsets near the density of the original dataset.

Each individual subset n is used as a separate dataset to train a corresponding n ANN using RBFN structures. Small margins can be added from neighboring subsets to increase the learning information, as shown in Fig. 6. Although small margins will merely increase the number of training dataset instances for each individual ANN, it will provide more informative data to ensure a well-trained ANN. The ANN has insufficient information regarding the instances that lay on the borders of the subset. Adding instances beyond the border to the training dataset will give the ANN a good generalization to respond to border instances. Those added instances will only be used for training purposes.

The process of chopping the dataset into an N subset and using those subsets to train N corresponding individual ANNs is further explained in Algorithm 1.

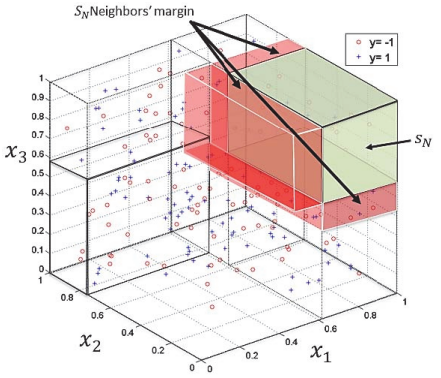

Fig. 6. Adding small margins from each adjacent neighbor to the current subset $\mathbf{S_N}$.

Algorithm 1 Dividing the Data Set Into N Subsets and Training Them

1: Input: ξ = (number of chopping process), $\{\mathbf{x}, y_d\}$ = (training dataset), (MDL: ANN training model)
2: Output: = $2^{\xi} \times MDL$ trained structure of individual ANNs.
3: $N = 2^{\xi}$ (number of resultant subsets and ANNs)
4: Compute original dataset density B
5: for $chop$ = 1 to ξ
6: for $subset$ = 1 to $(2^{chop} - 1)$
7: Find each feature median $\hat{x_i}$
8: Compute average density β_i for chopped subsets
9: Select feature i with minimum $|\beta_i - \mathrm{B}|$
10: Chop along median $\hat{x_i}$ dataset into two subsets.
11: end for
12: end for
13: for $subset$ = 1 to N
14: Train $MDL(subset)$ using $\{\mathbf{x}, y_d\} \subset \{subset + Neibghbors\ margin\}$
15: end for

C. MCRN Method

All trained ANNs are gathered and stacked in a multicolumn structure, named the MCRN structure, as shown in Fig. 7. Once a new testing data vector $\tilde{\mathbf{x}}$ is applied, the *input subset selector* will forward $\tilde{\mathbf{x}}$ to the appropriate ANNs. Each one of the selected k ANNs gives its own output $\tilde{y}_k$ to the *output combiner*, which will calculate the single output y, as explained in the steps detailed in the following.

1) Input Subset Selector: When a test input vector $\tilde{\mathbf{x}}$ is presented, only k ANNs are selected based on the KNN algorithm. By determining the Euclidean distance between the new testing data $\tilde{\mathbf{x}}$ and all training datasets $\mathbf{x}^P$, only the k nearest points with the k smallest Euclidean distances d_k are chosen, as in (11). Those k points belong to k subsets, as shown in Fig. 8

$$\mathbf{d}_k = \min_{1..k}(\|\tilde{\mathbf{x}} - \mathbf{x}^p\|) = \min_{1..k}\left(\sum_{i=1}^{} (\tilde{\mathbf{x}}_i - x_i^p)^2\right) \quad (11)$$

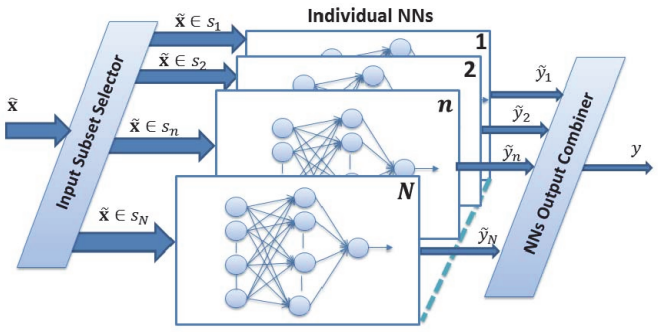

Fig. 7. Internal structure of the MCRN with the *input subset selector*, N individual ANNs and the *output combiner*.

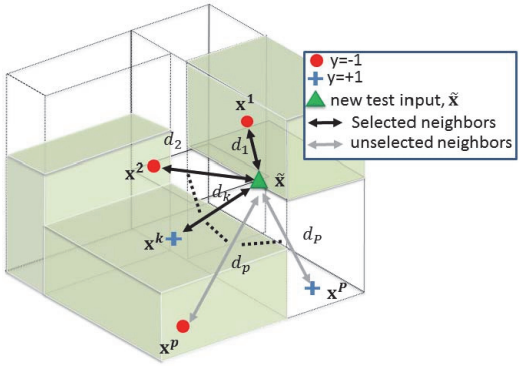

Fig. 8. Selecting k subsets, and hence, k neural networks, based on the KNN algorithm.

where $\mathbf{d}_k = \{d_1, d_2, \ldots, d_k\}$; d_k is the kth minimum Euclidean distance between new testing point $\tilde{\mathbf{x}}$ and the kth training point $\mathbf{x}^p$; $\tilde{\mathbf{x}}_i$ is the ith value of vector $\tilde{\mathbf{x}}$, and x_i^p is the ith value in the pth pattern of vector $\mathbf{x}$.

Based on the k nearest points, there are only k selected subsets δ_k; hence, only k ANNs are selected to be executed as follows:

$$\delta_k = \forall s_n, s_n \subset S, \mathbf{x}^k \in s_n, \mathbf{x}^k = \arg\min_{x^p} \|\tilde{\mathbf{x}} - \mathbf{x}^p\| \quad (12)$$

where δ_k is the kth selected subsets from all S subsets.

2) Individual ANNs : Each test input $\tilde{\mathbf{x}}$ has k selected RBFN output results. Those ANNs work individually in parallel to produce k results y_k; therefore, (2) becomes (13) for each k ANN

$$\tilde{y}_k = \sum_{h=0}^{} w_h^k \theta_h^k(\tilde{\mathbf{x}}) \quad \forall \delta_k. \quad (13)$$

The resultant output of each k selected ANNs is $\tilde{y}_k$, which gives the decision of those ANNs for a given input $\tilde{\mathbf{x}}$.

3) Output Combiner: Only k ANNs are selected; the remaining $N - k$ ANNs are not used for each new entry $\tilde{\mathbf{x}}$. The outputs of those k ANNs, $\tilde{y}_k$, contribute to the overall output decision y. The overall MCRN output y is calculated based on the average sum of all k outputs, as in (14). In classification problems, the real value output y should be hard limited to

give the 1 and −1 class values

$$y = \frac{1}{k} \sum_{i=1}^{k} \tilde{y}_i. \tag{14}$$

In fact, this equation still works as a single large RBFN compared with $y = \sum_{h=0}^{H} w_h \theta_h(\mathbf{x})$ shown in (2). Equation (2) has a linear relation between kernels $\theta_h(\mathbf{x})$ and output y through the weights w_h. Although that is true, the relation can also be held for $\tilde{y}_k = \sum_{h=0}^{H} w_h^k \theta_h^k(\tilde{\mathbf{x}})$ shown in (13), which has the same linear relation for each k selected ANNs.

To prove that the MCRN total output y still has a linear relation with weights, substituting y_k from (13) into (14) yields

$$y = \frac{1}{k} \sum_{i=1}^{k} \sum_{h=0}^{H} w_h^i \theta_h^i(\tilde{\mathbf{x}}). \tag{15}$$

Assume a vector $\psi_j(\tilde{\mathbf{x}})$ to be all hidden layer kernels of all k ANNs, as shown

$$\psi_j(\tilde{\mathbf{x}}) = \theta_h^i(\tilde{\mathbf{x}}), \quad j = 1 + h + H \times (i - 1).$$

Now also consider the vector υ_j representing all weights between hidden units and outputs for all k ANNs to be

$$\upsilon_j = \frac{1}{k} w_h^i, \quad j = 1 + h + H \times (i - 1).$$

Therefore, (15) becomes

$$y = \sum_{j=1}^{k \times (H+1)} \psi_j(\tilde{\mathbf{x}}) \upsilon_j \tag{16}$$

which represents a single large RBFN with input $\tilde{\mathbf{x}}$ and output y. Compared with (2), (16) works the same manner, regardless of how many ANNs are inside or how many ANNs are selected.

IV. Experimental Evaluation

This section is organized as follows. In Section IV-A, the characteristics of used datasets and the criteria of using them are explained. Section IV-B presents RBFN results for those datasets. Section IV-C presents results of applying the *k-d tree* algorithm to each dataset to prepare small subsets for individual training. Section IV-D discusses MCRN results after plugging trained individual ANNs. Section IV-E compares MCRN versus RBFN speed during training and testing. In Section IV-F, MCRN results are compared with other well-known classifiers.

A. Data and Criteria

In this paper, the RBFN and MCRN are tested using the different benchmark UCI datasets [27] shown in Table III. The variety of parameters that each dataset has increases the difficulty of training the ANN. A larger training dataset requires additional computations and hidden units. For simplicity, the Letter dataset is trained and tested with a single output letter "A". Considering a 26-output MIMO system is equivalent to considering 26 MISO systems. The Urban dataset [28], [29] has nine different classes, which represent the land-cover objects in an urban area. We use only the 'asphalt' class in our experiments. The Occupancy dataset is obtained from time-stamped pictures that were taken every minute [30].

TABLE III
Dataset Parameters

Dataset name	Features	Classes	Total instances	Training /Testing
Iris	4	3	150	120/30
Glass-Id	9	2	214	170/44
Seeds	7	3	210	168/42
Liver	5	2	341	306/35
Wisconsin	9	2	699	560/139
Thyroid	21	3	7200	5760/1440
Hepatitis	19	2	155	124/31
Ionosphere	34	2	351	280/71
Urban	148	9*	675	168/507
Occupancy	5	2	10808	8143/2665
Satimage	36	6	6435	4435/2000
Letter	16	26*	20000	16000/4000

*A single output is used: only 'asphalt' for the Urban dataset and only letter 'A' for the Letter dataset.

TABLE IV
Results of Testing a Single RBFN

Dataset	Tolerance	Hidden units	Accuracy	Recall
Iris	0.0001	91	93.3	90
Glass-Id	0.0001	145	81.8	66.7
Seeds	0.0001	126	89.7	85.4
Liver	0.01	274	65.7	60
Wisconsin	0.0001	324	91.4	90.6
Thyroid	0.01	3790	96.4	94.2
Hepatitis	0.0001	95	87.1	88.5
Ionosphere	0.0001	108	91.6	100
Urban	0.01	71	90.9	48.3
Occupancy	0.01	38	97	92.6
Satimage	0.02	500	95.5	87.1
Letter	0.0001	1510	97.1	100

The datasets in Table III are listed in ascending order from lower to higher memory use. Those datasets are diverse both in the number of features and/or the number of instances. Some datasets, such as Thyroid, Hepatitis, Ionosphere, Satimage, and Letter, have many features, while others, such as the Thyroid, Occupancy, Satimage, and Letter datasets, have many instances. Each dataset is divided into a training set and testing set, as shown in Table III. Some datasets are divided with an approximated ratio (80% for training and 20% for testing), whereas the Urban, Satimage, and Occupancy datasets are originally separated.

B. Traditional RBFN Training

The RBFNs in this paper are trained based on incremental insertions of the most violating input vectors until convergence is achieved [2]. For each dataset, the RMSE is calculated at each step until it meets a tolerance value, which is set as a goal to stop the learning process, as shown in Table IV. Table IV shows each dataset with the number of hidden units in each RBFN structure after tolerance is achieved. It also shows the accuracy and recall results for each dataset.

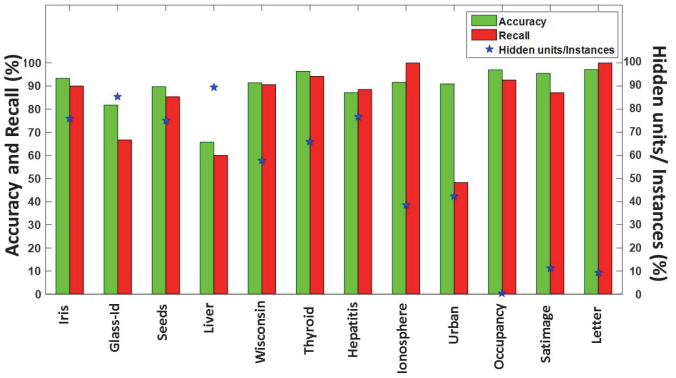

Fig. 9. Comparison of the RBFN results for accuracy, recall, and hidden units/dataset instances for each dataset.

The overall results demonstrate the good performance of the RBFN structure. Those results are achieved by incrementally inserting training set instances as the centers of added hidden units. This insertion means the kernel inner products will be increased. For a small dataset, adding new hidden units will not overly burden computations. For example, with the Iris dataset, 91 hidden units out of 120 training dataset instances represent 75.8% of the used instances as hidden units. This large ratio is still considered acceptable because the inner products of 120 input vectors by 91 hidden units will not take a long time or occupy a considerable amount of storage.

The difficulty increases when using larger datasets, such as Thyroid, Letter, and Satimage. Each training insertion step requires several computations and high memory usage. With such a large dataset, even a small ratio of hidden units and instances requires many computations. For instance, 1510 hidden units out of 16 000 training instances are used with the Letter dataset, equating to a ratio of 9.4%, which is considerably smaller than the Iris ratio. Moreover, increasing the number of hidden units will affect not only the training time but also the testing time. The problem of inner-product computations is still present for each testing instance. Moreover, a neat and light RBFN structure is as important as good results. To improve the RBFN, one must consider two important challenges: how to decrease the number of hidden units and how to obtain better results. These issues will be addressed throughout this paper.

Fig. 9 shows how many hidden units are used compared to the number of overall training instances for each dataset as well as a bar plot for the accuracy and recall results obtained using these RBFN structures for each dataset.

The high hidden units/instances ratios shown in Fig. 9 for small datasets, such as Iris, Glass-Id, Seeds and Liver, indicate that the RBFN requires considerable information (training set instances) to reach a reasonable tolerance with good accuracy and recall results. However, lower ratios for Occupancy, Letter, or Satimage are sufficient to achieve the required tolerance with good performance. Although small ratios are considered good, the large number of hidden units requires many computations to calculate the kernel inner products of each hidden unit and all training set instances.

A fair comparison between the RBFN and MCRN results is achieved by using the same conditions and same training set instances, as shown in Sections IV-C–IV-F. The accuracy and speed are used as key factors to compare the results.

TABLE V
EFFECT OF DIVIDING DATASETS INTO SUBSETS IN TERMS OF DENSITY

Dataset Experiment	Original Dataset Density*	Average Subset Density	Minimum Subset Density	Maximum Subset Density
Iris-2**	33.3	27.1	11.6	42.6
Iris-4**	33.3	32.9	0	70.9
Glass-Id-2	22.9	24.7	21.1	28.4
Glass-Id-4	22.9	27.7	9.7	52.6
Seeds-2**	32.7	27.8	18.3	39.3
Seeds-4**	32.7	33.9	14.3	51.2
Liver-2	41.5	43.4	40	46.8
Liver-4	41.5	40	35.2	42.8
Wisconsin-2	65.7	61.9	45.3	78.5
Wisconsin-4	65.7	56.9	21.5	93
Wisconsin-8	65.7	49.3	19.4	97.7
Thyroid-2**	2.5	2.3	1.7	2.9
Thyroid-4**	2.5	2	1.1	3.2
Thyroid-8**	2.5	2.2	0	4.4
Hepatitis-2	21.8	21.1	15.2	26.9
Hepatitis-4	21.8	10.8	0	27.5
Ionosphere-2	63.2	70.5	63.2	77.9
Ionosphere-4	63.2	70	56.7	82.9
Urban-2	8.3	7.1	6.3	7.9
Urban-4	8.3	8	4.8	10.7
Occupancy-2	21.2	19.9	11.5	28.4
Occupancy-4	21.2	19.9	15.2	29.6
Satimage-2**	24.2	27.3	24.2	30.5
Satimage-4**	24.2	26.5	14.9	35.4
Letter-2	3.9	3.7	3.4	4
Letter-4	3.9	3.6	2	5.1
Letter-8	3.9	3.9	1.5	6.6

* Density is the ratio of the positive labels in the training set.
**For simplicity purpose, the density results are shown for the first class only, while the training is performed for all classes.

C. Applying the k-d Tree Algorithm to the Datasets

To prepare a dataset for the MCRN, each training set is divided into semi-equal subsets using the *k-d tree* algorithm. A set of data is chopped in a specific dimension (i.e., feature) based on the median value of that feature. Many experimental cases are made to divide each dataset, as shown in Table V. Each experiment name has a dataset name followed by a hyphen and numerical value. The numerical value represents how many subsets are used in that experiment. For example, Thyroid-8 represents the original training set of Thyroid divided into 8 subsets. The results in Table V show the effect of dividing original datasets into subsets in terms of density. Here, density is the ratio of the number of the 1 class to the number of all classes in a training dataset.

The results in Table V show that a preferable chopping is considered when the average of the density of the resultant subsets is slightly affected by division. This shows that the resultant subsets have adequate information to train a small ANN. Small margins are added to each subset from its neighbor subsets. This guarantees a more regional generalized training for individual ANNs to avoid the risk of intermittent training of those instances, particularly on boundaries between subsets.

Each experiment shows that the original density value differs slightly from the average density value, whereas the

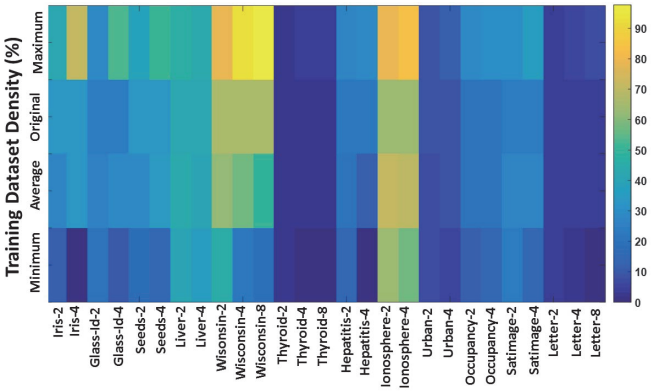

Fig. 10. Variation in dataset density due to the division of each dataset into two and four subsets for each experiment.

TABLE VI
RESULTS OF TESTING THE MCRN FOR EACH EXPERIMENT IN TABLE V

Dataset Experiment	Maximum Hidden units among N subsets*	Chopping Features in sequence	k	Accuracy	Recall
Iris-2	76	4	5	94.4	93.1
Iris-4	62	3,1	3	97.8	96.7
Glass-Id-2	113	2	5	84.1	85.7
Glass-Id-4	103	2,7	1	86.4	87.5
Seeds-2	87	6	7	88.9	86.8
Seeds-4	67	6,2	7	91.3	89.7
Liver-2	224	3	1	71.4	69.2
Liver-4	155	6,5	1	68.6	61.5
Wisconsin-2	323	5	7	95	95.6
Wisconsin-4	204	2,6	3	97.1	97.8
Wisconsin-8	164	2,6,2	5	97.1	98.9
Thyroid-2	3701	18	1	95.7	92.5
Thyroid-4	3341	18,1	5	96.3	94
Thyroid-8	2871	18,19,12	7	96.5	93.9
Hepatitis-2	44	17	1	96.8	100
Hepatitis-4	46	17,1	5	96.8	100
Ionosphere-2	150	8	3	93	100
Ionosphere-4	109	8,3	3	91.6	100
Urban-2	58	44	3	92.7	66.7
Urban-4	41	44,76	3	92.9	66.7
Occupancy-2	49	5	1	97.6	95
Occupancy-4	38	5,2	1	93.2	95
Satimage-2	300	34	5	95.9	89.4
Satimage-4	240	34,14	7	95.7	88.8
Letter-2	500	12	3	96.6	100
Letter-4	501	12,10	1	96.5	100
Letter-8	431	12,10,11	1	96.4	100

*Maximum number of hidden units is good indicator of how big the individual ANN is because all other ANNs are less time consuming.

minimum or maximum values differ considerably from the original density, as shown in Fig. 10. Each dataset is chopped as previously explained in Section II-B. The sequence of chopping features is shown in Table VI. At each chopping step, the best feature to use is the one that shows the best matching between the average subsets' density and original density. The difference in density is greater in small datasets because division is limited by the number of features and instances, as shown for Iris-2 and Iris-4. The resultant density is less affected in large-scale datasets, such as Thyroid-2, Thyroid-4, Thyroid-8, Urban-2, Urban-4, Occupancy-2, Occupancy-4, Letter-2, Letter-4, and Letter-8. In some cases, such as Iris-4, Thyroid-8, and Hepatitis-4, the minimum subset densities are zero. Each one of those zero-density subsets has training instances with single-labeled values. Therefore, training an RBFN for such subset will result in a small RBFN structure with few hidden units. With such single-class cases, one can use a simple mathematic relation to represent the subset functionality. However, to maintain the generality of this paper, we keep the RBFN training as our choice, even for such straightforward cases.

D. MCRN Results

Each experiment in Table V has N subsets, and each subset is used to train an individual ANN using the RBFN training method. Thus, each experiment has N trained individual ANNs. During training, the same RBFN conditions in Table IV, such as tolerance and training set instances, are kept to guarantee consistency while comparing the results.

After training, resultant N individual RBFNs are stacked in parallel in the MCRN structure, as shown in Fig. 7; this is done for each experiment in Table V separately. During testing time, the *input subset selector* selects the KNN vectors. Those selected vectors belong to k subsets. Those k subsets are used to train k individual ANNs. The results of applying the test input to those k ANNs are combined and averaged by the *output combiner*. The number of k should be odd to break the tie and should be between 1 and 7. The MCRN results for each experiment are shown in Table VI.

The results show that the maximum number of hidden units for the new individual ANNs is less than that of the RBFN shown in Table IV for all experiments, except the Ionosphere and Occupancy datasets. The MCRN accuracy and recall results are better than those of the RBFN. Some results, such as those for Thyroid and Letter, have no improvement but are still considered good because dividing the one large ANN into two or more sub-ANNs will increase the process in a parallel environment. The MCRN outperforms the RBFN in many cases, such as for Iris, Glass-Id, Seeds, Liver, Wisconsin, Ionosphere, Hepatitis, Satimage, and Urban.

The promising results shown in Table VI suggest that the MCRN can yield comparable or better results than the RBFN in many experimental cases. There are many inner products overall in individual ANNs, but those inner products are only in k selected ANNs. This means that the MCRN has fewer overall inner computations than the RBFN. The number of hidden units for each individual ANN is less than the number of its subset instances. The maximum number of hidden units is equal to the subsets instances in the worst case. During the testing phase, the MCRN selects k individual ANNs, and other $N - k$ ANNs are inactive for this specific test input vector. Each individual ANN is executed in a single processor in the parallel environment. In this case, the delay in the execution time is the delay caused by the largest individual ANN, which is considerably lower than the fully connected traditional RBFN time. Furthermore, all the k neighbors may lay in the same subset. In this special case, the test instance is solely affected by the ANN that was trained using this subset.

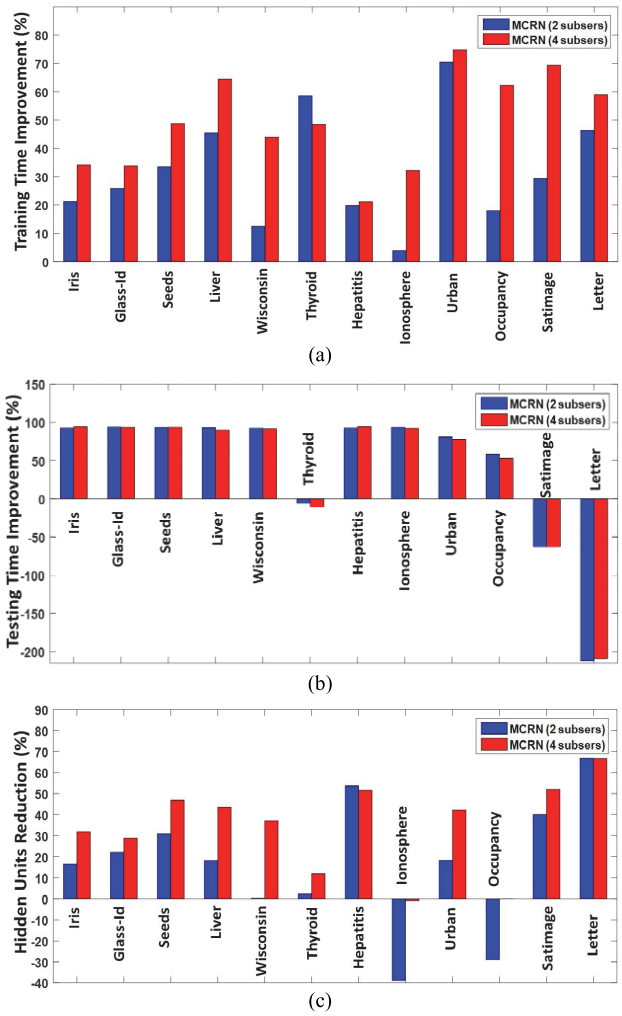

Fig. 11. Percentage comparisons for the MCRN (two subsets) and MCRN (four subsets) over the RBFN for each dataset according to (a) training time improvement, (b) testing time improvement, and (c) reduction in number of hidden units per ANN.

At that moment, this ANN will be the only fired one, and there will be no need for an average sum. This will reduce the testing time in such special case.

Each experiment has its own conditions; selecting the best k neighbors depends on how the instances are distributed and how many subsets are generated from the chopping process. Any odd number of k will break any tie in a decision and will yield good results. However, the best result using different values of k is reported in our experiments. Good results and smaller ANNs tip the scale for the MCRN over the RBFN. Moreover, those small ANNs can be parallelized during training and testing, which further improve the overall performance, whereas the RBFN cannot be parallelized as easily as the MCRN, because the RBFN has a fully connected internal structure.

E. Speed Comparison

One important goal in ANN classification problems is increasing the speed of the MCRN in the training and testing process. Both the RBFN and the MCRN are implemented using a Windows 7 64-b platform with an Intel core i7 processor and 16 GB of RAM. The individual ANNs of the MCRN are executed using the parallelism feature of MATLAB 2015. Fig. 11 shows the percentage improvement in training, testing, and hidden units for each dataset using the RBFN and MCRN experiments.

TABLE VII
COMPARISON OF THE ACCURACY RESULTS OF THE MCRN WITH OTHER CLASSIFIERS

Dataset	SVM	KNN (k=3)	RBFN	MCRN	Improved %
Iris	91.1	95.6	93.3	**97.8**	2.2
Glass-Id	**95.5**	88.6	81.8	86.4	-9.1
Seeds	88.9	90.5	89.7	**91.3**	0.8
Liver	57.1	60	65.7	**71.4**	5.7
Wisconsin	95.7	96.4	91.4	**97.1**	0.7
Thyroid	95.9	96.3	96.4	**96.5**	0.1
Hepatitis	87.1	90.3	87.1	**96.8**	6.5
Ionosphere	84.5	90.1	91.6	**93**	1.4
Urban	91.1	90.5	91.9	**92.9**	1.0
Occupancy	**97.9**	95.9	97	97.6	-0.3
Satimage	93.9	**96.5**	95.5	95.9	-0.6
Letter	99.3	**99.9**	97.1	96.6	-3.3

Fig. 11(a) shows noticeable improvements in training time using the MCRN compared with the RBFN for all datasets. As more division occurs in datasets, the training time decreases further, because fewer computations are needed in terms of the inner products for each individual ANN compared with a large RBFN. The RBFN suffers from excessive computations during training due to its large hidden layer. Each hidden unit must compute an inner product for all the training dataset instances. In contrast, small structures of individual ANNs and fewer training subsets reduce the training delay of the MCRN.

In offline classification problems, training is conducted only once. Thereafter, the training time is no longer as important as the testing time. The testing time is considerably more important because trained ANNs are plugged into an online environment to classify every new testing entry in real time. The improvements in testing time in Fig. 11(b) show that the MCRN outperforms the RBFN in the majority of cases. However, the RBFN outperforms the MCRN in certain cases, such as Thyroid, Letter, and Satimage. The MCRN testing time, T_{test}, requires three stages of delay: the *input subset selector* time, T_{ISS}, the individual ANN time, T_{INNs}, and the *output combiner* time, T_{oc}, as in (17). The first and last stages cannot be parallelized in the same manner as the middle stage. T_{oc} is considered extremely small compared with T_{ISS} and T_{INNs} and can be neglected, because T_{oc} only averages k ANN outputs. T_{ISS} is the time required to calculate the KNNs, which is also the time used to calculate the distances between the new testing instance and all training instances. T_{ISS} is smaller than T_{INNs} for small datasets. These calculations become more difficult and time consuming when considering large-scale datasets, and T_{ISS} will have a negative effect on the timing calculations. Nonetheless, the MCRN still exhibits good performance and speed improvement in the majority of cases

$$T_{\text{test}} = T_{\text{ISS}} + T_{\text{INNs}} + T_{\text{OC}}. \tag{17}$$

TABLE VIII
NOTATION DEFINITION

Symbol	Meaning
B	Original dataset density.
β_i	Subsets' average density.
$\mathbf{c}_h$ and σ_h	Center and width of an RBF, respectively.
$\mathbf{d}_k$	Set of k minimum distances between new input and P training vectors.
d_k	the k-th minimum Euclidean distance between new testing point $\tilde{\mathbf{x}}$ and the k-th training point $\mathbf{x}^p$.
$e_p(\mathbf{w})$	Error between p-th desired output and the output of ANN when applying the test input.
$E(\mathbf{w})$	Sum square error of all P sets.
H	Number of hidden layer units.
I	Number of input layer units.
k	Number of nearest neighbors, number of selected ANNs.
MDL	ANN training model.
N	Number of Individual ANNs, Number of subsets.
P	Number of training set patterns.
$\mathbf{Q}^+$	pseudo inverse matrix of all H hidden units.
R_1 , R_2	Regions results from chopping whole space in the first dimension x_1.
R_{11}, R_{12} , R_{21}, R_{22}	Regions results from chopping whole space in first dimension x_1 and then in second dimension x_2.
S	Set of all s subsets.
$s_1, s_2, \dots, s_n, \dots, s_N$	Subsets of original dataset.
δ_k	k-th selected subset from all S subsets.
T_{test}	ANN testing time.
T_{ISS}	Input subset selector delay time.
T_{INNs}	Individual ANNs delay time.
T_{OC}	Output combiner delay time.
w_h	Weight between the hidden unit h and the output y.
w_0	Bias weight with input $\theta_0 = 1$ to the output y.
$\mathbf{w}$	Weight matrix of ANN.
$\mathbf{x}^p$	Training input vector of the p-th training dataset.
$\tilde{\mathbf{x}}$	New testing input.
$\mathbf{x}$	Input vector.
x_i	i-th element of input vector $\mathbf{x}$.
$\hat{x}_1$	Median of x_1 values in x_1 dimension.
$\hat{x}_{21}$ and $\hat{x}_{22}$	Medians of x_2 values in x_2 dimension.
$\hat{x}_{31}, \hat{x}_{32}, \hat{x}_{33}, \hat{x}_{34}$	Medians of x_3 values in x_3 dimension.
y	Single ANN output.
$\mathbf{y_d}$	Vector of desired output.
y_d^p	Desired output of the p-th training dataset.
$\tilde{y}_k$	resultant output of each k selected ANNs.
ξ	Number of divisions in all dimensions.
$\theta_h(\mathbf{x})$	h-th hidden unit RBF.
$\boldsymbol{\theta}$	Hidden unit vector.

TABLE IX
ACRONYMS DEFINITIONS

Symbol	Meaning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DBN	Deep Belief ANN
DNA	novel deep neural architecture
ErrCor	Error correction algorithm
ISO	improved second order algorithm
k-d tree	k-dimension tree algorithm
KNN	k-nearest neighbors
LM	Levenberg-Marquardt algorithm
MCRN	Multi-Column RBF Network
MIMO	Multi Input Multi Output system
MISO	Multi Input Multi Single system
MRBF	median radial basis function
RBF	Radial Basis Function
RBFN	Radial Basis Function Network
RBM	Restricted Boltzmann machine
RMSE	Root Mean Square Error.
SVM	Support Vector Machine

Fig. 11 shows that the MCRN has a shorter training time and fewer hidden units for the Letter dataset; however, the testing time is longer than that for the RBFN. For the Ionosphere dataset, the MCRN has more hidden units than the RBFN, but the MCRN training and testing times are considerably shorter. Although the MCRN has shorter training and testing times for the Occupancy dataset, the maximum number of hidden units is higher. For all other datasets, the MCRN outperforms the RBFN in terms of training time, testing time, and number of hidden units.

The runtime complexity of a traditional RBFN network is $O(I \times H)$. Therefore, the process time increases with increases in the number of input vector features, I, and/or the number of the hidden units, H. In contrast, the MCRN runtime complexity is determined by $O(I \times P) + O(I \times H_{\text{biggestANN}})$. The first term indicates the time required to find the nearest neighbors to the new input vector among all P training instances, whereas the second term represents the process of applying the new input vector of I dimensionality to the largest structured individual ANN, which has $H_{\text{biggestANN}}$ hidden units. Similarly, the first term also represents T_{ISS}, and the second term represents T_{INNs}. When the number of training datasets, P, decreases, the effect of the first term nearly vanishes, and the speed of the MCRN method increases. With large-scale datasets, the effect of the first term will increase, and the system will be delayed, even if the structure of the individual ANNs is less than in the traditional RBFN structure. This issue can be utilized in future work to decrease the complexity of the neighbor searching technique by sorting instances or memorizing the last results instead of repeating the entire search process numerous times.

F. MCRN Compared With Other Classifiers

Many studies select the SVM and KNN to use as comparative classifiers for the RBFN, as in [8], and [31]–[33]. In this paper, the same training and testing instances used in the RBFN and MCRN are used to train and test the SVM and KNN. The accuracy results are compared with those obtained with the RBFN and the MCRN, as shown in Table VII. This comparison demonstrates how the performance of the MCRN compares with those of other well-known classification techniques.

These results demonstrate that the MCRN can compete with the machine learning techniques that have been considered. The MCRN is superior for the Iris, Seeds, Liver, Wisconsin, Thyroid, Hepatitis, Ionosphere, and Urban datasets. For the other results, the MCRN is considered comparable, with percentage decreases of -0.6% to -9.1% compared with the best result for each dataset. In general, the MCRN is still better than the RBFN for all cases and still shows promising results when compared with the SVM and KNN.

V. Conclusion

Using different benchmark UCI datasets, the MCRN shows a total accuracy improvement of 35.3% for all datasets compared with the SVM, 22.7% compared with KNN and up to 34.8% compared with the RBFN. Although the MCRN considers distance computations between every new entry and all training data instances, it shows a maximum reduction in training time of up to 70.5% with two subsets and up to 74.8% with four subsets compared with the RBFN. Furthermore, the MCRN shows a maximum reduction in testing time of up to 94.2% with two subsets and up to 94.7% with four subsets compared woth the RBFN. The RBFN requires excessive computations for kernel inner products with each new test input entry. Although the distance computations of the MCRN add considerable delays, particularly with a large-scale dataset, it still utilizes smaller individual ANN structures, resulting in fewer overall inner products. Moreover, the MCRN is suitable for a parallel environment because of the independence of its individual ANNs, while the RBFN structure cannot be deployed in parallel as easily because of its fully connected structure. Compared with recent results, the MCRN shows promise in terms of both accuracy and timing. Suggestions for future studies include determining a way to minimize distance computations with large-scale data and a way to use hardware ANN chips to obtain better results.

Appendix

See Tables VIII and IX.

Acknowledgment

The authors would like to thank A. Huynh and J. Perez for proofreading to help improve this paper.

References

[1] W. Kaminski and P. Strumillo, "Kernel orthonormalization in radial basis function neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1177–1183, Sep. 1997.

[2] H. Yu, P. D. Reiner, T. Xie, T. Bartczak, and B. M. Wilamowski, "An incremental design of radial basis function networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 10, pp. 1793–1803, Oct. 2014.

[3] L. Shao, D. Wu, and X. Li, "Learning deep and wide: A spectral method for learning deep networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 12, pp. 2303–2308, Dec. 2014.

[4] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2012, pp. 3642–3649.

[5] C. Yan *et al.*, "A highly parallel framework for HEVC coding unit partitioning tree decision on many-core processors," *IEEE Signal Process. Lett.*, vol. 21, no. 5, pp. 573–576, May 2014.

[6] C. Panchapakesan, M. Palaniswami, D. Ralph, and C. Manzie, "Effects of moving the center's in an RBF network," *IEEE Trans. Neural Netw.*, vol. 13, no. 6, pp. 1299–1307, Nov. 2002.

[7] L. Bruzzone and D. F. Prieto, "A technique for the selection of kernel-function parameters in RBF neural networks for classification of remote-sensing images," *IEEE Trans. Geosci. Remote Sens.*, vol. 37, no. 2, pp. 1179–1184, Mar. 1999.

[8] K. Z. Mao and G.-B. Huang, "Neuron selection for RBF neural network classifier based on data structure preserving criterion," *IEEE Trans. Neural Netw.*, vol. 16, no. 6, pp. 1531–1540, Nov. 2005.

[9] A. G. Bors and I. Pitas, "Median radial basis function neural network," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1351–1364, Nov. 1996.

[10] T. Xie, H. Yu, J. Hewlett, P. Rozycki, and B. Wilamowski, "Fast and efficient second-order method for training radial basis function networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 4, pp. 609–619, Apr. 2012.

[11] O. Arif and P. A. Vela, "Kernel map compression for speeding the execution of kernel-based methods," *IEEE Trans. Neural Netw.*, vol. 22, no. 6, pp. 870–879, Jun. 2011.

[12] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 8, pp. 1553–1565, Aug. 2014.

[13] L. Szymanski and B. McCane, "Deep networks are effective encoders of periodicity," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 10, pp. 1816–1827, Oct. 2014.

[14] A. Krizhevsky, I. Sulskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 25. 2012, pp. 1–9.

[15] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, Jan. 2013.

[16] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[17] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.

[18] W. K. Wong and M. Sun, "Deep learning regularized Fisher mappings," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1668–1675, Oct. 2011.

[19] A. Stuhlsatz, J. Lippel, and T. Zielke, "Feature extraction with deep neural networks by a generalized discriminant analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 4, pp. 596–608, Apr. 2012.

[20] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba, "Learning with hierarchical-deep models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1958–1971, Aug. 2013.

[21] S. Bu, Z. Liu, J. Han, J. Wu, and R. Ji, "Learning high-level feature by deep belief networks for 3-D model retrieval and recognition," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2154–2167, Dec. 2014.

[22] K. Chen and A. Salman, "Learning speaker-specific characteristics with a deep neural architecture," *IEEE Trans. Neural Netw.*, vol. 22, no. 11, pp. 1744–1756, Nov. 2011.

[23] M. Van De Steeg, M. M. Drugan, and M. Wiering, "Temporal difference learning for the game Tic-Tac-Toe 3D: Applying structure to neural networks," in *Proc. IEEE Symp. Ser. Comput. Intell.*, Dec. 2015, pp. 564–570.

[24] R. Mall, V. Jumutc, R. Langone, and J. A. K. Suykens, "Representative subsets for big data learning using k-NN graphs," in *Proc. IEEE Int. Conf. Big Data*, Oct. 2014, pp. 37–42.

[25] Q. Gu and J. Han, "Clustered support vector machines," in *Proc. AISTATS*, 2013, pp. 307–315.

[26] J. L. Bentley, "Multidimensional binary search trees in database applications," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 4, pp. 333–340, Jul. 1979.

[27] A. Asuncion and D. J. Newman. (2007). *UCI Machine Learning Repository*, accessed on Mar. 3, 2016. [Online]. Available: http://www.ics.uci.edu/~/MLRepository.html

[28] B. A. Johnson and Z. Xie, "Classifying a high resolution image of an urban area using super-object information," *ISPRS J. Photogram. Remote Sens.*, vol. 83, pp. 40–49, Sep. 2013.

[29] B. A. Johnson, "High-resolution urban land-cover classification using a competitive multi-scale object-based approach," *Remote Sens. Lett.*, vol. 4, no. 2, pp. 131–140, Feb. 2013.

[30] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and CO_2 measurements using statistical learning models," *Energy Buildings*, vol. 112, pp. 28–39, Jan. 2016.

[31] Y.-J. Oyang, S.-C. Hwang, Y.-Y. Ou, C.-Y. Chen, and Z.-W. Chen, "Data classification with radial basis function networks based on a novel kernel density estimation algorithm," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 225–236, Jan. 2005.

[32] R. N. Mahdi and E. C. Rouchka, "Reduced HyperBF networks: Regularization by explicit complexity reduction and scaled Rprop-based training," *IEEE Trans. Neural Netw.*, vol. 22, no. 5, pp. 673–686, May 2011.

[33] F. Dammak and L. Baccour, "Proposition of a classification system 'β–LS–SVM' and its application to medical data sets," in *Proc. 6th Int. Conf. Soft Comput. Pattern Recognit. (SoCPaR)*, 2014, pp. 101–105.

Ammar O. Hoori (S'17) received the B.Sc. and the M.Sc. degrees in computer engineering from University of Baghdad, Baghdad, Iraq, in 1999 and 2002, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Virginia Commonwealth University, Richmond, VA, USA.

From 2008 to 2013, he was a Teacher and Researcher with the Computer Engineering Department, University of Baghdad. His current research interests include machine learning, neural networks, computer networks, and distributed systems.

Yuichi Motai (S'00–M'03–SM'12) received the B.Eng. degree in instrumentation engineering from Keio University, Tokyo, Japan, in 1991, the M.Eng. degree in applied systems science from Kyoto University, Kyoto, Japan, in 1993, and the Ph.D. degree in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2002.

He is currently an Associate Professor of Electrical and Computer Engineering with Virginia Commonwealth University, Richmond, VA, USA. His current research interests include sensory intelligence; particularly in medical imaging, pattern recognition, computer vision, and sensory-based robotics.