Neurocomputing 73 (2010) 3012–3019

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

A sequential learning algorithm for self-adaptive resource allocation network classifier

S. Suresh [a],*, Keming Dong [b], H.J. Kim [b]

[a] School of Computer Engineering, Nanyang Technological University, Singapore
[b] CIST, Korea University, Seoul, Republic of Korea

A R T I C L E I N F O

Article history:
Received 12 August 2009
Received in revised form
14 July 2010
Accepted 14 July 2010
Communicated by F. Rossi
Available online 22 August 2010

Keywords:
Resource allocation network
Self-adaptive control parameters
Sequential learning
Multi-category classification
Extended Kalman filter

A B S T R A C T

This paper addresses sequential learning algorithm for self-adaptive resource allocation network classifier. Our approach makes use of self-adaptive error based control parameters to alter the training data sequence, evolve the network architecture, and learn the network parameters. In addition, the algorithm removes the training samples which are similar to the stored knowledge in the network. Thereby, it avoids the over-training problem and reduces the training time significantly. Use of misclassification information and hinge loss error in growing/learning criterion helps in approximating the decision function accurately. The performance evaluation using balanced and imbalanced data sets shows that the proposed algorithm generates minimal network with lesser computation time to achieve higher classification performance.

1. Introduction

Neural networks are powerful tools that can capture the underlying relationship between the input and output data by learning. In the classical batch learning algorithm, the objective is to construct a network that can predict the output of new data given complete training data set. Here, the samples are presented simultaneously and presented as often as desired. In most practical applications complete set of training samples may not be available a priori or contain large training set. Some of the practical problems like cancer classification, human behavior prediction and scene understanding in video surveillance [22] and some industrial problems [1] allow temporal changes in the task being learnt. Hence, classical batch learning is experienced to be rather infeasible and online/sequential learning is employed instead.

In neural network model, selection of network architecture is a critical issue. One has to find a minimal architecture that accurately fits the true function described by the training data. A large network may accurately fit the training data, but may have poor generalization performance due to over-fitting. On the other hand, small network requires lesser computational effort, but may not be able to approximate the given function. Research in architecture selection has resulted in many algorithms to solve this problem. A complete literature on architecture selection is given in [16]. Finding appropriate architecture for a given training data set itself is a challenging problem. When the training data set is not defined prior to the learning process, the complexity of finding the minimal architecture increases further. In this paper, we address learning algorithms which evolve the architecture by itself in online/sequential learning framework.

Online/sequential learning is performed in sequence of trails. Here, the training samples arrive one-by-one and the samples are discarded after learning. So, it can accommodate the temporal changes in the task and require less memory and computational time for learning process. A sequential learning has been analyzed extensively in the framework of radial basis function network (RBFN) [5–15]. One of the first sequential learning algorithm was the resource allocation network (RAN) [11]. The RAN starts with zero hidden neuron and add neurons based on the novelty of the incoming data [7,11]. It uses the least mean square algorithm for network parameter update. Similar approach is used in the minimal resource allocation network (MRAN) [13] and the extended minimal resource allocation network (EMRAN) [12]. Besides adding neuron, the MRAN and EMRAN prune insignificant neurons from the network based on the contribution of error over a window of samples. In the MRAN, extended Kalman filter (EKF) was used to update the network parameters. Since the EKF is computationally intensive, the EMRAN updates the parameter of nearest neuron for the current sample. In [14,15], a single stage

* Corresponding author.
E-mail address: ssundaram@ntu.edu.sg (S. Suresh).

incremental learning approach is presented. Here, the neurons are added to the network based on the local error history in a fixed interval which may leads to unnecessary or untimely insertions. In the RAN, MRAN and EMRAN algorithms add neuron continuously based on the error history and hence, the network generated by these algorithms are better than the incremental learning approach. In [6], neuron significance with respect to input distribution is used as a criterion for growing and pruning the network architecture, and is called growing and pruning RBFN (GAP-RBFN). Aforementioned algorithms learn samples one-by-one and only once and also evolve the network architecture automatically. It has been shown in literature that the aforementioned algorithms provide better generalization performance for function approximation problems than the classification problems [5].

For classification problems, the generalization performance depends heavily on the optimal selection of control parameters of the learning algorithms and the input data distribution as they use the basic function approximation approach for classification. Recently in [5], a sequential learning multi-category classifier is proposed. Here, the neurons are added based on misclassification information and distance between the current sample and nearest neuron in the same class. It is also shown that the use of hinge loss function improves the performance significantly [5].

For a fixed single-layer neural network architecture, online/sequential learning algorithm using recursive least square is presented in [8]. It is referred as an online sequential extreme learning machine (OS-ELM). In the OS-ELM algorithm, the input weights are selected randomly and output weights are calculated analytically using the least square error. For sequential learning, the output weights are updated using recursive least. Here, the training data was presented one-by-one or chunk-by-chunk. Recently in [10], incremental convex extreme learning machine was proposed. Both the OS-ELM and the incremental version produces better generalization performance with smaller computational time. In case of sparse and imbalance data sets, the random selection in the OS-ELM and incremental versions affects the performance significantly [19,23]. Similar to that of sequential learning neural network, in support vector machine framework, an incremental and decremental learning algorithm handles the training samples one-by-one or chunk-by-chunk [20]. Here, the solution is constructed recursively by retaining the Kucnh–Tucker (KT) conditions on all previously seen data, while adding a new data to the solution.

Aforementioned sequential learning algorithms (both constructive and fixed architecture networks) uses all training samples one-by-one and only once. If the training data set contains more similar data, then the resultant classifier has poor generalization due to over-fitting. In addition, the sequence in which the training samples presented to the sequential algorithms affects the performance significantly. Also, the problem dependent algorithm control parameters influence the approximation ability of the network. As the existing sequential algorithms suffer from these drawbacks, we need an algorithm which alters the sequence of training samples based on the information content to achieve a good generalization performance. We propose one such sequential learning algorithm, named as, self-adaptive resource allocation network (SRAN). The SRAN classifier uses radial function network as a basic building block. The control parameters in the proposed sequential algorithm are self-regulated, so, they are fixed, and are mostly independent of the problem considered. The control parameters alter the sequence in which the SRAN classifier approximates the decision function, based on the difference between the information contained in each sample and the knowledge acquired by the network. The higher the difference, the earlier a sample participates in learning. A few samples with lesser differences are pushed to the rear end of the sample data stack. These samples are later used to fine-tune the network parameters. Also, a few samples with redundant information are discarded from the training data set, thus avoiding over-training. Thus, the finally realized network is compact and provides better generalization performance.

The performance of the proposed SRAN classifier is evaluated by comparing it with other sequential learning neural algorithms like MRAN [13], EMRAN [12], GAP-RBF [6], SMC-RBF [5], fixed network online learning OS-ELM [8], and incremental and decremental SVM [20]. We also compare the results with batch learning SVM [4] classifier. For experimental evaluation, we consider: (i) image segmentation (IS), (ii) vehicle classification (VC), and (iii) glass identification (GI) problems from UCI machine learning repository [2]. Among the three real-world examples, GI and VC problems are sparse in nature with high sample imbalance. First, we use balanced IS data set to highlight the advantages of the proposed algorithm. Next, we use high sample imbalance VC and GI data sets to show the effectiveness of the proposed algorithm. The results clearly highlight that the proposed SRAN classifier is compact and provides better generalization performance. Finally, we highlight some issues related to SRAN algorithm.

The paper is organized as follows: Section 2 describes the proposed SRAN classifier. Section 3 presents experimental results and performance comparison with other existing sequential learning algorithms. Section 4 summarize the main conclusions from this study.

2. A sequential learning algorithm for self-adaptive resource allocation network (SRAN) classifier

In this section, we describe the principles behind self-adaptive resource allocation network (SRAN) first and then provide the various steps involved in the algorithm and finally summaries the algorithm in a pseudo code form.

2.1. Problem definition

Online/sequential learning for a multi-category classification problem can be stated in the following manner. The observation data arrives one-by-one and one at a time. After learning, the sample is discarded from the sequence. Suppose we have the observation data $\{(\mathbf{x}_1,\mathbf{y}_1),(\mathbf{x}_2,\mathbf{y}_2),\ldots,(\mathbf{x}_t,\mathbf{y}_t),\ldots\}$, where $\mathbf{x}_t \in \Re^m$ is an m-dimensional features of observation t and $\mathbf{y}_t \in \Re^n$ is its coded class label. Here, n represents the total number of classes. For notational convenience, the subscript t is left out in all further discussion. If the feature observation $\mathbf{x}$ is assigned to the class label c, then cth element of $\mathbf{y} = [y_1,\ldots,y_c,\ldots,y_n]^T$ is 1 and other elements are -1.

$$y_j = \begin{cases} 1 & \text{if } j==c \\ -1 & \text{otherwise,} \end{cases} \quad j=1,2,\ldots,n \tag{1}$$

The observation data are random variables and the observation $\mathbf{x}$ provides some useful information on probability distribution over the observation data to predict the corresponding class label with certain accuracy. Hence, the classification problem is to predict the coded class label $\mathbf{y}$ of a new observation $\mathbf{x}$. This requires us to estimate a functional relationship between the coded class label and feature space from sequential training data. In the SRAN classifier, a radial basis function network is used as a building block. The SRAN network approximates the functional relationship between the feature space and the coded class label.

The output of the SRAN classifier ($\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_n]^T$) with K hidden neurons has the following form:

$$\hat{y}_i = \sum_{j=1}^{K} \alpha_{ij} y_h^j, \quad i = 1,2,\ldots,n \tag{2}$$

$$y_h^j = \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_j^l\|^2}{(\sigma_j^l)^2}\right) \tag{3}$$

where $\boldsymbol{\mu}_j^l$ is the jth neuron center corresponding to the lth class, σ_j^l is the width of the jth neuron and α_{ij} is the weight connecting the ith output neuron and jth Gaussian neuron.

The predicted class label $\hat{c}$ for the new training sample is given by

$$\hat{c} = \arg \max_{i \in 1,2,\ldots,n} \hat{y}_i \tag{4}$$

In other sequential learning algorithms [6,13], the error ($\mathbf{e}$) is usually the difference between the actual output and predicted output ($\mathbf{y} - \hat{\mathbf{y}}$), which is used in the mean square error loss function. For classification problems, the above definition of error restricts the outputs of the neural classifier between ± 1. In [17,18], it is shown that the classifier developed using a hinge loss function can estimate the posterior probability more accurately than the mean square error loss function. Hence, in our formulation, we use a hinge loss function to calculate the error $\mathbf{e} = [e_1, e_2, \ldots, e_n]^T$ and is given below

$$e_i = \begin{cases} y_i - \hat{y}_i & \text{if } y_i \hat{y}_i < 1 \\ 0 & \text{otherwise} \end{cases} \quad i = 1,2,\ldots,n \tag{5}$$

With this hinge loss function, the network output can grow beyond ± 1 and prevent the saturation problems in the learning process. The truncated outputs of the classifier model approximate the posterior probability accurately [18]. The truncated output is defined as

$$T(\hat{y}_i) = \min(\max(\hat{y}_i, -1), 1), \quad i = 1,2,\ldots,n \tag{6}$$

Since, the target vectors are coded as -1 or 1, the posterior probability of observation vector $\mathbf{x}$ belonging to class c is

$$\hat{p}(c|\mathbf{x}) = \frac{T(\hat{\mathbf{y}}) + 1}{2} \tag{7}$$

2.2. SRAN learning algorithm

In the setting of standard online/sequential learning, the training sample arrives one at a time and the network adapts its parameters based on the difference in knowledge between the network and the current sample. Fig. 1 gives a bird's eye view of the SRAN algorithm. As each new sample ($\mathbf{x}_t$) is presented to the network, based on the sample error ($\mathbf{e}$), the sample is either

- used for network training (growing/learning) immediately, or;
- pushed to the rear end of the stack for learning in future, or;
- deleted from the data set.

In ideal conditions, training stops when there are no more samples to be presented to the network. However, in real-time, training is stopped when the samples get stacked repeatedly and do not participate in learning.

Similar to other sequential learning algorithms, the SRAN learning algorithm begins with zero hidden neurons and adds new hidden neuron based on the information present in the current sample. First sample ($\mathbf{x}_1, \mathbf{y}_1$) forms first hidden neuron as

$$\boldsymbol{\alpha}_1 = \mathbf{e}; \quad \boldsymbol{\mu}_1^c = \mathbf{x}_1; \quad \sigma_1^c = \kappa\sqrt{\mathbf{x}^T\mathbf{x}} \tag{8}$$

where κ is a positive constant which controls the overlap between the hidden neurons and c is the actual class label of the current sample.

In standard online/sequential learning, the samples are presented only once, and all the samples are learnt. In such a network, there is no control over the sample sequence which influences the learning ability of the network. This also means that arrival of similar samples leads to over-training of particular pattern. This will, therefore, influence the generalization ability of the sequential learning algorithm. In our proposed approach, the sequence of the training sample is controlled internally using self-regulated control parameters as explained below.

The self-regulated control parameters (η_a, η_l) identify the sample 't' with maximum information. The learning process of SRAN involves allocation of new hidden neurons, as well as adjusting network parameters. If the current sample 't' does not satisfy the learning criteria, then the current sample is stacked at the rear end of the sequence, for future use. These samples do not take part in training at this stage. Without loss of generality, let us assume that the network has K hidden neurons from $t-1$ training samples.

Now, we present the working principle of self-regulation system for deletion of samples, growing/learning, and sequence altering.

- *Deletion criterion*: If the absolute maximum error $E = \max_{i=1,2,\ldots,n} |e_i|$ is less than 0.05, then the sample is deleted without being used and thus prevents over-training.
- *Growing criterion*: The following criteria must be met for an observation ($\mathbf{x}_t$, $\mathbf{y}_t$) to be used to add a new hidden neuron to the network.

  $$\hat{c} \neq c \quad \textbf{AND} \quad E \geq \eta_a \tag{9}$$

  where E is the absolute maximum error in the current sample, and η_a is the self-adaptive growing threshold. Even though, the predicted output is not restricted between ± 1, the error ($\mathbf{e}$) is between ± 2 due to hinge loss function. Since, we use absolute maximum error in the criterion, the self-adaptive growing threshold is restricted in the range [0.75,1.5]. This range includes the misclassification region and correctly classified samples with high error. Before starting of the learning process, η_a is initialized to 1.5. The growth control parameters are adapted based on the current sample error ($\mathbf{e}$). Depending on the error that contributes to neuron growth, the respective growth control parameters are updated as

  $$\eta_a := \delta\eta_a - (1-\delta)E \tag{10}$$

  where δ is a parameter that controls the slope of decrease of the control parameter, usually kept close to one.
  If the criterion given in Eq. (9) is satisfied, then a new hidden neuron $K+1$ is added and its parameters are set as follows:

  $$\boldsymbol{\alpha}_{K+1} = \mathbf{e}; \quad \boldsymbol{\mu}_{K+1}^c = \mathbf{x}_t; \quad \sigma_{K+1}^c = \kappa\|\mathbf{x}_t - \boldsymbol{\mu}_{nr}^c\| \tag{11}$$

  where κ is a positive constant which controls the overlap between the hidden neurons, nr is the nearest neuron to the current sample, and c is the actual class label of the current sample. Like in the SMC-RBF [5], the SRAN also uses the nearest neuron in the same class for determining the width of new neuron.
  When a new hidden neuron is added, the dimensionality of error covariance matrix $P_{(t)}$ is increased to

  $$P_{(t)} = \begin{bmatrix} P_{(t-1)} & 0 \\ 0 & p_0 I \end{bmatrix} \tag{12}$$

  where I is the identity matrix, and p_0 is an estimate of uncertainty in the initial values assigned to the parameters.

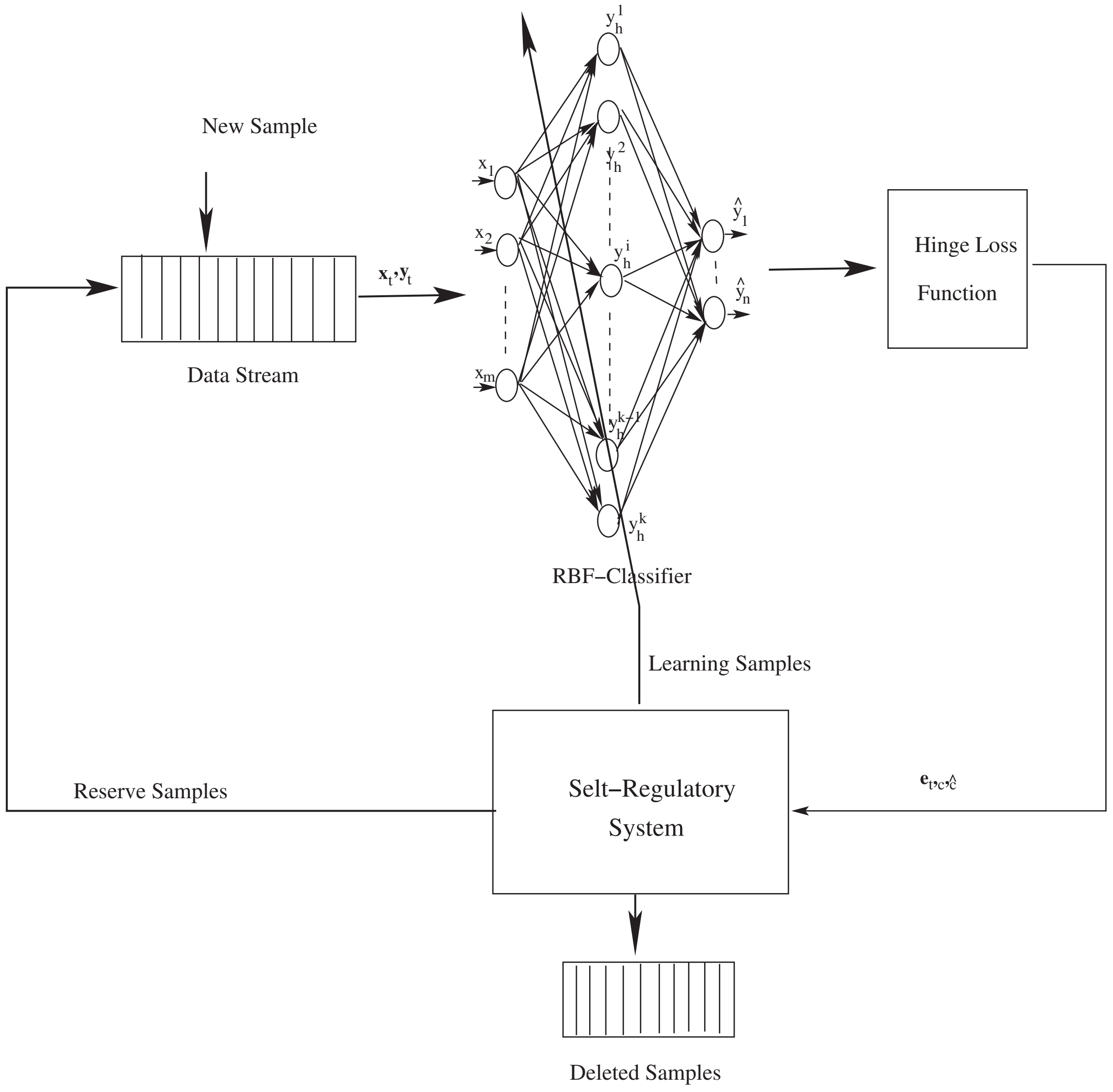

Fig. 1. SRAN learning algorithm in a Nutshell.

The dimensionality of identity matrix is equal to the number parameters introduced by the new hidden neuron.

- *Learning criterion*: The network parameters ($\mathbf{w} = [\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \boldsymbol{\mu}^l, \sigma_1^l, \ldots, \boldsymbol{\alpha}_K, \boldsymbol{\mu}_K^l, \sigma_K^l]$) are updated if the following condition is satisfied:

$$c == \hat{c} \quad \textbf{AND} \quad E \geq \eta_l \tag{13}$$

where η_l is the self-adaptive learning control parameter which is restricted between [0.05,0.75]. Here, the self-adaptive learning threshold is adapted based on the knowledge present in the current sample as

$$\eta_l := \delta \eta_l - (1-\delta)E \tag{14}$$

where δ is a parameter that controls the slope of decrease of the control parameter, usually kept close to one.

The SRAN uses the extended Kalman filter (EKF) to update the network parameters as

$$\mathbf{w}_{(t)} = \mathbf{w}_{(t-1)} + KL_{(t)}\mathbf{e} \tag{15}$$

where $KL_{(t)}$ is the Kalman gain and $\mathbf{e}$ is the error obtain from hinge loss function.

For the EKF, the parameter update equations for the new training sample are given by

$$\mathbf{a}_{(t)} = \Delta_{\mathbf{w}}(\hat{\mathbf{y}})_t \tag{16}$$

$$KL_{(t)} = P_{(t)}\mathbf{a}_{(t)}[R + \mathbf{a}_{(t)}^T P_{(t)} \mathbf{a}_{(t)}]^{-1} \tag{17}$$

$$P_{(t+1)} = [I - KL_{(t)}\mathbf{a}_{(t)}^T]P_{(t)} + IP_Q \tag{18}$$

where R is the variance of measurement noise, $\mathbf{a}_{(t)}$ is the partial derivatives for the output signal with respect to the parameters ($\mathbf{w}$), $P_{(t)}$ is the error covariance matrix, and $KL_{(t)}$ is the Kalman gains. The addition of artificial process noise (P_Q) helps in avoiding convergence to local minima [21].

The gradient vector $\mathbf{a}_{(t)}$ is given by

$$\mathbf{a}_{(t)} = \begin{bmatrix} 1, y_h^1, y_h^1 \dfrac{2\boldsymbol{\alpha}_1}{(\sigma_1^l)^2}(\mathbf{x}_t - \boldsymbol{\mu}_1^l)^T, y_h^1 \dfrac{2\boldsymbol{\alpha}_1}{(\sigma_1^l)^3}\|\mathbf{x}_t - \boldsymbol{\mu}_1^l\|^2 \ldots, \\ y_h^K, y_h^K \dfrac{2\boldsymbol{\alpha}_K}{(\sigma_K^l)^2}(\mathbf{x}_t - \boldsymbol{\mu}_K^l)^T, y_h^K \dfrac{2\boldsymbol{\alpha}_K}{(\sigma_K^l)^3}\|\mathbf{x}_t - \boldsymbol{\mu}_K^l\|^2 \end{bmatrix}^T \tag{19}$$

- *Sequence altering*: If the current sample t does not satisfy the growth and learning criteria, then the sample is pushed to the rear end of the stack, to be presented to the network, in future. These samples can be used to fine-tune the network parameters, when all the available samples in the data set are presented. Any new samples arriving in the sequence, are just stacked behind the current last sample.

Ideally, training stops when no further sample is available for training in the data stream. However, in real-time, training stops on meeting an error criteria.

To summarize, the SRAN algorithm in a pseudo code form is given below:

Pseudocode 1. Pseudo code for the SRAN Algorithm.

```
  Input : Present the training data one-by-one to the network from data stream.
  Output : Decision function that estimates the relationship between feature space and class label.
START
  Initialization : Assign the first sample as the first neuron ($K=1$). The parameters of the neuron are chosen
as shown in Eq. (8).
  Start learning for samples $t=2,3,\ldots$
  DO
    Compute significance of the sample to the network:
      Compute the network output $\hat{\mathbf{y}}_t$.
      Find the maximum absolute hinge error $E$ and predicted class label $\hat{c}$.
    Delete Redundant samples:
      IF $E \leq 0.05$ THEN
        Delete the sample from the sequence without learning.
      ENDIF
    IF $c \neq \hat{c}$ AND $E \geq \eta_a$THEN
      Add a neuron to the network ($K=K+1$).
      Choose the parameters of the network as in Eq. (11).
      Update the control adding parameters according to Eq. (10)
    ELSEIF $c==\hat{c}$ AND $E \geq \eta_l$ THEN
      Update the parameters of the network using EKF (Eqs. (15)–(18))
      Update the control parameters according to Eq. (14)
    ELSE
      The current sample $\mathbf{x}_t, \mathbf{y}_t$ is pushed to the rear end of the sample stack to be used in future. They can be later
used to fine-tune the network parameters.
    ENDIF
  ENDDO
END
```

3. Performance evaluation of SRAN classifier

In this section, we present performance evaluation of the SRAN classifier using three real world classification problems from UCI machine learning repository [2], namely: image segmentation (IS), vehicle classification (VC) and glass identification (GI) problems. The IS problem is a well-balanced data set. It consists of a databases of images with seven different categories randomly drawn from 2310 image regions. The aim is to recognize each region into one of seven categories using 19 attributes extracted from the region of 3×3 pixels. The VC and GI problems are having high sample imbalance for classifier development. In GI problem, nine features are used to identify six different types of glasses. Here, number of training samples in each classes are: 35, 38, 9, 7, 5 and 15. In addition to fewer training samples, the overlap between class labels c_3, c_4 and c_5 increases the complexity further [5]. Similar to GI problem, VC problem also have sample imbalance and significant overlap between classes. The detailed specification on number of input features, number of classes and number of training/testing samples are given in Table 1.

Table 1
Specification of UCI classification data set.

Data set	# Features	# Classes	# Samples	
			Training	Testing
Image segmentation	19	7	210	2100
Vehicle classification	18	4	424	422
Glass identification	9	6[a]	109	105

[a] Actual number of classes are 7 but there are no samples available for one class.

The performance of SRAN is compared with the well-known sequential learning algorithms like MRAN [13], GAP-RBFN [6], SMC-RBF [5] and incremental and decremental SVM [20] classifiers. In addition, we also compare the performance of SRAN with the fixed architecture OS-ELM classifier [8]. In the OS-ELM, for a given number of hidden neurons, the input weights are selected randomly and the output weights are calculated analytically. For new set of training samples, only the output weights are updated using the recursive least squares algorithm. For classification problems with sample imbalance and overlap, the random input weight selection will not be suitable [19]. Hence, we use the *k*-fold validation approach proposed in [19] to initialize the random input weights using the first batch of data. The input weights are kept constant for subsequent sequential samples.

All the simulations are conducted in MATLAB 7.0 environment on a desktop PC with Pentium duo processor and 4 GB memory. The simulation study for batch SVM is carried out using the popular LIBSVM package in C [3]. In all examples, the inputs to algorithm are scaled appropriately between ± 1. The performance measures used to compare the classifiers are described below.

3.1. Performance measures

In this paper, we use the global measures such as overall and average classification accuracies as a performance measures. The statistical measure of both local and global measures are given in confusion matrix (Q). Class-level performance is indicated by the percentage classification which tells us how many samples belonging to a particular class have been correctly classified. The percentage classification η_i for class c_i is

$$\eta_i = \frac{q_{ii}}{N_i^T} \tag{20}$$

where q_{ii} is the number of correctly classified samples and N_i^T is the number of samples for the class c_i in the testing data set. The global performance measures are the average (η_a) and overall (η_o) classification efficiency, which are defined as

$$\eta_a = \frac{1}{n_c}\sum_{i=1}^{n_c} \eta_i \tag{21}$$

$$\eta_o = \frac{1}{N^T}\sum_{i=1}^{n_c} q_{ii} \tag{22}$$

where n_c is the total number of classes, and N^T is the number of testing samples.

3.2. Performance comparison

The performance comparison results for the three real-world problems from the UCI Machine Repository are given below.

3.2.1. Balanced data set: image segmentation problem

The objective in image segmentation is partitioning an image into several constituent components. An important subsystem of image segmentation (IS) is image recognition, which extracts the interesting objects for further processing. Here in IS, 19 different features are extracted from the 3×3 region and each region is classified into one of 7 classes. For classifier development, we select 30 random samples from each class and the remaining 2100 samples (300 from each class) are used for testing. IS data set is a well-balanced data set. We use this data set to highlight the advantages of the proposed SRAN classifier. Note that the cost parameter c and Gaussian kernel width γ in batch SVM are optimized using a grid search and the best results are quoted here. Similarly, for the OS-ELM, the number of hidden neurons and initial weights are optimized using the k-fold validation scheme as explained in [19]. The control parameters for the MRAN, GAP-RBF and SMC-RBF are selected as in [5]. The number of hidden neurons, training time, and overall efficiency are given in Table 2. From the table, we can see that the proposed SRAN classifier outperforms the other existing sequential learning algorithms and batch learning SVM classifier. The overall efficiency is 5–6% more than MRAN and GAP-RBF classifiers, and 2–3% more than ID-SVM, batch-SVM and OS-ELM classifiers. The SRAN uses only 113 samples out of 210 training samples for generating best classifier. Hence, the computational effort is far less than its sequential counterpart and almost the same as the OS-ELM.

For IS problem, the SRAN uses 113 samples for growing/learning the network parameters, and the rest 97 samples are discarded without learning. The neuron growth history for the SRAN classifier is shown in Fig. 2. From this figure, we can see that 44 hidden neurons out of 47 neurons are added from 100 data, and remaining three neurons are added from the 13 reserved samples kept at the end of the data stream. From the neuron history, we can say that the 13 reserved samples pushed to the end of the data stream are generally used to fine-tune the network.

The SRAN classifier generates the best sequence of training samples from a given training sequence such that the classifier produces better generalization performance. Now, we conduct two studies using the MRAN classifier to highlight the advantages of proposed SRAN classifier. First, we use the best 113 sample sequence generated by the SRAN in the MARN algorithm. We call this classifier as MRAN*. Next, we append the deleted samples at the end of best sequence and develop a classifier MRAN$^+$. The number of neurons, overall training accuracy and testing accuracy are reported in Table 3. From the results, we can see that the training/testing performance of the MRAN* classifier (which uses 113 sample sequence) is better than the MRAN classifier. Also, the MRAN* achieves better generalization performance with smaller number of hidden neurons. The MRAN$^+$ classifier which uses entire 210 training samples sequence (best 113 samples sequence followed by 97 discarded samples) improves the training performance significantly, but the generalization performance reduces by 5%. This is due to overtraining using similar samples in MRAN$^+$ classifier. The neuron history of the MRAN* classifier is shown in Fig. 2. From this figure, we can see that the MRAN* requires more neurons than SRAN classifier. This is due to

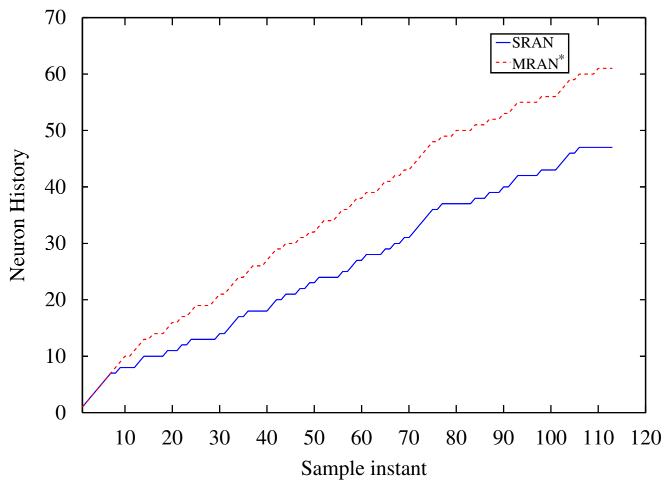

Fig. 2. Neuron growth history for SRAN and MRAN* classifier.

Table 2
Performance comparison for image segmentation problem. Note that batch SVM is implemented in C, whereas the rest in MATLAB.

Method	N_H neurons	Samples used	Training time (s)	Testing η_o
Batch-SVM	96[a]	210	721	90.62
MRAN	76	210	783	86.52
GAP-RBFN	83	210	365	87.19
OS-ELM	100	210	21	90.67
ID-SVM	88[a]	210	421	89.00
SMC-RBF	43	210	142	91.00
SRAN	47	113	22	**92.29**

[a] Support vectors.

Table 3
Performance comparison for MRAN, MRAN* and MRAN$^+$ classifiers.

Method	N_H neurons	Samples used	Training η_o	Testing η_o
MRAN	76	210	85.24	86.52
MRAN*	61	113	93.81	90.57
MRAN$^+$	69	210	99.05	85.81
SRAN	47	113	97.14	**92.29**

function approximation based criterion using in the MRAN algorithm.

Advantage of SRAN classifier: From the results of the SRAN algorithm, one can make the following observations:

- The control parameters are self-regulated, and hence, adapt themselves depending on the knowledge contained in the sample being considered and the knowledge acquired by the network.
- The self-regulated control parameters alter the sequence of training sample presentation, such that the network approximates the decision function accurately.
- Samples without significant information are deleted from the training set. This avoids over-training, reduces learning time, and minimizes the computational effort.
- The self-regulated control parameters result in selection of fewer neurons and efficient network structure.

3.2.2. *Sample imbalance data set: glass identification and vehicle classification problems*

Now, we look at the performance of the SRAN classifier on high sample imbalance data sets. The GI and VC data sets have fewer samples for classifier development and have strong overlap between the classes. In case of the VC problem, the number of training samples in each classes are: 119, 118, 98 and 89. In case of the GI problem, the number of training samples in each classes are: 35, 38, 9, 7, and 5. The presence of high sample imbalance and strong overlap between the classes influence the performance of existing sequential classifiers. In addition, the sequence in which the training samples are presented to the existing sequential classifiers will degrade the performance further.

Now, we present the experimental results for GI and VC problems. The number of hidden neurons, samples used in learning, training time, average classification efficiency (η_a) and overall classification efficiency (η_o) are reported in Table 4. For the OS-ELM and ID-SVM, 75% of training samples are presented first and the remaining 25% samples are presented one-by-one.

From the results, we can see that the proposed SRAN classifier outperforms the existing sequential classifiers and the batch SVM. Since, the VC problem has strong overlap between classes, the SRAN requires more number of hidden neurons (113 neurons) to approximate the decision function accurately. For the same problem, the SMC-RBF classifier requires 75 hidden neurons, However, the overall classification efficiency of SRAN is 3% more than the SMC-RBF classifier. Even though the number of neurons in the SRAN classifier is higher, the training time is less than the SMC-RBF classifier. This is due to reduction in number of samples used for learning. In case of the GI problem, the proposed SRAN classifier outperforms other algorithms by approximately by 5–20% with lesser number of hidden neurons.

Table 4
Performance comparison for GI and VC problems. Note that batch SVM is implemented in C, whereas the rest in MATLAB.

Data set	Method	N_H neurons	Samples used	Training time (s)	Testing	
					η_o	η_a
VC	Batch-SVM	234[a]	424	550	68.72	67.99
	MRAN	100	424	520	59.94	59.83
	GAP-RBFN	81	424	452	59.24	58.23
	OS-ELM	300	424	36	68.95	67.56
	ID-SVM	150[a]	424	350	55.45	58.23
	SMC-RBF	75	424	120	74.18	73.52
	SRAN	113	311	55	**75.12**	**76.86**
GI	Batch-SVM	102[a]	109	320	64.23	60.01
	MRAN	51	109	520	63.81	70.24
	GAP-RBFN	75	109	410	58.29	72.41
	OS-ELM	60	109	15	67.62	70.12
	ID-SVM	78[a]	109	212	54.22	50.01
	SMC-RBF	58	109	97	78.09	77.96
	SRAN	59	70	28	**86.21**	**80.95**

[a] Support vectors.

3.2.3. *Discussion*

The SRAN algorithm is a sequential learning algorithm with self-regulated control parameters. Since, the SRAN algorithm uses explicit classification error in growing/learning criterion and discarding similar samples, it prevents overtraining and provides better generalization performance. However, the SRAN sequential learning algorithm also suffers from the following issues:

1. Though the control parameters are self-regulated, the rate at which the control parameters regulate themselves is controlled by the parameter δ, which is fixed a priori. It is usually chosen close to 1. This parameter indirectly depends on the number of training samples, which is not known a priori in sequential learning framework. For problems having fewer training samples in a sequence, the parameter should be initialized to 0.95. The closeness to one depends on number of samples in a sequence.
2. The current algorithm does not include pruning strategy and hence they require more neurons to approximate the decision function. The exiting pruning conditions in MRAN and GAP-RBF depends on the entire training sequence. One needs to find better pruning algorithm which controls the criterion based on the information contained in the network.
3. The learning strategy involves an EKF, which is computationally expensive. The computational cost increases whenever a neuron is added to the network.

4. Conclusion

This paper presents a sequential learning algorithm for Self-adaptive Resource Allocation Network (SRAN). The self-adaptive nature of the algorithm identifies the reduced training data sequence with significant information (to avoid over-training) and produces a compact network using self-regulative control parameters. The SRAN also uses explicit misclassification error and hinge loss function in growing/learning criteria. This results in better approximation of decision function. The performance of the proposed SRAN algorithm has been studied with balanced and imbalance data sets. In all these studies, the performance of SRAN is compared with other existing algorithms like MRAN, GAP-RBF, OS-ELM, SMC-RBF, ID-SVM and batch-SVM. Based on these studies, it can be concluded that:

- SRAN uses the reduced training sample sequence which avoids over-training;
- SRAN produces a compact network with better generalization;
- For SRAN, the training time is small.

However, for large networks, the EKF algorithm used in parameter update equations increases the computational burden. Also, the SRAN does not use any pruning criterion and this may results in large network. Further studies in these directions are needed.

Acknowledgements

This work was in part supported by the IT R&D program of MKE/KEIT. 2008-F-036-02, Development of Anonymity-based-knowledge

Security Technology, and the CTRC program of MCST/KOCCA, and the 3D-Life project supported by the National Research Foundation, Korea.

The authors would like to thank the reviewers for their comments and suggestions which has improved the quality of presentation.

References

[1] S. Amari, Theory of adaptive pattern classifiers, IEEE Trans. EC 16 (3) (1967) 299–307.
[2] C. Blake, C. Merz, UCI repository of machine learning databases, University of California, Irvine, Department of Information and Computer Sciences URL: (⟨http://archive.ics.uci.edu/ml/⟩), 1998).
[3] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, National Taiwan University, Taiwan, Department of Computer Science and Information Engineering, URL ⟨http://www.csie.ntu.edu.tw/cjlin/libsvm/⟩, 2003.
[4] N. Cristianini, J.S. Taylor, An Introduction to Support Vector Machines, Cambridge University Press, Cambridge, UK, 2000.
[5] S. Suresh, N. Sundararajan, P. Saratchandran, A sequential multi-category classifier using radial basis function networks, Neurocomputing 71 (1) (2008) 1345–1358.
[6] G.-B. Huang, P. Saratchandran, N. Sundararajan, An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks, IEEE Trans. Syst., Man Cybern., Part B 34 (6) (2004) 2284–2292.
[7] V. Kadirkamanathan, M. Niranjan, A function estimation approach to sequential learning with neural networks, Neural Comput. 5 (1993) 954–975.
[8] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate on-line sequential learning algorithm for feedforward networks, IEEE Trans. Neural Networks 17 (6) (2006) 1411–1423.
[9] Y.W. Lu, N. Sundararajan, P. Saratchandran, A sequential minimal radial basis function (RBF) neural network learning algorithm, IEEE Trans. Neural Networks 9 (2) (1998) 308–318.
[10] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16–18) (2007) 3056–3062.
[11] J.C. Platt, A resource allocating network for function interpolation, Neural Comput. 3 (2) (1991) 213–225.
[12] L. Yan, N. Sundararajan, P. Saratchandran, Analysis of minimal radial basis function network algorithm for real-time identification of nonlinear dynamic systems, IEE Proc. Part-D: Control Theory Appl. 147 (4) (2000) 476–484.
[13] Y. Lu, N. Sundararajan, P. Saratchandran, A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks, Neural Comput. 9 (2) (1997) 461–478.
[14] B. Fritzke, Fast learning with incremental radial basis function networks, Neural Process. Lett. 1 (1) (1994) 2–5.
[15] B. Fritzke, A growing neural gas network learns topologies, in: G. Tesauro, D.S. Touretzky, T.K. Leen (Eds.), Advances in Neural Information Processing Systems, vol. 7, 1995, pp. 625–632.
[16] A.P. Engelbrecht, A new pruning heuristic based on variance analysis of sensitivity information, IEEE Trans. Neural Networks 12 (6) (2001) 1386–1399.
[17] S. Suresh, N. Sundararajan, P. Saratchandran, Risk sensitive hinge loss functions for sparse multi-category classification problems, Inf. Sci. 178 (12) (2008) 2621–2638.
[18] T. Zhang, Statistical behavior and consistency of classification methods based on convex risk minimization, Ann. Stat. 32 (1) (2003) 56–85.
[19] S. Suresh, R.V. Babu, H.J. Kim, No-reference image quality assessment using modified extreme learning machine classifier, Appl. Soft Comput. 9 (2) (2009) 541–552.
[20] G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning, Advances in Neural Information Processing Systems (NIPS 2000), vol. 13, MIT Press, Cambridge, MA, 2001.
[21] G.V. Puskorius, L.A. Feldkamp, Neurocontrol of nonlinear dynamics systems with Kalman filter trained recurrent networks, IEEE Trans. Neural Networks 5 (2) (1994) 279–297.
[22] R.V. Babu, S. Suresh, A. Makur, Online adaptive radial basis function networks for robust object tracking, Comput. Vision Image Understanding 114 (3) (2010) 297–310.
[23] S. Suresh, S. Saraswathi, N. Sundararajan, Performance enhancement of extreme learning machine for multi-category sparse data classification problems, Eng. Appl. Artif. Intell. 23 (7) (2010) 1149–1157.

Sundaram Suresh received the B.E degree in electrical and electronics engineering from Bharathiyar University in 1999, and M.E (2001) and Ph.D (2005) degrees in aerospace engineering from Indian Institute of Science, India. He was post-doctoral researcher in school of electrical engineering, Nanyang Technological University from 2005 to 2007. From 2007–2008, he was in INRIA-Sophia Antipolis, France as ERCIM research fellow. He was in Korea University for a short period as a visiting faculty in Industrial Engineering. From January 2009 to December 2009, he was in Indian Institute of Technology—Delhi as an Assistant Professor in Department of Electrical Engineering. Currently, he is working as an Assistant Professor in School of Computer Engineering, Nanyang Technological University, Singapore since 2010. His research interest includes flight control, unmanned aerial vehicle design, machine learning, optimization and computer vision.

Keming Dong is a research scholar in Graduate School of Information Management and Security, Korea University, Korea. His research interest includes machine learning, optimization and information security.

Hyoung Joong Kim received his B.S., M.S., and Ph.D. degrees from Seoul National University, Seoul, Korea, in 1978, 1986, 1989, respectively. He joined the faculty of the Department of Control and Instrumentation Engineering, Kangwon National University, Korea, in 1989. He is currently a Professor of the Graduate School of Information Management and Security, Korea University, Korea since 2006. His research interests include parallel and distributed computing, multimedia computing, and multimedia security.