# An Incremental Design of Compact Radial Basis Function Networks

Hao Yu, Philip D. Reiner, Tiantian Xie, Tomasz Bartczak, and Bogdan Wilamowski, *Fellow*, *IEEE*

***Abstract*—This paper proposes an offline algorithm for incrementally constructing and training radial basis function (RBF) networks. In each iteration of the Error Correction (ErrCor) algorithm, one RBF unit is added to fit and then eliminate the highest peak (or lowest valley) in the error surface. This process is repeated until a desired error level is reached. Experimental results on real world data sets show that the ErrCor algorithm designs very compact RBF networks compared to the other investigated algorithms. Several benchmark tests such as the duplicate patterns test and the two spiral problem were applied to show the robustness of the ErrCor algorithm. The proposed ErrCor algorithm generates very compact networks. This compactness leads to greatly reduced computation times of trained networks.**

***Index Terms*—Radial Basis Function Networks, Incremental Design, Error Correction, Levenberg Marquardt Algorithm**

## I. Introduction

ARTIFICIAL neural networks (ANNs) are considered as universal approximators and applied to solve various problems in industrial fields, such as motor control [1-2], robotic manipulators [3-4], nonlinear compensations [5] and pattern recognition [6].

There are two long-time discussed issues in artificial neural networks:

(1) How large should the network architecture should be?
(2) How is a proper training algorithm created or selected?

Issue (1) coincides with a common mistake in neural networks. It is quite easy to get error convergence when training larger than required networks, but these networks in most cases will respond very poorly to patterns not used for training. Therefore, in order to avoid over-fitting problems, the networks should be as compact as possible [7]. Also, it is often the case that first order gradient methods are not able to find solutions for these compact networks. In these situations second order algorithms are superior [8-9].

For issue (2), there is no doubt that gradient-based methods are the most straightforward ways to train neural networks. First order gradient methods [10] are very stable if the learning constant is small, but a tradeoff for good stability is long training time, especially for very accurate approximation. Usually, second order algorithms show faster convergence and more powerful search ability than first order algorithms [11-14].

For RBF networks, issue (1) becomes more complex than neural networks. This is because results depend not on network weights but also on the locations and widths of RBF units [15].

Several strategies and algorithms are proposed to find good initializations of RBF units and proper sizes of RBF network architectures [16-19]. Huang *et al.* proposed a generalized growing and pruning (GGAP) algorithm [20-21] to find proper sizes of RBF networks. Based on the evaluation of "significance", RBF units are added one-by-one with selected initial conditions or disregarded if they make little contribution to network performance. However, the GGAP algorithm could not properly handle problems with complex and high dimensional probability density distribution. This disadvantage of the GGAP algorithm was overcome in [22] by introducing the Gaussian mixture model (GMM) to approximate the GGAP evaluation formula. Compared to the GGAP algorithm, the GGAP-GMM algorithm can design RBF networks with less complexity for solving the same problem. Other methods also attempt to find a correct architecture for practical problems while maintaining a desired accuracy [23-25].

The algorithms mentioned above are primarily developed as online algorithms, but they work well as offline algorithms. The training time is very important for online training algorithms, because networks are being continuously re-trained. However, for offline trained systems which use algorithms such as: Support Vector Regression (SVR) [26-27], Incremental Extreme Learning Machine (I-ELM) [28], Convex I-ELM (CI-ELM) [29], Enhanced random search based I-ELM (EI-ELM) [30], and Decay RBF Neural Networks (DRNN) [31], the execution time of trained networks is much more important. All these offline algorithms perform two tests simultaneously: building a network with RBF units and training it. Notice that the execution time of the trained RBF networks is proportional to the number of RBF units used. Meaning that a compact network is more important for offline algorithms.

Most popular offline algorithms for training RBF networks are only adjusting the easy to train parameters (such as height of RBF units), while other parameters such as locations of RBF centers and widths of RBF units are fixed or selected randomly. The exception is a recently published ISO algorithm [32]. As a consequence, the ISO algorithm is capable of training RBF networks with tens rather than hundreds of units to reasonably low errors. However, the ISO algorithm requires that a network be initialized with a predetermined number of units and a set of randomly generated parameters for each unit. This randomization leads to inconsistent training in some situations. Consequently, the same problem must be trained many times to ensure that a good solution has been reached. And frequently, two or more RBF units are being adjusted to similar locations resulting in a larger than minimal network.

A common disadvantage of the algorithms mentioned above is that before a proper solution is reached, often tens or hundreds of experiments are needed with different settings of training parameters and random selections of initial conditions. However, with the proposed ErrCor algorithm, it is possible to reach an acceptable solution just with one try, because there are no learning parameters to be adjusted and there is no randomness in the process.

The ErrCor algorithm works like other hierarchical learning methods [19-22] and [28-30]. The number of RBF units is increased one by one from scratch, and each new RBF unit is trained using a second order method as in [32] to compensate for residual errors. Once the residual errors reach a desirable level the algorithm stops. The prime goal of this research is to develop algorithms which can reach a desired error with the least number of RBF units. Another benefit of the proposed approach is that such small RBF networks, 10 to 100 times smaller than obtained with other algorithms, are much easier for practical implementation and much faster.

## II. Computational Fundamentals

### A. Review of RBF Networks

Fig. 1 shows the standard three-layer architecture of RBF networks. It consists of $P$, $I$ dimensional inputs $\boldsymbol{x}_p=[\, x_{p,1}, x_{p,2} \ldots x_{p,I}\,]$ in the input layer, $H$ RBF units in the hidden layer and a single linear unit in the output layer. Notice that, for problems with multiple outputs, they can be processed as the combination of several single output sub-problems.

In this paper, equation (1) is applied as the kernel function of RBF units. Therefore, when applying pattern $p$, the output of RBF unit $h$ is calculated as:

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

Where: $\boldsymbol{c}_h$ and $\sigma_h$ are the center and width of RBF unit $h$, respectively. $\|\bullet\|$ represents the computation of Euclidean Norm.

The network output for pattern $p$ is calculated by

$$o_p = \sum_{h=1}^{H} w_h \varphi_h\left(\mathbf{x}_p\right) + w_0 \tag{2}$$

Where: $w_h$ presents the weight on the connection between RBF unit $h$ and network output. $w_0$ is the bias weight of output unit.

### B. ErrCorr Fundamentals

In order to improve the performance of RBF networks, training process is required to adjust parameters, such as centers, widths and output weights. In this paper, a modification of the Levenberg-Marquardt (LM) algorithm is used for parameter adjustment in RBF networks. The LM algorithm as described by [11] cannot be used to train problems with large numbers of patterns, because the size of the Jacobian becomes prohibitively large. Therefore, the algorithm was modified [14]. This way the second order algorithm can be used with data sets with basically unlimited number of training patterns.

For the LM algorithm, [11] [14], the update rule is given by:

$$\Delta_{k+1} = \Delta_k - \left(\mathbf{Q}_k + \mu_k \mathbf{I}\right)^{-1} \mathbf{g}_k \tag{3}$$

Where: vector $\boldsymbol{\Delta}$ consists of parameters, including centers $\boldsymbol{c}$, widths $\boldsymbol{\sigma}$ and weights $\boldsymbol{w}$; $\boldsymbol{Q}$ is the quasi Hessian matrix; $\boldsymbol{I}$ is the identity matrix; $\mu$ is the combination coefficient; $\boldsymbol{g}$ is the gradient vector.

Quasi Hessian matrix $\boldsymbol{Q}$ is calculated as the sum of sub quasi Hessian matrix $\boldsymbol{q}_p$

$$\mathbf{Q} = \sum_{p=1}^{P} \mathbf{q}_p \qquad \mathbf{q}_p = \mathbf{j}_p^T \mathbf{j}_p \tag{4}$$

and gradient vector $\boldsymbol{g}$ is calculated as the sum of sub gradient vector $\boldsymbol{\eta}_p$

$$\mathbf{g} = \sum_{p=1}^{P} \boldsymbol{\eta}_p \qquad \boldsymbol{\eta}_p = \mathbf{j}_p^T e_p \tag{5}$$

Where: the training error $e_p$ is calculated as the difference between desired output $y_p$ and actual output $o_p$ (2):

$$e_p = y_p - o_p \tag{6}$$

and elements of Jacobian row $\boldsymbol{j}_p$ is calculated by:

$$j_{p,n} = \frac{\partial e_p}{\partial \Delta_n} \tag{7}$$

Considering the network parameters $w_h$, $c_{h,i}$ and $\sigma_h$, for a given pattern $p$, elements the Jacobian row can be organized as

$$\begin{aligned}\mathbf{j}_p = \Bigg[ &\frac{\partial e_p}{\partial w_0}, \frac{\partial e_p}{\partial w_1} \cdots \frac{\partial e_p}{\partial w_h} \cdots \frac{\partial e_p}{\partial w_H}, \frac{\partial e_p}{\partial c_{1,1}} \cdots \frac{\partial e_p}{\partial c_{1,i}} \cdots \\ &\frac{\partial e_p}{\partial c_{1,I}} \cdots \frac{\partial e_p}{\partial c_{h,1}} \cdots \frac{\partial e_p}{\partial c_{h,i}} \cdots \frac{\partial e_p}{\partial c_{h,I}} \cdots \frac{\partial e_p}{\partial c_{H,1}} \cdots \\ &\frac{\partial e_p}{\partial c_{H,i}} \cdots \frac{\partial e_p}{\partial c_{H,I}}, \frac{\partial e_p}{\partial \sigma_1} \cdots \frac{\partial e_p}{\partial \sigma_h} \cdots \frac{\partial e_p}{\partial \sigma_H} \Bigg]\end{aligned} \tag{8}$$

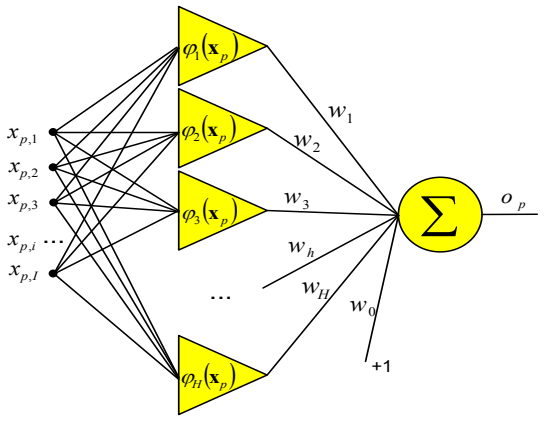



Fig. 1 RBF network with $I$ inputs, $H$ RBF units and one output unit.

Integrating equations (1), (2) and (6), with differential chain rule, the Jacobian row elements for pattern $p$ in (8) can be rewritten as:

$$\frac{\partial e_p}{\partial w_h} = -\varphi_h(\mathbf{x}_p) \qquad \frac{\partial e_p}{\partial w_0} = -1 \tag{9}$$

$$\frac{\partial e_p}{\partial c_{h,i}} = -\frac{2w_h\varphi_h(\mathbf{x}_p)(x_{p,i} - c_{h,i})}{\sigma_h} \tag{10}$$

$$\frac{\partial e_p}{\partial \sigma_h} = -\frac{w_h\varphi_h(\mathbf{x}_p)\|\mathbf{x}_p - \mathbf{c}_h\|^2}{\sigma_h^2} \tag{11}$$

With equations (9-11), all elements of Jacobian row $\boldsymbol{j}_p$ for the given pattern $p$ can be calculated. After applying all patterns, quasi Hessian matrix $\boldsymbol{Q}$ and gradient vector $\boldsymbol{g}$ are obtained by equations (4-5), so as to apply the update rule (3) for parameter adjustment.

## III. ErrCorr Description

The basic idea of the ErrCor algorithm is to use RBF units with kernel function (1) to create a peak/valley shape to compensate for the largest error in the error surface at the beginning of each iteration.

In each experiment, for comparison, the Mean Square Error (MSE) and the Root Mean Square Error (RMSE) is used:

$$RMSE = \sqrt{\frac{1}{P}\sum_{p=1}^{P} e_p^2} \tag{12}$$

where: $P$ is the number patterns; $e_p$ is the training/testing error for a pattern $p$, as given by (6).

### A. Graphical Interpretation of the Algorithm

In order to illustrate the error correction process, let us present an example. ErrCor will be used to approximate the sinusoidal function (13) as shown in Fig. 2.

$$y = \sin x \tag{13}$$

By going through the data of the curve in Fig. 2, the location ($\boldsymbol{X_A}$=4.7, $\boldsymbol{Y_A}$=-0.9999) of the lowest valley (marked as $\boldsymbol{A}$) is found. Then the first RBF unit can be added with initial center ($\boldsymbol{X_A}$=4.7), as shown in Fig. 3a. By applying the LM algorithm (section II.B) for parameter adjustment, the trained network is shown in Fig. 3b. Based on the training results, the outputs of the RBF network (Fig. 3b) are visualized in Fig. 4a and new error curve (Fig. 4b) is obtained as the difference between Fig. 2 and Fig. 4a. Comparing the error curves in Figs. 2 and Fig. 4b, one may notice that, the lowest valley (marked as $\boldsymbol{A}$) in Fig. 2 is eliminated from Fig. 4b. This results in an RMSE of 0.2000.

By observing the error curve in Fig. 4b, the lowest valley (marked as $\boldsymbol{B}$) with the coordinates ($\boldsymbol{X_B}$=0.0, $\boldsymbol{Y_B}$=-0.7238) can be found. Then, the second RBF unit is added with initial center ($\boldsymbol{X_B}$=0.0) as shown in Fig. 5a. After the training process, the network parameters have been adjusted as shown in Fig. 5b. Fig. 6 presents the actual outputs and errors obtained from the trained network (Fig. 5b). Again, one may notice that, the lowest valley (marked as $\boldsymbol{B}$) in error surface Fig. 4b is eliminated from Fig. 6b.

Following this procedure, as the size of the RBF network grows, the error values in the error curve continuously decrease. After two nodes are created in the network, the RMSE becomes 0.0025. TABLE I shows the training results as the network is constructed up to 5 nodes.

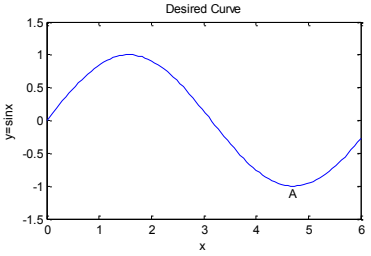
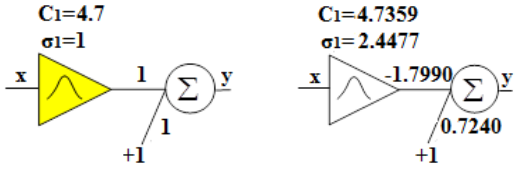
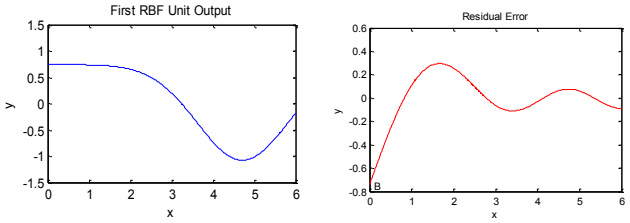



Fig. 2 Desired sinusoidal function



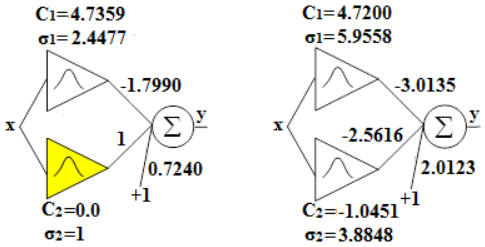

(a) (b)

Fig. 3 RBF network with 1 RBF unit: (a) initial RBF network; (b) trained RBF network. Yellow RBF unit is newly added



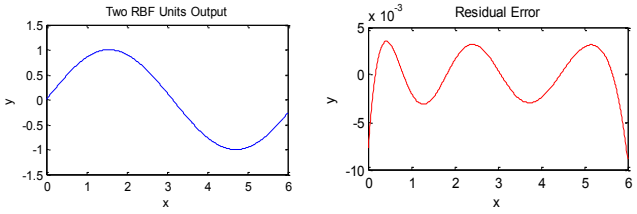

(a) (b)

Fig. 4 Result curves of the RBF network in Fig. 4b with 1 RBF unit: (a) actual output curve; (b) error curve




(a) (b)

Fig. 5 RBF network with 2 RBF units: (a) initial RBF network; (b) trained RBF network. Yellow RBF unit is newly added




(a) (b)

Fig. 6 Result curves of the RBF network in Fig. 5b with 2 RBF units: (a) actual output curve; (b) error curve

TABLE I
TRAINING RMSE AND #RBF UNITS FOR SINE PROBLEM

| # Units | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| RMSE | 0.2000 | 0.0025 | 0.0003 | 5.2e-5 | 2.5e-5 |

### B. Algorithm Implementation

Generally, the proposed ErrCor algorithm can be organized as the pseudo code shown in Fig. 7.

One may notice that, all the parameters are readjusted when the network is changed. It turns out that much better results can be obtained when the entire network is retrained after each RBF unit is added instead of training only the newly added RBF unit. Of course this makes training times longer, but the training time of off-line algorithms is less important than finding the optimal RBF network.

This algorithm is similar to the ISO algorithm described in [32] as both algorithms are trained using a modified version of the second order algorithm [14]. The difference, however, lies in the fact that the ISO algorithm requires a RBF network architecture initially to be specified. Then multiple training runs with randomly initialized RBF units are needed to reach satisfactory results (Fig. 8). The ErrCor algorithm, on the other hand, creates the network architecture by starting with an empty network and then adding RBF units one by one to compensate for the largest error at that time. The initial location of a new RBF unit is set at the pattern with the maximum error with can be recorded during error surface computation. Of course the algorithm is capable to move RBF centers to better location, but selection of location of the pattern with max error seems to be a better choice than random selection. This creates a network that is as small as possible and guarantees convergence with one training run. Fig. 8 depicts this difference in training for the ISO algorithm and the ErrCor algorithm.

## IV. EXPERIMENTS WITH HIGHLY NONLINEAR BENCH TESTS

Most of the real data sets are not highly nonlinear, and good results can be obtained with very few RBF units (see Table V). Therefore, in this section, the ErrCor algorithm is applied to many well-known nonlinear bench tests to demonstrate the power and robustness of the algorithm. Notice that if noise is not present, training and validation errors are similar. These benchmark tests are organized as follows: Rapidly Changing Function, Peaks Problem, and Two Spiral Problem.

The testing environment of the proposed algorithm consists of a Windows 7 64-bit operating system, an Intel Core i7-2600 CPU @ 3.4 GHz processor, and 8GB RAM.

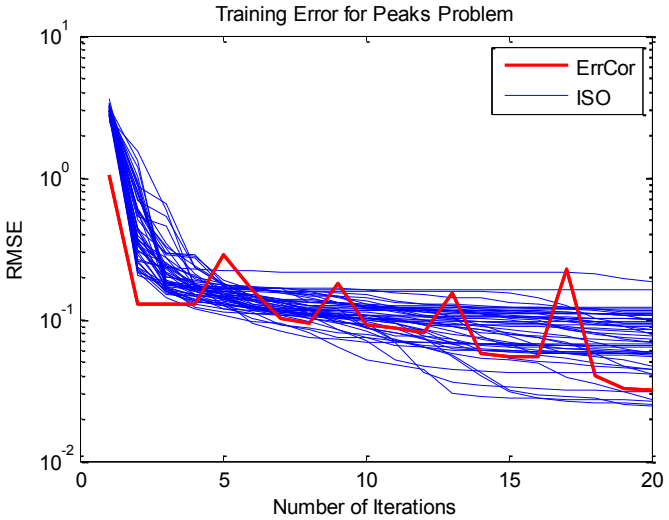



Fig. 8 The ISO algorithm and the ErrCor algorithm on the Peaks problem using 5 RBF units. Notice that the ISO algorithm errors vary greatly due to the random start points, while ErrCor reaches small error with a single try.

```
Initialization
for n = 1:NMAX    \\ for all new RBF units
    find error vector  using  eq. ( 6)   as a difference of desired and actual outputs
    find maximum of abs error  of the error vector e
    create a new RBF unit at location of maximum error by setting weight and width of the new RBF unit to 1
    calculate  RMSE (iter=1)
    for iter = 2:max_iteration
        for all patterns
            calculate Jacobian vector j_p using eq.  (8) and  eqs.  (9),(10),(11)
            calculate  sub quasi Hessian matrix   ,  calculate  sub gradient vector
        end (all patterns)
        calculate quasi-Hessian matrix Q using  eq. (4);  calculate and gradient vector  g using eq. (5)
        update network parameters using eq. (3);  calculate new output value and new error surface
        calculate RMSE(iter)
        while error is not reduced
            adjust the   parameter in eq. (3)  using the LM scheme [11]
        endwhile (error is not reduced)
        if RMSE(iter) < desired error then break  (max_iteration loop)
    endfor  (max_iterations)
    if RMSE(n) < desired_error,  then break  (new RBF unit loop)
endfor (main loop)
```

Fig. 7 Pseudo code of the proposed ErrCor algorithm

### A. Rapidly Changing Function

In this experiment, the proposed algorithm is applied to design RBF networks to approximate the following rapidly changing function this is the same function used to test many popular algorithms in [20].

The formula for this benchmark problem is the following [20]:

$$y(x) = 0.8\exp(-0.2x)\sin(10x) \tag{14}$$

In this problem, there are 3000 training patterns with x-coordinates uniformly distributed in range [0, 10]. The validation data set consists of 1500 patterns with x-coordinates randomly generated in the same range [0, 10].

Figs. 9 and 10 show the testing results of the proposed ErrCor algorithm, with the number of RBF units equal to 10 and 20 respectively. Fig. 11 shows the training results of proposed ErrCor algorithm and several other algorithms. One may notice that the proposed ErrCor algorithm can reach a similar training/testing error level with a 3 to 30 times smaller network.

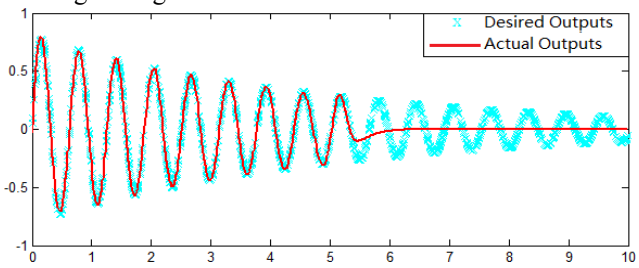



Fig. 9 Testing results of ErrCor algorithm with 10 RBF units; $E_{Train}=7.846\times10^{-3}$ and $E_{Test}=7.516\times10^{-3}$

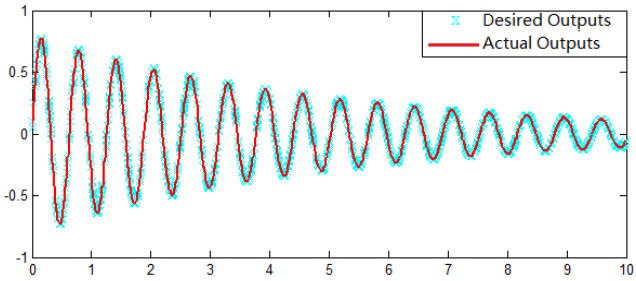



Fig. 10 Testing results of ErrCor algorithm with 20 RBF units; $E_{Train}=5.428\times10^{-6}$ and $E_{Test}=5.347\times10^{-6}$

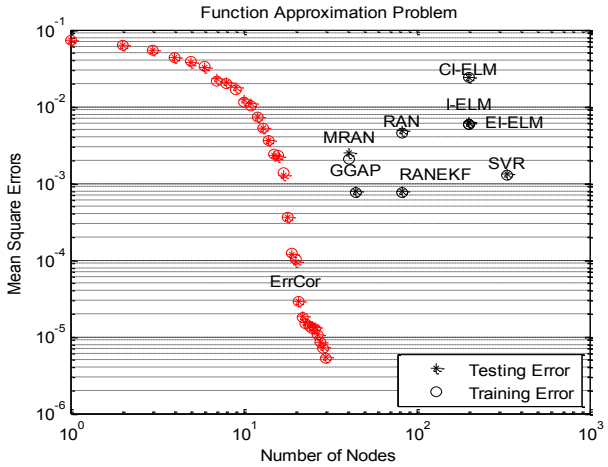



Fig. 11 Function approximation problem: training/testing average sum square errors vs. average number of RBF units

Table II presents the comparison of average training time, training errors, testing time, and testing error for each algorithm. For the proposed ErrCor algorithm, the computation time is counted until the RBF network with 20 units (with smaller training/testing errors than other algorithms) gets trained. For the ELM algorithms, the centers were generated from the input range [0,10] while impact factors were from the range (0,0.5]. For GAP-RBF the parameters are fixed at $\epsilon_{max} = 1.15, \epsilon_{min} = 0.04, \kappa = 0.10, and\ \gamma = 0.999$. For the MRAN algorithm, the threshold for growing and pruning was set as $e_{min}^{MR} = 0.06$, and the appropriate size of the sliding window was chosen as $M = 100$. The parameters for GGAP were $e_{min}^{GG} = 0.00001$. For SVR, the parameter $C$ was tuned to 1000 while γ was set at 1.

In order to provide a measure independent of physical CPU power, a normalized computation time was used to determine the efficiency of the constructed networks. The normalization was done by first testing two different data sets on networks of different sizes twenty times each. The average computation time per RBF unit per testing input was 1.195μs.

### B. Peaks Problem

The peaks problem is a problem with a two dimensional input that yields an output with many peaks and valleys; the peaks problem provides a way to easily visualize the training process of the various algorithms. The peaks problem consists of 2000 randomly generated patterns in the range (-1,+1) for both $x$ and $y$ directions using the formula :

$$\begin{aligned} z(x,y) = &-\frac{1}{30}\exp\left(-1-6x-9x^2-9y^2\right)+ \\ &-\left(0.6x-27x^3-243y^5\right)\exp\left(-9x^2-9y^2\right)+ \\ &+\left(0.3-1.8x+2.7x^2\right)\exp\left(-1-6y-9x^2-9y^2\right) \end{aligned} \tag{15}$$

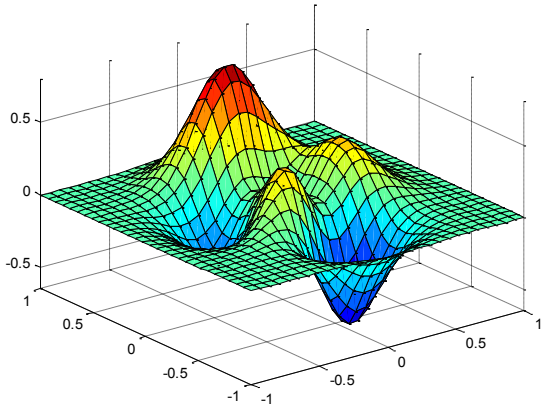


Fig. 12 The desired output for the peaks problem.

Another 1000 randomly generated patterns were used for the validation. Fig. 12 depicts the desired surface for the peaks problem.

As can be seen in Fig. 13, the major peaks and valleys of the desired output are targeted by the ErrCor algorithm with only five RBF units. This compact network achieves a validation RMSE of 0.031. As training continues, the error decreases steadily as units are added until the RMSE reaches about 0.0003 with 20 units. As was expected, after five RBF units

were added to the network, the centers of the RBF units in the trained network are located aproximately in the centers of the highest peaks and valleys. What is interesting however, is that after twenty RBF units were added, the centers had moved to completely different locations.

In comparison to the other algorithms, ErrCor was able to reach a much smaller RMSE with much fewer RBF units. This demonstrates that the ErrCor algorithm is very efficient when choosing heights, widths, and centers of the RBF units. The ELM family of algorithms was tested on this problem and was able to achieve an RMSE of about 0.03 with one thousand RBF units (See Figs. 13d, 14, and 15). This error is still 100 times larger than the error obtained with only 20 RBF units using the ErrCor algorithm (RMSE = 0.0003). The SVR algorithm used thirty-six support vectors to achieve an RMSE of 0.031 (See Fig. 16). Still, this requires about seven times more units than ErrCor for the same error.

TABLE II
COMPARISON OF TRAINING TIMES/ERRORS AND VALIDATION TIMES PER PATTERN/ERRORS FOR THE FUNCTION APPROXIMATION PROBLEM

| Function Approximation | | | | |
|---|---|---|---|---|
| Algorithm | Train Time (s) | Train RMSE | Test Time (μs) | Test RMSE |
| GGAP | 24.808 | 0.0265 | 54.16 | 0.0265 |
| MRAN | 78.572 | 0.0458 | 52.15 | 0.0490 |
| RANEKF | 105.72 | 0.0265 | 106.8 | 0.0265 |
| RAN | 45.514 | 0.0671 | 112.2 | 0.0686 |
| SVR | 0.2552 | 0.0346 | 2496 | 0.0361 |
| I-ELM | 0.5509 | 0.0831 | 239.0 | 0.0843 |
| CI-ELM | 0.5597 | 0.1356 | 239.0 | 0.1378 |
| EI-ELM | 5.3991 | 0.0728 | 239.0 | 0.0755 |
| ErrCor | 48.530 | 0.0141 | 23.90 | 0.0141 |

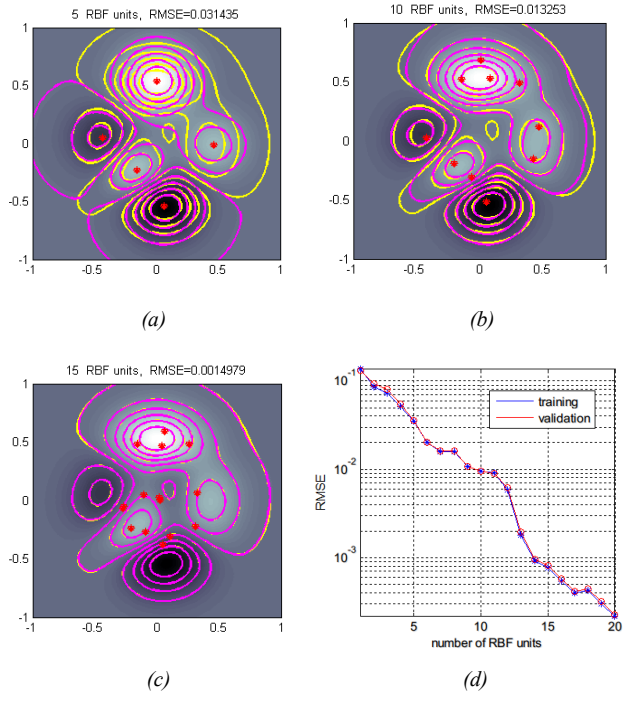



Fig. 13 ErrCor output. The yellow contour depicts the desired surface, the purple contour depicts the network output, and the red asterisks show where the centers of the RBF units are located.

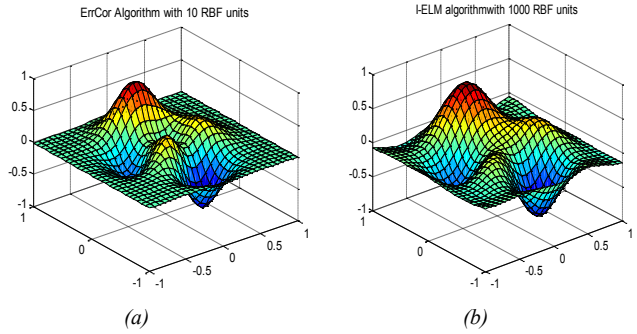



Fig. 14 ErrCorr output using 10 nodes, (a) compared to ELM output using 1000 nodes, (b).

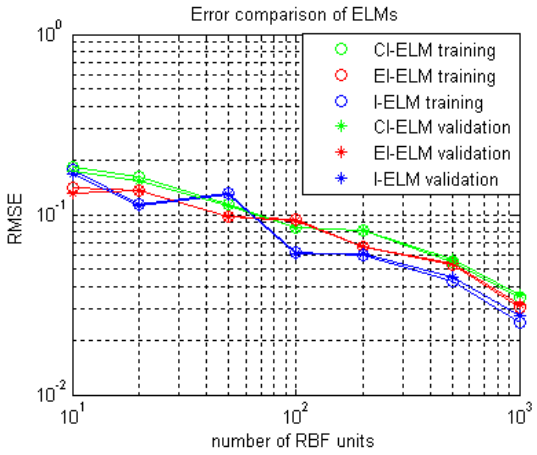



Fig. 15 Comparison of the three ELM algorithms on the peaks problem. All three attain similar errors. The random centers for the ELM algorithm were generated in the range of inputs [-1,1] while the impact factors were in the range (0,0.5].

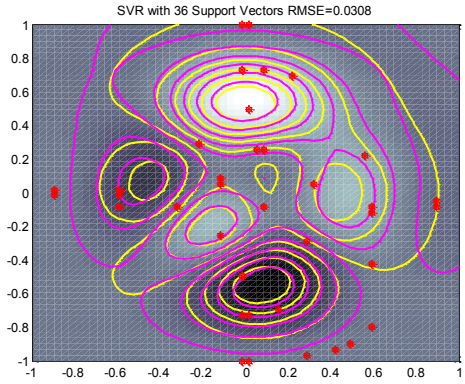



Fig. 16 SVR output. The yellow contour depicts the desired surface, the purple contour depicts the algorithm output, and the red asterisks show where the support vectors are located. The SVR parameters used were: G=0.3, Epsilon = 0.001, and C=10.

## C. *Two-Spiral Problem*

The two-spiral problem is primarily used as a benchmark for pattern classification. It can also be used as an approximation problem where patterns on one spiral should produce +1 outputs, while patterns on the other spiral should produce -1 outputs.

This problem is widely used as a challenging benchmark to evaluate the efficiency of learning algorithms and their network architectures. For the purpose of approximation the two-spiral

data set needs to be better defined, so in this paper 388 patterns were used instead of the typical 194 patterns.

This problem is widely used as a challenging benchmark to evaluate the efficiency of learning algorithms and their network architectures. For the purpose of approximation the two-spiral data set needs to be better defined, so in this paper 388 patterns were used instead of the typical 194 patterns.

The RBF-MLP networks proposed in [33] required at least 74 RBF units to solve the two-spiral problem. It was reported in [35] that the two-spiral problem was solved using 70 hidden RBF units. Using the ortho-normalization procedure in [34], the two-spiral problem can be solved with at least 64 RBF kernel functions.

Applying the ErrCor algorithm, Fig. 17 shows several steps in the training process. One may notice that, each newly added RBF unit contributes the error reduction during the training process. The ErrCorr algorithm constructs the network by adding one RBF unit at a time, and with 22 RBF units the training error drops below 0.003 (Fig 18). The SVR algorithm was tested using the LIBSVM package in [36]. SVR was trained to the two spiral problem using the parameters, C=1, G=0.5, and epsilon = 0.01. This output can be seen in Fig. 19.

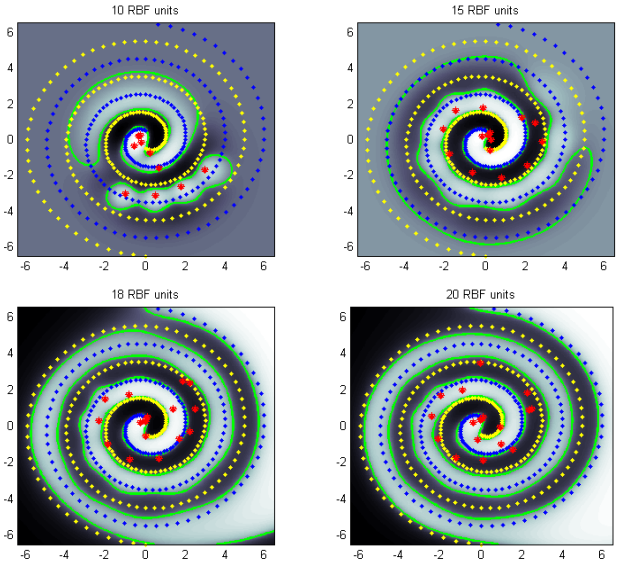



Fig. 17 The ErrCor algorithm incrementally solves the two spiral problem. The two classes of patterns are shown as blue and yellow asterisks, while the green contour shows the network output.

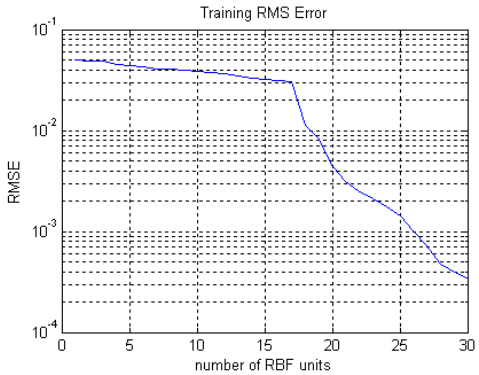



Fig. 18 The RMSE for ErrCor as each node is added to solve the two spiral problem.

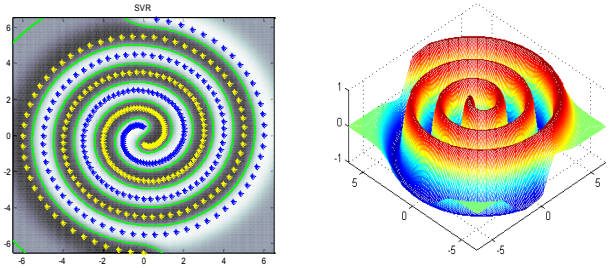



Fig. 19 The SVR algorithm output for the two-spiral problem. 297 patterns were used as support vectors to reach an RMSE of 0.003.

## V. Experiments With Real Life Problems

This section compares ErrCor with well-known algorithms on traditional benchmarks from various repositories, [37]. These are real life problems with many dimensions and with number of patterns from hundreds to thousands. Table III depicts the specifications of the benchmark data sets. In our experiments, all of the inputs have been normalized into the range [-1,1] while the outputs have been normalized into [0,1].

In each benchmark samples are randomly divided into two categories: training samples and validation samples. These experiments are repeated with 20 different random selections so the average and standard deviation results can be evaluated. Tables IV and V and Figs. 20 and 21 present more detailed comparisons on the Abalone and Auto-MPG datasets. These comparisons are given to compare the behavior of the ErrCor algorithm with other popular algorithms. Table VI presents a comparison of validation errors and units required to reach the desired errors by currently popular algorithms on all of the datasets in Table III.

The proposed algorithm was compared with other algorithms such as: GAP [20], GGAP [21], GGAP-GMM [22], SVR [23-24], I-ELM [28], CI-ELM [29], EI-ELM [30], MRAN [38], RAN-EKF [39], NME-ELM [40], and RAN [41]. The parameters for these algorithms were set based on the data presented in the aforementioned papers. For all data sets, the ELM algorithm parameters were centers in the range of inputs, [-1,1], and impact factors in the range (0, 0.5]. For GAP-RBF the parameters are fixed at $\epsilon_{max} = 1.15, \epsilon_{min} = 0.04, \kappa = 0.10,\ and\ \gamma = 0.999$. For the MRAN algorithm, the threshold for growing and pruning was set as $e_{min}^{MR} = 0.0001$, and the appropriate size of the sliding window was chosen as $M = 50$. The parameters for GGAP were $e_{min}^{GG} = 0.00008$ and $e_{min}^{GG} = 0.00007$ for Abalone and Auto-MPG respectively. For GGAP-GMM the parameters for the significance threshold are $e_{min}^{gmm} = 0.08, \eta = 0.1$ for Abalone and $e_{min}^{gmm} = 0.11, \eta = 0.06$ for Auto-MPG. The DRNN algorithm used a parameter of A=2000 and A=40 for the abalone and fuel consumption datasets respectively. The parameters for SVR are mentioned in Table V.

As before, the testing environment of the proposed algorithm consists of a Windows 7 64-bit operating system, an Intel Core i7-2600 CPU @ 3.4 GHz processor, and 8GB RAM.

TABLE III
SPECIFICATION OF BENCHMARK DATA SETS

| REAL WORLD PROBLEM | # TRAIN PATTERNS | # TEST PATTERNS | INPUT DIMENSIONS |
|---|---|---|---|
| Abalone | 2000 | 2177 | 8 |
| Auto-MPG | 320 | 78 | 7 |
| Auto-Price | 80 | 79 | 15 |
| Bos Housing | 250 | 256 | 13 |
| Cal Housing | 8000 | 12640 | 8 |
| Delta-Ailerons | 3000 | 4129 | 5 |
| Delta-Elevators | 4000 | 5517 | 6 |
| Machine CPU | 100 | 109 | 6 |

It can be noticed from Figs. 20 and 21 that, the proposed ErrCor algorithm reaches smaller training/testing errors with a more compact RBF architecture than the other algorithms. Longer training with more than four RBF units leads to smaller training errors, but greater validation errors due to over-fitting. One may notice that other offline algorithms such as ELM, SVR, or DRNN give much worse results. DRNN was omitted from Fig 20 because the best case yielded a validation error of RMSE=0.34.

A comparison of training times for different algorithms on both the Abalone and the Fuel Consumption data sets can be seen in TABLE III. Again, the proposed ErrCor algorithm has a larger training time than the SVR, I-ELM, and CI-ELM algorithms, but a faster training time than the GGAP, MRAN, RANEKF, RAN, and EI-ELM algorithms. Notice that the SVR algorithm may show a lower training error than ErrCor because ErrCor training was stopped when a very small validation error was reached.

A more important comparison for the purpose at hand is that of validation times. This comparison answers the question, "How efficient is the network once it has been trained?" In general, for RBF networks, this will be determined by how many units are in the network. As in section IV, a normalized computation time for RBF calculation was used to calculate the testing time for each algorithm. A comparison of computation time for testing patterns is shown in TABLE IV.

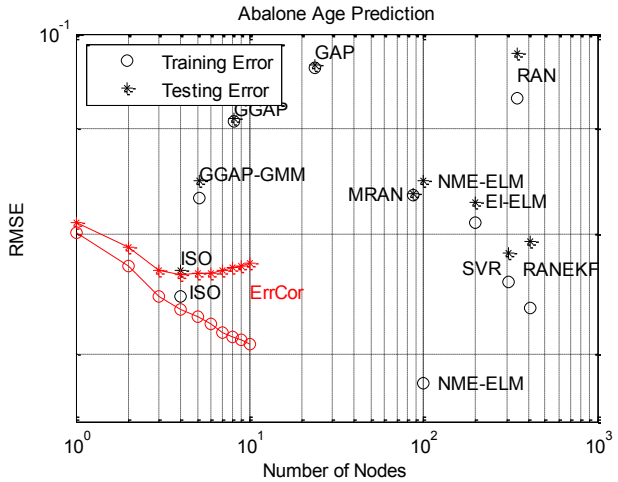



Fig. 20 Abalone age prediction problem: training/testing average sum square errors vs. average number of RBF units.

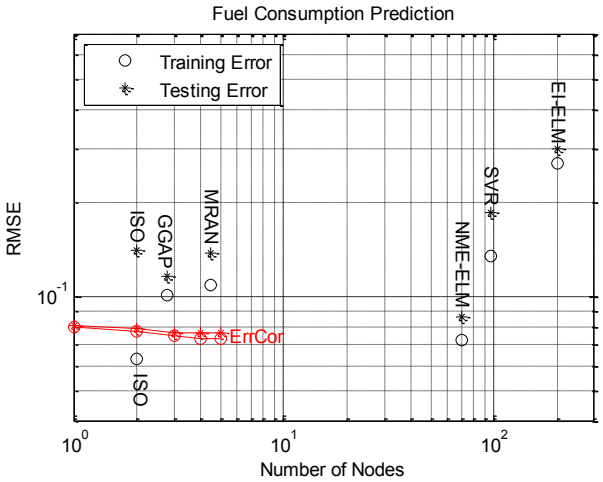



Fig. 21 Fuel consumption prediction problem: training/testing average sum square errors vs. average number of RBF units.

TABLE IV
COMPARISON BETWEEN TRAINING TIMES AND TRAINING ERRORS FOR ABALONE AND FUEL CONSUMPTION PROBLEM

| | Abalone | | Fuel Consumption | |
|---|---|---|---|---|
| Algorithm | Time(s) | RMSE | Time(s) | RMSE |
| GAP | 14.28 | 0.0963 | 0.4524 | 0.1144 |
| MRAN | 255.8 | 0.0836 | 1.4644 | 0.1086 |
| RANEKF | 15480 | 0.0738 | 1.0103 | 0.1088 |
| RAN | 105.17 | 0.0931 | 0.8042 | 0.2923 |
| SVR | 0.4446 | 0.0759 | 0.0210 | 0.0465 |
| I-ELM | 0.5990 | 0.0920 | 0.0593 | 0.0949 |
| CI-ELM | 0.6635 | 0.0827 | 0.0612 | 0.0929 |
| EI-ELM | 5.732 | 0.0811 | 0.5638 | 0.0930 |
| DRNN | 9.404 | 0.0820 | 0.0837 | 0.3506 |
| ISO | 8.497 | 0.0747 | 0.6657 | 0.0724 |
| NME-ELM | 0.0944 | 0.0678 | 0.0118 | 0.0328 |
| ErrCor | 4.808 | 0.0758 | 0.5030 | 0.0671 |

TABLE V
COMPARISON BETWEEN VALIDATION TIMES PER PATTERN AND VALIDATION ERRORS FOR ABALONE AND FUEL CONSUMPTION PROBLEM

| | Abalone | | Fuel Consumption | |
|---|---|---|---|---|
| Algorithm | Time(s) | RMSE | Time(s) | RMSE |
| GAP | 2.82e-5 | 0.0966 | 3.73e-6 | 0.1404 |
| MRAN | 1.05e-4 | 0.0837 | 5.33e-6 | 0.1376 |
| RANEKF | 4.89e-4 | 0.0794 | 6.14e-6 | 0.1387 |
| RAN | 4.13e-4 | 0.0978 | 5.31e-6 | 0.3081 |
| SVR | 6.75e-4 | 0.0784 | 1.15e-4 | 0.0785 |
| I-ELM | 2.39e-4 | 0.0938 | 2.39e-4 | 0.0970 |
| CI-ELM | 2.39e-4 | 0.0857 | 2.39e-4 | 0.1105 |
| EI-ELM | 2.39e-4 | 0.0829 | 2.39e-4 | 0.0892 |
| DRNN | 2.39e-3 | 0.3361 | 3.82e-4 | 0.3098 |
| ISO | 4.78e-6 | 0.0770 | 2.39e-6 | 0.1445 |
| NME-ELM | 1.20e-4 | 0.0849 | 8.37e-5 | 0.0861 |
| ErrCor | 3.59e-6 | 0.0765 | 3.59e-6 | 0.0792 |

TABLE VII
VALIDATION ERRORS FOR ErrCor, ELM FAMILY, AND SVR ALGORITHMS ON SEVERAL REAL LIFE PROBLEMS
(ELM ALGORITHMS WERE RUN USING 200 HIDDEN RBF UNITS)

| REAL WORLD PROBLEM | I-ELM | | EI-ELM (K=10) | | SVR (RBF) | | | | ErrCor | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | RMSE | | RMSE | | # | RMSE | | PARAMS | # | RMSE | |
| | MEAN | DEV | MEAN | DEV | UNITS | MEAN | DEV | $C, \gamma$ | UNITS | MEAN | DEV |
| Abalone | 0.0938 | 0.0053 | 0.0829 | 0.0027 | 310 | 0.0846 | 0.0013 | $(2^4, 2^{-6})$ | 4 | 0.0765 | 0.0012 |
| Auto-MPG | 0.0970 | 0.0142 | 0.0892 | 0.0095 | 96 | 0.0785 | 0.0087 | $(2^0, 2^0)$ | 3 | 0.0792 | 0.0092 |
| Auto-Price | 0.1261 | 0.0255 | 0.1139 | 0.0189 | 22 | 0.1052 | 0.0040 | $(2^8, 2^{-5})$ | 2 | 0.0909 | 0.0275 |
| Boston Housing | 0.1320 | 0.0126 | 0.1077 | 0.0084 | 47 | 0.1155 | 0.0079 | $(2^4, 2^{-3})$ | 4 | 0.0989 | 0.0341 |
| California Housing | 0.1731 | 0.0081 | 0.1503 | 0.0022 | 2189 | 0.1311 | 0.0011 | $(2^3, 2^1)$ | 10 | 0.1223 | 0.0016 |
| Delta-Ailerons | 0.0632 | 0.0116 | 0.0448 | 0.0065 | 83 | 0.0467 | 0.0010 | $(2^3, 2^{-3})$ | 3 | 0.0394 | 0.0007 |
| Delta-Elevators | 0.0790 | 0.0123 | 0.0575 | 0.0047 | 261 | 0.0603 | 0.0005 | $(2^0, 2^{-2})$ | 3 | 0.0532 | 0.0005 |
| Machine CPU | 0.0674 | 0.0177 | 0.0554 | 0.0148 | 8 | 0.0620 | 0.0180 | $(2^6, 2^{-4})$ | 1 | 0.0430 | 0.0293 |

## VI. CONCLUSION

This paper presents an algorithm for incrementally constructing single layer RBF networks. This algorithm works by adding a new RBF unit at the location of the highest error peak or lowest error valley. The widths, heights, and locations of the RBF units in the network are then optimized to reduce error using a modified second order algorithm [14]. The current training error for the problem is then evaluated by the difference between the network output and the training data output. This process is repeated until a desired level of training error is reached. This algorithm takes advantage of both a second order optimization process to converge very quickly at each step and an incremental network construction process to minimize the number of units needed to solve a problem.

There are three reasons why the ErrCor algorithm outperforms other commonly used algorithms to train RBF networks:

First, our algorithm is capable of solving all problems with a smaller number of RBF units than any other algorithm (based on our experiments and based on results published by others). Our algorithm requires a slightly longer training time than other algorithms, but in the case of offline systems, the training time is less important. However, if we make a fair comparison by measuring the training time required to reach a given accuracy, it turns out that our algorithm requires a shorter training time than other algorithms in many cases.

Second the execution time of a trained RBF network is proportional to the number of RBF units used. Because we are able to train the network with the smallest number of RBF units, our algorithm allows execution times that are tens to hundreds of times shorter than other algorithms. In the case of problems with a large number of training patterns, our execution is significantly faster than algorithms such as SVR (support vector regression) or algorithms from the ELM (extreme learning machine) family.

Third, training algorithms need experimentally chosen parameters for the best training results. For example, SVR needs experimental selection of: soft margin, radius of RBF units, or size of insensitive tube. Consequentially, tens or hundreds of training processes must be performed before a satisfactory solution can be found. The advantage of our algorithm is that there is no need for experimentally selected training parameters. Using the ErrCor algorithm, acceptable results are obtained with one training attempt.

## REFERENCES


[1] Q. N. Le and J. W. Jeon, "Neural-Network-Based Low-Speed-Damping Controller for Stepper Motor With an FPGA," *IEEE Trans. on Industrial Electronics*, vol. 57, no. 9, pp. 3167-3180, Sep. 2010.

[2] C. Xia, C. Guo and T. Shi, "A Neural-Network Identifier and Fuzzy-Controller-Based Algorithm for Dynamic Decoupling Control of Permanent-Magnet Spherical Motor," *IEEE Trans. on Industrial Electronics*, vol. 57, no. 8, pp. 2868-2878, Aug. 2010.

[3] H. Chaoui, P. Sicard and W. Gueaieb, "ANN-Based Adaptive Control of Robotic Manipulators With Friction and Joint Elasticity," *IEEE Trans. on Industrial Electronics*, vol. 56, no. 8, pp. 3174-3187, Aug. 2009.

[4] L. Y. Wang, T. Y. Chai and L. F. Zhai, "Neural-Network-Based Terminal Sliding-Mode Control of Robotic Manipulators Including Actuator Dynamics," *IEEE Trans. on Industrial Electronics*, vol. 56, no. 9, pp. 3296-3304, Sep. 2009.

[5] Y. L. Chow, L. L. Frank and etc., "Disturbance and Friction Compensations in Hard Disk Drives Using Neural Networks," *IEEE Trans. on Industrial Electronics*, vol. 57, no. 2, pp. 784-792, Feb. 2010.

[6] S. Ferrari, F. Bellocchio, V. Piuri and N. A. Borghese, "A Hierarchical RBF Online Learning Algorithm for Real-Time 3-D Scanner," *IEEE Trans. on Neural Networks*, vol. 21, issue 2, pp. 275-285, 2010.

[7] B. M. Wilamowski, "Neural Network Architectures and Learning Algorithms: *How Not to Be Frustrated with Neural Networks*," *IEEE Industrial Electronics Magazine,* vol. 3, no. 4, pp. 56-63, Dec. 2009.

[8] N. Fnaiech, F. Fnaiech, B. W. Jervis and M. Cheriet, "The Combined Statistical Stepwise and Iterative Neural Network Pruning Algorithm," *Intelligent Automation and Soft Computing*, vol. 15, no. 4, pp. 573-589, 2009.

[9] A. P. Engelbrecht, "A New Pruning Heuristic Based on Variance Analysis of Sensitivity Information," *IEEE Trans. on Neural Networks*, vol. 12, no. 6, pp. 1386 –1399, Nov. 2001.

[10] P. J. Werbos, "Back-propagation: Past and Future," *Proceeding of International Conference on Neural Networks*, San Diego, CA, 1, 343-354, 1988.

[11] M. T. Hagan, M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Trans. on Neural Networks*, vol. 5, no. 6, pp. 989-993, Nov. 1994.

[12] B. M. Wilamowski, N. Cotton, J. Hewlett, and O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms", 11th *INES 2007 -International Conference on Intelligent Engineering Systems,* Budapest, Hungary, June 29 2007-July 1 2007, pp. 127-132.

[13] B. M. Wilamowski, H. Yu, "Neural Network Learning without Backpropgation," *IEEE Trans. on Neural Networks*, vol. 21, no. 11, pp. 1793-1803, Nov. 2010.

[14] B. M. Wilamowski, H. Yu, "Improved Computation for Levenberg Marquardt Training," *IEEE Trans. on Neural Networks*, vol. 21, no. 6, pp. 930-937, June 2010.

[15] C. Panchapakesan, M. Palaniswami, D. Ralph and C. Manzie, "Effects of Moving the Centers in an RBF Network," *IEEE Trans. on Neural Networks*, vol. 13, no. 6, pp. 1299-1307, Nov. 2002.

[16] B. Fritzke, "Fast Learning with Incremental RBF Networks," *Neural Processing Letters*, vol. 1, no. 1, pp. 2-5, 1994.

[17] C. Constantinopoulos, A. Likas, "An incremental training method for the probabilistic RBF network," *IEEE Trans. on Neural Networks,* vol. 17, no. 4, pp. 966- 974, April 2006.

[18] S. Chen, L. Hanzo, S. Tan, "Symmetric Complex-Valued RBF Receiver for Multiple-Antenna-Aided Wireless Systems," *IEEE Trans. on Neural Networks,* vol. 19, no. 9, pp. 1659-1665, Sept 2008.

[19] S. Chen, C.F.N. Cowan, P.M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. on Neural Networks,* vol. 2, no. 2, pp. 302-309, Feb 1991.

[20] G. B. Huang, P. Saratchandran and N. Sundararajan, "An Efficient Sequential Learning Algorithm for Growing and Pruning RBF (GAP-RBF) Networks," *IEEE Trans. on System, Man, and Cybernetics, Part B*: vol. 34, no. 6, pp. 2284-2292, Dec. 2004.

[21] G. B. Huang, P. Saratchandran and N. Sundararajan, "A Generalized Growing and Pruning RBF (GGAP-RBF) Neural Network for Function Approximation," *IEEE Trans. on Neural Networks*, vol. 16, no. 1, pp. 57-67, Jan. 2005.

[22] M. Bortman and M. Aladjem, "A Growing and Pruning Method for Radial Basis Function Networks," IEEE Trans. on Neural Networks, vol. 20, no. 6, pp. 1039-1045, June 2009.

[23] Qiao Junfei, Han Honggui. A repair algorithm for radial basis function neural network and its application to chemical oxygen demand modeling. *International Journal of Neural Systems,* 2010, 20(1):63-74.

[24] Han Honggui, Qiao Junfei. A self-organizing fuzzy neural network based on growing-and-pruning algorithm, *IEEE Transactions on Fuzzy Systems*, 2010, 18(6): 1129-1143.

[25] Han Honggui, Chen Qili, Qiao Junfei. An efficient self-organizing RBF neural network for water quality predicting, *Neural Networks*, 2011, 24(7): 717-725.

[26] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[27] A. Smola and B. Schölkopf, A tutorial on support vector regression. NeuroCOLT2 Tech. Rep. NC2-TR-1998-030, 1998.

[28] Guang-Bin Huang; Lei Chen; Chee-Kheong Siew; , "Universal approximation using incremental constructive feedforward networks with random hidden nodes," Neural Networks, IEEE Transactions on , vol.17, no.4, pp. 879- 892, July 2006.

[29] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," Neurocomputing, vol. 70, no. 16–18, pp. 3056–3062, Oct. 2007.

[30] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," Neurocomputing, vol. 71, no. 16-18, pp. 3460-3468.

[31] Muzhou Hou; Xuli Han, "Constructive Approximation to Multivariate Function by Decay RBF Neural Network," *Neural Networks, IEEE Transactions on* , vol.21, no.9, pp.1517,1523, Sept. 2010

[32] Tiantian Xie, Hao Yu, Joel Hewlett, Paweł Rózycki, and Bogdan Wilamowski, "Fast and Efficient Second-Order Method for Training Radial Basis Function Networks," *IEEE Trans. On Neural Networks and Learning Systems,* vol. 23, no. 4, pp. 609-619, April 2012.

[33] N. Chaiyaratana and A. M. S. Zalzala, "Evolving Hybrid RBF-MLP Networks Using Combined Genetic/Unsupervised/Supervised Learning," *UKACC International Conference on Control '98*, Swansea, UK, Sep. 01-04, 1998, vol. 1, pp. 330-335.

[34] W. Kaminski and P. Strumillo, "Kernel Orthonormalization in Radial Basis Function Neural Networks," *IEEE Trans. on Neural Networks*, vol. 8, no. 5, pp. 1177-1183, Sep. 1997.

[35] R. Neruda and P. Kudová, "Learning Methods for Radial Basis Function *Networks*," *Future Generation Computer Systems*, vol. 21, issue. 7, July 2005, pp. 1131-1142.

[36] Chih-Chung and Chih-Jen Lin, LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1—27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[37] C. Blake and C. Merz, UCI Repository of Machine Learning Databases, Dept. Inform. Comput. Sci., Univ. California, Irvine, 1998.

[38] N. Sundararajan, P. Saratchandran and Y. W. Li, *Radial Basis Function Neural Networks With Sequential Learning: MRAN and Its Applications*. Singapore: World Scientific, 1999.

[39] V. Kadirkamanathan and M. Niranjan, "A Function Estimation Approach to Sequential Learning With Neural Networks," *Neural Computation*, vol. 5, no. 6, pp. 954-975, 1993.

[40] P. Reiner and B. M. Wilamowski, "Nelder-mead enhanced extreme learning machine," in *2013 IEEE 17th International Conference on Intelligent Engineering Systems (INES)*, 2013, pp. 225–230.

[41] J. Platt, "A Resource-Allocating Network for Function Interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213-225, 1991.