Adaptive Training of Radial Basis Function Networks Based on Cooperative Evolution and Evolutionary Programming

Alexander P. Topchy, Oleg A. Lebedko, Victor V. Miagkikh and Nikola K. Kasabov[1]

Research Institute for Multiprocessor Computer Systems,

2 Chekhova Str., GSP-284, Taganrog, 347928, Russia, apt@tsure.ru

[1] Department of Information Science, University of Otago,

Dunedin, P.O. Box 56, New Zealand, nkasabov@otago.ac.nz

Abstract

Neuro-fuzzy systems based on Radial Basis Function Networks (RBFN) and other hybrid artificial intelligence techniques are currently under intensive investigation. This paper presents a RBFN training algorithm based on evolutionary programming and cooperative evolution. The algorithm alternatively applies basis function adaptation and backpropagation training until a satisfactory error is achieved. The basis functions are adjusted through an error goal function obtained through training and testing of the second part of the network. The algorithm is tested on bench-mark data sets. It is applicable to on-line adaptation of RBFN and building adaptive intelligent systems

1. Introduction

Radial Basis Function Networks became very popular due to several important advantages over traditional multilayer perceptrons [1,2,14]:

- Locality of radial basis function and feature extraction in hidden neurons, that allows usage of clustering algorithms and independent tuning of RBFN parameters.
- Sufficiency of one layer of non-linear elements for establishing arbitrary input-output mapping.
- Solution of clustering problem can be performed independently from the weights in output layers.
- RBFN output in scarcely trained areas of input space is not random, but depends on the density of the pairs in training data set [3].

These properties lead to potentially quicker learning in comparison to multilayer perceptrons trained by back propagation. In some extent, RBFNs allow us to actualise a classical idea about training layer by layer. The standard approach to RBFN training includes k-means clustering for calculation of radial functions centres, P-nearest neighbour heuristic for definition of cluster widths, and subsequent training of output layer weights by least squares techniques [4, 14]. The last step is conventionally implemented by means of direct methods like singular value decomposition (SVD) or iterative gradient descent. It has been shown [5] that such a training procedure converges to a local minimum of the evaluation function. Thus, the problem of RBFN learning remains rather complex for large practical applications, and finding global search training algorithms is the subject of interest.

Evolutionary simulation is a promising approach to solving many AI problems. The use of evolutionary algorithms for neural network parametric and structural learning has been shown to be efficient in a number of applications, see e.g. [6]. However, the standard approach, when the instances in the population are the networks, has a number of drawbacks. The worst of them is much larger computational complexity in comparison to iterative search procedures processing a single network. Moreover, functional equivalence of the hidden layer elements leads to redundant genetic description of the network in traditional genetic algorithms for parametric optimisation [7].

The alternative approach presented here uses a population of hidden layer neurons, but not a population of RBF networks. Similar ideas appear in a number of recent papers for various types of architectures and evolutionary paradigms including multilayer perceptrons trained by means of genetic [8] and evolutionary programming [9] techniques, RBFN cooperative competitive genetic training algorithm [10]. All of these consider neurons in a single network as a population.

The presented algorithm itself is also based on the principle of cooperative evolution. The result of such an algorithm will be not the best instance, but the best population as a whole. Under the principle of cooperative evolution, each instance being evolved solves a part of the problem; and we are interested in

obtaining not the best possible instance, but a population solving the whole problem to obtain an optimal overall result. Thus, instances have to adapt in such a way, that their synergy solves the problem. Such an approach is very natural for neural networks, where we have many small sub-components achieving the goal together.

The solution of the RBFN learning problem can be decomposed into a number of sub-problems:

1. The search for optimal location and size of clusters in input features space.
2. Definition of parameters (weights and thresholds) of the output layer by means of gradient procedure or other methods, like singular values decomposition.

The first sub-problem is approached here by using cooperative evolution. The RBFN learning (problem 2) is based on the evolutionary programming paradigm, which employs a cooperative search strategy for optimal network parameters oriented to pattern classification tasks.

Evolutionary Programming (EP) [11] is an evolutionary computational technique. In contrast to Genetic Algorithms (GA), EP is based on the assumption that evolution optimizes the behavior of an instance, but not the underlying genetic code. Thus, EP is focused on the phenotypic level of evolution. Mutations in EP are the single source of the modifications in feasible solutions. Crossover and similar genetic operators are not used. The offspring in EP is created from parental solutions by means of cloning with subsequent mutations. Mutations are implemented as addition of normally distributed random values with zero mean and dynamically adjustable variances to components of solutions. Standard deviation in mutations is inversely proportional to the quality of parental solutions. The selection procedure is also different in EP and can be viewed as a form of stochastic tournament among parents and progeny.

The paper is organised as follows. Section 2 is devoted to the formal statement of RBFN training problem. Fitness function and radial basis functions crowding elimination are described in the sections 3 and 4 respectively followed by a description of the algorithm (section 5), experimental results (section 6) and conclusion in section 7.

2. The Statement of the RBFN Training Problem

The activity F_j of each, jth output neuron of RBFN, depends on the input vector $\boldsymbol{x}$ as follows:

$$F_j(\vec{x}) = \omega_{j0} + \sum_{i=1}^{L} v_{ij}\phi_i(\vec{x}), \quad (1)$$

where ω_{j0} is the value of the threshold on jth output; v_{ij} is the weight between the ith hidden neuron and the jth output; ϕ_i – non-linear transformation, performed by hidden neuron i.

L radial symmetrical basis functions perform non-linear transformation $\phi_i(\boldsymbol{x})=\phi(||\boldsymbol{x} - \boldsymbol{c}_i|| / d_i)$, where $\boldsymbol{c}_i \in \Re^n$ is the centre of the basis function ϕ_i, d_i is deviation or scaling factor for radius $||\boldsymbol{x} - \boldsymbol{c}_i||$, and $\|\cdot\|$ is Euclidean norm in $\Re^n$. The Gaussian function $\phi(r)= exp(-r^2/2)$ is frequently used as non-linear transformation ϕ.

RBFN training can be considered as an optimisation problem, where error function E is an evaluation function being minimised. E is usually defined as the average squared deviation of network outputs from desired output values on given training data set:

$$E = \frac{1}{2KM}\sum_{i=1}^{K}\sum_{j=1}^{M}\left(t_j^i - o_j^i\right)^2, \quad (2)$$

where: K is the number of input-output pairs, t_j^i is the target value for the output neuron j in reaction to ith input pattern; $o_j^i = F_j(\boldsymbol{x}_i)$ is the actual value generated by output neuron j after feeding ith input pattern; M *is* dimensionality of output space. For convenience, the target values were limited to {0,1} in our experiments.

3. Fitness Estimation

The main purpose of hidden elements in a network with conventional architecture solving a classification problem is to provide separating hyper-surfaces separating the patterns (the points in input space) of different classes in such a way that the patterns belonging to the same class appeared in the same side only. In networks with radial basis units such a surface can be thought of as a union of hyper-spheres or ellipses. The patterns of the class must be grouped by basis units in such a way that they do no overlap with patterns of other classes. One of the mentioned properties of radial basis Gaussian functions is locality of activity, which means that the influence of the function ϕ_i on some distance from the centre $\boldsymbol{c}_i$ can be neglected. It does not hold for common multilayer perceptron networks, where the effect of all hidden

neurons should be taken into account in each point of the input space. Locality in RBFN creates an opportunity to estimate the efficiency of each element separately from others. As a function for estimation of the quality of the element ϕ_j, the following function can be used:

$$e_j = \frac{\sum_{\vec{x}_k \in r} \phi_j(\vec{x}_k)}{\sum_{l=1}^{K} \phi_j(\vec{x}_l)}, \qquad (3)$$

where e_j is the value of jth element efficiency (quality); $\phi_j(\boldsymbol{x})$ is the value on the output of the jth element in response to presentation of the input pattern $\mathbf{x}$; $\boldsymbol{x}_k$ is the pattern belonging to the class r, which has maximal sum of activities for jth neuron, and $\boldsymbol{x}_l$ are the patterns of all classes. In other words, in the course of pattern presentation the partial sums of activities of a given unit for each class is calculated: S_k = sum of $\phi(\boldsymbol{x})$ over all $\boldsymbol{x}$ belonging to the class k, $k=1,..,C$, where C is the number of classes. Only the class r with maximal S_r contributes to the nominator of (3). During fitness calculation its possible to find the values, which each neuron gets on each of the classes. Than we find the maximal among them. In other words, this function defines how much element ϕ_j distinguishes class r from the other classes. The goal of the learning procedure is to maximize the values of e_j for all hidden neurons. However, the location of basis units in input space should be different in order to achieve optimal ' niching', i.e. to have no units performing the same function. ' Niching' of neurons should distribute them over the patterns, which is not enforced in expression (3). This problem can be solved by taking into account boundaries between the classes as discussed in detail below.

Generally, e_j is used for guiding the search through the space of RBF centres and widths, and estimation of the amount of effort to be spent for improvement of current clustering. This mechanism of credit assignment provides the appropriate direction for search by means of evolutionary programming.

Another advantage of the fitness function is that it does not require the values of output layer weights to be calculated. Thus, the search for the best values for the centres can be performed before training of the parameters of the output layer, that significantly reduces the complexity. The output layer training is only required for determination of the total error of the network (2), which is used in the termination condition of the algorithm.

4. RBF Crowding Avoidance

In the cooperative approach to NN learning the problem of "division of the work" among the neurons should be solved. There should not exist elements, which performs identical functions in pattern classification. If such competing neurons exist, some of them should be changed in a way that they perform more useful functions, which comprises the process of 'niching'.

It is obvious that function (3) does not satisfy this requirement. There exist local maximums, which many elements tend to occupy. These maximums have basins of attraction of different sizes, which invoke crowding of several Gaussians in the same area. However, some other areas could remain uncovered. The simplest way of solving this problem is to calculate the distance $\| \boldsymbol{c}_i - \boldsymbol{c}_j \|$ between the centres of the Gaussians and compare it with threshold distance. If the distance between the Gaussians ϕ_i and ϕ_j is less than this threshold then RBFs are considered to be competitive. However, this way is not invariant in respect to the distance between the patterns of different classes.

More adequate measure of the overlapping between two elements ϕ_i and ϕ_j can be expressed by the following function:

$$R_{ij} = \frac{\sum_{k=1}^{K} \phi_i(\vec{x}_k)\phi_j(\vec{x}_k)}{\sqrt{\sum_{k=1}^{K} \phi_i^{\,2}(\vec{x}_k) \sum_{l=1}^{K} \phi_j^{\,2}(\vec{x}_l)}} \qquad (4)$$

which measures orthogonality of normalised neuron activities. It approaches zero for totally non-overlapping neurons and equals to 1 for neurons performing identical functions. However, such a crowding function is computationally expensive to be applied on large number of training patterns. A trade-off heuristic for determining overlapping units is used here instead. It compares only the patterns for which a neuron's output is maximal (and next to maximal in our implementation). Thus it requires only one (or two in our case) additional comparisons for each pattern. In the general case in (4), only n ($n<<K$) points can be taken into account, for which the neuron's output has maximal value. If the obtained value is greater than 0, then the two elements are considered to be overlapping. This is an efficient and inexpensive way to find approximate values for R_{ij} .

In the presented below the elements ϕ_i and ϕ_j are considered to be competing, if they are the most close to the same pattern $\boldsymbol{x}_k$ (or n patterns as mentioned above). If competing elements are found, the one having maximal value of the fitness e has to be kept unchanged and the rest of them can be modified. Since

we can easily find the patterns, having the maximum impact into the error during output layer training, this information can be used for placement of the centres of the elements to be changed in the points, corresponding to such patterns.

5. The Description of the Algorithm

The pseudo-code of the algorithm can be outlined as follows:

I. FOR each number of basis unit (centres) from a given set DO the following steps:

1. Generation of initial values for centres and deviations of all elements ϕ_j.
2. Calculate the efficiency e_j of each basis unit.
3. Train output layer by I iterations of gradient procedure.
4. Find total error E of the network.
5. If E is less than desired threshold, then go to II.
6. Find elements performing almost identical functions. If there are no such units go to II.
7. Reallocate crowded Gaussians to the areas the worst classified patterns.
8. Generate a new offspring of basis function neurons.
9. Calculate the fitness of the new offspring of neurons e_j.
10. Choose the better population between the offspring and the parent.
11. Go to step 3.

II. Select the optimum RBFN's structure having number of centres with a minimum total error E.

The training of the output layer is by a small number of gradient descent iterations (I steps of delta-rule, the value of I being incremented in the example below every 20 generations starting from 5.

Mutation of the parent neuron and creation of the offspring is performed in step 8. Definition of the centres and deviations of offspring Gaussians are calculated in accordance with the following expressions:

$$c_{ij} = c_{ij} + N\left(0,\ \alpha\frac{E}{e_i}\right), d_i = d_i + N\left(0,\ \beta\frac{E}{e_i}\right) \quad (4)$$

where c_{ij} is the jth component of the ith neuron; $N(a,b)$ is normal distribution with mean a and variance b; α, β are the scaling factors; E is the error of the network. In experiments these parameters were set to α=0,16 and β=0,05.

Since, in contrast to the corresponding learning algorithms with logistic functions [9], it is possible to perform evaluation and selection of the Gaussians of the same parent independently from the others, several offspring can be processed during one generation. This is attributed to the local character of the basis functions and on the premises function (3) is derived. The selection procedure performed in step 10 can be performed either probabilistically or deterministically. If several offspring are created, different tournament-like strategies can be used. However, it is possible to perform deterministic selection too.

6. Experimental results

We are presenting the results for two well-known classification problems. The iris data classification problem, used by Fisher in 1936, remains the standard benchmark for testing pattern classification methods. The data set contains 150 patterns of three classes (50 for each class). Four continuous features correspond to sepal width and length, petal width and length. Three classes of plant are versicolour, setosa and virginica. One of them is linearly separable from two others.

RBFNs with 5, 10, 15, 20 and 25 radial Gaussian elements, four inputs and three outputs were used for classification. Results were averaged over 10 trials. Obtained measurements were compared with the traditional k-means clustering with SVD training and with a cooperative, competitive genetic training as in [10]. Fig.1 shows the number of miss-classified patterns in the training data set after approximately equal amount of CPU time corresponding to 100 generations of our algorithm. In each generation we evaluate two offspring networks, therefore the average number of objective function evaluations is two times bigger than the number of generations. It is clear that for larger network the relative performance of the algorithm increases. Total classification has been achieved for all runs with 25 elements in contrast to other methods.

Locations and relative widths of basis units for RBFN with 5 elements are shown in a 2D projection in the feature 3 and feature 4 plane , at the end of training in Fig. 2.

Figure 3 shows the dependence of the total network error on the number of iteration for different number of radial basis functions L. One iteration corresponds on average to 2 presentations of the training data set. MSE decreases quickly and reaches required for this

problem level 0.1 after 10-40 iterations. In general, the time for getting the error threshold MSE = 0.1 was comparable with the time of k-means algorithm.

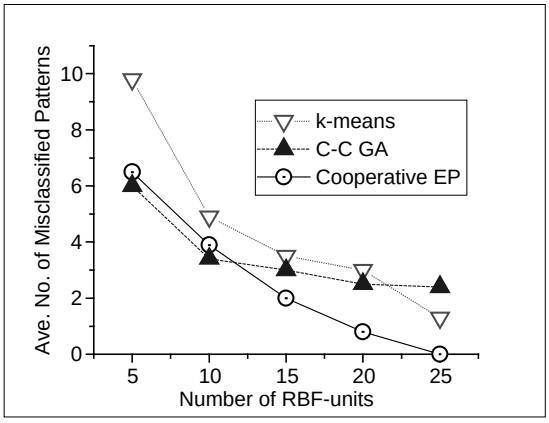

Figure 1: Iris problem: Comparing different algorithms after 100 generations of cooperative EP

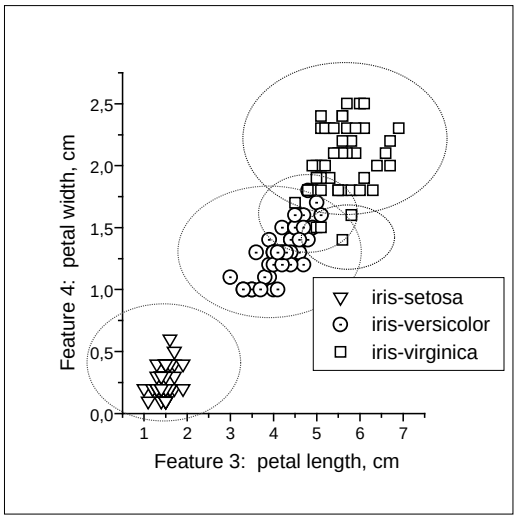

Figure2: Distribution of basis elements being evolved.

After reaching certain value of the error, the learning rate significantly decreases. This is attributed to the property of EP as a global search algorithm, to find quickly the value in the vicinity of optimum, but then spend a rather long time for final tuning. Several iterations of the gradient search procedure can be used to speed up the convergence to final values.

Other experiments were performed on the Glass Test problem. This problem has nine features for each of 214 patterns, divided in training and testing set, to be separated into 6 classes. The results of training after 200 generations were equal to results obtained on MLP with architecture 9-16-8-6 (see e.g. [12]) with 572 weights and sigmoidal non-linearities.

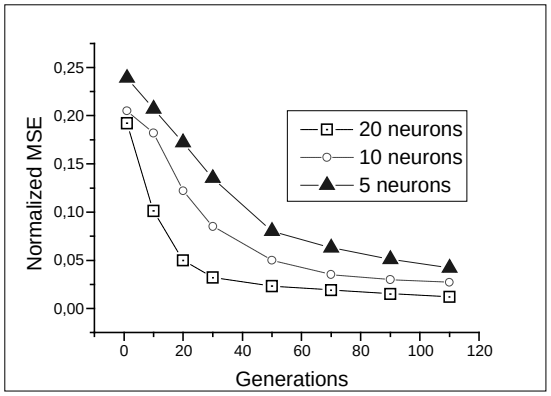

Figure 3: Iris problem. Convergence plot for various number of basis units

The authors have chosen an RBFN with 36 Gaussian elements having almost the same number of adjustable parameters. The values of MSE on training and testing data sets show approximately equal generalisation properties for MLP and RBFN for the given test problem. However, the RBFN error for the training data set is less after the same training time than the one for the MLP from [12].

7. Conclusion

The paper presents a novel algorithm for evolutionary training of RBFN. Evolution of a single network as a population of neurons, but not a collection of networks, seems to be an approach allowing avoidance of large computational complexity in traditional evolutionary algorithms for NN training. Moreover, many difficulties associated with standard procedures of genetic encoding can be successfully solved. However, this approach requires more precise analysis of the network decomposition and introduction of relative fitness functions for the elements. In this paper, RBFN adaptation is performed by means of evolutionary programming. The described algorithm for classification problems is competitive with the traditional RBFN training techniques, and shows better results for problems with large dimensionality.

A strong advantage of the new algorithm is its ability to gradually change and adapt basis functions within the learning procedure which includes alternative refinement of the basis functions and a gradient descent training. This advantage makes it applicable to on-line adaptive systems.

The presented learning algorithm can be easily extended and applied for the solution of a large class

of approximation problems by changing the fitness function. In this way, the described evolutionary learning approach can be used in almost every application of RBFN from control to image processing. In particular, this algorithm was used in neuro-fuzzy system for production sales analysis. The algorithm can be applied to other neuro-fuzzy architectures such as the fuzzy neural network FuNN [13,14,15]. A significant difference between the RBFN and the FuNN architectures is that the former is based on basis units which define cluster centres in the whole input space, while the latter one uses fuzzy membership functions for quontisation of the space of each individual input variable. FuNN has the advantage of adjusting the membership functions and the fuzzy rules embedded in its structure during the operation of the system (on-line) [15,16]. In addition to expanding the number of the basis units and finding their optimum number for a particular training data set, an on-line adaptive RBFNs and FuNNs structures are being developed at present which allow for 'shrinking', so the number of the basis units (membership functions in the FuNN case) can be reduced if necessary according to new data coming on-line.

Acknowledgements

This research is supported by Research Institute for Multiprocessor Computer Systems, Taganrog, Russia, and partially supported by a research grant PGSF UOO-606 from the New Zealand Foundation for Research Science and Technology.

References

[1] M. J. D. Powell, The Theory of Radial Basis Functions Approximation, in *Advances of Numerical Analysis* , pp. 105–210 , Oxford: Clarendon Press, 1992.

[2] F. Girosi, Some Extensions of Radial Basis Functions and their Applications in Artificial Intelligence, Computers Math. Applic., vol. 24, no. 12, pp. 61-80, 1992.

[3] J.A. Leonard, M.A. Kramer and L.H. Ungar, Using Radial Basis Functions to Approximate a Function and Its Error Bounds, *IEEE Trans. on Neural Netwirks*, vol.3, no. 4, pp.624-627, 1992.

[4] J. Moody and C. J. Darken, Fast Learning in Networks of Locally Tuned Processing Units, *Neural Computation*, vol. 1, pp. 281–294, 1989.

[5] Y. Linde, A. Buzo and R. Gray, An Algorithm for Vector Quantizer Design, *Proc. Of IEEE* , Com-28 (1), pp. 84-95, 1980.

[6] Schaffer, D. Whitley and L.J. Eshelman, Combinations of genetic algorithms and neural networks: A survey of the state of the art, in *Combinations of Genetic Algorithms and Neural Networks*, pp. 1-37, IEEE Computer Society Press, 1992.

[7] J. Angeline, G.M. Saunders, and J.B. Pollak, An evolutionary algorithm that constructs recurrent neural networks, IEEE *Transactions on Neural Networks*, vol.5, no. 1, pp.54-65, 1994.

[8] D.Prados, A fast supervised learning algorithm for large multilayered neural networks, in *Proceedings of 1993 IEEE International Conference on Neural Networks* , San Francisco, v.2, pp.778-782, 1993

[9] A.Topchy, O.Lebedko, V. Miagkikh, Fast Learning in Multilayered Neural Networks by Means of Hybrid Evolutionary and Gradient Algorithm, in *Proc. of the First Int. Conf. on Evolutionary Computations and Its Applications* , ed. E. D. Goodman et al., (RAN, Moscow), pp.390–399, 1996.

[10] B. A. Whitehead and T.D. Choate, Cooperative - Competitive Genetic Evolution of Radial Basis Function Centres and Widths for Time Series Prediction", *IEEE Transactions on Neural Networks*, vol. 7, no. 8, pp.869-880, 1996.

[11] Fogel L.J., Owens A.J. and Walsh M.J. "Artificial Intelligence through Simulated Evolution", John Wiley & Sons, 1966.

[12] L. Prechelt, Proben1-A set of neural network benchmark problems and rules, University Karlsruhe, Technical Report 21/94, 1994

[13 N. Kasabov, Kozma, R., Watts, M. Optimisation and adaptation of fuzzy neural networks through genetic algorithms and learning- with- forgetting methods and applications for phoneme-based speech recognition. *Information Sciences* (1997) accepted

[14] N. Kasabov , *Foundations of Neural networks, Fuzzy Systems and Knowledge Engineering*, MIT Press, 1996

[15] N. Kasabov, Kim, JS, Watts, M. and Gray, A. FuNN/2 - A fuzzy neural network architecture for adaptive learning and knowledge acquisition. *Information Sciences:Applications*,1997, in print