Initialization of Radial Basis Function Neural Networks with Prior Domain Information for Good Generalization

Dr. Ganka Kovacheva

Abstract: *The effect of initialization of Radial Basis Function (RBF) Neural Network (NN) with prior domain information is determining for generalization ability of the network. It defines the number of hidden units in a hidden layer in advance and minimizes the time of learning. The paper describes how to create RBF NN simulator including prior domain information and how the initialization works on the final result. A task for technical diagnostics of objects with fuzzy domain is examined by moving casual centers of clusters and moving domain centers and widths.*

Key words: *Radial Basis Function (RBF) Neural Networks, Initialization of RBF Neural Networks, Application in Technical Diagnostics.*

INTRODUCTION

Inductive learning methods such as artificial neural networks generally rely on large amounts of training data and require many training cycles to form concepts. One of encountered problems in these methods is an insufficient training example to achieve good generalization. The use of approximately correct domain parameters to initialize the learning algorithm prior to training has been shown to be effective in improving generalization in many cases [1], [5], [6]. Our main goal is to reduce the need for training data and to bias the learner towards a solution that fits with a domain expert's knowledge. The effects on training time, network size, and generalization accuracy of RBF network initialized with prior domain knowledge in comparison to networks that start learning from tabula rasa configuration are significant. In RBF NN there is two phases of training – a hidden unsupervised phase and an output supervised phase. Our goal is to put hidden units near to expert's domain and to perform supervised learning for hidden and output layers together.

This paper is organized as follows: The next sections describe the RBF network architecture and two different strategies for initialization of hidden units. Then is given information about practical realization of program simulator and its implementation in technical diagnostics problems.

RADIAL BASIS FUNCTION NEURAL NETWORK

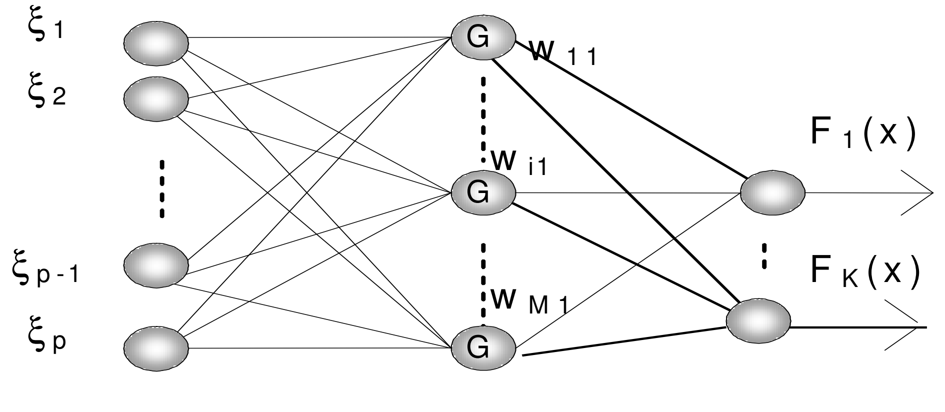

Figure 1 – Radial basis function neural network

RBF neural networks are three (input, hidden, output) layered networks, which contain a set of locally responsive units in a hidden layer (Figure 1). Each hidden unit is a node with an RBF activation function. Every unit has two important parameters – the center and the width [5] and shall be taken in this paper as a Gaussian function. Nodes in output layer are linear and yield a weighted sum of its inputs. The number of input nodes is equal to the dimension p of the input vector $x = (\xi_1, \xi_2, ..., \xi_p)^T$. The number of output nodes depends on the number k of classes we want to recognize.

The training data are given as a set of pairs $\{(x_i,d_i):1 \le i \le N\}$, where x_i is the measurement input vector in R^p and d_i is the corresponding target vector in R^K. If x_i belongs to the class k, then d_i is the vector consisting of 0 elements except the k-th element equal to 1. Using these training data in a procedure for determining the prior domain we can determine the number M of hidden units (M≤N) and initial values of their centers $\{t_i:1\le i \le M\}$ and widths $\{\sigma_i:1\le i\le M\}$.

Let $G(\xi:\sigma)$ be the Gaussian function with mean zero and the standard deviation σ. The j-th output of the Gaussian RBF network is expressed by

$$F_j(x) = \sum_{i=1}^{M} w_{ij} G(\|x - t_i\| : \sigma_i), \qquad (1)$$

where $x \in R^p$ is an input vector, $\|x\|$ is the Euclidian norm of the vector x and w_{ij} is the weight parameter, which connect i-th hidden node with j-th output node.

STRATEGIES FOR INITIALIZATION OF HIDDEN UNITS

The hidden layer's activation functions evolve slowly in accordance with some nonlinear optimization strategy, while the output layer's weights adjust themselves rapidly through a linear optimization strategy. It is necessary to involve in the optimization of the hidden layer of the network some improvement by using different techniques [2]:

Fixed centers randomly selected

The location of the centers may be chosen randomly from the training data, which are distributed in a representative manner for the problem. Specifically, a normalized RBF centered at t_i is defined as

$$G(\|x - t_i\|^2) = \exp\left(-\frac{M}{d^2}\|x - t_i\|^2\right)_{i=1,2,...,M}, \qquad (2)$$

where M is the number of centers and d is the maximum distance between the chosen centers. The standard deviation (i.e. width) of all the Gaussian RBF is fixed at

$$\sigma = \frac{d}{\sqrt{2M}}. \qquad (3)$$

Such a choice for the standard deviation σ avoids some extremes like too peaked or too flat gaussian function.

Supervised method for center and width selection

The centers of the radial-basis functions and all other free parameters of the network undergo a supervised learning process. For a gradient-descent procedure that represents a generalization of the LMS algorithm it is necessary to define the instantaneous value of the cost function

$$\varepsilon = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{K} e_{ij}^2 \qquad (4)$$

where N is the number of training examples and k is the number of classes. The error signal for i-th training datum at the j-th output node is defined by

$$e_{ij} = d_{ij} - F_j(x_i) \qquad 1 \le i \le N, 1 \le j \le K, \tag{5}$$

where d_{ij} is the j-th element of the target vector d_i.

It is necessary to find the parameters w_{ij}, t_i and σ_i, so as to minimize ε. Let w_i and e_j are the corresponding vectors of weights and errors. The results are the following adaptive formulas [2]:

Weights between hidden and output layers;

$$\frac{\partial\varepsilon(n)}{\partial w_i(n)} = \sum_{j=1}^{N} e_j(n) G(\|x_j - t_i(n)\| : \sigma_i(n)) \tag{6}$$

$$w_i(n+1) = w_i(n) - \eta_1 \frac{\partial\varepsilon(n)}{\partial w_i(n)} \qquad i=1, 2,..., M \tag{7}$$

Positions of centers (hidden layer);

$$\frac{\partial\varepsilon(n)}{\partial t_i(n)} = 2w_i(n) \sum_{j=1}^{N} e_j(n) G'(\|x_j - t_i(n)\| : \sigma_i(n)) \sigma_i^{-2}(n)[x_j - t_i(n)] \tag{8}$$

$$t_i(n+1) = t_i(n) - \eta_2 \frac{\partial\varepsilon(n)}{\partial t_i(n)} \qquad i=1, 2,... M \tag{9}$$

Widths of the functions (hidden layer);

$$\frac{\partial\varepsilon(n)}{\partial\sigma_i(n)} = -w_i(n) \sum_{j=1}^{N} e_j(n) G'(\|x_j - t_i(n)\| : \sigma_i(n)) Q_{ji}(n) \qquad i=1,2,... M \tag{10}$$

$$Q_{ji}(n) = [x_j - t_i(n)][x_j - t_i(n)]^T \tag{11}$$

$$\sigma_i(n+1) = \sigma_i(n) - \eta_3 \frac{\partial\varepsilon(n)}{\partial\sigma_i(n)} \tag{12}$$

The cost function ε is convex to linear parameters w_i, but nonconvex with respect to the centers t_i and widths σ_i and the search for the optimum values of t_i and σ_i may get stuck at a local minimum in parameter space. For the initialization of the gradient-descent procedure, it is often desirable to begin the search in parameter space from a structured initial condition. It limits the region of parameter space to be searched to an already known useful area. Implementing an RBF network with prior domain selection helps to avoid falling in a local minimum.

PRACTICAL REALIZATION OF THE SIMULATOR

A program simulator [3], [4] is realized in Borland C++ and consists of four program modules. The first module prepares input data for simulation. They are number N of training data, their dimension p, number of classes, all input vectors, all target vectors and coordinate of cluster centers and widths. All data are normalized within the interval [0.1 , 0.9]. They are stored in a file.

The second module consists of three sub modules. The first sub module creates network architecture with arbitrary number of units in each layer. In dialog we determine the appropriate architecture depending on different strategies for initialization of hidden units. The initial data for the positions of the centers and the widths of the units in hidden layer are loaded from a file. The weight coefficients linking the hidden and the output layers are generated randomly.

The second sub module trains neural network after reading the normalized data from the file. It optimizes the parameters and minimizes the sum of the mean square errors for all the outputs.

The training continues until the criterion for damping of fluctuations of standard error is satisfied or all epochs defined in advance are spent. It is possible to change the learning rate in the process of learning. We can save all the parameters of the trained network in a file in order to use them later with new data or to make an additional training.

The last sub module evaluates the standard error of the trained network by using an entire training data.

The third module graphically shows performance of the final results for better perceiving and understanding.

The fourth module is designed for preparation of new data for classification. Their target vectors are unknown and the network responds are available after the test procedure. Such a sample is being classified into one of the existing state classes.

REALIZATION OF DIFFERENT TYPE OF INITIALIZATION

The design of an RBF neural network consists of determining the number of RBF functions in hidden layer, determining their initial centers and widths and finding the weights that connect them to the output nodes. It is trivial to place RBF on every given training pattern and then make the kernel function peak when that pattern or similar patterns are presented but this becomes practically impossible, when the number of the data to be trained is large. Sometimes when the data isn't precise and even have a noise, it can cause over-fitting. Even when the number of centers is less and they are distributed in a representative manner for the problem, the learning is slowly and generalization error cannot fall under definite value.

Clustering algorithm for input data is needed, like k-means algorithm or another but when a sample of one specific class is in the envelope of another class some faulty and erroneous results could be received. The clustering algorithm, which determines the number of hidden neurons, must take into account the attachment of the sample to a definite class as this would result in the accuracy and the time for training.

The clustering algorithm below is appropriate for determining the number of the hidden units, their initial centers and widths [6]. This algorithm creates a group of samples belonging to the same class, which is enough discriminate from opposite classes. The realization is made in C++ and includes the following steps:

1. Every training point is a different cluster;
2. Randomly choose the label k=1,2,3…c for every cluster;
3. Start from k=1;
4. Search for cluster from the same class;
5. Merge these two clusters and evaluate of the new centre;
6. Evaluate the distance d between the new centre and the centre of the nearest cluster from an opposite class;
7. Evaluate the distance between the new centre and the most outlying point of this cluster, which is a radius R of the cluster;
8. If $d>\alpha R$, where α=const than accept the merge from point 5 and continue from point 4, linking current k with the new created cluster and decreasing c=c-1. If this inequality is not true reject the merging and restore the two initial clusters, leave k and c without changing and continue from point 4. Repeat steps from 4 to 8 for all the clusters and than increase k=k+1;
9. Repeat steps 4 to 8 until k=c.

Practically the range of α is [1, 3] because big values cause good accuracy but limitary reduction of clusters.

This information is used for initialization of weights between input and hidden layer. Building of this links is a dynamic structure, which uses the number of neuron-source, the number of neuron-receiver and weighted link between them. The dynamic structure of neuron connection is:

```
neuron[i]- > struct a
{
float state;
float potential;
float error;
float radii;
struct connect*next;
};
```

```
struct connect
{
int nnumb;
float weight;
struct connect*post;
};
```

Every neuron[i] is connected to another neuron with number nnumb with weighted connection weight. The initial information for state, potential and error can be 0 when clustering of input data is not used, but radii is equal to some positive initial value when there is not cluster information or equal to the value of width, received after clustering procedure. Thus the region is localized easier and covered by Gaussian with calculated in advance parameters.

PRACTICAL RESULTS

The task is to classify different states of compressors. There are 7 different classes of state, which are distributed between good working order and different kinds of faulty operating. The measurement vector consists of monitor indicator values of the compressor. It serves for indirect evaluation of the state and has 24 symptoms. Some of them are the pressure and the temperature of the in-coming and out-coming gas and vibrations at many points of observation.

The network size is 24-26-7, where 26 cluster centers are randomly selected from the training data set. They are distributed in a representative manner depending on the quantity of available data for every class. The width for every cluster is a constant. Network training involves adjusting the output weight, centre and width of each hidden cluster by gradient descent on the error surface – formulas (6)-(12).

Graphical results for the dynamic of the learning process in a case with moving casual centers of clusters and moving domain centers and widths is shown at Figure 2:

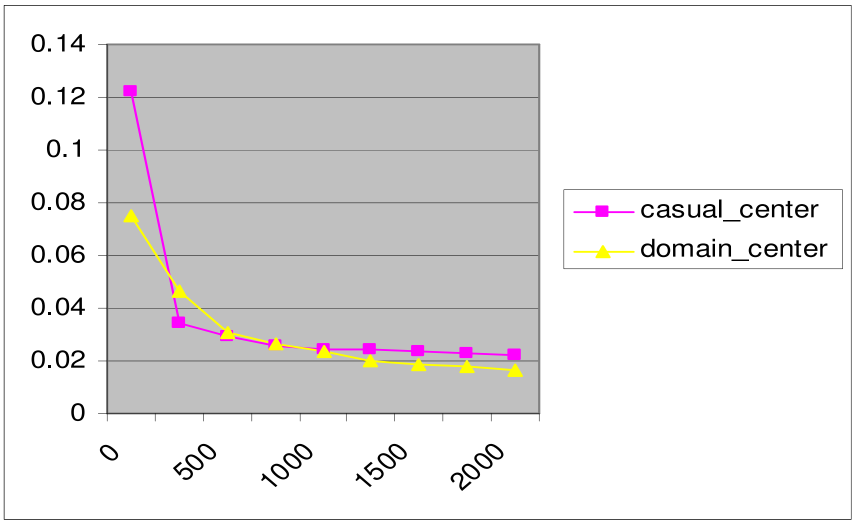

Figure2: Random initialization of centers and widths by comparison with determinated by clustering method centers of domain and their widths

CONCLUSIONS AND FUTURE WORK

The practical application of the developed software simulator in the technical diagnostics has led to the following conclusions: Adapting the position of centers of radial-basis functions is beneficial when a minimal network configuration is required. The same performance on generalization can be achieved by using a larger RBF network – network with a larger number of fixed centers in the hidden layer, and only adapting the output layer of the network by linear optimization.

The use of prior domain information improves learning speed, accuracy and results in smaller networks with sufficient training examples for concept learning. The randomly initialized network have produced an initial quick reduction in error but then not been able to converge to an optimal solution. Also, for randomly initialized networks, accuracy cannot improve as the number of local kernel functions used to initialize the network increased.

REFERENCES

1. Andrews R. and Geva S.: "On the effects of initializing a neural network with prior knowledge", ICONIP'99, Perth, Australia, vol.1, pp.251-256, 1999
2. Haykin S.: "Neural networks", Mc Master University, 1994
3. Kovacheva G. and Ogawa H.: "Radial basis function classifier for fault diagnostics", ISICT'03, Dublin, Ireland, 2003
4. Kovacheva G. and Ogawa H.: "Incremental learning method for RBF classifiers", WISICT'04, Cancun, Mexico, 2004
5. Moody J. and Darken C.: "Fast learning in network of locally- tuned processing units", Neural Computation, 1, 1989
6. Musavi M., Ahmed W. and Chan K.: "On the training of Radial Basis Function classifiers", Neural Networks, 5, pp. 595-603, 1992

ABOUT THE AUTHOR

Major Assistant Professor Ganka Petkova Kovacheva, PhD, Department of Computer Systems and Technology, Technical University of Varna, Phone: +359 52 622978, E-mail: gpp_k@mail.bg