

GROWING RBF NETWORK MODELS FOR SOLVING NONLINEAR APPROXIMATION AND CLASSIFICATION PROBLEMS

Gancho Vachkov
School of Engineering and
Physics
The University of the South Pacific
(USP)
Suva, Fiji Islands
gancho.vachkov@gmail.com

Valentin Stoyanov
Department of Automatics and
Mechatronics
University of Ruse "Angel
Kanchev"
Ruse, Bulgaria
vstojanov@uni-ruse.bg

Nikolinka Christova
Department of Automation of
Industry
University of Chemical Technology and
Metallurgy (UCTM)
Sofia, Bulgaria
nchrist@uctm.edu

KEYWORDS

Radial Basis Function Networks, Growing RBFN models, Particle Swarm Optimization, Nonlinear Approximation, Classification.

ABSTRACT

In this paper a multi-step learning algorithm for creating a Growing Radial Basis Function Network (RBFN) Model is presented and analyzed. The main concept of the algorithm is to gradually increase by one the number of the Radial Basis Function (RBF) units at each learning step thus gradually improving the total model quality. The algorithm stops automatically, when a predetermined (desired) approximation error is achieved. An essential part in the whole algorithm plays the optimization procedure that is run at each step for increasing the number of the RBFs, but only for the parameters of the newly added RBF unit. The parameters of all previously RBFs are kept at their optimized values at the previous learning steps. Such optimization strategy, while suboptimal in nature, leads to significant reduction in the number of the parameters that have to be optimized at each learning step. A modified constraint version of the particle swarm optimization (PSO) algorithm with inertia weight is developed and used in the paper. It allows obtaining *conditional optimum* solutions within the range of preliminary given boundaries, which have real practical meaning. A synthetic nonlinear test example is used in the paper to estimate the performance of the proposed growing algorithm for nonlinear approximation. Another application of the Growing RBFN model is illustrated on two examples for data classification, one of them from the publicly available red wine quality data set.

INTRODUCTION

Radial Basis Function (RBF) Networks have been widely used in the last decade as a power tool in modeling and simulation, because they are proven to be *universal approximators* of nonlinear input-output relationships with any complexity (Poggio and Girosi 1990; Park and Sanderberg 1993; Musavi et al. 1992). In fact, the RBF Network (RBFN) is a composite multi-input, single output model, consisting of a

predetermined number of N RBFs, each of them playing the role of a local model (Musavi et al 1992). Then the aggregation of all the local models as a weighted sum of their output produces the overall nonlinear output of the RBFN.

There are some important differences between the RBF networks and the classical feed forward networks, such as the well known multilayer perceptron (MLP), often called back-propagation neural network (Poggio and Girosi 1990; Park and Sanderberg 1993). The most important difference is that the RBFN are *heterogeneous* in parameters, in a sense that they have three different groups of parameters, which need to be tuned by using different learning (optimization) algorithms. This makes the total learning process of the RBFN more complex, because it is usually done in a sequence of several learning phases. This obviously may affect the accuracy of the produced model.

In this paper we briefly analyse the internal structure and the properties of the classical RBFN model with a pre-determined number of RBF units and the respective learning algorithm, based on the Particle Swarm Optimization (PSO) with constraints (Eberhart and Kennedy 1995). Then we present and analyse a new learning algorithm for creating an *incremental structure* of the RBF network, called *Growing RBFN* model.

The algorithm starts with a RBFN model that has only one RBF unit and at each step of the algorithm a new RBF unit is inserted to the RBFN model. This process obviously leads to gradual increase of the model complexity, but at the same time the model accuracy is gradually increased. The important point here is that the optimization procedure is only performed for the small group of parameters of the newly inserted single RBF unit. This leads to a reduction in the number of the simultaneously optimized parameters. Such repetitive (step-wise) optimization scheme is able to eventually produce a model with the desired approximation error at a lower computation cost.

The merits of the proposed learning algorithm for creating a growing RBFN model are discussed and analysed in the paper by solving nonlinear approximation problem on a test nonlinear example. Another novel part in the paper is the proposed classification criterion that is calculated by a specially introduced quantization procedure. It converts the real valued outputs from the

growing RBFN model into respective integer predicted classes. This allows the RBFN model to be used as a classifier for solving nonlinear classification problems. Two such classification problems are solved and discussed in the paper by using the proposed algorithm for creating the Growing RBFN models.

BASICS OF THE CLASSICAL RBF NETWORK MODEL

Our objective is to create a model of a real process (system) with K inputs and one output based on a preliminary available collection of M experiments (*input-output* pairs) in the form:

$$\{(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_i, y_i), \dots, (\mathbf{X}_M, y_M)\} \quad (1)$$

Here $\mathbf{X}=[x_1, x_2, \dots, x_K]$ is the vector consisting of all K inputs of the system (process) under investigation. The produced experimental output from the process is denoted by y . Then the predicted output, calculated by the RBF network model can be denoted, as follows:

$$y_m = f(\mathbf{X}, \mathbf{P}) \quad (2)$$

Here $\mathbf{P}=[p_1, p_2, \dots, p_L]$ represents the vector of all L parameters participating in the RBFN model. They will be explained later on.

The classical RBFN has a fixed structure of *three layers*, namely the *input* layer, *hidden* layer and *output* layer as shown in Fig. 1.

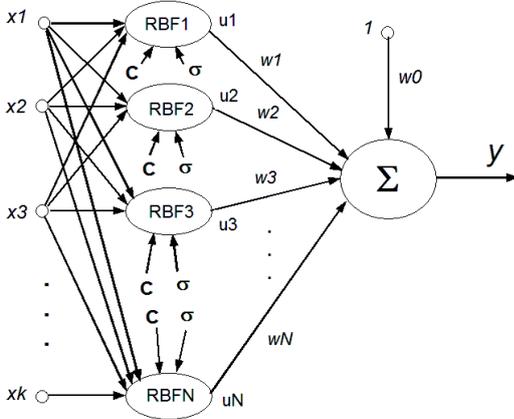


Figure 1: Structure of the Classical Radial Basis Function Network with K inputs and N RBFs

The output calculated by the RBFN model with a given number of N Radial Basis Functions is in the form of a *weighted sum* of the outputs from all N RBFs:

$$y_m = w_0 + \sum_{i=1}^N w_i u_i \quad (3)$$

Here $u_i, i=1, 2, \dots, N$ are the outputs of the RBFs with K inputs x_1, x_2, \dots, x_K and $w_i, i=0, 1, 2, \dots, N$ are the *weights* associated with the RBFs, including one additional *offset* weight w_0 as seen in the figure.

The output u of each RBF is calculated by the following *Gaussian* function in the K -dimensional input space:

$$u = \exp\left(-\sum_{j=1}^K (x_j - c_j)^2 / (2\sigma^2)\right) \in [0, 1] \quad (4)$$

It is easy to notice that each RBF is determined by $K+1$ parameters, namely the K -dimensional vector $\mathbf{C} = [c_1, c_2, \dots, c_K]$ representing the *center* (location) of the RBF in the input space and the *scalar width* (*spread*) σ . An example of a RBF with a scalar width $\sigma=0.15$ and a center $\mathbf{C} = [0.4, 0.6]$ in the two-dimensional space ($K=2$) is shown in Fig. 2.

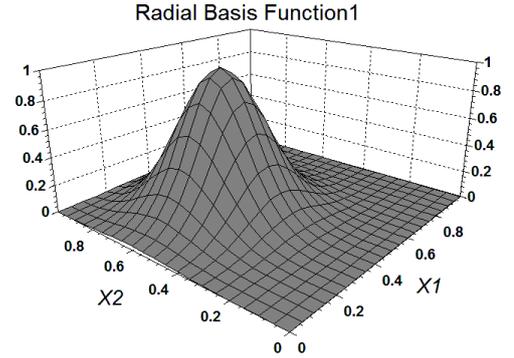


Figure 2: Example of a RBF with $K=2$ Inputs, Center Located at $[0.4, 0.6]$ and a Single Width $\sigma = 0.15$.

Using the above notations, it is easy to calculate the total number L of all parameters that define the Classical RBF Network from Fig. 1 having K inputs and N RBFs. This number is given as:

$$L = N \times K + N + (N + 1) = N \times K + 2 \times N + 1 \quad (5)$$

All these parameters constitute the parameter vector \mathbf{P} from (2). It is worth to note that the parameters are *heterogeneous* in nature, unlike the parameters of the classical backpropagation neural network (the *multilayer perceptron*), where all the parameters are homogeneous and represent one category – the connection weights between the units (neurons).

In the classical RBF Network, the whole group of L parameters are split into 3 distinct sets (categories): set \mathbf{C} of all *Centers* with $N \times K$ parameters; set σ of N scalar *Widths* (Spreads) and a set \mathbf{W} of $N+1$ *Weights*, as follows:

$$\mathbf{P} = [p_1, p_2, \dots, p_L] = \mathbf{C} \cup \sigma \cup \mathbf{W} \quad (6)$$

It is obvious from (5) that the number of the parameters will rapidly grow with increasing the size of the RBFN model, namely the number N of RBFs and the number K of inputs. This leads to a respective increase in the CPU time for running the optimization algorithm that tunes the parameters of the RBFN model. In addition, the increase of the number of optimization parameters L leads to a respective decrease of the probability for finding the *global minimum* for the approximation error. This means that a reliable optimization algorithm being able to find the global optimum should be used.

OPTIMIZATION CRITERIA FOR TUNING THE PARAMETERS OF THE RBF NETWORK MODEL

The general assumption for creating a RBF Network model is that a collection (1) of M input-output pairs of experiments is available and the structure of the RBFN (the number N of the RBFs) is fixed prior to the learning process.

Then, the objective of the procedure for *off-line* tuning (optimizing) of all L parameters of the RBFN model is to *minimize* a preliminary formulated performance index (criterion). This is a typical case of *supervised learning*.

Criterion for Optimization of the RBFN Model Used as Universal Nonlinear Approximator

When the RBFN model is created in order to perform a *nonlinear approximation*, the most often used and natural criterion is to *minimize* the prediction (approximation) error RMSE of the RBFN model, based on all available M experiments:

$$RMSE = \sqrt{\frac{1}{M} \sum_{i=1}^M (y_i - y_{im})^2} \rightarrow \min \quad (7)$$

Further on in the text a synthetic example is given for illustration of the optimization of the parameters of the RBFN model when used as nonlinear approximator.

Criterion for Optimization of the RBFN Model Used as Classifier for Solving Classification Problems

When a classification problem has to be solved, the outputs of all M samples (experiments) from the training set are not real values, but rather *integer numbers* that represent T ordered classes with their integer labels: $t=1, 2, \dots, T$. Since the classical RBFN model produces real valued outputs, some *post processing* of the outputs is needed in order to transform the RBFN model into a *Classifier*. Here we propose to do this by the following *quantization* procedure:

1) Define the *ranges* of the real values $\mathbf{R}_t = [B_{t_{\min}}, B_{t_{\max}}]$, $t=1, 2, \dots, T$ that correspond to the respective classes, as follows:

$$B_{t_{\min}} = (2t - 1) / 2, \quad t=2, 3, \dots, T \quad (8)$$

$$B_{t_{\max}} = (2t + 1) / 2, \quad t=1, 2, \dots, T - 1 \quad (9)$$

Here we apply the following assumption (heuristics):

$$B_{1_{\min}} = 0.0; \quad B_{T_{\max}} = T + 1; \quad (10)$$

2) Then the total range \mathbf{R} of the real values that correspond to all T classes will be:

$$\mathbf{R} = \mathbf{R}_1 \cup \cup \cup \dots \cup [B_{1_{\min}}, B_{T_{\max}}] \quad (11)$$

3) For every real output value, predicted by the RBFN model (the classifier), the following integer *predicted class* will be assigned:

$$\text{if } (B_{t_{\min}} \leq y < B_{t_{\max}}) \text{ then Predicted Class} = t \quad (12)$$

4) The *misclassified samples* (experiments) is calculated by applying the following rule:

IF the predicted class for the current sample does not coincide with the real (given) class for this sample, THEN q is incremented by *one*.

Now, the proposed criterion for optimization is in the form of *classification error Cerr* that has to be *minimized*. This is a *real value* bounded between *0.0 and 1.0* that shows the portion of all misclassified samples from the whole list of M samples:

$$Cerr = q / M; \quad 0 \leq q \leq M; \quad 0.0 \leq Cerr \leq 1.0 \quad (13)$$

Further on in the text an illustration of how the above criterion (13) is used for optimizing the parameters of the RBFN model is given on two classification examples: synthetic data set and a real data set.

TUNING THE PARAMETERS OF THE RBFN MODEL BY THE ALGORITHM OF PARTICLE SWARM OPTIMIZATION WITH CONSTRAINTS

The problem of tuning the parameters of the RBF network has been investigated by many authors for a long time by using different learning algorithms that include separate or simultaneous tuning of the 3 groups of parameters [Musavi et al. 1992; Yousef 2005] and by applying different optimization approaches. In this paper we use a nonlinear optimization strategy for simultaneous learning of all three groups of parameters in (6) and we use for this purpose a modified version of the Particle Swarm Optimization algorithm with constraints, as described in the sequel.

The PSO algorithm belongs to the group of the *multi-agent* optimization algorithms (Eberhart and Kennedy 1995; Yousef 2005; Zhang et al. 2007). It uses a heuristics that mimics the behavior of flocks of flying birds (particles) in their collective search for a food. The main concept is that a single bird has not enough power to find the best solution, but in cooperation and exchanging information with other birds in the neighborhood it is likely to find the best (global) solution. The swarm consists of a predetermined number n of particles (birds).

The classical version of the PSO algorithm does not include any constraints (boundaries) on the search in the input space, i.e. this is an *unconstrained* search in the input space. Here the general idea is that birds should be free to explore the whole unlimited space until eventually find the global optimum.

However, when solving practical engineering problems it is often mandatory to impose certain constraints (limits) to the parameters $[x_1, x_2, \dots, x_K]$ in the input space in order to produce a physically meaningful solution. From a theoretical point of view, this is a *constrained optimization* procedure, which finally produces a *conditional* optimum that can be practically implemented.

In order to achieve such plausible solutions, we have made slight modification in the original version of the PSO algorithm with inertia weight that uses constraints in the form of boundaries (*minimum* and *maximum*) on all input parameters, as follows:

$$x_{i \min} \leq x_i \leq x_{i \max}, i = 1, 2, \dots, K \quad (14)$$

The idea of the constrained optimization is very simple. The input parameter that has violated the input space is *moved back* to its boundary value from (14), as follows:

$$\begin{aligned} \text{if } (x_i < x_{i \min}) \text{ then } x_i &= x_{i \min}; i = 1, 2, \dots, K \\ \text{if } (x_i > x_{i \max}) \text{ then } x_i &= x_{i \max}; i = 1, 2, \dots, K \end{aligned} \quad (15)$$

As a result, at the next iteration of the search, a velocity (step) with different amount and direction will be generated. Then the particle is likely to escape from the “prohibited” area beyond the boundary. This of course, may take sometimes a few trials (iterations).

THE LEARNING ALGORITHM FOR CREATING GROWING RBFN MODELS

The simultaneous learning of all L parameters of the classical RBFN model with predetermined number of N RBFs has the serious disadvantage that there is no way to predict in advance the correct number of RBFs that will produce a model with the desired RMSE (7). Therefore it is a normal case to run several times the whole algorithm for tuning the RBFN model with different number N of RBFs, until achieving the right model with the desired accuracy (RMSE). This is a tedious process that leads to *redundant* and computationally expensive optimizations.

There is another serious demerit of the simultaneous learning of all L parameters of the classical RBFN model. It is that the increase in the number N of RBFs will linearly increase the number L of the learning parameters, according to (5). As a result, the computation time for running the larger optimization task will also increase even faster than linearly, because a larger number of iteration will be needed to get closer to the global optimum in the higher L -dimensional space. The main idea of the algorithm for creating a *Growing* RBF Network Model has been explained in our previous work (Vachkov and Sharma 2014). It is experimentally shown there that this algorithm creates an economical model, in terms of computation time and reduced model complexity.

The proposed Growing RBFN model is free of the above demerits of the simultaneous learning of the fixed RBFN model. Instead of optimizing simultaneously all L parameters from (5) it decomposes the entire optimization problem into a sequence of optimizations, each of them with smaller number of parameters, namely those of the newly added RBF unit only. Thus the initial large optimization task is replaced by a sequence of optimization sub-task, each of them with smaller dimensionality (smaller number of tuning parameters).

In this algorithm, we run a *multi-step* procedure that starts with the simplest possible RBFN model with just $N=1$ RBF and a small number of parameters: $L = K+2+1 = K+3$ according to (5). These parameters are: K centers, one width, one weight and the offset weight. The optimization of these parameters is done by the PSO, explained in the previous section.

Then, if the desired accuracy $RMSE_{min}$ is not yet achieved, we proceed with a second step of the algorithm

by adding another RBF to the current BBFN model. Thus N becomes 2 and the number of all parameters will be: $L = 2 \times K + 2 \times 2 + 1$. This is the *growing step* of the algorithm. The new RBFN model obviously has a higher complexity. However the PSO algorithm is run for only the *new added* parameters $L' = K+2$, namely: K centers, one width and one weight for the newly added RBF unit. The previously optimized parameters (for $N=1$) are kept at their optimal values from the previous step.

If the desired approximation error is not yet achieved, *another growing step* is performed by adding another RBF unit ($N=3$) and the PSO algorithm is run for the new added $L' = K+2$ parameters only. The process stops until the desired error is satisfied, that is: $RMSE \leq RMSE_{min}$.

As a result, the Growing RBFN model automatically creates the RBFN model with the *minimal* number of RBFs, thus avoiding the problem of over-parameterizing of the model.

PERFORMANCE OF THE GROWING RBFN MODEL ON A TEST NONLINEAR EXAMPLE

Detailed comparison of the standard learning algorithm for the classical RBFN model with the learning algorithm for creating the Growing RBFN model was given in (Vachkov and Sharma 2014). Here we make a brief summary from comparing the two learning methods, based on the test nonlinear example shown in Fig. 3. This is an example with $K=2$ inputs and one output, specially generated for testing purposes.

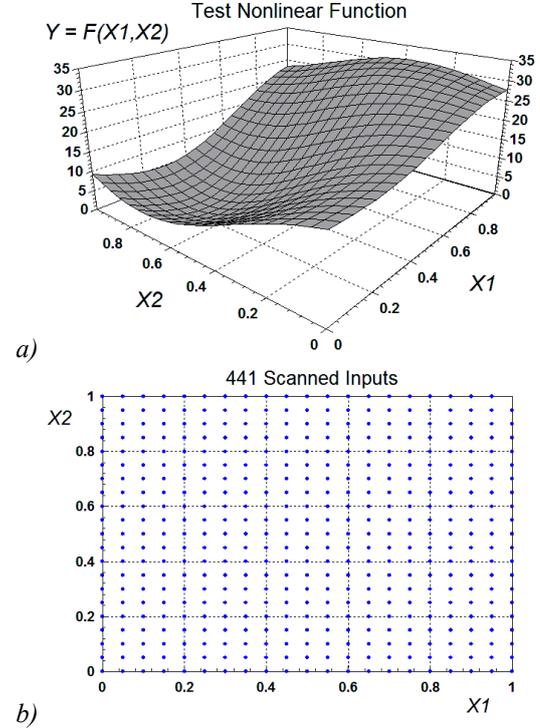


Figure 3: Test Nonlinear Example with two $K=2$ Inputs and the Training Set of Evenly Distributed $M=441$ Experimental Data Used in Simulations

In order to partly eliminate the influence of the random nature of the PSO algorithm, both learning algorithms

were run 10 times each, by using the same tuning parameters of the PSO for both of them.

The standard RBFN model was learned with a fixed number of $N=10$ RBF units and the Growing RBFN model started from $N=1$ and was grown until $N=10$ RBFs.

The learning results from all 10 trials (runs) for the classical RBFN model are displayed in Fig. 4. In a similar way the results from learning of the Growing RBFN model until $N=10$ (for all 10 trials) are shown in Fig. 5. Here the gradual and stable decrease of the training error RMSE can be clearly seen.

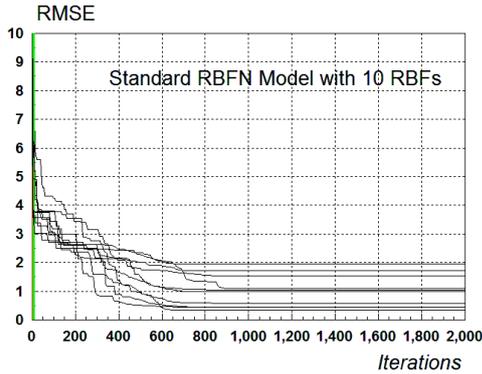


Figure 4: Results from Learning of the Classical RBFN Model after 2000 Iteration and 10 Trials

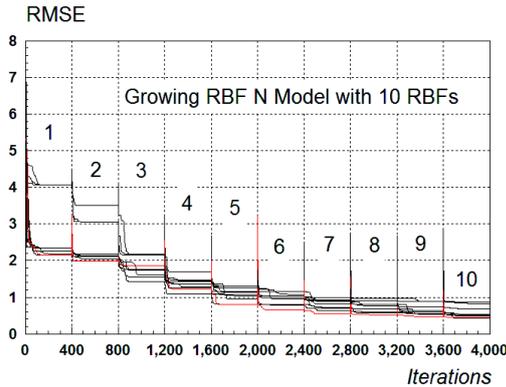


Figure 5: Results from Learning of the Growing RBFN Model with 400 Iterations for all Steps $N=1,2,\dots,10$ and for 10 Trials

It is easy to notice the smaller deviation of the results produces by the Growing RBFN model after all 10 trials. The minimal obtained error RMSE is similar for both models, but the mean of the RMSE for the Growing RBFN model: $RMSE = 0.5785$ is smaller than the mean of the RMSE for the Growing RBFN model: $RMSE=1.0196$. In addition, the standard deviation for the classical RBFN model is: 0.5641 , while for the Growing model it is only: 0.1556 .

From these experimental results a conclusion can be drawn that the learning algorithm for the Growing RBFN model is more stable, in a sense that it produces results with a smaller deviation of the results for multiple runs. Both learning methods have similar computation cost (about 50 sec in these experiments) with the assumed number of iterations. However, this balance may change

with changing the maximal number of iterations for each run of the PSO.

USING THE GROWING RBFN MODELS FOR CLASSIFICATION

In this Section we present the experimental results from using the proposed Growing RBFN model for solving classification problems, according to the explained *quantization* procedure in (8) – (12) and the criterion for calculating the *classification error* (13).

Classification Based on the Test Nonlinear Example

Here the same nonlinear test example from Fig. 3 was used for classification purpose and $T=3$ classes were artificially created by assuming the following levels of quantization for the output:

$$\begin{aligned} \text{if } (0 \leq y < 10) & \text{ then Class} = 1; \\ \text{if } (10 \leq y < 20) & \text{ then Class} = 2; \\ \text{if } (20 \leq y < 35) & \text{ then Class} = 3; \end{aligned} \quad (16)$$

The surface of all these 3 classes in the 2-dimensional feature (input) space is shown in the following Fig. 6 and Fig. 7.

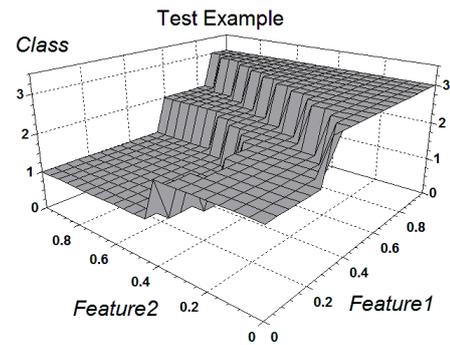


Figure 6: Surface of the Generated 3 Classes

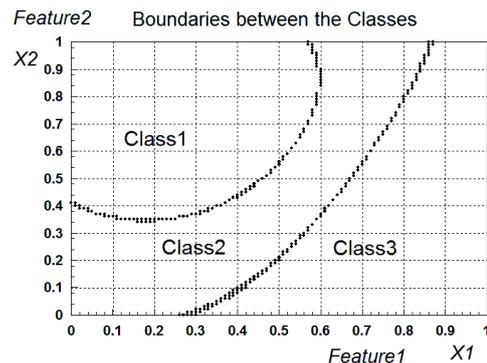


Figure 7: Boundaries between the Three Classes in the Two-Dimensional Input Space

The learning algorithm for the Growing RBFN model was applied by using a training set with 400 randomly generated samples in the 2-dimensional space, as shown in Fig. 8. As a *Classifier*, the optimal Growing RBFN model with 8 RBFs was used, as shown for the training results in Fig. 9. Here the minimal classification error (13) was $C_{err} = 0.0025$, which means *one* misclassification only.

The response surface of the optimal Classifier is shown in Fig. 10. The results from classification of two *test* data sets, consisting of 200 and 300 random samples are shown in the next Fig. 11 and Fig. 12 respectively.

It is worth noting that the popular *k*-nearest neighbors (kNN) algorithm (Suguna and Thanushkodi 2010) has produced in average worse results for the same classification task with $k = 3, 5$ and 7 , as follows:

Test Data Set with 200 Samples: 14, 14 and 11 misclassifications; Test Data set with 300 samples: 15, 10 and 14 misclassifications respectively.

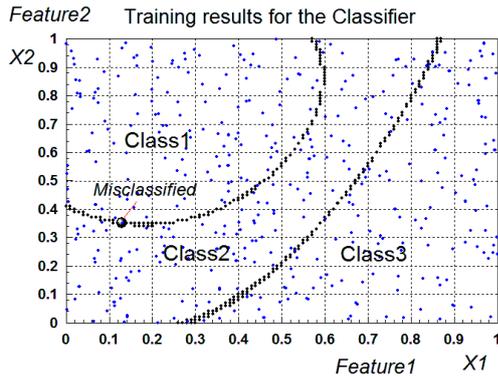


Figure 8: Training Results of the Classifier (Growing RBFN Model with 8 RBFs), based on 400 Random Samples.

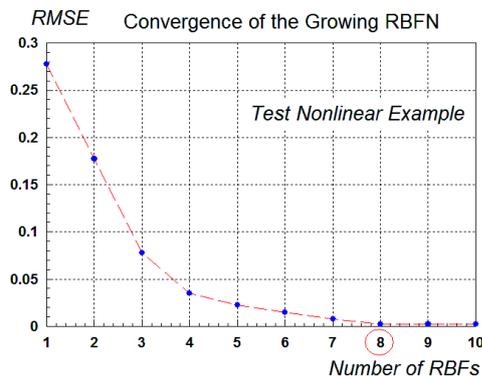


Figure 9: Gradual Decrease of the Training Error During the Learning of the Growing RBFN Model

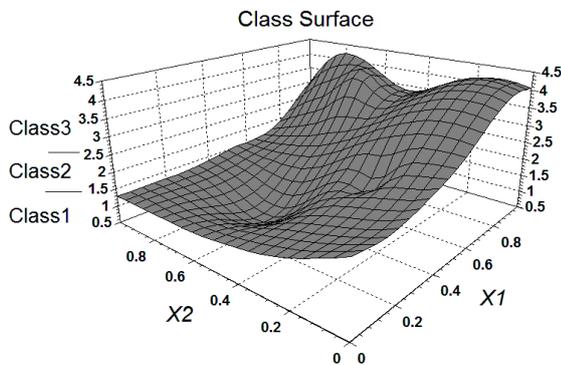


Figure 10: The Response Surface of the Optimal Classifier (the RBFN Model with 8 RBFs)

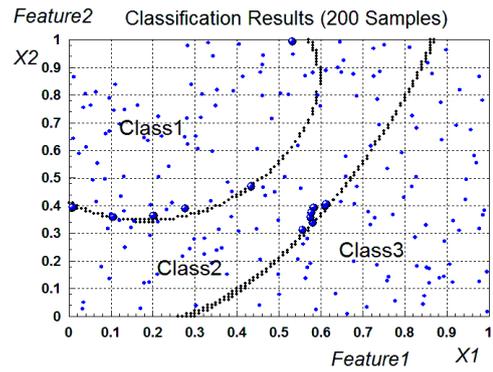


Figure 11: Classification Results from a Test Set with 200 Random Samples (12 Misclassifications)

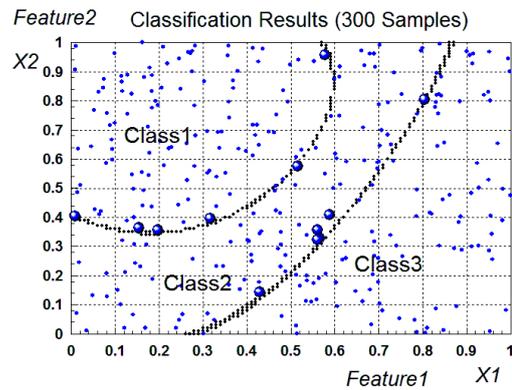


Figure 12: Classification Results from a Test Set with 300 Random Samples (11 Misclassifications)

Classification Based on the Wine Quality Data Set

In (Cortez et al. 2009), two datasets were used as real examples for testing several data mining methods for classification. The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. They comprise of 1599 samples for the red wine and 4898 samples for the white wine. The datasets are available in *UCI Machine Learning Repository* (Asuncion and Newman 2007).

There are 11 input attributes that can be used as possible features for classification. They represent different real variables based on physicochemical laboratory tests. Some of the attributes are correlated, which makes sense to apply some techniques for feature selection before the classification.

The output variable (based on sensory data) is the *quality* of the wine, expressed as an integer value between 0 and 10. The worst quality is marked as 0 and the best wine quality is denoted by 10. The classes are ordered (from 0 to 10) but are not balanced (e.g. there are much more normal wines than excellent or poor ones). The distribution of all classes for the full data set of the red wine with 1599 samples is shown in Fig. 13. It is seen that some classes are not represented by any samples. Therefore they are omitted and only the real classes (3,4,5,6,7 and 8) are taken into consideration. They are renumbered as "new classes" with respective new labels: **1,2,3,4,5** and **6** for classification, as shown in Fig. 13.

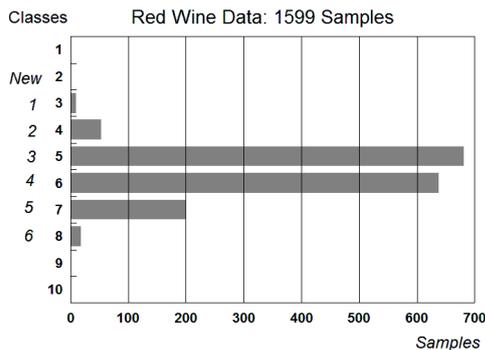


Figure 13: Distribution of the Classes for the Dataset of Red Wine with 1599 Samples

As for the inputs, we have omitted input 8 (wine density), since it has very small variation throughout the whole range of samples (between 0.9901 and 1.0037). Thus only the remaining 10 inputs were used as features for the classification problem with the new 6 classes.

The training curve for creating the Growing RBFN model is shown in Fig. 14. Here the first 1000 samples have been used as a *training set*. The minimal classification error $C_{err}=0.3720$ was obtained with $N=12$ RBFs. This translates into 372 misclassifications.

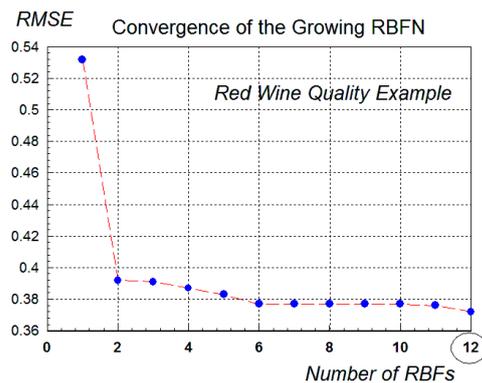


Figure 14: Gradual Decrease of the Training Error During the Learning of the Growing RBFN model.

The above simulations show that the Data set with the red wine samples is much more “difficult” for classification than the test nonlinear example because it has a higher dimension and obviously some of the classes are overlapped. The performance of the Classifier, based on the Growing RBN model is slightly better than that one of the kNN algorithm for different assumed numbers of neighbors.

CONCLUSIONS

A multi-step learning algorithm for creating a Growing RBFN model is proposed in this paper. It gradually increases the model complexity by inserting one new RBF unit at each step. An optimization strategy, based on the PSO algorithm with constraints is used to optimally tune the parameters of the newly inserted RBF unit only, while keeping the parameters of all “older” RBF units at their previously optimized values. In such way a significant reduction in the dimensionality of the optimization problem is achieved, compared with the case of learning the classical RBFN model. In addition,

the approximation error RMSE is gradually decreased at each step.

The obtained Growing RBFN model is optimal in a sense that it is the model with minimal number of RBFs that satisfies the predetermined desired RMSE. Thus the problem of over-parameterizing of the model is avoided. The proposed Growing RBFN model can be used successfully for nonlinear approximation of multidimensional processes (functions) that have real valued outputs. This was demonstrated in the paper on a test nonlinear example.

The proposed Growing RBFN model can be successfully used as a *classifier* for solving nonlinear classification problems. Here a special post-processing *quantization procedure* was designed that transforms the real valued outputs from the model into respective predicted classes of the classifier. The training and classification process of the classifier, based on the Growing RBFN model was demonstrated on the Red Wine Quality Data set.

A future direction of this research is aimed at solving some currently existing problems. First, the problem of automatic selection of the tuning parameters of the PSO algorithm should be solved in such way, so that to decrease the variations in the results from the different runs. Second, a more detailed and elaborated method is needed for possible automatic generation of the boundaries between the different classes. This could significantly improve the results from classification.

REFERENCES

- Asuncion A. and D. Newman. 2007. *UCI Machine Learning Repository*, University of California, Irvine, <http://www.ics.uci.edu/~mllearn/MLRepository.htm>.
- Cortez, P.; A. Cerdeira; F. Almeida; T. Matos; and J. Reis. 2009. “Modeling Wine Preferences by Data Mining from Physicochemical Properties”, In *Decision Support Systems*, Elsevier, 47(4): 547-553. ISSN: 0167-9236.
- Eberhart, R.C. and J. Kennedy. 1995. “Particle Swarm Optimization”, in: *Proc. of IEEE Int. Conf. on Neural Networks*, Perth, Australia, 1942-1948.
- Musavi, M.; W. Ahmed.; K. Chan.; K. Faris; and D. Hummels. 1992. “On the Training of Radial Basis Function Classifiers”, *Neural Networks*, 5, 595-603.
- Park, J. and I.W. Sandberg. 1993. “Approximation and Radial-Basis-Function Networks”. *Neural Computation*, 5, 305-316.
- Poggio, T. and F. Girosi. 1990. “Networks for Approximation and Learning”. *Proceedings of the IEEE*, 78, 1481-1497.
- Suguna N. and K. Thanushkodi. 2010. “An Improved k-Nearest Neighbor Classification Using Genetic Algorithm. *IJCSI International Journal of Computer Science Issues*, 7 (4), No. 22, ISSN (Print): 1694-0814, 18-21.
- Vachkov G. and A. Sharma. 2014. “Growing Radial Basis Function Network Models”, Proc. of the *IEEE Asia-Pacific World Congress on Computer Science and Engineering*, APWC on CSE, November 4-5, 2014, Plantation Island Resort, Fiji Island, 111-116.
- Yousef, R. 2005. “Training Radial Basis Function Networks using Reduced Sets as Center Points”. *International Journal of Information Technology*, 2, 21-27.
- Zhang, J.-R.; J. Zhang; T. Lok; and M. Lyu. 2007. “A Hybrid Particle Swarm Optimization, Back-Propagation Algorithm for Feed Forward Neural Network Training”, *Applied Mathematics and Computation* 185, 1026-1037.