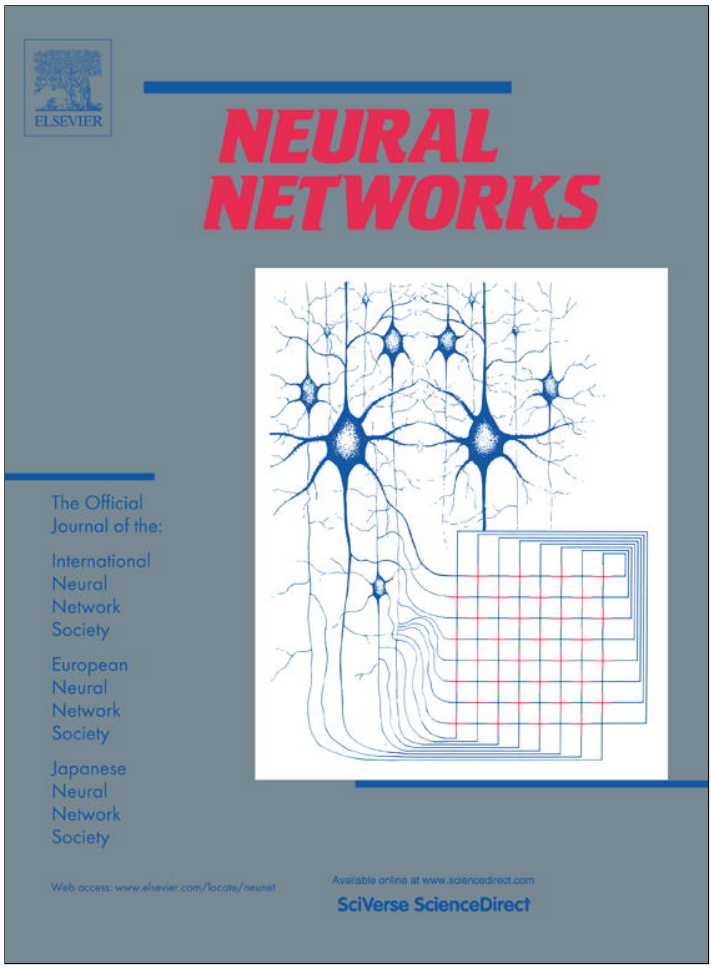

Neural Networks 46 (2013) 210–226

Contents lists available at SciVerse ScienceDirect

Neural Networks

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

A growing and pruning sequential learning algorithm of hyper basis function neural network for function approximation

Najdan Vuković [a,*], Zoran Miljković [b,1]

[a] University of Belgrade - Faculty of Mechanical Engineering, Innovation Center, Kraljice Marije 16; 11120 Belgrade 35, Serbia

[b] University of Belgrade - Faculty of Mechanical Engineering, Production Engineering Department, Kraljice Marije 16; 11120 Belgrade 35, Serbia

ARTICLE INFO

Article history:
Received 12 September 2012
Received in revised form 22 April 2013
Accepted 6 June 2013

Keywords:
Feedforward neural networks
Hyper basis function
Sequential learning
Neuron's significance
Extended Kalman filter

ABSTRACT

Radial basis function (RBF) neural network is constructed of certain number of RBF neurons, and these networks are among the most used neural networks for modeling of various nonlinear problems in engineering. Conventional RBF neuron is usually based on Gaussian type of activation function with single width for each activation function. This feature restricts neuron performance for modeling the complex nonlinear problems. To accommodate limitation of a single scale, this paper presents neural network with similar but yet different activation function—hyper basis function (HBF). The HBF allows different scaling of input dimensions to provide better generalization property when dealing with complex nonlinear problems in engineering practice. The HBF is based on generalization of Gaussian type of neuron that applies Mahalanobis-like distance as a distance metrics between input training sample and prototype vector. Compared to the RBF, the HBF neuron has more parameters to optimize, but HBF neural network needs less number of HBF neurons to memorize relationship between input and output sets in order to achieve good generalization property. However, recent research results of HBF neural network performance have shown that optimal way of constructing this type of neural network is needed; this paper addresses this issue and modifies sequential learning algorithm for HBF neural network that exploits the concept of neuron's significance and allows growing and pruning of HBF neuron during learning process. Extensive experimental study shows that HBF neural network, trained with developed learning algorithm, achieves lower prediction error and more compact neural network.

1. Introduction

Radial basis function (RBF) neural networks are among the most used single layered neural networks. The attractiveness of this network topology comes as a result of high accuracy and generalization property these networks are able to achieve when faced with complex engineering problems. More than three decades of research have witnessed a number of successful applications of RBF models for function approximation, classification, regression, system identification, dynamical modeling and control, and pattern recognition (Wu, Wang, Zhang, & Du, 2012). Compared to multilayer perceptron networks (MLP), the RBF network has one hidden layer and the network output is linear combination of basis functions, weighted with connecting weights. This makes them suitable for modeling those problems where fast neural solutions are needed, because simple matrix inversion is sufficient for determining weights, provided that parameters of all RBFs are known. Furthermore, as stated in Billings, Wei, and Balikhin (2007), regardless of the number of input vector dimensions and the total number of the RBF parameters, the RBF produces scalar output through nonlinear mapping, which is important because the RBF network model is not large and does not increase when the number of input variables increases. A number of different basis functions have been suggested and analyzed (thin plate spline, multiquadratic, inverse multiquadratic etc. (Simon, 2001)) but the most common used is Gaussian basis function. However, if the number of neurons (processing units) is large, the performance is poor due to over-fitting. On the other hand, small number of processing units may cause under-fitting. Therefore, optimal number of hidden units is important for RBF performance. Changing of RBF structure can be performed on-line, during the learning process (Alexandridis, Sarimveis, & Bafas, 2003), or off-line, when learning procedure ends (Islam, Sattar, Amin, Yao, & Murase, 2009). In this paper we focus on on-line structure changing, for class of generalized Gaussian basis function.

Learning algorithms for RBF networks spread from supervised approaches based on extended Kalman filter (Simon, 2001; Todorović, Stanković, & Moraga, 2002) or gradient descent methods (Karayiannis, 1999), to unsupervised learning approaches

* Corresponding author. Tel.: +381 11 3302 468.
E-mail addresses: nvukovic@mas.bg.ac.rs, najdanvuk@gmail.com (N. Vuković), zmiljkovic@mas.bg.ac.rs (Z. Miljković).
[1] Tel.: +381 11 3302 468.

0893-6080/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.neunet.2013.06.004

(Lazaro, Santamari, & Pantaleo, 2003) and combined unsupervised–supervised approaches (Schwenker, Kestler, & Palm, 2001). Recently, an interesting learning algorithm called extreme learning machine (ELM) (Huang, Zhu, & Siew, 2006) is proposed for single layered feedforward neural networks; in the case of the RBF networks, ELM randomly assigns RBF parameters and analytically determines network weights. In contrast to other approaches, ELM does not require various threshold parameters.

Another possibility for RBF parameters optimization is to use algorithms able to find global minimum (Gonzalez et al., 2003). However, these algorithms are computationally expensive and not designed for fast modeling needed in engineering practice.

If one excludes variant of ELM for online sequential learning (Huang, Zhou, Ding, & Zhang, 2012), majority of learning algorithms are batch, meaning that learning algorithm uses all training examples in each iteration. Although this approach can result in a higher accuracy of the RBF model, there are disadvantages of using all training examples in one learning iteration. Firstly, whenever new training data is available, the RBF model needs to be relearned with new data by going through learning process from the very beginning using all data. Furthermore, if the model assumes on-line modification of parameters, a common application of neural network in engineering practice, batch algorithms are not suitable. In contrast, when new data is available, the sequential learning continues learning process by updating the existing neural network, instead of going through entire learning process from the beginning, and this is why sequential algorithms are preferred. Sequential learning has the following characteristics (Huang, Saratchandran, & Sundararajan, 2005):

(1) learning system uses one and only one training example in iteration. The examples are presented to the learner sequentially, one following the other.
(2) training example is erased from memory after learning procedure finishes the update of network parameters.
(3) learning system has no prior knowledge of the total number of examples.

These features of sequential learning are important for modeling of engineering problems, where new data might be available after neural network model was built. Sequential learning enables learning system to continue learning process if new data is received, without the need to memorize and use old data.

In this paper, we develop and analyze a class of generalized RBF neural networks with Gaussian basis function that allows scaling of local input dimensions, a feature not present in "standard" RBF formulation. Compared to the RBF, the hyper basis function (HBF) has different scale for each dimension of the input vector $\mathbf{x}_i \in \mathbb{R}^{n_x}$ (n_x is the number of dimensions of input vector $\mathbf{x}_i$). Research results (Mahdi & Rouchka, 2011; Nishida, Yamauchi, & Omori, 2006) have shown that HBF generates neural network with same accuracy as RBF or even higher accuracy than RBF, but with less number of basis functions. Developed HBF network learning algorithm is sequential and during learning process it changes number of HBF neurons according to predefined optimality criteria. In this paper, growing and pruning ability of HBF neural network is founded on the original contribution of the neuron's significance, introduced by Huang et al. in series of papers (Huang, Saratchandran, & Sundararajan, 2004; Huang et al., 2005), which is modified by Bortman (Bortman & Aladjem, 2009), with introduction of Gaussian mixture model for modeling of complex input densities.

Experimental analysis on several datasets is used to show the applicability of the developed HBF network. The HBF's performance, in terms of accuracy, generalization property and the compactness of the generated network, is compared to RBF networks performance: MRAN (Lu, Sundararajan, & Saratchandran, 1997), GAP–RBF (Huang et al., 2004), GGAP–RBF (Huang et al., 2005) and GGAP–GMM (Bortman & Aladjem, 2009).

The paper is structured as follows. The second part of the paper provides an analysis of research results and compares features of the proposed sequential learning algorithm with the current state in the field. Third part of the paper introduces HBF neuron and HBF neural network and provides basic information related to their advantages over conventional RBF neuron and RBF network; this section of paper introduces concept of neuron's significance needed for growing and pruning ability, and presents the developed HBF learning algorithm. Section 4 of the paper analyzes stability of the EKF for HBF parameter optimization to prove that it is convergent and analyzes the influence of growing and pruning of HBF neurons on EKF convergence. Computational complexity and memory requirements of HBF–GP are provided in Section 5 of the paper. Section 6 shows the performance of developed sequential learning algorithm for HBF, for three simulated nonlinear engineering problems. The performance is directly compared to the well known RBF learning algorithms. Another performance comparison for three real world data is given in Section 7. Finally, in Section 8 we provide an extensive discussion of the obtained results and in Section 9 of the paper we summarize the conclusions from the study.

2. Related work and contributions of the paper

Determination of the optimal number of neural network neurons is open research problem and a number of possible solutions have been proposed (Alexandridis et al., 2003). It is a well known fact that too large network architectures may cause over-fitting to the data, while on the other hand, too small network architecture may underfit the data (Han, Chen, & Qiao, 2011; Huang et al., 2004, 2005; Islam et al., 2009; Miljković & Aleksendrić, 2009). In this part of the paper, we analyze main research contributions to this problem.

Resource allocation network (RAN) developed by Platt (Platt, 1991) was one of the first RBF networks able to add new neuron into hidden layer based on novelty of the data. The disadvantage of this approach is the inability to prune neurons, which results in a large RBF network. Kadirkamanathan and Niranjan (Kadirkamanathan & Niranjan, 1993) tried to enhance the RAN concept and developed RANEKF, RAN RBF network trained with extended Kalman filter (EKF), to improve the learning accuracy and get more parsimonious network topology. However, the disadvantage of RANEKF is the same as the RAN's—the inability to delete neurons, which is an important feature because in the case of complex problems the size of RBF network can become unnecessarily large.

Numerous improvements to RAN concept have been advised. Lu et al. (1997) and Lu, Sundararajan, and Saratchandran (1998) realized the disadvantage of the RAN and RANEKF and developed minimal resources allocating network (MRAN). Compared to RAN and RANEKF, MRAN was specifically developed to enable pruning of neurons. MRAN is based on sliding window that calculates the contribution of neuron to the overall network output. The basic idea of MRAN can be summarized in the following sentence: if the neuron's output is smaller than the previously defined threshold for a given number of iterations, then that neuron should be deleted. However, the size of the sliding window heavily influences the speed of learning process and its size is determined by user's experience during experimental process. Another improvement of RAN concept is based on QR decomposition (Salmeron, Ortega, Puntonet, & Prieto, 2001) which is used to determine the optimal structure of network. The disadvantage of this approach is a large number of new parameters, various thresholds and necessity for all data in pruning process.

Huang et al. (Huang et al., 2004; Zhang, Huang, Sundararajan, & Saratchandran, 2007) suggested another improvement of RANEKF and introduced growing and pruning RBF (GAP–RBF) based on the

concept of neuron's significance, where significance is defined as the contribution of the neuron to the network output averaged over all the input data received so far. However, this concept relies on hard assumption that the input density is uniform, an assumption not suitable for various types of problems. This flaw has been addressed in Huang et al. (2005) where the concept of neuron's significance is improved and generalized to various classes of input density. Generalized growing and pruning RBF (GGAP–RBF) is based on a similar concept of neuron's significance, but its analytical expression is more rigorously introduced using q-norm of the vectors (Huang et al., 2005). Compared to RANEKF and MRAN, GGAP–RBF generates better RBF models in terms of accuracy and compactness of the network. Nevertheless, GGAP–RBF lacks flexibility when defining the input density, because one needs to analyze the input data and define the (approximate) mathematical expression for input density. This is a big disadvantage of the GGAP–RBF, because in real life applications it relies on all data for the estimation of underlying input density, which is not suitable for complex engineering problems where fast design of RBF network models is required.

Another GGAP–RBF improvement was suggested by Bortman and Aladjem (2009), where GGAP–RBF concept of neuron significance is furthermore improved by estimating input density with Gaussian mixture model (GMM). The proposed algorithm uses same features of GGAP–RBF but estimation of input density is carried out using GMM (GGAP–GMM). The direct comparison of GGAP–GMM and GGAP–RBF (Bortman & Aladjem, 2009) shows that the GMM estimation of input density is able to improve the accuracy of the RBF model and network compactness.

In Han et al. (2011) the on-line optimization of RBF network structure is carried out in a different manner. Instead of growing and pruning the network, flexible structure RBF (FS–RBF) developed by Han et al. uses splitting and deleting of neighboring neurons; splitting is based on average firing rate of the neuron while deleting criteria uses mutual information between hidden layer and output layer and neuron is deleted if normalized mutual information is less than the previously defined threshold. With minor adjustments of initial formulation, this idea has been tested in a series of papers by the same authors for system identification (Qiao & Han, 2012) and model predictive control (Han, Qiao, & Chen, 2012).

Interesting idea of growing and pruning of RBF neural network is suggested by Todorović et al. (2002). They applied extended Kalman filter (EKF) for recurrent RBF training and proposed statistical tests for growing and pruning of network based on EKF's estimates of estimation error and innovation statistics. However, the bottleneck of this solution is the necessity to maintain correlations between variables in EKF framework, especially if state vector has large number of dimensions.

This paper differs from aforementioned research results along following lines. Instead of Gaussian radial basis function, in this paper the more general formulation of the Gaussian with diagonal weighting matrix is introduced and analyzed—hyper basis functions (HBF) (Mahdi & Rouchka, 2011; Nishida et al., 2006; Poggio & Girosi, 1989, 1990). Other research results (Bortman & Aladjem, 2009; Han et al., 2011, 2012; Huang et al., 2004, 2005; Qiao & Han, 2012) are focused on RBF implementation and analysis. The HBF network is the least studied and this paper analyzes its performance and shows its ability in experimental analysis carried out on several benchmark datasets. The HBF neural network is recently studied in Mahdi and Rouchka (2011); the HBF model of neural network is enhanced with soft local dimension reduction and weight decay. The overall conclusion is that HBF is able to achieve accuracy competitive to that of the Support Vector Machines (SVM). However, the study revealed that for some problems HBF network is unnecessarily large; authors emphasize the necessity to have the HBF network able to optimally change the number of neurons. With this in mind, research presented in this paper focuses on the sequential learning paradigm with the ability to optimally change the number of neurons during learning process (Bortman & Aladjem, 2009; Huang et al., 2004, 2005). One of the first efforts in this direction is (Nishida et al., 2006), where authors developed HBF network able to simultaneously learn new relationships between input–output pairs and determine optimal number of processing units. Authors developed RAN type of HBF network with modified EKF formulation, re-formulated pruning strategy, and introduced merging strategy; these changes have resulted in a higher accuracy and less number of neurons. However, as any MRAN type of algorithm, it heavily relies on various thresholds which are generally dependent on the problem being analyzed.

Having the previous analysis in mind, we develop sequentially trained HBF neural network and exploit the concept of neuron's significance (Huang et al., 2004, 2005) needed for growing and pruning of neurons during learning process. The input density of data is modeled with Gaussian mixture model as in Bortman and Aladjem (2009). However, these concepts are re-formulated for the case of HBF network, which is the main distinction between HBF neural network developed and analyzed in this paper and research results presented in Bortman and Aladjem (2009) and Huang et al. (2004, 2005). Other distinctions are emphasized later on, when we introduce and develop concepts needed for sequential learning of HBF neural network.

3. Proposed sequential learning algorithm for hyper basis functions with growing and pruning (HBF–GP)

3.1. Hyper basis functions

The RBF network is a feed forward type of neural network consisting of three layers: input layer, hidden layer and output layer. Input layer passes input vector $\mathbf{x}_i$ directly to the hidden layer, where each neuron in the hidden layer applies the same activation function. When shown input vector $\mathbf{x}_i$, neuron calculates Euclidean distance between input vector $\mathbf{x}_i$ and prototype vectors and passes calculated distances through activation function. The output function of RBF neural network is computed as:

$$\mathbf{y}_i = f(\mathbf{x}_i) = \sum_{j=1}^{J} w_j g_j\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \sigma_j\right) \tag{1}$$

where $g_j(\cdot,\cdot,\cdot)$ stands for the j-th basis function; $\mathbf{x}_i \in \mathbb{R}^{n_x}$ is the input vector in the RBF neural network (n_x stands for the number of the input dimensions, i.e. $n_x = \dim \mathbf{x}_i$), $\boldsymbol{\mu}_j \in \mathbb{R}^{n_x}$ is the center of j-th basis function (also referred as the prototype vector (Simon, 2001)) and $\sigma_j \in \mathbb{R}^1$ represents the scalar parameter; w_j is connecting weight of the j-th basis function. A common choice for basis function $g_j(\cdot,\cdot,\cdot)$ is the Gaussian activation function defined as:

$$\begin{aligned} g_j\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \sigma_j\right) &= \exp\left(-0.5\left\|\mathbf{x}_i - \boldsymbol{\mu}_j\right\|^2/\sigma_j^2\right) \\ &= \exp\left\{-0.5\left(\mathbf{x}_i - \boldsymbol{\mu}_j\right)^T\left(\mathbf{x}_i - \boldsymbol{\mu}_j\right)/\sigma_j^2\right\}. \end{aligned} \tag{2}$$

In this equation, scaling scalar parameter $\sigma_j \in \mathbb{R}^1$ defines the width of the j-th basis function. The Gaussian basis function peaks to one when the input vector $\mathbf{x}_i$ is close to the j-th center $\boldsymbol{\mu}_j$ (where closeness is defined by the Euclidean distance $\|\mathbf{x}_i - \boldsymbol{\mu}_j\|$), and decreases to zero in the opposite case. The Gaussian basis function can be regarded as a similarity measure between the input vector $\mathbf{x}_i$ and j-th center vector $\boldsymbol{\mu}_j$; as Euclidean distance $\|\mathbf{x}_i - \boldsymbol{\mu}_j\|$ increases, the Gaussian activation function outputs the small value; in the opposite case, when Euclidean distance $\|\mathbf{x}_i - \boldsymbol{\mu}_j\|$ decreases, the Gaussian outputs value is close to one. In other words, greater similarity between input vector $\mathbf{x}_i$ and j-th center vector $\boldsymbol{\mu}_j$ implies

bigger output of the Gaussian activation function. This feature is one of the reasons why RBF neural networks with Gaussian basis functions are a popular choice for modeling problems in regression, classification, system identification and signal processing.

A straight forward generalization of the simple Gaussian function is to consider the case where closeness is measured with a Mahalanobis-like distance instead of Euclidean-like distance. To be more mathematically stringent, this is not the real Mahalanobis distance measure but rather Mahalanobis-like distance measure. Hyper basis function using the Mahalanobis-like distance is given in the following form:

$$g_j\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) = \exp(-0.5\,\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2_{\boldsymbol{\Sigma}_j}) = \exp\left\{-0.5\,(\mathbf{x}_i - \boldsymbol{\mu}_j)^T \boldsymbol{\Sigma}_j (\mathbf{x}_i - \boldsymbol{\mu}_j)\right\} \quad (3)$$

where $\|\cdot\|^2_{\boldsymbol{\Sigma}_j}$ is Mahalanobis-like norm weighted with positive definite square matrix $\boldsymbol{\Sigma}_j$. Weighting matrix $\boldsymbol{\Sigma}_j$ makes similarity between the input vector $\mathbf{x}_i$ and j-th center vector $\boldsymbol{\mu}_j$ invariant to scaling and local orientation of the data (Mahdi & Rouchka, 2011). The Mahalanobis-like distance can be thought of as a generalization of the Euclidean distance for similarity measure, therefore the following forms of the matrix $\boldsymbol{\Sigma}_j$ are possible (Mahdi & Rouchka, 2011; Schwenker et al., 2001):

1. all neurons have the same width $\sigma \in \mathbb{R}^1$, i.e. $\boldsymbol{\Sigma}_j = (1/\sigma^2)\mathbf{I}$, where $\mathbf{I}$ is the identity matrix of appropriate dimensions.
2. all neurons have different width parameter: $\sigma_j \in \mathbb{R}^1, j = 1, \ldots, J$, i.e. $\boldsymbol{\Sigma}_j = (1/\sigma_j^2)\mathbf{I}$.
3. every neuron has a unique diagonal weighting matrix $\boldsymbol{\Sigma}_j$, with varying size and restricted orientation: $\boldsymbol{\Sigma}_j = \mathrm{diag}\left(1/\sigma_1^2, 1/\sigma_2^2, \ldots, 1/\sigma_{n_x}^2\right)$.
4. the general case, where weighting matrix $\boldsymbol{\Sigma}_j$ is a full matrix, rather than diagonal. Compared to the previous case, both size and orientation vary:

$$\boldsymbol{\Sigma}_j = \begin{bmatrix} 1/\sigma_{11}^2 & 1/\sigma_{12}^2 & \cdots & 1/\sigma_{1n_x}^2 \\ 1/\sigma_{21}^2 & 1/\sigma_{22}^2 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 1/\sigma_{n_x1}^2 & \cdots & \cdots & 1/\sigma_{n_xn_x}^2 \end{bmatrix}.$$

The first case of RBF neural network is constructed of J neurons with same (Gaussian) width σ for each neuron. Compared to other cases, this one has the least number of parameters to optimize, which makes it attractive for usage. However, the majority of engineering problems cannot be modeled with RBF neural network with (hard) assumption that each neuron has the same width because smaller number of parameters may result in under-fitting. Therefore, the second case relaxes the previous assumption and allows each neuron to have different width. This is the most popular choice for RBF neural network with Gaussian basis functions (the reader is kindly referred to Bortman & Aladjem, 2009, Huang et al., 2004, 2005 and references therein). However, in some applications there is a need to take into account scaling of dimensions of the input vector $\mathbf{x}_i$; Billings et al. (2007) provide excellent insight where additional scaling of input dimensions is necessary and emphasize its importance.

Cases three and four are more flexible than the previous ones in terms of the shape of Gaussian basis function. Being the most general case, the fourth case provides flexibility and biggest number of parameters to optimize. Although flexibility in terms of the neuron shape is a desirable, the large number of parameters, i.e. high degree of freedom of the RBF model, leads to severe over-fitting. Therefore, the third case provides a trade-off between two extremes: on one hand, we have cases one and two, where local scaling of data is not allowed, and on the other hand, the case four with high degree of freedom of the RBF neural network model.

In this research, we developed a RBF neural network based on the case three, where Gaussian basis functions are given with Eq. (3) and the weighting matrix $\boldsymbol{\Sigma}_j$ is diagonal, enabling scaling of local dimensions.

The case three is a least studied and RBF neural network using the Mahalanobis-like distance and diagonal weighting matrix $\boldsymbol{\Sigma}_j$ is often called the Hyper basis function neural network (HBF) (Mahdi & Rouchka, 2011; Nishida et al., 2006; Poggio & Girosi, 1989, 1990). Billings et al. (2007) introduced the similar model and named it generalized multi-scale radial basis function network (MSRBF). In this paper we will use hyper basis function network and HBF as its acronym.

The main advantage of HBF network compared to RBF network is the smaller number of neurons. Nevertheless, some disadvantages need to be stressed. Firstly, high degree of freedom may result in over-fitting of the HBF model to data, as pointed in Mahdi and Rouchka (2011). To solve this problem, we developed a HBF neural network model able to add or delete neuron on-line, during learning process. This feature should enable optimal (in some previously defined sense) determination of final number of HBF neurons. The next section of the paper introduces criteria for growing and pruning of HBF neurons during learning process.

Another useful feature of HBF neuron, and generally speaking HBF neural network, is in the ability to provide local scaling of the data which is important for modeling of nonlinear dynamical systems. Let us observe NARX model (Billings et al., 2007)

$$\mathbf{y}(t) = f(\mathbf{y}(t-1), \ldots, \mathbf{y}(t-l_y), \mathbf{u}(t-1), \ldots, \mathbf{u}(t-l_u)) + \mathbf{e}(t) \quad (4)$$

where $\mathbf{u}(t)$, $\mathbf{y}(t)$ and $\mathbf{e}(t)$ represent system input, output and noise variables, l_y and l_u are maximum lags of the input and output, and $f(\cdot)$ is some unknown nonlinear function. Furthermore, let us imagine we would like to find model of Eq. (4) based on RBF neural network (Eq. (1)). Then, input vector into neural network is formed as follows

$$\mathbf{x} = [\mathbf{x}_1(t), \ldots, \mathbf{x}_l(t)]^T, \quad (5)$$

where $l = l_y + l_u$ and

$$\mathbf{x}_k = \begin{cases} \mathbf{y}(t-k), & k = 1, \ldots, l_y \\ \mathbf{u}(t-(k-l_y)), & k = l_y+1, \ldots, l_y+l_u. \end{cases} \quad (6)$$

Now, let us assume that we have no prior knowledge about which element of input vector $\mathbf{x}$ is more significant, that is, which of the input dimensions has more influence on the output through nonlinear function $f(\cdot)$. Taking everything into consideration, the activation function $g(\cdot)$ can be written as:

$$g_j(\cdot) = \exp\{-0.5C\}$$
$$C = \left(\mathbf{y}(t-1) - \boldsymbol{\mu}_{j,1}\right)^2/\sigma_j^2 + \left(\mathbf{y}(t-l_y) - \boldsymbol{\mu}_{j,l_y}\right)^2/\sigma_j^2 + \cdots + \left(\mathbf{u}(t-1) - \boldsymbol{\mu}_{j,l_y+1}\right)^2/\sigma_j^2 + \cdots + \left(\mathbf{u}(t-l_y) - \boldsymbol{\mu}_{j,d}\right)^2/\sigma_j^2. \quad (7)$$

Without loss of generality let us assume that input $\mathbf{u}(t)$ and output $\mathbf{y}(t)$ are bounded in $\left[\underline{u}, \bar{u}\right], \left[\underline{y}, \bar{y}\right]$ and let us define associated ranges as $r_u = \left[\underline{u}, \bar{u}\right]$ and $r_y = \left[\underline{y}, \bar{y}\right]$. One can consider two common cases: (i) $r_y \ll r_u$ and (ii) $r_y \gg r_u$. In the first case, influence of lagged input variables $\mathbf{u}(t)$ may be exaggerated while the role of states $\mathbf{y}(t)$ may be downplayed; in the second case the opposite is valid: the role of system variables $\mathbf{y}(t)$ will be exaggerated and the role of system inputs $\mathbf{u}(t)$ will be downplayed. This problem is commonly encountered in black box modeling of nonlinear dynamical systems when designers have no prior knowledge of the influence input dimensions have on the output.

As Eq. (7) shows, single scale RBF networks are especially sensitive to this problem. The normalization of input variables can solve some part of this problem but not completely. HBF neuron provides natural extension of the RBF neuron able to tackle general case of the problem.

3.2. Definition of neuron's significance

This part of the paper briefly presents the concept of neuron's significance. As stated in Huang et al. (2004, 2005) the significance of neuron is defined as averaged contribution of the neuron to the overall network output, averaged over all training data received so far. Consider the case that RBF or HBF network has J neurons after seeing n sequentially presented training examples. The output of the network for the i-th training example is computed as:

$$f_1(\mathbf{x}_i) = \sum_{j=1}^{J} w_j g_j\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right). \tag{8}$$

If we remove the k-th processing unit, the output of the network will be changed to

$$f_2(\mathbf{x}_i) = \sum_{j=1}^{k-1} w_j g_j\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) + \sum_{j=k+1}^{J} w_j g_j\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right). \tag{9}$$

Removing the k-th neuron will result in the error:

$$E(k) = \|f_1 - f_2\|_q = \|w_k\|_q g_k(\mathbf{x}_i) \tag{10}$$

where $\|\cdot\|_q$ is the q-norm of vectors, which provides the measure for distance between two points in Euclidean space. In Huang et al. (2005), the neuron's significance is defined as L_q norm of $E(k)$ weighted by the input density function $p(\mathbf{x})$:

$$E_{\text{sig}}(k) = \|\mathbf{w}_k\|_q \left(\int_{\mathbb{R}^{n_x}} \exp(-q\|\mathbf{x} - \boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}_j}) p(\mathbf{x}) d\mathbf{x}\right)^{1/q}. \tag{11}$$

Observing the previous expression indicates that neuron's significance $E_{\text{sig}}(k)$ strongly depends on input density $p(\mathbf{x})$ and Huang et al. (2005) proposed an analytically tractable solution for simple probability density functions. However, this approach implies modeling of input density $p(\mathbf{x})$ with carefully chosen "simple" densities. To accommodate this, Bortman and Aladjem (2009) proposed Gaussian mixture modeling (GMM) for approximation of possible very complex input density $p(\mathbf{x})$, i.e.

$$\hat{p}(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m N(\mathbf{x}; \boldsymbol{\nu}_m, \boldsymbol{\Sigma}_m) \tag{12}$$

where $N(\mathbf{x}; \boldsymbol{\nu}_m, \boldsymbol{\Sigma}_m)$ is a multivariate Gaussian probability density function of random variable $\mathbf{x}$ parameterized with mean vector $\boldsymbol{\nu}_m \in \mathbb{R}^{n_x}$ and covariance matrix $\boldsymbol{\Sigma}_m \in \mathbb{R}^{n_x \times n_x}$, and α_m are mixing coefficients $\left(\sum_{m=1}^{M} \alpha_m = 1;\ \alpha_m > 0\ \forall m\right)$.

To derive a closed form (analytical) expression for the calculation of $E_{\text{sig}}(k)$, consider rewriting the Gaussian term under integral in the following manner:

$$\exp(-q\|\mathbf{x} - \boldsymbol{\mu}\|^2_{\boldsymbol{\Sigma}_j}) = (2\pi/q)^{n_x/2} \det(\boldsymbol{\Sigma}_j)^{-1/2} \times N(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j^{-1}/q). \tag{13}$$

Introduction of GMM for the approximation of input density $p(\mathbf{x})$ allows us to rewrite neuron's significance as:

$$\hat{E}_{\text{sig}}(k) = \|\mathbf{w}_k\|_q \left[(2\pi/q)^{n_x/2} \det\left(\boldsymbol{\Sigma}_j\right)^{-1/2} \sum_{m=1}^{M} \alpha_m \times \left(\int_{\mathbb{R}^{n_x}} N\left(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j^{-1}/q\right) N\left(\mathbf{x}; \boldsymbol{\nu}_m, \boldsymbol{\Sigma}_m\right) d\mathbf{x}\right)\right]^{1/q}. \tag{14}$$

Table 1
Sequential learning algorithm of hyper basis function neural network with growing and pruning ability.

Hyper basis function network with growing and pruning (HBF–GP):
input $(J_0, \varepsilon_{\min}, \varepsilon_{\max}, E_{\min}, \kappa, e_{\min})$

1. Estimate input density $p(\mathbf{x})$ with GMM to obtain estimate $\hat{p}(\mathbf{x})$
2. Set initial parameters of HBF network: $\mathbf{w}_J$, $\boldsymbol{\mu}_J$, $\boldsymbol{\Sigma}_J$
3. **For** each observation $(\mathbf{x}_i, \mathbf{y}_i)$ do:
3.1. Calculate parameters for growth criterion
$\varepsilon_i = \max\left[\varepsilon_{\max}\gamma^i, \varepsilon_{\min}\right],\ (0 < \gamma < 1)$
Calculate current error of the HBF network
$\mathbf{e}_i = \mathbf{y}_i - \mathbf{f}(\mathbf{x}_i)$
Find neuron k nearest to the input sample $\mathbf{x}_i$
$k = \arg\min \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k}$
3.2. Calculate parameters for potential new neuron
$\mathbf{w}_{J+1} = \mathbf{e}_i,\ \boldsymbol{\mu}_{J+1} = \mathbf{x}_i,\ \boldsymbol{\Sigma}_{J+1} = \kappa\|\mathbf{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k}\mathbf{I}$
3.3. Compute significance of newly added neuron $J+1$, i.e. $\hat{E}_{\text{sig}}(J+1)$
$\hat{E}_{\text{sig}}(J+1) = \|\mathbf{w}_{J+1}\|_q \left((2\pi/q)^{n_x/2} \det\left(\boldsymbol{\Sigma}_{J+1}\right)^{-1/2} \mathbf{N}_{J+1}^T \mathbf{A}\right)^{1/q}$
3.4. Update parameters of HBF network
If $\|\mathbf{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k} > \varepsilon_i$ and $\hat{E}_{\text{sig}}(J+1) > E_{\min}$
allocate new unit with parameters $\mathbf{w}_{J+1},\ \boldsymbol{\mu}_{J+1},\ \boldsymbol{\Sigma}_{J+1}$
Else
Update parameters $(\mathbf{w}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ of nearest neuron k using EKF
Compute significance of nearest neuron k, i.e. $E_{\text{sig}}(k)$
$\hat{E}_{\text{sig}}(k) = \|\mathbf{w}_k\|_q \left((2\pi/q)^{n_x/2} \det\left(\boldsymbol{\Sigma}_k\right)^{-1/2} \mathbf{N}_k^T \mathbf{A}\right)^{1/q}$
If $\hat{E}_{\text{sig}}(k) < E_{\min}$
Remove k-th neuron
EndIf
EndIf
EndFor
output $\left(J,\ \mathbf{w}_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right), j = 1, \ldots, J$

The reader may notice that $\hat{E}_{\text{sig}}(k)$ represents estimate of $E_{\text{sig}}(k)$ due to the introduction of $\hat{p}(\mathbf{x})$. The closed form analytical solution is easily derived as an integral of product of two Gaussians distributions:

$$\int_{\mathbb{R}^{n_x}} N\left(\mathbf{x}; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j^{-1}/q\right) N\left(\mathbf{x}; \boldsymbol{\nu}_m, \boldsymbol{\Sigma}_m\right) d\mathbf{x} = N\left(\boldsymbol{\mu}_j - \mathbf{x}_i; \mathbf{0}, \boldsymbol{\Sigma}_j^{-1}/q + \boldsymbol{\Sigma}_m\right). \tag{15}$$

When this result is taken into account, the estimate of neuron's significance becomes:

$$\hat{E}_{\text{sig}}(k) = \|\mathbf{w}_k\|_q \left((2\pi/q)^{n_x/2} \det\left(\boldsymbol{\Sigma}_j\right)^{-1/2} \mathbf{N}_j^T \mathbf{A}\right)^{1/q} \tag{16}$$

where $\mathbf{A}$ denotes vector of mixing coefficients of GMM, i.e. $\mathbf{A} = [\alpha_1, \alpha_2, \ldots, \alpha_M]^T$, and $\mathbf{N}_j$ is defined as:

$$\mathbf{N}_j = \left[N\left(\boldsymbol{\mu}_j - \boldsymbol{\nu}_1; \mathbf{0}, \boldsymbol{\Sigma}_j^{-1}/q + \boldsymbol{\Sigma}_1\right), N\left(\boldsymbol{\mu}_j - \boldsymbol{\nu}_2; \mathbf{0}, \boldsymbol{\Sigma}_j^{-1}/q + \boldsymbol{\Sigma}_2\right), \ldots, N\left(\boldsymbol{\mu}_j - \boldsymbol{\nu}_M; \mathbf{0}, \boldsymbol{\Sigma}_j^{-1}/q + \boldsymbol{\Sigma}_M\right)\right]^T. \tag{17}$$

In this paper, the 1-norm ($q = 1$) and 2-norm ($q = 2$) of the neuron's significance is applied and tested for the case of HBF network.

3.3. Hyper basis function sequential learning algorithm with growing and pruning ability (HBF–GP)

Having introduced HBF basis function and concept of neuron's significance for HBF type of neuron, the next part of paper introduces hyper basis function sequential learning algorithm with growing and pruning ability (HBF–GP). HBF–GP algorithm is given in Table 1.

HBF–GP sequential learning algorithm requires several input parameters: initial number of HBF neurons (J_0), parameters required for growth criterion ($\varepsilon_{\min}, \varepsilon_{\max}$), threshold for neuron significance $E_{\min}$, overlap of the hyper basis functions κ, and desired learning accuracy $e_{\min}$.

Prior to beginning of the learning process, the estimation of input density $p(\mathbf{x})$ requires $N_{\text{pre\_history}}$ data points. Experimental results in Section 4 show that this number is smaller than the total number of training examples needed for the estimation of input density $p(\mathbf{x})$ in the original GGAP–RBF algorithm, a conclusion already made by Bortman et al. in their research (Bortman & Aladjem, 2009).

Algorithm continues with calculation of parameter ε_i needed for growth criterion. In the same step, the algorithm calculates current error of the network $\mathbf{e}_i$ and nearest neuron k to the newest input sample $\mathbf{x}_i$. Given this information, the parameters for possible neuron are easily defined; its significance is calculated according to Eq. (12). The reader is kindly reminded that $\widehat{E}_{\text{sig}}(k)$ is calculated with simple matrix operations.

The next step of the HBF–GP determines whether new neuron should be added. This criterion is given in the step 3.4: as any RAN type sequential learning algorithm (Bortman & Aladjem, 2009; Huang et al., 2004, 2005; Lu et al., 1997, 1998; Platt, 1991), the HBF–GP adds new neuron to the hidden layer only if the possible new neuron $J+1$ is sufficiently far from the existing neurons, i.e. $\|\mathbf{x}_i - \boldsymbol{\mu}_k\|_{\boldsymbol{\Sigma}_k} > \varepsilon_i$; the reader is kindly reminded that the Mahalanobis-like distance metrics is used to define closeness between newest input vector $\mathbf{x}_i$ and center of HBF neuron $\boldsymbol{\mu}_k$, instead of Euclidean-like distance used in Bortman and Aladjem (2009), Huang et al. (2004, 2005), Lu et al. (1997, 1998) and Platt (1991). The second criterion is same as in Bortman and Aladjem (2009), Huang et al. (2004, 2005), and insures that significance of possible new neuron $\hat{E}_{\text{sig}}(J+1)$ is greater than threshold significance $E_{\min}$. When neuron $J+1$ satisfies these conditions it is added to the HBF structure, and learning procedure goes back to step 3 and presents new learning pair $(\mathbf{x}_i, \mathbf{y}_i)$. Otherwise, if growing condition is not fulfilled, the HBF–GP will not add new neuron $J+1$ to the HBF hidden layer and the HBF structure with J neurons will remain unchanged. In this case, the learning procedure continues with update of the nearest neuron k (calculated at step 3.2) with extended Kalman filter (EKF). The reader may notice that only parameters of neuron k nearest to the input sample $\mathbf{x}_i$ are updated with EKF, while parameters of other neurons are not modified. The rationale for this is quite simple: greater difference between input vector $\mathbf{x}_i$ and the j-th prototype vector $\boldsymbol{\mu}_j$ $(j = 1, \ldots, J)$ will result in smaller output/activation of the Gaussian basis function. The Jacobian matrix $\mathbf{H} = \left[\partial\mathbf{f}(\mathbf{x}_i)/\partial\mathbf{w}_j \quad \partial\mathbf{f}(\mathbf{x}_i)/\partial\boldsymbol{\mu}_j \quad \partial\mathbf{f}(\mathbf{x}_i)/\partial\boldsymbol{\Sigma}_j\right]$ will have non-zero elements only for the nearest neuron k; for other neurons $(\forall j \notin k)$ the Jacobian will approach to zero or small neglecting value. This approach is adopted in Bortman and Aladjem (2009) and Huang et al. (2004, 2005) as well.

Having updated parameters of the nearest neuron, the algorithm calculates the significance of the nearest neuron $\hat{E}_{\text{sig}}(k)$ and checks if it is greater than threshold significance $E_{\min}$; pruning of neuron occurs if its significance $\hat{E}_{\text{sig}}(k)$ is smaller than $E_{\min}$.

This learning procedure is sequentially applied for all training samples; when final sample N is processed, the learning stops and algorithm outputs HBF network. Outputs of the learning algorithm are parameters of HBF network: total number of HBF processing units (J), connecting weights $\mathbf{w}_j$, centers $\boldsymbol{\mu}_j$ and widths $\boldsymbol{\Sigma}_j$ of HBFs $(j = 1, \ldots, J)$.

Learning algorithm developed and presented in this paper (HBF–GP) differs from its closest predecessors GAP–RBF (Huang et al., 2004), GGAP–RBF (Huang et al., 2005) and GGAP–GMM (Bortman & Aladjem, 2009):

(1) Compared to the GAP–RBF and the GGAP–RBF, the HBF–GP estimates input density using GMM in the first step (Table 1). The expression for neuron's significance $E_{\text{sig}}(k)$ is the same as in GGAP–RBF case, but main difference is in estimate of $E_{\text{sig}}(k)$, i.e. $\widehat{E}_{\text{sig}}(k)$—Eq. (16), because approximation of input density $p(\mathbf{x})$ with GMM overcomes restrictions of GAP–RBF and GGAP–RBF. Furthermore, both GAP–RBF and GGAP–RBF use radial basis functions, hence their acronyms; in contrast this paper is based on hyper basis functions and ability for scaling of local dimensions.
(2) The HBF–GP has been built using same idea as GGAP–GMM in terms of input density $p(\mathbf{x})$ approximation with GMM. However, the main difference between HBF–GP compared to GGAP–GMM is in terms of basis function; similarly to GAP–RBF and GGAP–RBF, GGAP–GMM is based on radial basis functions, while HBF–GP uses generalization of RBF—hyper basis function. Yet another important distinction between these two algorithms is in terms of neuron's significance; GGAP–GMM was developed for $q = 2$, while in this research we extrapolate neuron's significance for the class of generalized RBF—the HBF and analyze influence of q norm on compactness of HBF neural network. Two distinct cases for q norm are analyzed: $q = 1$ and $q = 2$. General case of q norm for HBF neuron's significance could be easily derived.

3.4. EKF for HBF network parameter optimization

In this subsection of the paper we introduce Extended Kalman Filter as algorithm for optimization of HBF parameters. Linear discrete dynamical system is given as (Vuković, 2012):

$$\begin{aligned}\boldsymbol{\lambda}_i &= \boldsymbol{\lambda}_{i-1} + \boldsymbol{\varepsilon}\\ \mathbf{y}_i &= \mathbf{f}(\boldsymbol{\lambda}_i, \mathbf{x}_i) + \boldsymbol{\omega}\end{aligned} \tag{18}$$

where $\mathbf{x}_i, \mathbf{y}_i$ denote i-th input–output pair from the training set, and $\mathbf{f}(\boldsymbol{\lambda}_i, \mathbf{x}_i)$ denotes HBF neural network, while $\boldsymbol{\varepsilon}$ and $\boldsymbol{\omega}$ denote process and measurement noises ($\boldsymbol{\varepsilon} \sim N(\boldsymbol{\varepsilon}, \mathbf{Q})$ and $\boldsymbol{\omega} \sim N(\boldsymbol{\omega}, \mathbf{R})$) with zero mean and covarianceces $\mathbf{Q}$ and $\mathbf{R}$, respectively. The state vector $\boldsymbol{\lambda}$ is defined as: Eq. (19) is given in Box I where $\mathbf{w}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ represent weights, centers of HBF neurons and weighting matrix. To ease representation we introduced following notation for weighting matrix of HBF neurons $\boldsymbol{\Sigma}_k^{n_x}$: subscript denotes the particular neuron k while the superscript provides the local dimension. Therefore, subscript indexes are $1, \ldots, J$ while the superscript indexes are $1, \ldots, n_x$.

As one can see in Eq. (18), the state vector $\boldsymbol{\lambda}_i$ in time step i linearly evolves from the state vector $\boldsymbol{\lambda}_{i-1}$ in the previous time step $(i-1)$. We introduce standard assumptions:

$$\begin{aligned}&E[\boldsymbol{\lambda}_0] = \hat{\boldsymbol{\lambda}}_0; \qquad E\left[\left(\boldsymbol{\lambda}_0 - \hat{\boldsymbol{\lambda}}_0\right)\left(\boldsymbol{\lambda}_0 - \hat{\boldsymbol{\lambda}}_0\right)^T\right] = \mathbf{P}_0\\ &E[\boldsymbol{\varepsilon}_i] = \mathbf{0}; \qquad E\left[\boldsymbol{\varepsilon}_{i-1}\boldsymbol{\varepsilon}_i^T\right] = \mathbf{Q}\delta_{i-1,i}\\ &E[\boldsymbol{\omega}_i] = \mathbf{0}; \qquad E\left[\boldsymbol{\omega}_{i-1}\boldsymbol{\omega}_i^T\right] = \mathbf{R}\delta_{i-1,i}\end{aligned} \tag{20}$$

where $\delta_{i-1,i}$ represents Kronecker delta. Standard predictor–corrector cycle of EKF can be easily derived (Vuković, 2012):

$$\begin{aligned}&\hat{\boldsymbol{\lambda}}_{i|i-1} = \hat{\boldsymbol{\lambda}}_{i-1}; \qquad \mathbf{P}_{i|i-1} = \mathbf{P}_{i-1} + \mathbf{Q}\\ &\hat{\mathbf{y}} = \mathbf{f}(\hat{\boldsymbol{\lambda}}_{i|i-1}, \mathbf{x}_i);\\ &\mathbf{H}_i = \nabla_{\boldsymbol{\lambda}}\hat{\mathbf{y}}_i = \left[\nabla_{\mathbf{w}}\mathbf{f}(\cdot,\cdot)\ \nabla_{\boldsymbol{\mu}}\mathbf{f}(\cdot,\cdot)\ \nabla_{\sum}\mathbf{f}(\cdot,\cdot)\right]\\ &\mathbf{K} = \mathbf{P}_{i|i-1}\mathbf{H}_i^T\left(\mathbf{H}_i\mathbf{P}_{i|i-1}\mathbf{H}_i^T + \mathbf{R}\right)^{-1}\\ &\hat{\boldsymbol{\lambda}}_i = \hat{\boldsymbol{\lambda}}_{i|i-1} + \mathbf{K}_i(\mathbf{y}_i - \hat{\mathbf{y}}_i); \qquad \mathbf{P}_i = (\mathbf{I} - \mathbf{K}_i\mathbf{H}_i)\mathbf{P}_{i|i-1}\end{aligned} \tag{21}$$

$$\boldsymbol{\lambda} = \left[\mathbf{w}_1^T \quad \mathbf{w}_2^T \quad \cdots \quad \mathbf{w}_J^T \quad \boldsymbol{\mu}_1^T \quad \boldsymbol{\mu}_2^T \quad \cdots \quad \boldsymbol{\mu}_J^T \quad \boldsymbol{\Sigma}_1^1 \quad \boldsymbol{\Sigma}_1^2 \quad \ldots \quad \boldsymbol{\Sigma}_1^{n_x} \quad \cdots \quad \boldsymbol{\Sigma}_J^1 \quad \cdots \quad \boldsymbol{\Sigma}_J^{n_x}\right]^T \tag{19}$$

Box I.

where $\mathbf{K}_i$ denotes Kalman gain; all other parameters have already been defined.

4. Discussion of convergence

4.1. Stability of EKF for HBF parameter optimization

To prove that EKF is stable, one typically uses Lyapunov's direct method applied on discrete-time. Lyapunov's direct method is based on Lyapunov function defined with the following quadratic form system (Crassidis & Junkins, 2011; Jazwinski, 1970):

$$V(\tilde{\mathbf{e}}) = \tilde{\mathbf{e}}_i^T \mathbf{P}_i^{-1} \tilde{\mathbf{e}}_i \tag{22}$$

where estimation error is defined as $\tilde{\mathbf{e}}_i = \hat{\boldsymbol{\lambda}}_i - \boldsymbol{\lambda}_i$. By definition, error covariance matrix $\mathbf{P}_i$ is required to be positive definite, which implies that $V(\tilde{\mathbf{e}}) > 0$, $\forall \mathbf{x}_i \neq \mathbf{0}$. Difference between Lyapunov function in the next time step and in the present time step is given as follows:

$$\Delta V(\tilde{\mathbf{e}}) = V(\tilde{\mathbf{e}}_i) - V(\tilde{\mathbf{e}}_{i-1}) = \tilde{\mathbf{e}}_i^T \mathbf{P}_i^{-1} \tilde{\mathbf{e}}_i - \tilde{\mathbf{e}}_{i-1}^T \mathbf{P}_{i-1}^{-1} \tilde{\mathbf{e}}_{i-1}. \tag{23}$$

Now, to prove stability it suffices to show that $\Delta V(\tilde{\mathbf{e}}) < 0$. Estimation error in i-th time step is defined as

$$\tilde{\mathbf{e}}_i = \hat{\boldsymbol{\lambda}}_i - \boldsymbol{\lambda}_i = (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i) \tilde{\mathbf{e}}_{i-1} + \mathbf{K}_i \boldsymbol{\omega} - \boldsymbol{\varepsilon}. \tag{24}$$

Substituting homogeneous term of Eq. (24) $(\mathbf{I} - \mathbf{K}_i \mathbf{H}_i) \tilde{\mathbf{e}}_{i-1}$ into Eq. (23) gives us

$$\tilde{\mathbf{e}}_{i-1}^T \left\{ (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^T \mathbf{P}_i^{-1} (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i) - \mathbf{P}_{i-1}^{-1} \right\} \tilde{\mathbf{e}}_{i-1} < 0. \tag{25}$$

Now, to prove stability it suffices to show that weighting matrix in this quadratic form is negative definite, i.e.

$$(\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^T \mathbf{P}_i^{-1} (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i) - \mathbf{P}_{i-1}^{-1} < \mathbf{0}. \tag{26}$$

By rewriting the previous equation in the following form

$$\mathbf{I} - \mathbf{P}_i (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^{-T} \mathbf{P}_{i-1}^{-1} (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^{-1} < \mathbf{0}. \tag{27}$$

And substituting $\mathbf{P}_i$:

$$\mathbf{P}_i = (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i) \mathbf{P}_{i-1} (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^T + \mathbf{K}_i \mathbf{R} \mathbf{K}_i^T + \mathbf{Q}$$

into Eq. (27) we obtain

$$-\left(\mathbf{K}_i \mathbf{R} \mathbf{K}_i^T + \mathbf{Q}\right) (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^{-T} \mathbf{P}_{i-1}^{-1} (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^{-1} < \mathbf{0}. \tag{28}$$

By definition, $(\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^{-T} \mathbf{P}_{i-1}^{-1} (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)^{-1}$ is always positive definite, therefore Eq. (28) reduces to

$$-\left(\mathbf{K}_i \mathbf{R} \mathbf{K}_i^T + \mathbf{Q}\right) < \mathbf{0}. \tag{29}$$

Lyapunov's condition given in Eq. (29) is satisfied if measurement noise covariance $\mathbf{R}$ is positive definite and process noise covariance $\mathbf{Q}$ at least positive semi-definite. Knowing that both measurement noise covariance $\mathbf{R}$ and process noise covariance $\mathbf{Q}$ can be set to be positive definite, it is safe to conclude that EKF as learning algorithm for HBF neural network is stable. This ends the analysis of EKF stabilization properties.

4.2. Growing of HBF network

In the next step we show that adding HBF neuron to the network structure does not change the convergence of the Extended Kalman Filter. Let us assume that our HBF neural network with J processing units has seen N training samples so far. Now, let us add additional neuron to network structure producing total of $J + 1$ neurons; the error between desired output $\mathbf{y}_i$ and HBF network output is given as:

$$\begin{aligned}
\mathbf{e}_{i,J+1} &= \mathbf{y}_i - \sum_{j=1}^{J+1} \mathbf{w}_j g\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) \\
&= \mathbf{y}_i - \left(\sum_{j=1}^{J} \mathbf{w}_j g\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) + \mathbf{w}_{J+1} g\left(\mathbf{x}_i, \boldsymbol{\mu}_{J+1}, \boldsymbol{\Sigma}_{J+1}\right) \right) \\
&= \mathbf{e}_i - \mathbf{w}_{J+1} g\left(\mathbf{x}_i, \boldsymbol{\mu}_{J+1}, \boldsymbol{\Sigma}_{J+1}\right) \\
&= \mathbf{e}_i - \mathbf{e}_i.
\end{aligned} \tag{30}$$

Therefore, by adding additional HBF neuron to the network structure the resulting error would not jeopardize accuracy or convergence of the learning system; growing of network will decrease the overall error and it will speed up the convergence. In Eq. (30) we have used neuron initialization as given in the Table 1 (step 3.2): $\mathbf{e}_i = \mathbf{y}_i - \mathbf{f}(\mathbf{x}_i)$, $\mathbf{w}_{J+1} = \mathbf{e}_i$, and $(\boldsymbol{\mu}_{J+1} = \mathbf{x}_i) \Rightarrow g(\mathbf{x}_i, \boldsymbol{\mu}_{J+1}, \boldsymbol{\Sigma}_{J+1}) = 1$.

4.3. Pruning of HBF network structure

The final step in convergence discussion requires analyzing the influence when neuron is to be deleted. Consider a HBF neural network of J processing units and let us assume that we are to delete one unit (k-th HBF neuron). The error is defined as:

$$\begin{aligned}
\mathbf{e}_{i,J-1} &= \mathbf{y}_i - \sum_{j=1, j\neq k}^{J} \mathbf{w}_j g\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) \\
&= \mathbf{y}_i - \left(\sum_{j=1, j\neq k}^{J} \mathbf{w}_j g\left(\mathbf{x}_i, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right) + \mathbf{w}_k g\left(\mathbf{x}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) \right. \\
&\quad \left. - \mathbf{w}_k g\left(\mathbf{x}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) \right) \\
&= \mathbf{y}_i - \left(\mathbf{f}(\mathbf{x}_i) - \mathbf{w}_k g\left(\mathbf{x}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right) \\
&= \mathbf{e}_i + \mathbf{w}_k g\left(\mathbf{x}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right).
\end{aligned} \tag{31}$$

In other words, we have introduced an additional error term in Eq. (31). Now, q-norm of the error induced by removing k-th neuron from network is given by Eq. (6); for entire training set $i = 1, \ldots, n$ this error is given as:

$$\begin{aligned}
E(k, i) &= \| \left(E(k, 1), \ldots, E(k, n)\right)^T \|_q \\
&= \left(n^{-1} \sum_{i=1}^{n} (E(k, i))^q \right)^{1/q} \\
&= \| w_k \|_q \left(n^{-1} \sum_{i=1}^{n} \left(g\left(\mathbf{x}_i, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)\right)^q \right)^{1/q}.
\end{aligned} \tag{32}$$

By following the procedure given for RBF neuron (Huang et al., 2005), we are able to provide estimate of the HBF neuron significance, defined as statistical contribution of k-th HBF neuron to the overall output of the HBF network, i.e.

$$\begin{aligned}
E_{\text{sig}}(k) &= \lim_{n\to\infty} E(k) \\
&= \|\mathbf{w}_k\|_q \left(\int_{\mathbb{R}^{n_x}} \exp(-q \|\mathbf{x} - \boldsymbol{\mu}_k\|^2_{\boldsymbol{\Sigma}_k}) p(\mathbf{x}) d\mathbf{x} \right)^{1/q}.
\end{aligned} \tag{33}$$

All parameters in the previous equation have already been defined. This result shows that the error introduced by removing the k-th neuron introduced in Eq. (31) can be estimated using Eq. (33). In Table 1 (step 3.4) we explicitly calculate significance of HBF neuron after EKF update: if its significance is lower than threshold $E_{\min}$ that neuron and all its connections are deleted from network structure. In the opposite case, if its significance is bigger than threshold

Table 2
Computational complexity and memory requirements of tested algorithms.

Algorithm	Computational complexity			Memory requirements
	# of additions	# of multiplications	# of matrix inversions	
MRAN	$O(N_{RBF}^2)$	$O(N_{RBF}^2)$	1, $O(n_y)$	$O(N_{RBF}^2 J_{RBF})$
GAP–RBF	$O(N_{RBF}^2)$	$O(N_{RBF}^2)$	1, $O(n_y)$	$O(N_{RBF}^2 J_{RBF})$
GGAP–RBF	$O(N_{RBF}^2)$	$O(N_{RBF}^2)$	1, $O(n_y)$	$O(N_{RBF}^2 J_{RBF})$
GGAP–GMM	$O(N_{RBF}^2)$	$O(N_{RBF}^2)$	1, $O(n_y)$	$O(N_{RBF}^2 J_{RBF}) + O(N_{GMM} K)$
HBF–GP	$O(N_{HBF}^2)$	$O(N_{HBF}^2)$	1, $O(n_y)$	$O(N_{HBF}^2 J_{HBF}) + O(N_{GMM} K)$

$E_{\min}$ we keep the updated neuron in the network structure. The reader is reminded that we model input density $p(\mathbf{x})$ in Eq. (33) with Gaussian mixture which allows us to handle problems having complex input densities.

Having analyzed three important cases of the HBF convergence we conclude:

(1) as a learning algorithm, the EKF is stable, which means that it is convergent;
(2) growing of HBF network will improve accuracy and will not jeopardize convergence of the EKF;
(3) pruning of HBF network introduces additional error term (Eq. (31)), but we are able to estimate its influence by applying the concept of neuron significance (Eq. (33)) during learning.

5. Computational complexity

Computational complexity of the HBF–GP algorithm is easily determined knowing that the update of parameters is performed with Extended Kalman Filter (EKF). Theoretically, the computational complexity of EKF is $O(n^2)$ for prediction and $O(n^3)$ for the update step of the filter, where n is the number of dimensions of the EKF's state vector $\mathbf{x}$ (not to be confused with the input into HBF neural network).

As stated previously, we employed a simplified version of the EKF which neglects interdependencies between EKF states in the error covariance matrix. The reason for this approach is very simple: all elements of the Jacobian matrix, $\mathbf{H} = \left[\partial \mathbf{f}(\mathbf{x}_i)/\partial \mathbf{w}_j \quad \partial \mathbf{f}(\mathbf{x}_i)/\partial \boldsymbol{\mu}_j \quad \partial \mathbf{f}(\mathbf{x}_i)/\partial \boldsymbol{\Sigma}_j\right]$ will be close to zero except for the elements of the Jacobian corresponding to the nearest neuron. The sparse structure of the Jacobian reduces computational complexity in the update step to $O(n^2)$. A similar approach with the same intuition has been applied in Bortman and Aladjem (2009) and Huang et al. (2005) as well. However, in contrast to these approaches, we estimate the nearest neuron to the current input sample $\mathbf{x}_i$ using the Mahalanobis distance, and this neuron is used for the update. Table 2 shows computational complexity of the tested algorithms, and their memory requirements.

Total number of parameters per one RBF neuron with different width σ_j for each neuron $j = 1, \ldots, J$ is:

$$N_{RBF} = \underbrace{\dim(\mathbf{x})}_{n_x} + \underbrace{\dim(\mathbf{y})}_{n_y} + 1 = n_x + n_y + 1. \tag{34}$$

On the other hand, total number of parameters for HBF neuron is:

$$N_{HBF} = 2\dim(\mathbf{x}) + \dim(\mathbf{y}) = 2n_x + n_y. \tag{35}$$

The Table 2 reveals that computational complexity of the HBF–GP is of the same order of magnitude as other learning algorithms $\sim O(n_x^2 + n_y^2)$ (all other elements except squares are omitted) but it grows linearly with the square of the input dimension n_x, i.e. $O(4n_x^2 + n_y^2)$.

Table 3
Specification of benchmark datasets. n_x denotes the number of attributes (dimension of input vector $\mathbf{x}_i$, N—the total number of examples, N_{train}—the number of training examples, and N_{test}—the number of testing examples.

Dataset	Type of data	Type of problem	n_x	N	N_{train}	N_{test}
2D sine wave	Simulated	Function approximation	2	600	200	400
Dynamical system with long input delays	Simulated	System identification	2	2000	1000	1000
MacKey–Glass chaotic time series	Simulated	Time series prediction	4	1000	500	500
Abalone age prediction	Real	Regression	8	4177	3000	1177
Weather Ankara	Real	Regression	9	1609	1100	509
Auto-MPG	Real	Regression	7	398	320	78

The memory requirements are given in the Table 2 as well: J_{RBF} and J_{HBF} represent the total number of processing units for RBF and HBF neural networks respectively; K stands for the total number of Gaussians used in the GMM for modeling the input density $p(\mathbf{x})$, and N_{GMM} is number of parameters per one Gaussian component in the GMM defined as $N_{GMM} = (n_x^2 + 3n_x + 2)/2$.

Observing the Table 2 it seems that HBF–GP requires more storage space, but in experiments, we find out that on an average, HBF–GP needs less number of neurons $J_{RBF} > J_{HBF}$, which significantly reduces memory requirements. Furthermore, some components in the GMM can be redundant, therefore we employ a method for estimating parameters of Gaussian mixture that simultaneously estimates optimal number of components K based on minimum message length criterion (Figueiredo & Jain, 2002). Less number of processing units and optimal number of GMM components reduce memory requirements.

6. Simulation studies and performance comparison

HBF–GP performance in terms of accuracy, generalization and compactness of the network is tested in the experimental process. In this section of the paper, the performance of the proposed HBF–GP learning algorithm is tested for several benchmark problems for system identification, function approximation, time series prediction and regression. These benchmark problems are chosen, so that the HBF–GP performance over different complex problems can be assessed. Characteristics of benchmark problems are given in Table 3. All experiments were conducted in Matlab 7.8 environment running on laptop computer (Intel Core2Duo on 2 GHz with 2 GB of RAM). Furthermore, we have directly compared HBF–GP to the well known algorithms in the field: MRAN (Lu et al., 1997), GAP–RBF (Huang et al., 2004), GGAP–RBF (Huang et al., 2005) and GGAP–GMM (Bortman & Aladjem, 2009).

6.1. 2D sine wave

The first simulation study analyzes HBF–GP performance for a 2D sine wave problem defined as (Orr et al., 2000):

$$y = 0.8\,\sin(x_1/4)\sin(x_2/2); \quad x_1 \in [0, 10],\ x_2 \in [-5, 5]. \tag{36}$$

The output value is corrupted with additive Gaussian noise $N(0, 0.1^2)$. Training set consists of 200 examples while testing sets have 400 examples from defined input range. The test data is generated from the same range and it is noise free.

Two measures of learning accuracy are used: the mean arithmetic error (MAE) and the root mean square error (RMS). These accuracy measures are calculated for training and testing sets and compared.

To test the influence of q norm—Eq. (16), two cases are considered: $q = 1$ (1-norm HBF–GP) and $q = 2$ (2-norm HBF–GP).

Table 4
Experimental results of HBF–GP performance for 2D *sine wave* averaged over 50 independent runs. Table shows initial number of HBF neurons J_0, numerical value for q norm in definition of neuron's significance (Eq. (12)), final number of HBF neurons J after processing N training examples (with their rounded values in brackets), root mean square error for testing and training sets (RMS_{test} and RMS_{train}), and mean arithmetic error for testing and training sets (MAE_{test} and MAE_{train}).

J_0	q	J	RMS_{test}	RMS_{train}	MAE_{test}	MAE_{train}
1	1	1.90 ± 0.36 (2 ± 0)	0.0954 ± 0.0679	0.1041 ± 0.0432	0.0760 ± 0.0509	0.1321 ± 0.0562
	2	2.78 ± 0.65 (3 ± 1)	0.0735 ± 0.0257	0.0944 ± 0.0126	0.0597 ± 0.0220	0.1201 ± 0.0150
2	1	2.06 ± 0.31 (2 ± 0)	0.0776 ± 0.0379	0.0990 ± 0.0213	0.0626 ± 0.0307	0.1253 ± 0.0262
	2	2.78 ± 0.65 (3 ± 2)	0.0735 ± 0.0257	0.0944 ± 0.0126	0.0597 ± 0.0220	0.1201 ± 0.0150
3	1	2.08 ± 0.34 (2 ± 0)	0.0734 ± 0.0451	0.0964 ± 0.0258	0.0588 ± 0.0340	0.1209 ± 0.0338
	2	2.68 ± 0.89 (3 ± 1)	0.0760 ± 0.0257	0.0948 ± 0.0122	0.0606 ± 0.0198	0.1194 ± 0.0160
5	1	2.18 ± 0.52 (2 ± 1)	0.0758 ± 0.0611	0.1018 ± 0.0368	0.0606 ± 0.0448	0.1289 ± 0.0495
	2	2.56 ± 0.73 (3 ± 1)	0.0626 ± 0.0246	0.0935 ± 0.0125	0.0515 ± 0.0206	0.1173 ± 0.0147
10	1	2.74 ± 1.01 (3 ± 1)	0.0615 ± 0.0289	0.0927 ± 0.0161	0.0497 ± 0.0231	0.1176 ± 0.0194
	2	3.96 ± 1.61 (4 ± 2)	0.0501 ± 0.0219	0.0873 ± 0.0100	0.0402 ± 0.0174	0.1111 ± 0.0121
15	1	4.6 ± 2.59 (5 ± 3)	0.0710 ± 0.0443	0.0942 ± 0.0255	0.0565 ± 0.0339	0.1166 ± 0.0328
	2	6.34 ± 2.34 (6 ± 2)	0.0537 ± 0.0210	0.0850 ± 0.0099	0.0441 ± 0.0177	0.1054 ± 0.0118

The parameters for HBF–GP are chosen as: $\varepsilon_{\max} = 5$, $\varepsilon_{\min} = 0.1$, $\kappa = 1$, and $\gamma = 0.99$. The EKF parameters for both setups are: initial state uncertainty $p_0 = 1$, $\mathbf{P}_0 = p_0\mathbf{I_P}$, state transition uncertainty $q_0 = 0.01$, $\mathbf{Q} = q_0\mathbf{I_Q}$, and measurement uncertainty $r_0 = 1$, $\mathbf{R} = r_0\mathbf{I_R}$, where $\mathbf{I_P}$, $\mathbf{I_Q}$, and $\mathbf{I_R}$ are the identity matrices of appropriate dimensions.

To test HBF–GP's growing and pruning ability, HBF–GP is tested with different initial number of neurons: 1, 2, 3, 5, 10, and 15. HBF–GP is tested for 50 independent runs. Training results are given in Table 4.

As Table 4 indicates, for any chosen number of initial neurons J_0, the final number of HBF neurons J converges to the value in the range [2, 5]. The final number of neuron J increases for $J_0 = 15$ (on average, HBF converges to [5, 6] neurons in this setup), but, compared to other experimental results for different number of initial units J_0 (Table 4), the overall difference between estimated optimal number of HBF neurons is one to two processing units. The rationale for this discrepancy is: it is much harder to define optimal number of neurons in sequential learning setup than in batch learning. For instance, in Islam et al. (2009) one can see that it takes a several retraining attempts with all available training examples to find optimal number of hidden neurons. This is the advantage of batch algorithm, but the sequential learning is a faster learning procedure and off-line pruning is not allowed. For these reasons, the optimal number of neurons J (optimal in the sense of neuron's significance defined in Eq. (12)) slightly differs when the initial number of neurons J_0 is increased. Furthermore, all predecessors of HBF–GP (GAP–RBF, GGAP–RBF and GGAP–GMM) are not tested to assess the influence of different number of initial neurons J_0; by definition, these algorithms are initialized with one initial neuron. HBF–GP shares the same roots with these algorithms, but we decided to test it with different number of initial neurons J_0.

It is interesting to analyze generalization ability of HBF–GP. Table 4 shows that RMS_{test} is fairly constant and for each test it falls in the range [0.05, 0.08]; values for MAE_{test} is [0.04, 0.08] for each run, regardless on the initial number of neurons or chosen value for q norm. Knowing that the initial number of neurons J_0 is varied in the process, it is safe to conclude that HBF–GP has converged to the optimal number of neurons, which in turn generates constant performance in terms of generalization ability.

The influence of the q norm on the optimal number of HBF neurons J is given in Fig. 1; the figure shows evolution of optimal number of HBF neurons J averaged over 50 independent trials.

Although the initial number of HBF neurons is different, experimental results after 50 independent trials (Fig. 1) show that HBF–GP converges to the optimal number of neurons J regardless of the starting point. Obtained experimental results clearly indicate that the HBF–GP can modify number of neurons in the hidden layer according to the introduced concept of neuron's significance and approximation of input density with GMM. Regardless of the initial number of neurons, the HBF–GP changes the total number of neurons and converges to some optimal value.

It is interesting to notice that in this example, 1-norm neuron's significance is less enthusiastic about addition of new HBF neurons. On average, 2-norm neuron's significance seems to be more enthusiastic in terms of need for new HBF processing units. As Fig. 1(a)–(c) depicts, at the end of sequential learning, both networks converge to different number of neurons. A similar trend is noticeable when the initial number of neurons is increased ($5 \leq J_0 \leq 15$): Fig. 1(d)–(f) shows that higher number of initial neurons J_0 induces faster rate of pruning for 1-norm HBF–GP than for 2-norm HBF–GP. On the basis of experimental results for 2D sine wave problem, one may conclude that 1-norm HBF–GP neural network needs less processing units than 2-norm HBF–GP neural network.

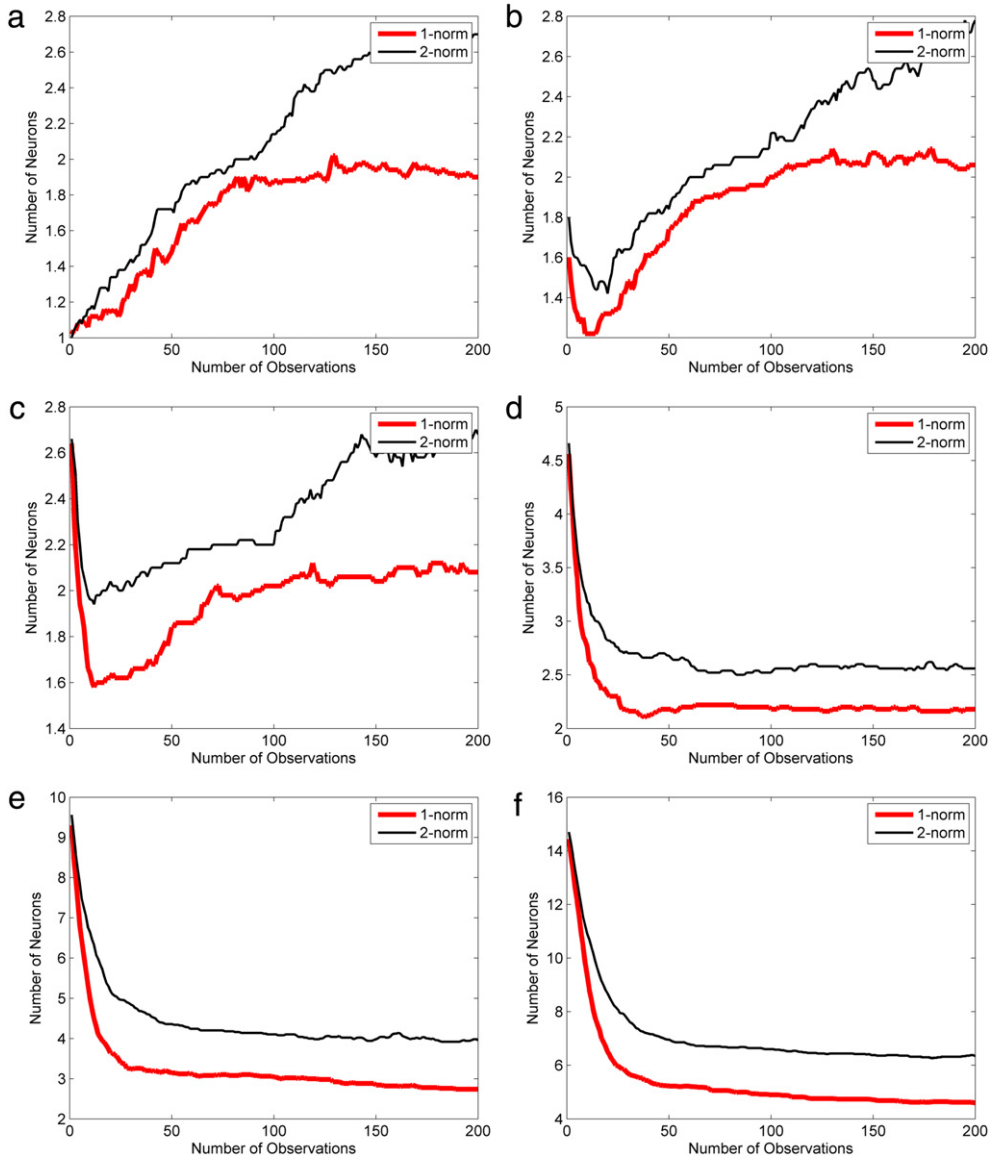

Fig. 1. Evolution of optimal number of HBF neurons J during training for two different choices of q norm, $q = 1$ (1-norm) and $q = 2$ (2-norm), averaged over 50 independent trials: (a) $J_0 = 1$, (b) $J_0 = 2$, (c) $J_0 = 3$, (d) $J_0 = 5$, (e) $J_0 = 10$, (f) $J_0 = 15$.

6.2. Dynamical system with long input delays

Consider the dynamical plant represented with following equation (Han et al., 2011):

$$y_p(t+1) = 0.72y_p(t) + 0.025y_p(t-1)u(t-1) + 0.01u^2(t-2) + 0.2u(t-3). \quad (37)$$

There are only two input values $y_p(t)$ and $u(t)$; these values are fed into HBF–GP neural network to generate desired output value $y_p(t+1)$. The training inputs are uniformly distributed in the $[-2, 2]$ interval for about half of the training time and a single sinusoid signal $1.05\sin(\pi t/45)$ for the remaining training time. The training data has 1000 training examples. To verify the performance of the generated HBF–GP network and analyze identification results, the testing signal is applied:

$$u(t) = \begin{cases} \sin(\pi t/25), & 0 < t < 250, \\ 1.0, & 250 \le t < 500 \\ -1.0, & 500 \le t < 750 \\ 0.3\sin(\pi t/25) + 0.1\sin(\pi t/32) \\ \quad + 0.6\sin(\pi t/10), & 750 \le t < 1000. \end{cases} \quad (38)$$

The testing set consists of 1000 examples. The input into HBF neural network is given by the following vector $\mathbf{x} = \begin{bmatrix} y_p(t) & u(t) \end{bmatrix}^T$. Performance of the HBF–GP model is compared to MRAN radial basis function neural network (Lu et al., 1997) and GAP–RBF (Huang et al., 2004).

To fully show how HBF neural network can be used to model nonlinear dynamical system with long input delays we test how the initial spread of HBF neuron influences learning performance. We tested two different cases of weighting matrix $\boldsymbol{\Sigma}$ initialization:

(1) in the first setup, the initial spreads of neurons is a scalar κ, ($\kappa \in \mathbb{R}^1$) multiplying all dimensions of the weight matrix $\boldsymbol{\Sigma}$, i.e.

$$\boldsymbol{\Sigma}_j = \kappa \cdot \mathbf{I} = \mathrm{diag}\left(1/\sigma_1^2, 1/\sigma_2^2, \ldots, 1/\sigma_{n_x}^2\right), \quad \kappa = 1/\sigma^2,\ n_x = \dim(\mathbf{x}).$$

this way of initialization was adopted in Nishida et al. (2006) as well.

(2) in the second setup, we adopt initial values of weighting matrix $\boldsymbol{\Sigma}_j$ in the following manner. Similar to the approach presented in Billings et al. (2007) where nonlinear input–output system

Table 5
Performance of HBF–GP neural network for dynamical system with long input delays ($\varepsilon_{max} = 0.1, \varepsilon_{min} = 0.01$). Table shows final number of processing units J and root mean square error RMS_{test} for different setup of scale parameter κ and weighting matrix $\mathbf{\Sigma}$. Results are averaged over 30 independent trials.

Initial spread	Algorithm	$\kappa = 1/\sigma^2$	J	RMS_{test}
Case #1 $\mathbf{\Sigma} = \kappa\,\mathbf{I}$	HBF–GP ($q = 1$)	0.5	3.73	0.0813
		0.75	4.85	0.0905
		0.95	6.1	0.0733
		2	8.7	0.0727
		3	12	**0.0680**
	HBF–GP ($q = 2$)	0.5	7.66	0.0705
		0.75	9.25	0.0746
		0.95	9.45	0.0689
		2	9.56	**0.0660**
		3	13.95	0.0679
Case #2 $\mathbf{\Sigma} = \kappa \cdot \mathrm{diag}(\mathbf{\Sigma}_{y_p}, \mathbf{\Sigma}_u)$	HBF–GP ($q = 1$)	0.5	4.15	0.0832
		0.75	5.8	0.0731
		0.95	6.1	0.0799
		2	8.60	0.0663
		3	10.5	**0.0660**
	HBF–GP ($q = 2$)	0.5	7.45	0.0913
		0.75	8.55	0.0759
		0.95	9.15	0.0776
		2	12.05	**0.0626**
		3	14.4	0.0641

Table 6
Performance of HBF–GP neural network for dynamical system with long input delays ($\varepsilon_{max} = 1, \varepsilon_{min} = 0.01$). Table shows final number of processing units J and root mean square error RMS_{test} for different setup of scale parameter κ and weighting matrix $\mathbf{\Sigma}$. Results are averaged over 30 independent trials.

Initial spread	Algorithm	$\kappa = 1/\sigma^2$	J	RMS_{test}
Case #1 $\mathbf{\Sigma} = \kappa\mathbf{I}$, $\kappa = 1/\sigma^2$	HBF–GP ($q = 1$)	0.5	3.15	0.0849
		0.75	3.22	0.0854
		0.95	4.65	**0.0763**
		2	5.15	0.0824
		3	5.65	0.0919
	HBF–GP ($q = 2$)	0.5	3.4	0.0799
		0.75	5	**0.0758**
		0.95	5	0.0759
		2	5.6	0.0833
		3	6	0.1007
Case #2 $\mathbf{\Sigma} = \kappa \cdot \mathrm{diag}(\mathbf{\Sigma}_{y_p}, \mathbf{\Sigma}_u)$	HBF–GP ($q = 1$)	0.5	3.2	0.0777
		0.75	3.7	**0.0766**
		0.95	3.4	0.0786
		2	5.35	0.0844
		3	5.5	0.0842
	HBF–GP ($q = 2$)	0.5	3.65	0.0717
		0.75	4.65	0.0736
		0.95	5.01	0.0721
		2	6.05	**0.0691**
		3	5.6	0.0892

with delays is modeled with neural network, we partition the weighting matrix $\mathbf{\Sigma}_j$ into two blocks, i.e.

$$\mathbf{\Sigma}_j = \kappa \cdot \mathrm{diag}\left(\mathbf{\Sigma}_{y_p}, \mathbf{\Sigma}_u\right). \tag{39}$$

The first block $\mathbf{\Sigma}_{y_p}$ defines spreads of state variables $y_p(t)$, while the second block $\mathbf{\Sigma}_u$ defines spread of controls $u(t)$; scalar $\kappa \in \mathbb{R}^1$ represents initial scale. Initial values of weighting matrix $\mathbf{\Sigma}$ are adopted as:

$$\begin{aligned} \mathbf{\Sigma}_{y_p} &= \left\{\max(y_p(t)) - \min(y_p(t))\right\}_{t=1}^{1000}; \\ \mathbf{\Sigma}_u &= \{\max(u(t)) - \min(u(t))\}_{t=1}^{1000}. \end{aligned} \tag{40}$$

Experimental results are given in Table 5. HBF parameters are set as: $\varepsilon_{max} = 0.1, \varepsilon_{min} = 0.01$, and $\gamma = 0.99$, the initial state uncertainty $p_0 = 0.9$, $\mathbf{P}_0 = p_0\mathbf{I_P}$, state transition uncertainty $q_0 = 0.001$, $\mathbf{Q} = q_0\mathbf{I_Q}$, and measurement uncertainty $r_0 = 1$, $\mathbf{R} = r_0\mathbf{I_R}$, where $\mathbf{I_P}$, $\mathbf{I_Q}$, and $\mathbf{I_R}$ are the identity matrices of appropriate dimensions.

In addition, Table 6 presents experimental results obtained in the second experiment in which we increased ε_{max} and decreased ε_{min}: adopted values of parameters are: $\varepsilon_{max} = 1$, $\varepsilon_{min} = 0.01$, and $\gamma = 0.99$. EKF parameters are the same as in the previous case.

Experimental results given in Tables 5 and 6 point out the following conclusions. Firstly, in Table 5, increase in scale parameter κ induces increase in processing units J and decrease of test statistics RMS_{test}. One may notice that on average, RMS_{test} is lower in the second case: the minimal RMS_{test} is $RMS_{test} = 0.0660$ for 1-norm and $RMS_{test} = 0.0626$ for 2-norm HBF–GP neural network.

Secondly, Table 6 shows similar trend when scale parameter κ is increased: number of neurons J is increased as well, but with lower rate of increase than in the first experiment presented in Table 5. Minimal values of test statistics RMS_{test} are bigger than in the previous setup of parameters: average RMS_{test} (for both cases in terms of scale parameter κ) tends to $\sim$0.0800, which is higher than average results obtained in the first experiment given in Table 5. Similar to the first case, on average test statistics RMS_{test} is lower for 2-norm HBF–GP.

Based on these results we adopted the following setup of parameters for HBF–GP:

Table 7
Experimental results for dynamical system with long input delays; table shows final number of processing units J, root mean square error RMS_{test} and final number of parameters for GAP–RBF, MRAN and HBF–GP.

Algorithm		J	RMS_{test}	# parameters
GAP–RBF	$\kappa = 0.87$	14.01 ± 1.42 (14 ± 1)	0.0860 ± 0.0146	$\sim$64
	$\kappa = 2$	13.55 ± 1.81 (14 ± 2)	0.0865 ± 0.0150	$\sim$64
MRAN	$\kappa = 0.87$	26.57 ± 12.6 (27 ± 13)	0.1127 ± 0.0202	$\sim$108
	$\kappa = 2$	29.85 ± 12.37 (30 ± 12)	0.1032 ± 0.0160	$\sim$120
HBF–GP	$q = 1$	8.60 ± 1.72 (9 ± 2)	0.0663 ± 0.0183	$\sim$45
	$q = 2$	12.05 ± 3.1 (12 ± 3)	0.0626 ± 0.0136	$\sim$60

$\varepsilon_{max} = 0.1$, $\varepsilon_{min} = 0.01$, $\kappa = 2$, and $\gamma = 0.99$, while initial estimate of neurons spread is adopted according to the second case $\mathbf{\Sigma} = \kappa \cdot \mathrm{diag}(\mathbf{\Sigma}_{y_p}, \mathbf{\Sigma}_u)$. For GAP–RBF and MRAN we set:

$\varepsilon_{max} = 0.1$, $\varepsilon_{min} = 0.01$, $\kappa = 0.87$, and $\gamma = 0.99$. Both GAP–RBF and MRAN were tested for two cases: $\kappa = 2$ (same as HBF–GP) and $\kappa = 0.87$ (usual setting for GAP–RBF) keeping other parameters unchanged. Additionally, for MRAN we set $e_{min} = 0.01$ and $e'_{min} = 0.1$, and growing threshold is $\delta = 0.01$, while growing/pruning window is $M = 30$. EKF parameters are unchanged: $p_0 = 0.9$, $q_0 = 0.001$, and $r_0 = 1$. Neural networks are initialized with one neuron $J_0 = 1$. Experimental results showing performance of GAP–RBF, MRAN and HBF–GP over 30 independent trials are given in Table 7.

As seen in the Table 7, the HBF–GP outperforms GAP–RBF and MRAN in terms of generalization. The generalization ability of HBF–GP, measured with RMS_{test}, is better than other self-organizing algorithms. Average optimal number of neurons for HBF–GP is fewer than in GAP–RBF and MRAN. In general, the main advantage of HBF over RBF concept is fewer number of parameters needed. Total number of parameters for Gaussian basis function neural network with different width σ_j for each neuron

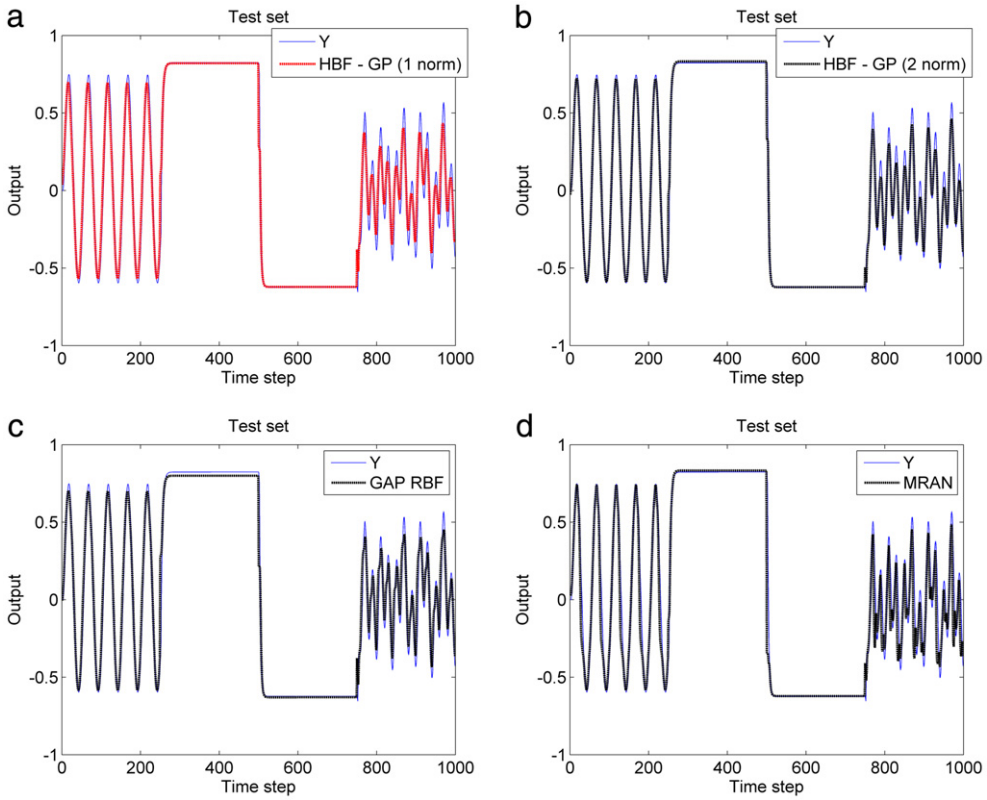

Fig. 2. Output of tested neural networks compared to the test signal: (a) 1-norm HBF–GP, (b) 2-norm HBF–GP, (c) GAP–RBF and (d) MRAN. All tested networks "recognize" given test pattern.

$j = 1, \ldots, J$ used in MRAN and GAP–RBF is:

$$N_p^{\mathrm{RBF}} = J_{\mathrm{RBF}}(n_x + n_y + 1) = J_{\mathrm{RBF}} N_{\mathrm{RBF}}. \tag{41}$$

On the other hand, total number of HBF neural network parameters is:

$$N_p^{\mathrm{HBF}} = J_{\mathrm{HBF}}(2n_x + n_y) = J_{\mathrm{HBF}} N_{\mathrm{HBF}}. \tag{42}$$

Table 7 reveals that the HBF–GP needs less parameters to memorize information about relationship between input vector **x** and output vector **y**. This is important insight into HBF's performance: less number of neurons implies less number of parameters, as shown in Table 7.

Fig. 2(a)–(d) shows the output of HBF–GP neural network compared to the test signal (Eq. (18)), while the error between the test signal and the output of four neural networks is given in Fig. 3(a)–(d). Figs. 2 and 3 indicate that 2-norm HBF–GP generates more accurate output given the test signal which may be seen in Fig. 3: the error peaks of 2-norm HBF–GP are smaller of 1-norm HBF–GP error peaks. Consequently, the root mean square error of 2-norm HBF–GP is smaller as well (Tables 5 and 6). Performance of GAP–RBF is similar to 1-norm HBF–GP but with notable difference: the error peaks of GAP–RBF at switching points $t = 250$ and $t = 500$ (Eq. (38), Figs. 2 and 3) are bigger (shown in Fig. 3), which results in a higher value of root mean square error (Table 7). All algorithms diverge from test signal in switching points ($t = 250$; $t = 500$, and $t = 750$) but as experimental results given in Table 7 and Fig. 3 indicate, on average the HBF–GP generates smaller error between the test signal and generated output; compared to performance of GAP–RBF and MRAN, HBF–GP achieves better generalization with less number of parameters.

Fig. 4 shows evolution of the root mean square error during learning process for 1-norm HBF–GP, 2-norm HBF–GP, GAP–RBF and MRAN averaged over 30 trials. Fig. 5 depicts evolution of the number of neurons averaged over 30 trials. As it can be seen, average number of neurons converges for all algorithms to some optimal value. HBF neural network, trained with proposed HBF–GP sequential learning algorithm, outperforms both GAP–RBF and MRAN in terms of needed number of neurons. Compared to MRAN, the HBF–GP converges to smaller number of neurons. In contrast to MRAN, the GAP–RBF is closer to HBF–GP in terms of number of neurons but it still needs bigger number of neurons than HBF–GP.

Although the performance of GAP–RBF in terms of number of neurons is similar to that of HBF–GP, the HBF–GP achieves better generalization (smaller RMS_{test}) and needs less number of parameters, as experimental results shown in Table 7 and Fig. 3 indicate.

This illustrative example shows HBF–GP's performance directly compared to GAP–RBF and MRAN. Experimental results demonstrate that HBF–GP achieves higher accuracy and needs less number of neurons (and parameters in general) than MRAN and GAP–RBF. These features of HBF–GP are important, especially from an engineering perspective, where fast neural solutions are required.

6.3. MacKey–Glass chaotic time series prediction

MacKey–Glass chaotic time series has been recognized as one of the benchmark problems for assessing the performance of learning algorithms. The time series is generated with following discrete equation:

$$x(t+1) = (1-a)x(t) + bx(t-\tau)/(1 + x^{10}(t-\tau)). \tag{43}$$

For $a = 0.1$, $b = 0.2$, $\tau = 17$ and choosing $x(0) = 1.2$ the goal is to predict $x(t + \Delta t)$ as a function of the previous states. In this setup, the prediction model of $x(t + \Delta t)$ is given by:

$$x(t + \Delta t) = f(x(t), x(t-6), x(t-12), x(t-18)). \tag{44}$$

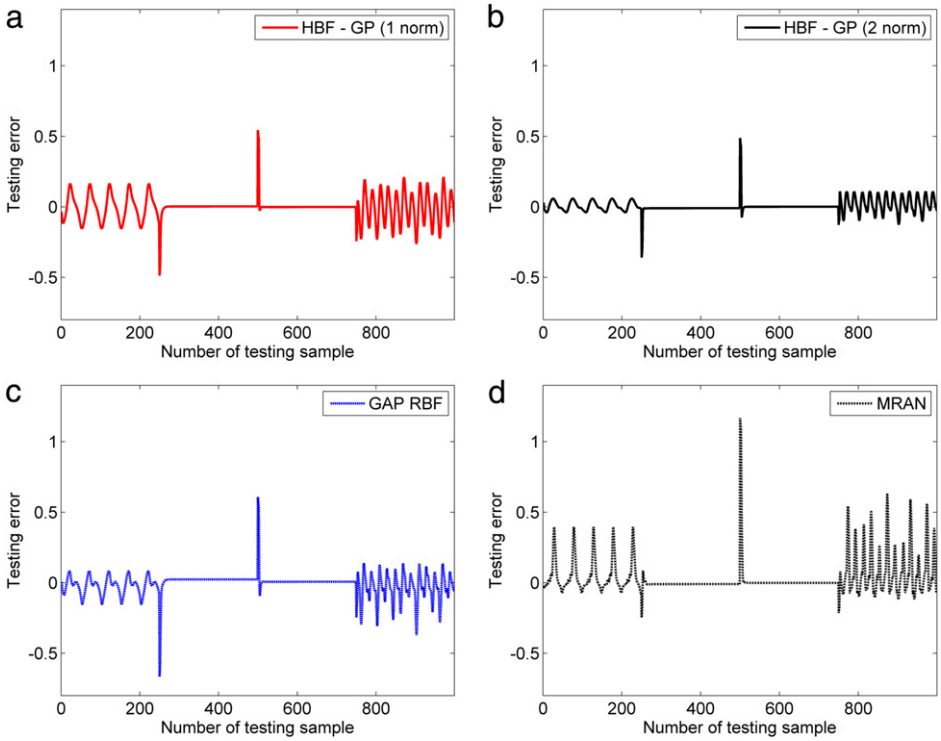

Fig. 3. The error between the test signal and output of neural network: (a) 1-norm HBF–GP, (b) 2-norm HBF–GP, (c) GAP–RBF and (d) MRAN. Compared to GAP–RBF and MRAN, HBF–GP generates smaller error over test interval.

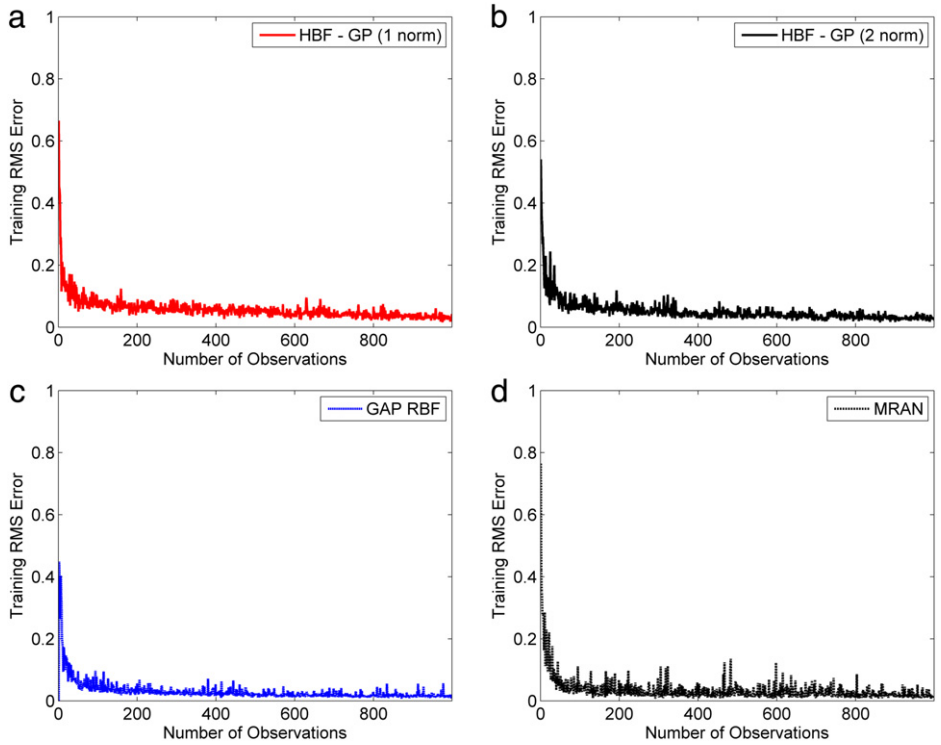

Fig. 4. Evolution of the root mean square error during learning process: (a) 1-norm HBF–GP, (b) 2-norm HBF–GP, (c) GAP–RBF and (d) MRAN. Results are averaged over 30 experimental trials.

500 data points are generated for training ($t = 1, \ldots, 500$) and another 500 data points for testing ($t = 501, \ldots, 1001$).

Performance of HBF–GP neural networks in terms of accuracy and compactness of the generated network is compared to the GGAP–RBF (Huang et al., 2005) the sequential learning algorithm of the RBF neural network, which is known for producing good results for this problem. Since HBF–GP can be seen as a direct offspring of GGAP–RBF, we have decided to test both learning algorithms for two characteristic cases: (i) norm for calculating neuron significance is $q = 1$ and (ii) norm is $q = 2$. For additional information related to GGAP–RBF the reader is referred to the seminal paper (Huang et al., 2005).

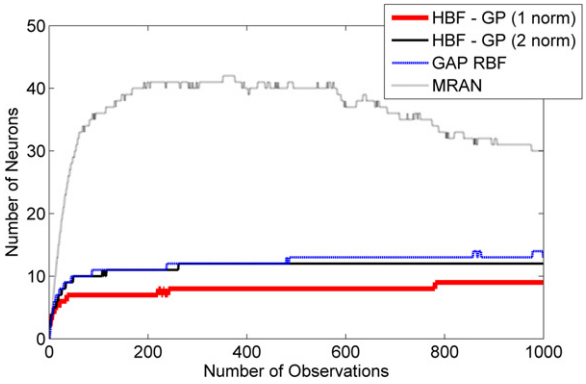

Fig. 5. Evolution of the number of neurons during learning process. HBF–GP outperforms both GAP–RBF and MRAN. On average, 1-norm HBF–GP needs less neurons than 2-norm HBF–GP. Results are averaged over 30 experimental trials.

Table 8
Experimental results for MacKey–Glass time series prediction. Table shows performance in terms of number of processing units J, testing root mean square error RMS_{test} and overall number of parameters for GGAP–RBF and HBF–GP. Results are averaged over 30 trials.

Algorithm		J	RMS_{test}	# parameters
GGAP–RBF	$q = 1$	11 ± 1.72 (11 ± 2)	0.0348 ± 0.0061	∼77
	$q = 2$	18 ± 1.61 (18 ± 2)	0.0364 ± 0.0050	∼126
HBF–GP	$q = 1$	2.13 ± 0.31 (2 ± 0)	0.0272 ± 0.0059	∼18
	$q = 2$	2.23 ± 0.57 (2 ± 1)	0.0278 ± 0.0058	∼18

Numerical values for HBF–GP parameters are: $\varepsilon_{\max} = 1$, $\varepsilon_{\min} = 0.1$, and $\gamma = 0.99$. The EKF parameters for both setups are: initial state uncertainty $p_0 = 0.9$, $\mathbf{P}_0 = p_0\mathbf{I_P}$, state transition uncertainty $q_0 = 0.001$, $\mathbf{Q} = q_0\mathbf{I_Q}$, and measurement uncertainty $r_0 = 0.1$, $\mathbf{R} = r_0\mathbf{I_R}$, where $\mathbf{I_P}$, $\mathbf{I_Q}$, and $\mathbf{I_R}$ are the identity matrices of appropriate dimensions. Initial estimate of neuron's spread is adopted as $\mathbf{\Sigma}_j = \kappa \cdot \mathbf{I}$, $\kappa = 0.4$. For GGAP–RBF we follow seminal paper RBF (Huang et al., 2005) and set: $\varepsilon_{\max} = 0.1$, $\varepsilon_{\min} = 0.01$, $\kappa = 0.87$, and $\gamma = 0.99$. Both neural networks are initialized with one neuron $J_0 = 1$. Table 8 presents the experimental results of HBF–GP (both 1-norm and 2-norm) performance and GGAP–RBF (1-norm and 2-norm) performance after 30 repetitions of training process.

Table 8 reveals that HBF–GP outperforms both 1-norm and 2-norm GGAP–RBF in terms of generalization and compactness of the network. Compared to the GGAP–RBF, the HBF–GP needs less number of parameters to memorize the relationship between input and output. The root mean square error calculated for testing set is smaller for HBF–GP (both 1-norm and 2-norm) than for GGAP–RBF. Fig. 6 shows the evolution of number of neurons during training process while output of the HBF–GP after training compared to the desired output is given in Fig. 7.

As experimental results given in Fig. 6 and Table 8 indicate, HBF–GP outperforms GGAP–RBF in terms of accuracy and compactness of generated neural network architecture.

7. HBF–GP for real data regression

The final study of HBF–GP performance analysis is carried out using well-known datasets in Machine Learning community (Asuncion & Newman, 2010; Guvenir & Uysal, 2000). These datasets are carefully chosen and HBF–GP performance on real

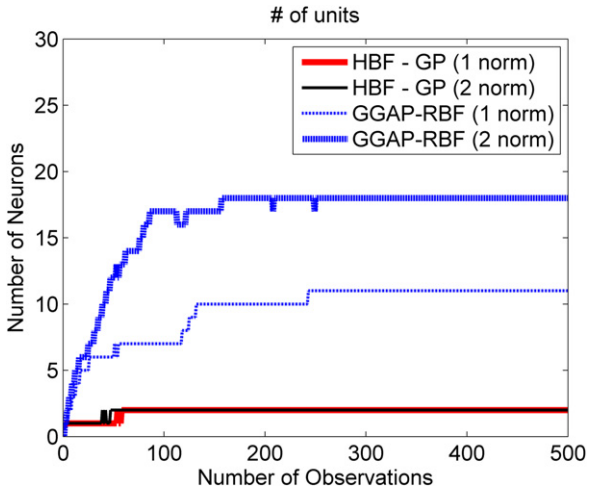

Fig. 6. Evolution of number of neurons during learning process for MacKey–Glass time series prediction. Both 1-norm and 2-norm HBF–GP generate more compact neural network with less number of processing units than 1-norm and 2-norm GGAP–RBF. Results are averaged over 30 experimental trials.

world datasets is compared to the results of its predecessors: GGAP–RBF (Huang et al., 2005) and GGAP–GMM (Bortman & Aladjem, 2009). Our implementation of GGAP–GMM produces slightly worst results than the original reported, therefore, to give fair comparison, we will compare HBF–GP results to those reported in Bortman and Aladjem (2009) for both GGAP–RBF and GGAP–GMM. Following (Bortman & Aladjem, 2009), parameters for GGAP–RBF and GGAP–GMM are: $\varepsilon_{\max} = 1.15$, $\varepsilon_{\min} = 0.04$, $\kappa = 0.1$, and $\gamma = 0.99$. The parameters for HBF–GP are chosen as: $\varepsilon_{\max} = 10$, $\varepsilon_{\min} = 2$, and $\gamma = 0.999$. Two different cases have been tested to assess the performance of HBF–GP for real data regression problems. Initial weighting matrix $\mathbf{\Sigma}_j$ is chosen as follows:

(1) all dimensions of weighting matrix are identical: $\mathbf{\Sigma}_j = \kappa\,\mathbf{I}$, where κ is some scaling parameter provided by user (for example $\kappa = 1/\sigma^2$);
(2) initial values are different along all dimensions of the weighting matrix $\mathbf{\Sigma}_j$:

$$\begin{aligned} &\mathbf{\Sigma}_j^l = \{\max(\mathbf{x}_l) - \min(\mathbf{x}_l)\}_{t=1}^{N_{\text{train}}}, \quad l = 1, \ldots, n_x; \\ &\mathbf{\Sigma}_j = \kappa \cdot \operatorname{diag}\left(\mathbf{\Sigma}_j^1, \ldots, \mathbf{\Sigma}_j^{n_x}\right) \end{aligned} \tag{45}$$

for some arbitrary κ. This way of initialization is similar to initialization of weighting matrix $\mathbf{\Sigma}$ given in Eq. (40).

The first case of weighting matrix initialization does not emphasize any of the dimensions of the input vector $\mathbf{x}$; all dimensions have equal initial weight. On the other hand, in the second case different scale is enabled so as to provide additional information to the HBF network. Both cases are tested for three values of scaling parameter κ: (i) $\kappa = 0.5$, (ii) $\kappa = 1$, and (iii) $\kappa = 2$.

The EKF parameters are: initial state uncertainty $p_0 = 1$, $\mathbf{P}_0 = p_0\mathbf{I_P}$, state transition uncertainty $q_0 = 0.01$, $\mathbf{Q} = q_0\mathbf{I_Q}$, and measurement uncertainty $r_0 = 1$, $\mathbf{R} = r_0\mathbf{I_R}$, where $\mathbf{I_P}$, $\mathbf{I_Q}$, and $\mathbf{I_R}$ are the identity matrices of appropriate dimensions. Prior to training, all data are normalized into [0, 1] range, that is $0 \le \mathbf{x}_l \le 1$, $l = 1, \ldots, \dim(\mathbf{x})$. Following (Bortman & Aladjem, 2009), available examples from data sets have been divided randomly fifteen times into training and testing sets. 5% of training data is used to estimate input density $p(\mathbf{x})$ with GMM. Table 9 summarizes results of HBF–GP training.

The HBF–GP outperforms both GGAP–RBF and GGAP–GMM in terms of accuracy and compactness of generated network. Experimental results (Table 9) show that HBF–GP is able to achieve better

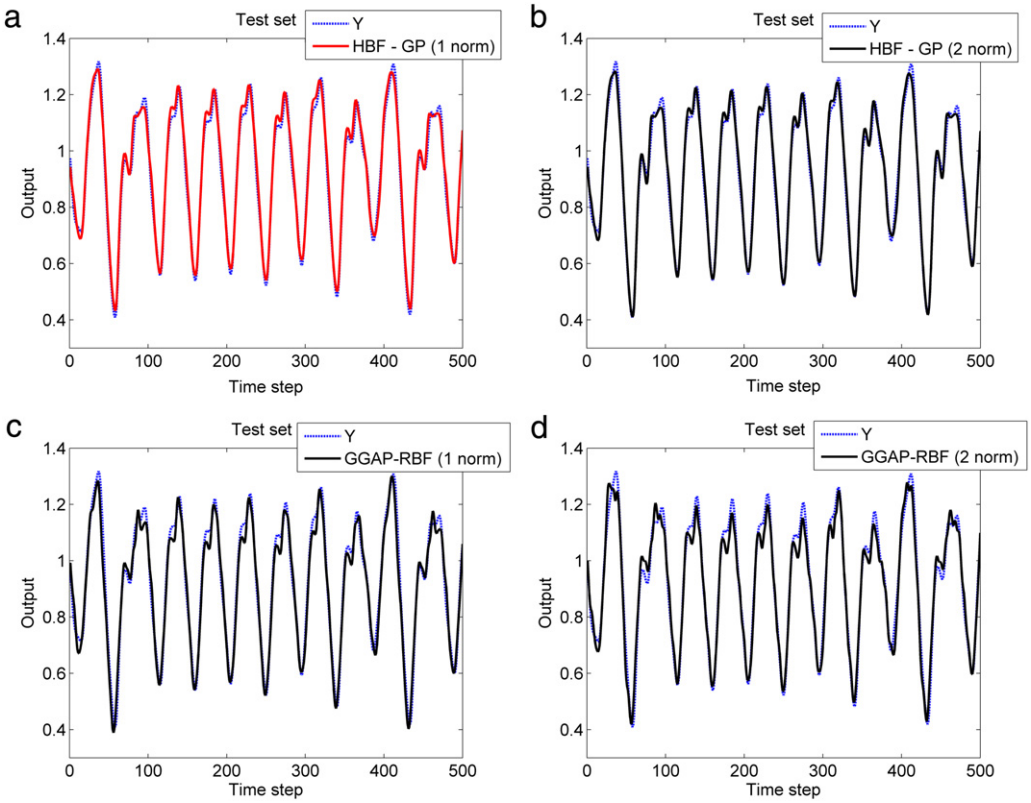

Fig. 7. Approximated MacKey–Glass time series—test output versus generated output of neural networks after learning: (a) 1-norm HBF–GP, (b) 2-norm HBF–GP, (c) 1-norm GGAP–RBF and (d) 2-norm GGAP–RBF. Networks were trained with observations up to $t = 500$, while testing is performed from $t = 501$ to $t = 1001$.

generalization (for dataset Weather Ankara) or at least the same level of generalization (Abalone dataset and Auto-mpg) as its predecessors GGAP–RBF and GGAP–GMM, but with smaller number of neurons. As Table 9 reveals, final number of parameters is significantly smaller for HBF–GP neural network. These conclusions are in congruence with initial intuition for building this type of neural network: the generated network should have smaller number of neurons, hence the smaller number of parameters, while local scaling of input dimensions exploits relationship between variables to improve generalization.

It is interesting to notice that HBF–GP performs better when weighting matrix $\mathbf{\Sigma}_j$ is initialized in accordance with Eq. (45); as it can be seen in Table 9. Different scaling along input dimensions allows HBF neural network to flexibly adapt to data and to extract additional information. For Weather Ankara and Abalone age prediction the best results are obtained when weighting matrix $\mathbf{\Sigma}_j$ is initialized in this manner. On the other hand, experimental results indicate that for Auto-mpg dataset the best results are obtained when weighting matrix $\mathbf{\Sigma}_j$ is initialized as $\mathbf{\Sigma}_j = \kappa\,\mathbf{I}$. We believe this comes as a result of dataset itself; Weather Ankara and Abalone age prediction show periodicity, which is in contrast to Auto-mpg dataset.

As experimental results given in Table 9 show, the scaling parameter κ influences HBF–GP performance; increase in scaling parameter κ induces decrease in generalization ability of HBF–GP. This can be observed especially if weighting matrix $\mathbf{\Sigma}_j$ is initialized in accordance with Eq. (45). In a similar manner, if weighting matrix $\mathbf{\Sigma}_j$ is initialized as $\mathbf{\Sigma}_j = \kappa\,\mathbf{I}$, increased scaling parameter κ results in a lower accuracy of HBF–GP. Obtained experimental results indicate that optimal manner of initialization of the weighting matrix $\mathbf{\Sigma}_j$ and numerical value of scaling parameter κ for problem being modeled and analyzed should be determined empirically in trial and error procedure.

8. Discussion

Experimental studies conducted on simulated engineering problems confirmed that HBF–GP needs less parameters to memorize nonlinear mapping between input and output. For example, as shown in Section 6.2 (Table 7), the number of neurons generated by HBF–GP averaged over 30 independent trials is lower than GAP–RBF and especially MRAN algorithms. Furthermore, the total number of parameters of the HBF–GP is less than or at least equal to the number of parameters needed for GAP–RBF networks; on the other hand, HBF–GP greatly outperforms MRAN in terms of needed number of neurons (and consequently parameters). HBF–GP shows better generalization ability than both GAP–RBF and MRAN, since the calculated root mean square error over test set is smaller for HBF–GP. Similarly, the Section 6.3 shows performance of HBF–GP when highly nonlinear and stochastic function is to be approximated. HBF–GP shows good performance, outperforming GGAP–RBF in terms of number of neurons (and parameters) and in terms of generalization ability. The needed number of parameters for HBF network is less than the number of parameters generated by the GGAP–RBF.

These two studies show that HBF–GP outperforms well studied and established neural networks and learning algorithms when a nonlinear dynamical system is to be modeled. In the first conducted study (Section 6.2), the HBF–GP shows better performance when long input delays of the system are present. In the second study (Section 6.3), the HBF–GP generates more compact network structure and achieves better generalization when dealing with highly nonlinear and chaotic system. This finding is in congruence with basic intuition behind the construction of HBF neuron and generally HBF network: to allow local scaling of the data. This feature of HBF is especially useful when dealing with nonlinear dynamical system having input dimensions with different scale.

Table 9
Test RMSE and final number of processing units for GGAP–RBF, GGAP–GMM and HBF–GP neural networks on real world datasets averaged over 15 repetitions of experiment.

Dataset	Algorithm				Number of units J/(appr. # of parameters)		RMS_{test}
Abalone age prediction	1. GGAP–RBF (Bortman & Aladjem, 2009)				8.2 ± 1.32	74	0.0910 ± 0.0024
	2. GGAP–GMM (Bortman & Aladjem, 2009)				5.13 ± 0.83	46	0.0850 ± 0.0027
	3. HBF–GP	Case #1 Initial $\mathbf{\Sigma}$ is: $\mathbf{\Sigma} = \kappa\,\mathbf{I}$	$\kappa = 0.5$	$q = 1$	1.9333 ± 0.2582	33	0.0926 ± 0.0095
				$q = 2$	1.6667 ± 0.4880	28	0.0868 ± 0.0031
			$\kappa = 1$	$q = 1$	1.8667 ± 0.3519	32	0.0900 ± 0.0047
				$q = 2$	2 ± 0	34	0.0944 ± 0.0124
			$\kappa = 2$	$q = 1$	1.7333 ± 0.4577	29	0.0913 ± 0.0061
				$q = 2$	1.6 ± 0.5071	27	0.0910 ± 0.0054
		Case #2 Initial $\mathbf{\Sigma}$ has different scales	$\kappa = 0.5$	$q = 1$	1.9333 ± 0.4577	33	0.0886 ± 0.0032
				$q = 2$	2 ± 0	34	0.0891 ± 0.0073
			$\kappa = 1$	$q = 1$	1.7333 ± 0.457	29	0.0922 ± 0.0104
				$q = 2$	1.8 ± 0.414	30	$\mathbf{0.0881 \pm 0.0030}$
			$\kappa = 2$	$q = 1$	1.8667 ± 0.3519	32	0.0897 ± 0.0053
				$q = 2$	1.9333 ± 0.2582	33	0.0952 ± 0.0140
Weather Ankara	1. GGAP–RBF (Bortman & Aladjem, 2009)				9.53 ± 1.85	104	0.0610 ± 0.0097
	2. GGAP–GMM (Bortman & Aladjem, 2009)				5.2 ± 1.86	60	0.0611 ± 0.0075
	3. HBF–GP	Case #1 Initial $\mathbf{\Sigma}$ is: $\mathbf{\Sigma} = \kappa\,\mathbf{I}$	$\kappa = 0.5$	$q = 1$	1 ± 0	19	0.0234 ± 0.0016
				$q = 2$	1 ± 0	19	0.0230 ± 0.0010
			$\kappa = 1$	$q = 1$	1 ± 0	19	0.0259 ± 0.0016
				$q = 2$	1 ± 0	19	0.0256 ± 0.0017
			$\kappa = 2$	$q = 1$	1 ± 0	19	0.0299 ± 0.0022
				$q = 2$	1 ± 0	19	0.0294 ± 0.0016
		Case #2 Initial $\mathbf{\Sigma}$ has different scales	$\kappa = 0.5$	$q = 1$	1 ± 0	19	0.0234 ± 0.0011
				$q = 2$	1 ± 0	19	0.0238 ± 0.0017
			$\kappa = 1$	$q = 1$	1 ± 0	19	$\mathbf{0.0219 \pm 0.0009}$
				$q = 2$	1 ± 0	19	0.0225 ± 0.0012
			$\kappa = 2$	$q = 1$	1 ± 0	19	0.0250 ± 0.0023
				$q = 2$	1 ± 0	19	0.0250 ± 0.0028
Auto-mpg	1. GGAP–RBF (Bortman & Aladjem, 2009)				2.8 ± 0.77	25	0.1162 ± 0.0156
	2. GGAP–GMM (Bortman & Aladjem, 2009)				3.6 ± 0.74	32	0.1167 ± 0.0134
	3. HBF–GP	Case #1 Initial $\mathbf{\Sigma}$is: $\mathbf{\Sigma} = \kappa\,\mathbf{I}$	$\kappa = 0.5$	$q = 1$	1 ± 0	15	0.1194 ± 0.0122
				$q = 2$	1 ± 0	15	0.1162 ± 0.0091
			$\kappa = 1$	$q = 1$	1 ± 0	15	0.1147 ± 0.0145
				$q = 2$	1 ± 0	15	$\mathbf{0.1094 \pm 0.0071}$
			$\kappa = 2$	$q = 1$	1 ± 0	15	0.1154 ± 0.0099
				$q = 2$	1 ± 0	15	0.1156 ± 0.0089
		Case #2 Initial $\mathbf{\Sigma}$ has different scales	$\kappa = 0.5$	$q = 1$	1 ± 0	15	0.1138 ± 0.0093
				$q = 2$	1.0667 ± 0.2582	16	0.1174 ± 0.0216
			$\kappa = 1$	$q = 1$	1 ± 0	15	0.1154 ± 0.0071
				$q = 2$	1.0667 ± 0.2582	16	0.1372 ± 0.0716
			$\kappa = 2$	$q = 1$	1 ± 0	15	0.1273 ± 0.0325
				$q = 2$	1.1333 ± 0.3519	17	0.1338 ± 0.0358

The obtained experimental results presented in this paper prove effectiveness and accuracy of the proposed method. HBF–GP performance for real data regression shows that developed neural network generates same level of accuracy but with smaller number of parameters. In these two terms, HBF–GP outperforms both GGAP–RBF and GGAP–GMM. Less number of neurons and better generalization ability than its predecessors are important for HBF neural network, especially from an engineering perspective, where fast models of complex problems are being built using neural networks.

Yet another observation of HBF–GP performance should be stressed. 1-norm HBF–GP needs less neurons than 2-norm HBF–GP. In other words, 1-norm HBF–GP seems to be less enthusiastic in terms of need for neurons. This finding is in congruence with results reported in Huang et al. (2005). It is interesting to notice that generalization of Gaussian type of basis function (the hyper basis function) and modeling of input density with GMM have not changed the main concept of neuron's significance.

Experiments revealed limitation of the proposed concept as well. The limitation is directly linked to the parameters ε_{max}, ε_{min}, and κ; these parameters are problem specific, and one needs to find their optimal values. None of the values for ε_{max} and ε_{min} stated in the paper for analyzed problems (Sections 6.1, 6.2, 6.3, and Section 7) are optimal; these values have been determined in the experimental process. ε_{max} and ε_{min} directly influence the speed of growing and pruning of the neurons; if values of ε_{max} and ε_{min} are not optimal, HBF–GP neural network may over-fit the data, situation which occurs when developed sequential learning algorithm keeps adding HBF neurons (number of neurons is larger than needed) or under-fit the data (smaller number of neurons than needed). This situation is resolved during experimental process, by observing generalization property of the generated HBF neural network and assessing its performance. Numerical values for ε_{max}, ε_{min} and scaling parameter κ are problem specific and are not meant to be optimal for wider set of problems; optimal bounds for these parameters are to be defined in next stages of research.

In our experiments we noticed that it takes a great number of trial and error attempts to find optimal values of parameters for MRAN, GAP–RBF and GGAP–RBF. This is especially noticeable

when MRAN is employed since the number of free parameters is higher than in other algorithms. For example, on average, it took us more than 120 min to find optimal values of parameters; this feature of MRAN is verified in Huang et al. (2005). On the other hand, it takes fewer steps for HBF–GP to converge to the optimal network structure; in this sense it is more user friendly than MRAN. Furthermore, in contrast to GAP–RBF and GGAP–RBF, modeling of input density with Gaussian mixture is even more simplified and speeded up when the approach which simultaneously estimates parameters of mixture and number of mixture components (Figueiredo & Jain, 2002) is employed.

9. Conclusion

The hyper basis function (HBF) is multi-scale Gaussian type of activation function, which provides local scaling of input dimensions. This feature is important, especially for modeling of nonlinear dynamical systems, because single-scale Gaussian basis function cannot include all pieces of information; some input dimensions inevitably get neglected due to single scale property. Motivated by this observation, this paper presented a growing and pruning sequential learning algorithm for hyper basis function feedforward neural network (HBF–GP) for function approximation.

Developed sequential learning algorithm exploits the concept of neuron's significance for growing and pruning of neurons during learning process, defined as contribution of neuron averaged over all processed training examples so far (Huang et al., 2004, 2005). This contribution is weighted with input density of the data, modeled with Gaussian mixture model; this approach enables modeling of possible very complex input distributions, and results in calculation of neuron's significance based on simple matrix calculations.

Examples using simulated engineering problems and real data sets have clearly shown that hyper basis function neural network, trained with developed sequential learning algorithm with growing and pruning ability, generates compact network topologies with good generalization performance.

Acknowledgments

This work is supported by the Serbian Government—the Ministry of Education, Science and Technological Development—Project title: An innovative, ecologically based approach to the implementation of intelligent manufacturing systems for the production of sheet metal parts (2011–2014) under grant TR35004.

Authors would like to gratefully acknowledge the critical reviews by anonymous reviewers whose effort helped to improve an earlier version of the manuscript.

References

Alexandridis, A., Sarimveis, H., & Bafas, G. (2003). A new algorithm for online structure and parameter adaptation of RBF networks. *Neural Networks*, *16*, 1003–1017.

Asuncion, A., & Newman, D. (2010). UCI machine learning repository. Irvine, CA: University of California, School of Information and Computer Science. [http://archive.ics.uci.edu/ml/] (last date of access: September 10, 2012).

Billings, S. A., Wei, H. L., & Balikhin, M. A. (2007). Generalized multiscale radial basis function networks. *Neural Networks*, *20*(10), 1081–1094.

Bortman, M., & Aladjem, M. (2009). A growing and pruning method for radial basis function networks. *IEEE Transactions on Neural Networks*, *20*(6), 1039–1045.

Crassidis, J. L., & Junkins, J. L. (2011). *Optimal estimation of dynamic systems (2nd edition)*. New York: CRC Press, Taylor and Francis (Chapter 3).

Figueiredo, M. A. T., & Jain, A. K. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(3), 381–396.

Gonzalez, J., Rojas, I., Ortega, J., Pomares, H., Fernandez, F. J., & Diaz, A. F. (2003). Multi-objective evolutionary optimization of the size, shape, and position parameters of radial basis function networks for function approximation. *IEEE Transactions on Neural Networks*, *14*(6), 1478–1495.

Guvenir, H. A., & Uysal, I. (2000). Bilkent university function approximation repository. [http://funapp.cs.bilkent.edu.tr] (last date of access: September 10, 2012).

Han, H. G., Chen, Q. L., & Qiao, J. F. (2011). An efficient self-organizing RBF neural network for water quality prediction. *Neural Networks*, *24*(7), 717–725.

Han, H. G., Qiao, J. F., & Chen, Q. L. (2012). Model predictive control of dissolved oxygen concentration based on a self-organizing RBF neural network. *Control Engineering Practice*, *20*(4), 465–476.

Huang, G. B., Saratchandran, P., & Sundararajan, N. (2004). An efficient sequential learning algorithm for growing and pruning RBF (GAP–RBF) networks. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, *34*(6), 2284–2292.

Huang, G. B., Saratchandran, P., & Sundararajan, N. (2005). A generalized growing and pruning RBF (GGAP–RBF) neural network for function approximation. *IEEE Transactions on Neural Networks*, *16*(1), 57–67.

Huang, G. B., Zhou, H., Ding, X., & Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, *42*(2), 513–529.

Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*, 489–501.

Islam, M. M., Sattar, M. A., Amin, M. F., Yao, X., & Murase, K. (2009). A new adaptive merging and growing algorithm for designing artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, *39*(3), 705–722.

Jazwinski, A. (1970). *Stochastic processes and filtering theory*. New York: Academic Press (Chapter 7).

Kadirkamanathan, V., & Niranjan, M. (1993). A function estimation approach to sequential learning with neural networks. *Neural Computation*, *5*, 954–975.

Karayiannis, N. B. (1999). Reformulated radial basis neural networks trained by gradient descent. *IEEE Transactions on Neural Networks*, *10*(3), 657–671.

Lazaro, M., Santamari, I., & Pantaleo, C. (2003). A new EM-based training algorithm for RBF networks. *Neural Networks*, *16*, 69–77.

Lu, Y., Sundararajan, N., & Saratchandran, P. (1997). A sequental learning scheme for function approximation using minimal radial basis function (RBF) neural networks. *Neural Computation*, *9*, 461–478.

Lu, Y., Sundararajan, N., & Saratchandran, P. (1998). Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm. *IEEE Transactions on Neural Networks*, *9*(2), 308–318.

Mahdi, R. N., & Rouchka, E. C. (2011). Reduced hyperBF networks: regularization by explicit complexity reduction and scaled rprop based training. *IEEE Transactions on Neural Networks*, *22*(5), 673–686.

Miljković, Z., & Aleksendrić, D. (2009). Artificial neural networks—solved examples with theoretical background (in Serbian). University of Belgrade—Faculty of Mechanical Engineering, Belgrade.

Nishida, K., Yamauchi, K., & Omori, T. (2006). An online learning algorithm with dimension selection using minimal hyper basis function networks. *Systems and Computers in Japan*, *37*(11), 11–21.

Orr, M., Hallam, J., Takezawa, K., Murray, A., Ninomiya, S., Oide, M., et al. (2000). Combining regression trees and radial basis function networks. *International Journal of Neural Systems*, *10*(6), 453–465.

Platt, J. (1991). A resource allocating network for function interpolation. *Neural Computation*, *3*, 213–225.

Poggio, T., & Girosi, F. (1989). A theory of networks for approximation and learning. In *A. I. Memo 1140*. MIT.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, *1990*, 1481–1497.

Qiao, J. F., & Han, H. G. (2012). Identification and modeling of nonlinear dynamical systems using a novel self-organizing RBF-based approach. *Automatica*, *48*(8), 1729–1734.

Salmeron, M., Ortega, J., Puntonet, C. G., & Prieto, A. (2001). Improved RAN sequential prediction using orthogonal techniques. *Neurocomputing*, *41*, 153–172.

Schwenker, F., Kestler, H. A., & Palm, G. (2001). Three learning phases for radial-basis-function networks. *Neural Networks*, *14*(4–5), 439–458.

Simon, D. (2001). Training radial basis function neural networks with the extended Kalman filter. *Neurocomputing*, *48*, 455–475.

Todorović, B., Stanković, M. S., & Moraga, C. (2002). Extended Kalman filter trained recurrent radial basis function network in nonlinear system identification. In *ICANN(2002)* (pp. 819–824).

Vuković, N. (2012). Machine learning of intelligent mobile robot based on artificial neural networks. *Ph.D. dissertation* (in Serbian). University of Belgrade—Faculty of Mechanical Engineering.

Wu, Y., Wang, H., Zhang, B., & Du, K.-L. (2012). Using radial basis function networks for function approximation and classification. *ISRN Applied Mathematics*, http://dx.doi.org/10.5402/2012/324194. Article ID 324194.

Zhang, R., Huang, G. B., Sundararajan, N., & Saratchandran, P. (2007). Improved GAP–RBF network for classification problems. *Neurocomputing*, *70*(16–18), 3011–3018.