Faculty of Mathematics and Physics, Charles University, Prague
Institute of Computer Science, Academy of Sciences of the Czech Republic

Learning with Regularization Networks

Petra Kudová

PhD Thesis Summary

I–1 Theoretical Computer Science

Prague 2006

Matematicko-fyzikální fakulta, Univerzita Karlova, Praha
Ústav informatiky, Akademie věd České Republiky

Learning with Regularization Networks

Petra Kudová

Autoreferát dizertační práce

I–1 Teoretická informatika

Praha, 2006

Dizertační práce byla vypracována v rámci doktorského studia uchazeče na Matematicko--fyzikální fakultě Univerzity Karlovy v Praze (MFF UK) v letech 2001–2006.

Uchazeč	RNDr. Petra Kudová
Školitel	Mgr. Roman Neruda, CSc.
Školicí pracoviště	Ústav informatiky Akademie věd České Republiky Pod Vodárenskou věží 2 182 07 Praha 8
Oponenti	
Předseda OR I–1	Prof. RNDr. Petr Štěpánek, DrSc..

Autoreferát byl rozeslán dne ....

Obhajoba dizertační práce se koná dne .... ve .... hodin v budově MFF UK, ... S dizertační prací je možno se seznámit na studijním oddělení doktorského studia MFF UK, Ke Karlovu 3, Praha 2.

Contents

1 Introduction

In the last few years, machine learning has witnessed an increase of interest, which is a consequence of rapid development of the information industry and its need for an "intelligent" data analysis.

A learning problem can be described as finding a general rule that explains data given only by a sample of limited size. In addition, collected data may contain measurement errors or noise. Efficient algorithms are required to filter out the noise and capture the true underlying trend.

In this thesis we will deal with *supervised learning*. In such a case we are given pairs of input objects (typically vectors), and desired outputs. The output can be of continuous value, or it can predict a class label of the input object. The task of supervised learning is to predict the output value for any valid input object after having seen a number of examples, i.e. input-output pairs. To achieve this, a "reasonable" generalization from the presented data to unseen situations is needed. Such a problem appears in a wide range of application areas, covering various approximation, classification, and prediction tasks.

Artificial neural networks (ANNs) represent one of the approaches that are applicable to the learning problem. Though their original motivation was to model neural systems of living organisms, neural networks are successfully used in such diverse fields as modeling, time series analysis, pattern recognition, signal processing, and control.

The primary property of ANNs is their ability to learn from their environment and to improve their performance through learning. There is a good supply of network architectures and corresponding supervised learning algorithms (e.g. [6]). The model, that is, a particular type of neural network, is usually chosen in advance and its parameters are tuned during learning so as to fit the given data. The difficulties that might occur during the learning process include slow convergence, getting stuck in local optima, *over-fitting* etc.

Over-fitting typically occurs in cases where learning was performed for too long or where training examples are rare. Then the network may learn very specific random features of the training data that have completely no relation to the underlying function. In this process the performance on the training examples still increases while the performance on unseen data becomes worse.

The problem of over-fitting can be cured by regularization theory. The regularization theory is a rigorous approach that formulates the learning problem as a function approximation problem. Given a set of examples obtained by random sampling of some real function, possibly in the presence of noise, the goal of learning is to find this function or its estimate. Since this problem is generally ill-posed, some a priori knowledge about the function should be added. Usually it is assumed that the function is *smooth*, where the smoothness is understood in a very general sense. Often it means that two similar inputs correspond to two similar outputs and/or that the function does not oscillate too much. The solution is found by minimizing the functional containing both the data and stabilizer, i.e. the term representing our a priori knowledge.

It was shown that for a wide class of stabilizers the solution can be expressed in a form of feed-forward neural network with one hidden layer, called a *regularization network*. Different types of stabilizers lead to different types of activation functions called *kernel functions* in the hidden layer.

The regularization network as a solution derived from the regularization theory has as many units in the hidden layer as the number of training examples was. Such a solution is unfeasible for bigger data sets, therefore the class of *generalized regularization networks* was introduced.

Radial basis function networks (RBF networks) represent a subclass of generalized regularization networks. They belong among more recent types of neural networks. In contrast to classical models (multilayer perceptrons, etc.), the RBF network is a network with local units, the proposal of which was motivated by presence of many local response units in human brain. Other motivation came from numerical mathematics, where radial basis functions were first introduced in the solution of real multivariate problems. It was shown that RBF networks possess the universal approximation property.

Based on a good theoretical background, the classes of regularization networks and RBF networks represent promising approaches to learning and deserve further investigations. Though the theoretical knowledge is very beneficial, it does not cover all aspects of their practical applicability. Experimental study of corresponding learning algorithms may justify the theoretical results and give an idea of real complexity and efficiency of the algorithms. Both good theoretical and good empirical knowledge are the best starting point for successful applications, further improvements of the existing algorithms, or creating new learning approaches.

2 Goals and Objectives

The main goal of the work is to study and develop learning algorithms for networks based on the regularization theory. In particular, learning possibilities for a family of regularization networks and RBF networks should be studied. The available approaches, including gradient techniques, genetic algorithms, and linear optimizations methods, should be investigated and potential improvements suggested. Based on the obtained theoretical and experimental results, new algorithms should be proposed, possibly as hybrid methods combining the existing algorithms. All the algorithms should be implemented and their behavior studied experimentally.

The goal of the work will be achieved by means of the following objectives:

- **Study of regularization network learning and its properties**

  The regularization theory leads to a solution of the regularized learning problem in the form of regularization network having as many hidden units as the number of data points is. Such a solution leads to quite a straightforward learning algorithm, since the centers of hidden units are fixed to the given data points and the output weights are estimated as a solution of linear optimization problem. Despite its seeming simplicity, problems may occur due to round-off errors, numerical unstability etc. Therefore the real applicability and behavior of the algorithm should be studied.

  In addition, the regularization network learning algorithm requires knowledge of the regularization parameter and kernel function. To set up these parameters, good knowledge of the role of regularization parameter and kernel function in the learning is needed. The impact of regularization parameter and kernel function choice on the performance of the learning algorithm should be studied and different types of kernel functions compared.

- **Design of autonomous learning algorithm for regularization network**

  The regularization parameter and kernel function are parameters that represent our prior knowledge of the given problem. In fact, they are a part of the learning problem definition, and therefore assumed to be known in the theory. However, this is not always true in practice. In most applications such knowledge is not available and a desired learning algorithm should be able to estimate these parameters itself. The framework above the basic learning algorithm should be created to establish a fully autonomous learning procedure.

- **Study and design of learning algorithms for generalized regularization networks**

  A regularization network as an exact solution of the regularized learning problem has as many kernel functions as the number of data points is. This fact makes its learning quite straightforward and simple, but limits its practical applicability. Since such a solution is unfeasible for larger data sets, generalized regularization networks were introduced.

  Different learning approaches for the generalized regularization networks should be studied, with focus on RBF networks. The RBF networks already possess a wide range of learning possibilities, including gradient techniques, combinations of clustering and linear optimization, and genetic algorithms. The main concern of further research is to create hybrid approaches.

- **Performance comparison of regularization network and RBF network**

  Unlike the regularization network, the RBF network typically has a much smaller number of basis functions than the size of the data set is. The basis functions are usually distributed using various heuristics, so that the resulting network may be far from the optimal solution. On the other hand, the solution is usually much cheaper in terms of space complexity.

  The comparison of learning performance of regularization networks and RBF networks might throw new light on the difference between the "exact" solution and the "approximate" solution. Optimally, if this difference is reasonable, the RBF networks might represent a cheaper alternative to the regularization networks.

3 Methods and algorithms

3.1 Regularization Networks

Regularization networks [5, 14] belong to a family of feed-forward neural networks with one hidden layer. They are based on the regularization theory, where the problem of supervised learning is handled as an function approximation problem.

Definition 3.1 *(Learning from Examples) We are given a set of examples (pairs)*

$$\{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^N \tag{1}$$

that was obtained by random sampling of a real function f, generally in presence of noise. The problem of recovering the function f from data, or finding the best estimate of it, is called learning from examples *(or* supervised learning*). The set of input-output pairs (1) is called the* training set*.*

The problem is generally ill-posed [14, 7]. We are interested only in such solutions that possess the *generalization ability*, i.e. functions that give relevant outputs also for the data not included in the training set. It is usually assumed that the function is *smooth*, two similar inputs correspond to two similar outputs, or the function does not oscillate too much.

Based on this assumption, the solution is stabilized by means of an auxiliary non-negative functional that embeds prior information about the solution. Then the solution is sought as a minimum of the functional:

$$H[f] = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 + \gamma \Phi[f], \tag{2}$$

where Φ is called a *stabilizer* or *regularization term* and $\gamma > 0$ is *the regularization parameter* controlling the trade-off between closeness to the data and degree of satisfaction of the desired property of the solution.

The functional (2) was shown to have a unique solution for a wide class of stabilizers. The solution has a form:

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i K(\mathbf{x}, \mathbf{x}_i), \tag{3}$$

where N is the number of training samples, $\mathbf{x}_i \in \mathbb{R}^d$ are the training samples, $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a positive semi-definite function, called a *kernel function*, $w_i \in \mathbb{R}$ are coefficients, called *weights*. Different stabilizers lead to different types of kernel functions. In fact, the choice of kernel function is equivalent to the choice of the stabilizer, i.e. to the choice of our prior assumption. Solution derivation can be found in [5, 4, 14, 16], and others.

The regularized learning problem solution (3) can be represented by a feed-forward neural network with one hidden layer. Such a network is called a *regularization network*. Its learning algorithm is sketched in Algorithm 3.1.

Definition 3.2 *(Regularization Network) A* regularization network *is a feed-forward neural network with one hidden layer of kernel units and one linear output unit. It represents a function*

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i K(\mathbf{x_i}, \mathbf{c_i}), \tag{4}$$

where N is the number of hidden neurons (i.e. the number of basis functions), $w_i \in \mathbb{R}$, $\mathbf{c}_i \in \mathbb{R}^d, \mathbf{x}_i \in \mathbb{R}^d$, $K : \mathbb{R}^d \to \mathbb{R}$ is a chosen kernel (basis) function. To coefficients of the linear combination w_i we refer as to weights*, the vectors $\mathbf{c}_i$ are called* centers*.*

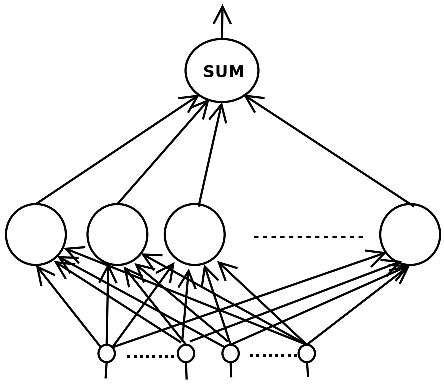


Figure 1: Regularization network scheme.

Input: Data set $\{\mathbf{x}_i, y_i\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}$
Output: Regularization Network

1. Set the centers of kernels:
   $\forall i \in \{1, \ldots, N\} : \mathbf{c}_i \leftarrow \mathbf{x}_i$
2. Compute the values of weights $w_1, \ldots, w_N$:
$$(\mathbf{K} + \gamma \mathbf{I})\mathbf{w} = \mathbf{y}, \tag{5}$$
   where $\mathbf{I}$ is the identity matrix, $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, and $\mathbf{y} = (y_1, \ldots, y_N)$, $\gamma > 0$.

Algorithm 3.1. The RN learning algorithm.

3.2 Product and Sum Kernels

The kernel function used in a particular application of regularization network is typically supposed to be given in advance, for instance chosen by the user. As a kernel function we can use any symmetric, positive semi-definite function, possibly a conditionally positive semi-definite function.

Since different kernel functions correspond to different prior assumptions, the choice of kernel function always depends on the particular task. However, we often have to deal with heterogeneous data, in the sense that different attributes differ in type or quality, or that the character of data differs in different parts of the input space. Then for the different parts of the data different kernel functions are suitable.

In such situations, kernel functions created as a combination of simpler kernel functions might better reflect the character of the data. We can benefit from the fact that the set of positive semi-definite functions is closed with respect to several operations, such as sum, product, difference, limits, etc. [1, 15]. Based on the operations of product and sum we introduce two types of composite kernel units, namely a *product kernel* and a *sum kernel*.

Product Kernel

Definition 3.3 *(Product Kernel) Let $K_1, \ldots, K_k$ be kernel functions defined on $\Omega_1, \ldots, \Omega_k$ ($\Omega_i \subset \mathbb{R}^{d_i}$), respectively. Let $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$. The kernel function K defined on Ω that satisfies*

$$K(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k) = K_1(\mathbf{x}_1, \mathbf{y}_1)K_2(\mathbf{x}_2, \mathbf{y}_2) \cdots K_k(\mathbf{x}_k, \mathbf{y}_k), \tag{6}$$

where $\mathbf{x}_i, \mathbf{y}_i \in \Omega_i$, is called a product kernel.

A computational unit that realizes the product kernel function will be called a *product unit* (see Figure 2). A regularization network with the hidden layer formed by product kernels is called a *product kernel regularization network* (PKRN).

Product kernels might be useful if a priori knowledge of data suggests looking for the solution as a member of a product of two or more function spaces. This is typically in a

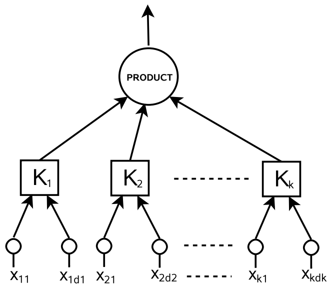


Figure 2: A unit realizing a product kernel.

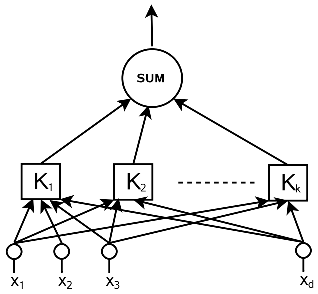


Figure 3: A unit realizing a sum kernel.

situation when the individual attributes, or groups of attributes, differ in type or quality. In such situations, we can split the attributes into groups, which means that instead of one input vector $\mathbf{x} \in \mathbb{R}^d$ we will deal with k input vectors $\mathbf{x}_i \in \mathbb{R}^{d_i}$, for $i = 1, \ldots, k$. Then the training set has the form

$$\{(\mathbf{x}_1^i, \mathbf{x}_2^i, \ldots, \mathbf{x}_k^i, y^i) \in \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \times \ldots \times \mathbb{R}^{d_k} \times \mathbb{R}\}_{i=1}^N. \tag{7}$$

Using a product kernel on such training data enables us to process different $\mathbf{x}_i$ separately.

Sum Kernel

Definition 3.4 *(Sum Kernel) The kernel function $K : \Omega \times \Omega \to \mathbb{R}$ that can be obtained as a sum of two or more other kernel functions $K_1, \ldots, K_k$*

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k} K_i(\mathbf{x}, \mathbf{y}), \tag{8}$$

where $\mathbf{x}, \mathbf{y} \in \Omega$, is called a sum kernel.

A computational unit realizing the sum kernel is shown in Figure 3. We call it a *sum unit*. A regularization network that has sum units in its hidden layer we call a *sum kernel regularization network* (SKRN).

The sum kernel is intended for use in cases when prior knowledge or analysis of data suggests looking for a solution as a sum of two or more functions. For example, when the data is generated from a function influenced by two sources of different frequencies, we can use a kernel obtained as a sum of two parts corresponding to high and low frequencies.

Restricted Sum Kernel

Approximation of data with different distributions in different parts of the input space may be done with the help of a *restricted sum kernel*.

Definition 3.5 *(Restricted Sum Kernel) Let $K : \Omega \times \Omega \to \mathbb{R}$ be a kernel function and let A be a subset of Ω. Then the kernel K_A defined by*

$$K_A(\mathbf{x}, \mathbf{y}) = \begin{cases} K(\mathbf{x}, \mathbf{y}) & \textit{if } \mathbf{x}, \mathbf{y} \in A, \\ 0 & \textit{otherwise;} \end{cases} \tag{9}$$

is called a restricted sum kernel.

In situations when different kernels are suitable for different parts of the input space, we can divide the input space into several subsets $A_1, \ldots, A_k$ and choose different kernels K_i for each A_i.

Then we obtain the kernel as a sum of kernels K_i restricted to the corresponding sets:

$$K(\mathbf{x}, \mathbf{y}) = \begin{cases} K_i(\mathbf{x}, \mathbf{y}) & \text{if } \mathbf{x}, \mathbf{y} \in A_i \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Divide et Impera

The second application of restricted sum kernels is a derivation of the *Divide et Impera* approach that represents a technique for dealing with bigger data sets.

Note that an SKRN with restricted sum kernels represents a function

$$f(\mathbf{x}) = \sum_{\mathbf{x}_i \in A_1} w_i K_1(\mathbf{x}, \mathbf{x}_i) + \ldots + \sum_{\mathbf{x}_i \in A_k} w_i K_k(\mathbf{x}, \mathbf{x}_i), \tag{11}$$

which can be also interpreted as a sum of k regularization networks, each using its own kernel function $K_s, s = 1, \ldots, k$.

For the case of disjoint sets A_s, always exactly one member of (11) is nonzero. Thus for an input $\mathbf{x}$ only the regularization network corresponding to the set A_s, for which $\mathbf{x} \in A_s$, has the nonzero output. Conversely, to determine the value of w_i, only the training samples $\{(\mathbf{x}_j, y_j) | \mathbf{x}_j \in A_s\}$ are needed.

So the partitioning of the input space defines the partitioning of the training set into k subsets. The weights of the SKRN with restricted sum kernels can be determined by solving k smaller linear systems instead of a big one.

Replacing one linear system by k smaller ones reduces both the space and time requirements of learning. The drawback of this approach is a slightly bigger approximation error that is caused by the lack of information on the borders of the input space areas, i.e. sets A_i.

3.3 Autonomous Learning Framework

The RN learning algorithm (Algorithm 3.1) is based on the assumption that the regularization parameter γ and the kernel function K are given in advance. We call these parameters *metaparameters* to distinguish them from the parameters of the network itself (i.e. weights, centers).

Ideally, the metaparameters are selected by the user based on their knowledge of the given problem. Since this is not possible or very difficult in majority of practical applications, we need to build a framework above this algorithm to make it capable of finding not only the network parameters but also optimal metaparameters.

We propose the following procedure:

1. Setup of the algorithm

    (a) Choice of a type of the kernel function: By the type we mean that we decide whether to use a Gaussian, multi-quadratic, sum, product or another kernel function (For sum and product kernels it is necessary to determine the type for all kernels used in the sum and product respectively.).

(b) Choice of the additional parameters of the kernel function: Some kernels have additional parameters that have to be estimated (such as the width in the case of the Gaussian function).

(c) Choice of the regularization parameter γ.

2. Running the RN learning algorithm (Algorithm 3.1).

Typically, we suppose that the type of kernel function — Step 1(a) — is given by the user. The Step 1(b) is performed as a search for the metaparameters with the minimal cross-validation error.

Clearly, it is not possible to go through all possible metaparameter values. Therefore we create a grid of couples $[\gamma, p]$ (where p is a kernel parameter) using a suitable sampling and evaluate the cross-validation error for each point of this grid. The point with the lowest cross-validation error is picked.

To speed up the process, we proposed the *adaptive grid search* algorithm (see [8, Section 4.5]). It starts with a coarse grid, i.e. sparse sampling, and then creates a finer grid around the point with the minimum.

The disadvantage of this approach is the high number of evaluations of the Algorithm 3.1 needed during the search. Nevertheless, these evaluations are completely independent, so they can be performed in parallel.

In addition, we proposed a more sophisticated search algorithm, called *genetic parameter search* (see [8, Section 4.6]). It applies genetic algorithms to search for the optimal metaparameters. It can be simply extended to search also for the kernel function type.

3.4 RBF Networks

An RBF network is a feed-forward neural network with one hidden layer of RBF units and a linear output layer (see Fig. 4). By the RBF unit we mean a neuron with d real inputs and one real output, realizing a radial basis function φ (such as the Gaussian function). Instead of the most commonly used Euclidean norm, we use the *weighted norm* $\|\cdot\|_C$, where $\|\mathbf{x}\|^2_C = (C\mathbf{x})^T(C\mathbf{x}) = \mathbf{x}^T C^T C\mathbf{x}$.

Definition 3.6 *(RBF Network) An* RBF network *is a 3-layer feed-forward network with the first layer consisting of d input units, a hidden layer consisting of h RBF units and an output layer of m linear units. Thus, the network computes the following function:* $\mathbf{f} = (f_1, \dots, f_s, \dots, f_m) : \mathbb{R}^d \to \mathbb{R}^m$:

$$f_s(\mathbf{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\| \mathbf{x} - \mathbf{c}_j \|_{\mathbf{C}_j}}{b_j}\right), \tag{12}$$

where $w_{ji} \in \mathbb{R}$, φ is a radial basis function, $\mathbf{c}_j \in \mathbb{R}^d$ are called centers*, $b \in \mathbb{R}$ are called* widths*, and $\mathbf{C}_j$ are the weighted norm matices.*

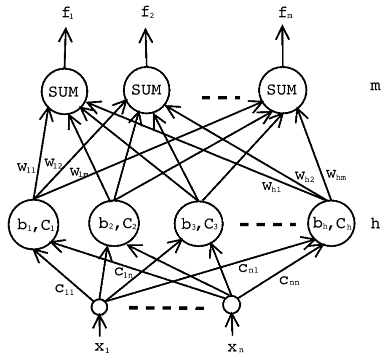


Figure 4: An RBF network.

From the regularization framework point of view, the RBF network belongs to the family of *generalized regularization networks*. The generalized regularization networks are RNs with lower number of kernels than data points and also it is not necessary that the kernels are uniform.

The goal of RBF network learning is to find the parameters (i.e. centers $\mathbf{c}$, widths b, norm matrices C and weights w) so that the network function approximates the function given by the training set $\{(\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}^n \times \mathbb{R}^m\}_{i=1}^N$. There is a variety of algorithms for RBF network learning, in our previous work we studied their behavior and possibilities of their combinations [11, 12]. The three most significant approaches are the *gradient learning*, *three-step learning*, and *genetic learning*.

Input: Data set $S = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}^m$
Output: Network parameters:
$\mathbf{c}_k, b_k, \Sigma_k^{-1}, w_{ks}, s = 1, \cdots, m$ and $k = 1, \cdots, h$

1. $\tau := 0$
   Setup randomly $\mathbf{c}_k(0)$, $b_k(0)$, $\Sigma_k^{-1}(0)$, $w_{ks}(0)$ and $\Delta\mathbf{c}_k(0)$, $\Delta b_k(0)$, $\Delta\Sigma_k^{-1}(0)$, $\Delta w_{ks}(0)$ for $s = 1, \cdots, m$ and $k = 1, \cdots, h$
2. $\tau := \tau + 1$
3. Evaluate: $s = 1, \cdots, m$ and $k = 1, \cdots, h$

$$\begin{aligned} \Delta\mathbf{c}_k(\tau) &= -\epsilon\frac{\partial E}{\partial \mathbf{c}_k} + \alpha\Delta\mathbf{c}_k(\tau-1) & \Delta b_k(\tau) &= -\epsilon\frac{\partial E}{\partial b_k} + \alpha\Delta b_k(\tau-1) \\ \Delta\Sigma_k^{-1}(\tau) &= -\epsilon\frac{\partial E}{\partial \Sigma_k^{-1}} + \alpha\Delta\Sigma_k^{-1}(\tau-1) & \Delta w_{ks}(\tau) &= -\epsilon\frac{\partial E}{\partial w_{ks}} + \alpha\Delta w_{ks}(\tau-1), \end{aligned}$$

   $\epsilon \in (0,1)$ is the learning rate, $\alpha \in \langle 0,1\rangle$ is the momentum coefficient.
4. Change the values of parameters: $s = 1, \cdots, m$ and $k = 1, \cdots, h$

$$\begin{aligned} \mathbf{c}_k(\tau) &= \mathbf{c}_k(\tau-1) + \Delta\mathbf{c}_k & b_k(\tau) &= b_k(\tau-1) + \Delta b_k \\ \Sigma_k^{-1}(\tau) &= \Sigma_k^{-1}(\tau-1) + \Delta\Sigma_k^{-1} & w_{ks}(\tau) &= w_{ks}(\tau-1) + \Delta w_{ks} \end{aligned}$$

5. Evaluate the error of the network.
6. If the stop criterion is not satisfied, go to 2.

Algorithm 3.2. Gradient learning.

Gradient Learning

The most straightforward approach to RBF network learning is based on the well-known back-propagation algorithm for the multilayer perceptrons (MLPs). The back-propagation learning is a non-linear gradient descent algorithm that modifies all network parameters proportionally to the partial derivative of the training error. The trick is in clever ordering of the parameters so that all partial derivatives can be computed consequently.

Since the RBF network has a structure formally similar to the MLP, it can be trained by the modification of the back-propagation algorithm. Unlike the MLP, the RBF network has always only one hidden layer, so evaluating its derivatives is rather simple.

The gradient learning algorithm is sketched in Algorithm 3.2. It uses a gradient descent enhanced with a momentum term [13] for the stepwise parameter modifications. Since it depends on the random initialization and may suffer from local minima, it is recommended to try several different initializations and pick the best solution.

Three-Step Learning

The gradient learning described in the previous section unifies all parameters by treating them in the same way. The *three-step learning*, on the contrary, takes advantage of the well-defined meaning of RBF network parameters.

The learning process is divided into three consequent steps corresponding to the three distinct sets of network parameters. The first step consists of determining the hidden unit centers, in the second step the additional hidden unit parameters (widths, weighted norm matrices) are estimated. During the third step the output weights are determined. The algorithm is listed in Algorithm 3.3.

Input: `Data set` $S = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}^m$
Output: `Network parameters:`
$\mathbf{c}_k, b_k, \Sigma_k^{-1}, w_{ks}, s = 1, \cdots, m$ `and` $k = 1, \cdots, h$

1. `Determine the centers` $\mathbf{c}_i, i = 1, \ldots, h$ `using a vector quantization method`
2. `Set up widths` b_i `and matrices` Σ_i^{-1} `for` $i = 1, \ldots, h$ `by minimization of`

$$E(b_1 \cdots b_h; \Sigma_1^{-1} \cdots \Sigma_h^{-1}) = \frac{1}{2} \sum_{r=1}^{h} \left[\sum_{s=1}^{h} \varphi(\xi_{sr}) \xi_{sr}^2 - P \right]^2 \qquad \xi_{sr} = \frac{\| \mathbf{c}_s - \mathbf{c}_r \|_{\mathbf{C}_r}}{b_r}, \quad (13)$$

`where` P `is the overlap parameter.`

3. `Compute the values for` w_{js} `for` $j = 1, \ldots, h$ `and` $s = 1, \ldots, m$ `by`

$$\mathbf{W} = \mathbf{P}^{+}\mathbf{Y}, \quad (14)$$

`where the matrix` $\mathbf{P}$ `is a` $d \times h$ `matrix of outputs of RBF units and` $\mathbf{P}^{+}$ `is its pseudoinverse,` $\mathbf{W}$ `is a` $d \times m$ `matrix of weights and` $\mathbf{Y}$ `is a` $h \times m$ `matrix of the desired outputs` $\mathbf{y}_i$`.`

Algorithm 3.3. Three-step learning.

Genetic Learning

The third learning method introduced here is based on the genetic algorithms (GAs). It is sketched in Algorithm 3.4. Unlike the traditional GA approaches, we use a direct float encoding for the RBF network parameters. An individual is formed by a sequence of blocks, where each block contains a vector of one RBF unit parameter values.

A selection operator is used to choose individuals to a new population. Each individual is associated with the value of the error function of the corresponding network. Selection is a stochastic procedure, in which the individual probability of being chosen to the new population is the higher, the smaller the error function of the corresponding network is.

The crossover operator composes a pair of new individuals combining parts of two old individuals. The positive effect of the crossover is the creation of new solutions recombining the current individuals. Finally, the mutation operator represents small local random changes of an individual.

The GAs represent a robust mechanism that usually does not suffer from the local extremes problem. The price for this robustness is a bigger time complexity, especially for problems with bigger individuals resulting in a huge search space.

Input: `Data set` $S = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{N} \subseteq \mathbb{R}^d \times \mathbb{R}^m$
Output: `Network parameters:`
$\mathbf{c}_k, b_k, \Sigma_k^{-1}, w_{ks}, s = 1, \cdots, m$ `and` $k = 1, \cdots, h$

1. Create random population of N individuals $P_0 = \{I_1, \cdots, I_N\}$.
   $i \leftarrow 0$
2. For each individual compute the error on the training set.
3. If the minimal error in the population is small enough, stop.
4. Create an empty population P_{i+1} and while the population has less than N individuals repeat:
   - Selection: Select two individuals from P_i.
     $I_1 \leftarrow selection(P_i)$
     $I_2 \leftarrow selection(P_i)$
   - Crossover: with probability p_{cross}:
     $(I_1, I_2) \leftarrow crossover(I_1, I_2)$
   - Mutation: with probability p_{mutate}:
     $I_k \leftarrow mutate(I_k), k = 1, 2$
   - Insert: insert I_1, I_2 into P_{i+1}
5. Go to 2.

Algorithm 3.4. Genetic learning.

Hybrid Methods

The three described algorithms represent three main branches of the wide range of RBF learning algorithms. Since these learning algorithms have been studied quite well, we believe that the main potential for the further improvements lies in clever combinations rather than further modifications of the available algorithms. Hybrid approaches based on combinations of the well-known algorithms may achieve a synergy effect and thus over--perform the single algorithms.

We proposed two hybrid approaches — the *hybrid genetic learning* and the *four-step learning algorithm*.

The hybrid genetic learning replaces both the first and second step of the three-step learning by the GAs. The third step setting the output weights is performed by a linear optimization technique, see Algorithm 3.5 for the sketch of the error evaluation procedure. There are good reasons for such combinations. The first two steps are based on heuristics so the use of the GAs is appropriate for them. On the other hand, the determination of output weights is a linear optimization task, for which many efficient algorithms exist.

Input: Individual I, data set T
Output: Error associated with I

1. Create the RBF network f represented by the individual I
2. Run the least squares method to set the weights of f
3. Compute the error of network f on the data set T

Algorithm 3.5. Error evaluation in hybrid genetic learning.

Input: Data set $S = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}^m$
Output: Network parameters: $\mathbf{c}_k, b_k, \Sigma_k^{-1}, w_{ks}, s = 1, \cdots, m$ and $k = 1, \cdots, h$

1. Run the Algorithm 3.3.
2. Run the Algorithm 3.2, but instead of the random initialization use the result of step 1.

Algorithm 3.6. Four-step learning.

The four-step learning — Algorithm 3.6 — is based on the three-step learning followed by the gradient learning. The result of the three-step learning is used as an initial value for the gradient learning that further tunes the values of all parameters.

4 Experimental Results

The main goals of our experiments can be summarized as following:

1. demonstrate the behavior of regularization networks;
2. study the role of regularization parameter and kernel function;
3. compare different types of kernel functions;
4. demonstrate the behavior of our product kernels and sum kernels and compare them to the classical solutions;

5. demonstrate the behavior of RBF networks as the representatives of generalized regularization networks;

6. compare the regularization networks and RBF networks in order to find out the difference between an "exact solution" and an "approximate solution".

In order to achieve high comparability of our results, we have chosen frequently used tasks for the experiments with learning algorithms. As benchmark tasks we use the data sets from the PROBEN1 repository, the artificial task *two spirals* and the well-known image of *Lenna*. In addition, the task of flow rate prediction was picked to represent real-life problems.

In all our experiments we work with distinct data sets for training and testing, referred to as the *training set* and the *test set*. The learning algorithm is run on the training set, including the possible cross-validation. The test set is never used during the learning phase, it is only used for the evaluation of the resulting network error.

For the data set $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N \subset \mathbb{R}^d \times \mathbb{R}^m$ and the network representing a function f, the normalized error is computed as follows:

$$E = 100 \frac{1}{Nm} \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i)||^2, \tag{15}$$

where $|| \cdot ||$ denotes the Euclidean norm. We use the notation E_{train} and E_{test} for the error computed over the training set and test set, respectively.

The experiments have been performed using implementation in system Bang [2], the standard numerical library LAPACK [9] was used to solve linear least square problems (step 2 in Algorithm 3.1). Most experiments were run on the computer clusters *Lomond* [10], *Joyce* and *Blade*. The former cluster is the Sun cluster available in Edinburgh Parallel Computing Centre, University of Edinburgh. The latter two are clusters of workstations with the Linux operating system at the Institute of Computer Science, Academy of Sciences of the Czech Republic. Time requirements listed in following sections refer to an Intel Xeon 2.80 GHz processor.

4.1 Regularization Networks

We demonstrated the role of metaparameters on the tasks from the PROBEN1 repository and on an approximation of the Lenna image — see [8, Subsection 6.3.1] and Figure 5. Both the experiments show that the choice of the regularization parameter and the kernel function is crucial for the successful learning, one cannot choose arbitrary values and a search for optimal metaparameters is necessary. In addition, a wrong choice of the kernel function cannot be cured by any change of the regularization parameter, and might result in a completely useless solution.

The two methods for the metaparameters setup were tested on the tasks from the PROBEN1 repository — see [8, Subsection 6.3.2]. It was shown that the simpler method — adaptive grid search algorithm — is sufficient for this tasks. The more sophisticated genetic parameter search suffers from high time requirements.

Several common kernel functions were compared on the collection of benchmarks PROBEN1 — see [8, Subsection 6.3.3] and Figure 6. In most cases, the lowest error was

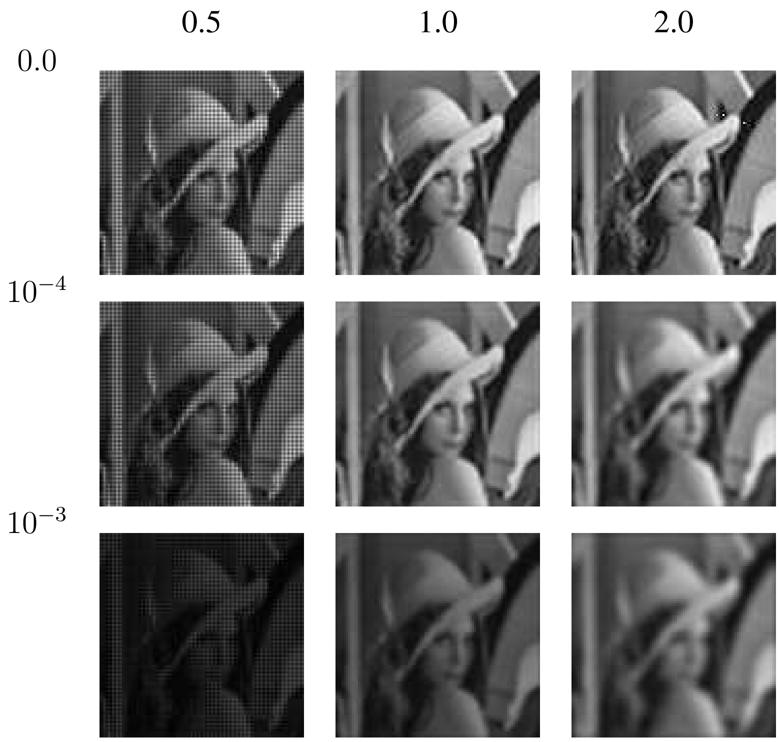


Figure 5: Images generated by the regularization network learned on the Lenna image (50×50 pixels) using Gaussian kernels with the widths from 0.5 to 2.0 and the regularization parameters from 0.0 to 0.001.

achieved by the RN with the inverse multi-quadratic kernel function. For many cases, the Gaussian function achieves the second lowest test error. Both the functions are functions with local response, i.e. they give a relevant output only in the local area around its center. The results justify usage of local functions, including the Gaussian function, and show that the commonly used Gaussian function is a good first choice.

Our product and sum kernels were tested on the tasks from PROBEN1 — see [8, Subsection 6.3.4]. The SKRN achieved the lowest error on 23 tasks, the RN on 13 tasks, and the PKRN on two tasks. However, the errors of all the three networks are comparable. In addition, the SKRN achieved very low test errors on several data sets without the loss of generalization ability. Table 1 shows the the results on a part of PROBEN1 tasks.

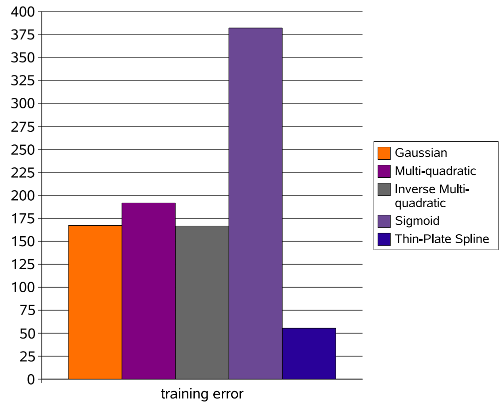
Also the *Divide et Impera* approach was demonstrated — see [8, Subseciton 6.3.6]. It represents an alternative for the bigger data sets that cannot be processed by the simple RN learning algorithm. Dividing into two or three subtasks brings significant reduction of time complexity, while it only slightly increases the result errors.

4.2 RBF Networks

The three studied algorithms for RBF network learning, as well as our hybrid methods, were tested on tasks selected from the PROBEN1 repository.

Table 2 summarizes the results from two experiments with various RBF network learning algorithms.

First, consider the main three approaches, the gradient learning, three-step learning, and



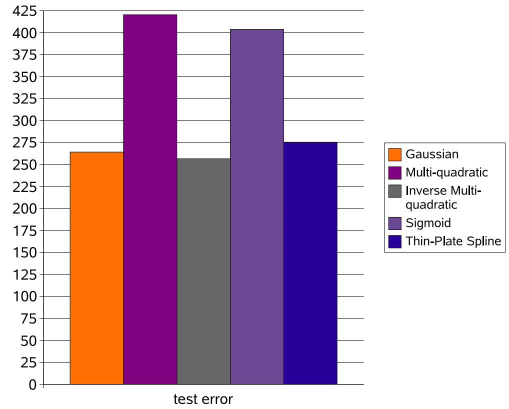


Figure 6: Comparison of overall training error (left) and test error (right) for different kernels.

	RN		**SKRN**		**PKRN**	
Task	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}
cancer1	2.28	**1.75**	0.00	1.77	2.68	1.81
cancer2	1.86	3.01	0.00	**2.96**	2.07	3.61
cancer3	2.11	2.79	0.00	**2.73**	2.28	2.81
card1	8.75	**10.01**	8.81	10.03	8.90	10.05
card2	7.55	**12.53**	0.00	12.54	8.11	12.55
card3	6.52	12.35	6.55	**12.32**	7.01	12.45
flare1	0.36	0.55	0.35	**0.54**	0.36	**0.54**
flare2	0.42	0.28	0.44	**0.26**	0.42	0.28
flare3	0.38	0.35	0.42	**0.33**	0.40	0.35
glass1	3.37	6.99	2.35	**6.15**	2.64	7.31
glass2	4.32	7.93	1.09	**6.97**	2.55	7.46
glass3	3.96	7.25	3.04	**6.29**	3.31	7.26

Table 1: Comparisons of errors on training and test set for the RN with Gaussian kernels, the SKRN, and the PKRN.

genetic learning. The gradient learning is able to achieve better results in terms of error measured on both the training and test set. The three-step learning is the fastest method, due to the unsupervised phase to set the centers, and rather fast linear optimization to set the output weights. The errors achieved are still competitive. The genetic learning is in general slower about an order of magnitude. While most of the measured running times were in the order of seconds and minutes, it takes minutes to hours for the GA to converge to the desired values. The results are still not as good as with the gradient learning. Nevertheless, the GA — as a general learning procedure — has its potential in learning the networks with heterogeneous units; and it is suitable for parallelization.

Second, the table includes the two hybrid methods. The four-step learning further improves the results obtained by the three-step learning. On CANCER it achieves comparable results with the gradient learning. The hybrid genetic learning achieves very good results, slightly better than the gradient learning. However, it suffers from high time requirements.

Used learning method	Cancer (5 units) E_{train}	E_{test}	Time h:m:s	Glass (15 units) E_{train}	E_{test}	Time m:s
Gradient	2.19	2.76	00:00:28	3.25	7.13	13:41
Three-step	3.67	3.57	00:00:01	7.50	9.90	00:17
Four-step	2.20	2.55	00:00:36	7.04	9.55	03:32
Genetic	4.69	4.60	07:24:16	–	–	–
Hybrid genetic	2.09	2.75	02:30:31	–	–	–

Table 2: Comparison of learning methods on the cancer data set for network with 5 hidden units and on the glass data set for the network with 15 hidden units. Average training and test error.

	RN		**RBF**			**MLP**		
	E_{test}	# units	E_{test}		# units	E_{test}		arch.
			mean	std		mean	std	
cancer1	**1.76**	525	2.11	0.01	15	1.60	0.41	4+2
cancer2	**3.01**	525	3.12	0.07	15	3.40	0.33	8+4
cancer3	**2.80**	525	3.19	0.13	15	2.57	0.24	16+8
card1	**10.00**	518	10.16	0.56	10	10.53	0.57	32+0
card2	**12.53**	518	12.81	0.01	10	15.47	0.75	24+0
card3	12.32	518	**12.09**	0.00	10	13.03	0.50	16+8
flare1	0.54	800	**0.37**	0.00	10	0.74	0.80	32+0
flare2	**0.27**	800	0.31	0.00	10	0.41	0.47	32+0
flare3	**0.34**	800	0.38	0.00	10	0.37	0.01	24+0
glass1	6.95	161	**6.76**	0.02	20	9.75	0.41	16+8
glass2	**7.91**	161	7.96	0.00	20	10.27	0.40	16+8
glass3	**7.33**	161	8.06	0.97	20	10.91	0.48	16+8

Table 3: Comparison of E_{test} of RN, RBF, and MLP. For RBF and MLP the mean and standard deviation from 10 repetitions of the runs are listed (RN learning is deterministic). The numbers of neurons in the first and second hidden layer are listed for the MLP.

4.3 Regularization Networks vs. RBF Networks

The main aim of the experimental part of this work was to assess the relative performance of the RNs and RBF networks. The PROBEN1 repository was used to perform the comparison.

The regularization networks have been trained with the RN learning algorithm (Algorithm 3.1) with the metaparameters setup done by the adaptive grid search. The RBF networks have been trained by the gradient learning.

Table 3 compares the results obtained by the RNs and RBF networks by means of the test error. In addition, the results are related to the performance of the MLP.

In terms of the test error, the regularization networks achieved the best results on 23 tasks; the RBF networks on 8 tasks (see Table 3 for a part of the results). Both the training and testing errors are quite comparable, the difference is in average about 6%. In addition, the RBF networks need a 10 to 50 times lower number of hidden units to obtain comparable approximation and generalization performance. The time requirements needed to achieve the listed errors varied from 1 to 30 minutes depending on the size of the particular data set, and were similar for both the regularization networks and RBF networks.

The regularization networks, in their exact form, are therefore suitable rather for the tasks with smaller data sets, where is a high danger of over-fitting. For the tasks possessing large data amounts, “cheaper” alternatives represented by the generalized regularization networks, such as the RBF networks, are more competent.

To show that the RNs and RBF networks represent competitive learning methods not only to the MLP, but also to modern learning algorithms, we picked the comparison to the support vector machine (SVM). The comparison was made on the classification tasks CANCER and GLASS. The SVM was trained using the available library [3], which represents a current standard of SVM learning.

Table 4 compares the RN, RBF network, and SVM in terms of classification accuracy on the test set. The results obtained by the three methods are comparable, the differences in accuracy are not high. We see that both the regularization networks and RBF networks are vital alternatives to the SVM.

	RN	**RBF**	**SVM**
cancer1	**98.85%**	98.74%	97.12%
cancer2	95.40%	**96.84%**	96.55%
cancer3	95.98%	**96.95%**	95.97%
glass1	**75.00%**	72.45%	73.58%
glass2	**73.07%**	64.53%	66.03%
glass3	76.92%	72.26%	**79.24%**

Table 4: Comparison of classification accuracy of RN, RBF and support vector machines (SVM).

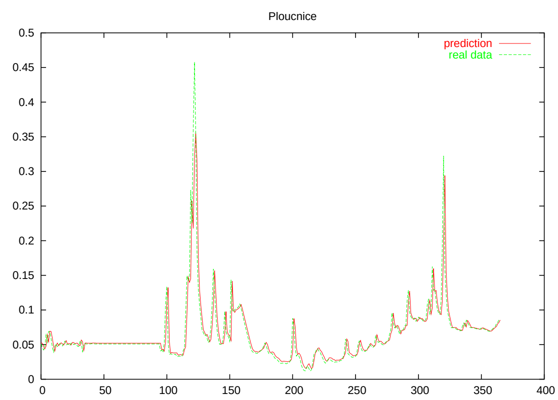


Figure 7: Prediction of flow rate by the regularization network.

4.4 Rainfall-Runoff Modeling

Both the RBF network and the RN were applied to rainfall-runoff modeling, i.e. modeling of river-flow rates based on daily flow and rainfall values. The research is realized in cooperation with University of J. E. Purkyně and the Czech Hydrometeorological Institute in Ústí nad Labem. The Ploučnice River in North Bohemia has been chosen as an experimental catchment to calibrate and evaluate the models.

See Figure 7 for an example of flow rate prediction by the RN. It has been shown that both the RBF networks and regularization networks can be successfully used for creating small rainfall-runoff models. These models can be built from historical time series data, without knowing anything about the physics of the process.

5 Conclusion

The main goal of our work was to study the possible ways of learning based on the regularization theory. Learning algorithms, including the RN learning algorithm derived directly from the theory, and various learning algorithms for RBF networks were investigated.

The RN learning algorithm (Algorithm 3.1) represents an incomplete tool for learning, since it requires a nontrivial setup of metaparameters. It was shown in the experiments that these metaparameters, the regularization parameter and the kernel function, significantly influence the quality of the solution (Subsection 4.1). Therefore a framework above the basic RN learning algorithm was created, including the estimation of the metaparameters. Two techniques for this setup were introduced — the adaptive grid search and the genetic parameter search (Subsection 3.3).

Since the choice of the kernel function plays a crucial role in learning, we decided that it deserves more attention. It resulted in proposing the composite types of kernel functions — a product kernel and sum kernel (Subsection 3.2). In the experiments (Subsection 4.1) we showed that they are a vital alternative to the classical (i.e. simple) kernels. They are especially useful on tasks that are heterogenous, either varying in attributes or different parts of the input space. Good behavior was observed while experimenting with the sum kernels. The setup phase adjusted the widths of the two Gaussians addends, so that one Gaussian was very narrow and the other one wide. Such a kernel function obtained good results even without the regularization term. Almost zero training errors were achieved, while the generalization property was preserved. Such kernel functions may be very useful for tasks with a low level of noise. Inspired by the concept of restricted sum kernels, we proposed the "Divide et Impera" approach. It is a simple procedure that splits the tasks into several disjoint subtasks. The learning algorithm is applied on each of these subtasks, possibly in parallel. The solution is then obtained as a sum of networks obtained. Such an approach does not only save the space, but also significantly reduces the time requirements.

Despite the good theoretical background, the regularization network may be not feasible in some situations. Particularly, the solution is too large for tasks with huge data sets. Therefore the notion of generalized regularization networks was introduced. We focused on one concrete subclass — RBF networks. The RBF networks benefit from a wide range of learning possibilities. Three main approaches were described (Subsection 3.4) and compared in the experiments (Subsection 4.2). The best results, in terms of the training and test error, were obtained by the gradient learning. The three-step learning, on the other hand, represented the fastest approach, while the resulting errors were still competitive. The genetic learning was significantly slower, and still it does not outperform the other methods. Inspired by these results, the two hybrid approaches were proposed — the four-step learning (Algorithm 3.6) and the hybrid genetic learning (Algorithm 3.5). Their behavior was demonstrated experimentally (Subsection 4.2) and it was shown that they, in some aspects, improve the original algorithms. The four-step learning adds a gradient phase after the three-step learning. The first part formed by the three-step learning saves time, while the second gradient part further improves the solution. The hybrid genetic learning represents a combination of the genetic learning and the third part of the three-step learning. Only the hidden layer is estimated by the GAs, the output weights are found by linear optimization. Such an approach achieved very good results, outperforming the other approaches; however, it suffers from very high time requirements.

When studying the learning from the point of view of both the regularization networks and RBF networks, the comparison of both the approaches is inevitable. In our experiments, the regularization networks and RBF networks achieved comparable results. So we claim that the RBF networks represent a cheaper alternative to the regularization networks. Finally, we presented an application of the studied algorithms to a real-life problem. Both the regularization networks and RBF networks were successfully applied on the prediction of the river-flow rate (Subsection 4.4).

References

[1] N. Aronszajn. Theory of reproducing kernels. *Transactions of the AMS*, 68:337–404, 1950.

[2] Project Bang, `http://bang.sf.net/`.

[3] Ch. Chih-Chung and L. Chi-Jen. Libsvm: a library for support vector machines, 2002. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

[4] F. Girosi. An equivalence between sparse approximation and support vector machines. Technical report, Massachutesetts Institute of Technology, 1997. A.I. Memo No. 1606.

[5] F. Girosi, M. Jones, and T. Poggio. Regularization theory and Neural Networks architectures. *Neural Computation*, 2:219–269, 7 1995.

[6] S. Haykin. *Neural Networks: a comprehensive foundation*. Tom Robins, 2nd edition, 1999.

[7] V. Kůrková. Learning from data as an inverse problem. In Antoch J., editor, *Computational Statistics*, pages 1377–1384. Heidelberg, Physica Verlag, 2004.

[8] P. Kudová. *Learning with regularization networks*. PhD thesis, MFF UK, 2006.

[9] LAPACK. Linear algebra package, `http://www.netlib.org/lapack/`.

[10] Lomond machine. Introduction to the university of edinburgh HPC service, `http://www.epcc.ed.ac.uk/computing/services/sun/documents/hpc-intro/hpc_introdoc.pdf`.

[11] R. Neruda and P. Kudová. Hybrid learning of RBF networks. *Neural Network World*, 12(6):573–585, 2002.

[12] R. Neruda and P. Kudová. Learning methods for radial basis functions networks. *Future Generation Computer Systems*, 21:1131–1142, 2005.

[13] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, 1986.

[14] T. Poggio and S. Smale. The mathematics of learning: Dealing with data. *Notices of the AMS*, 50:536–544, 5 2003.

[15] B. Schoelkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, Massachusetts, 2002.

[16] T. Šidlofová. Existence and uniqueness of minimization problems with fourier based stabilizers. In *Proceedings of Compstat, Prague*, 2004.