FACULTY OF MATHEMATICS AND PHYSICS
CHARLES UNIVERSITY, PRAGUE
&
INSTITUTE OF COMPTURE SCIENCE
ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

Learning with Regularization Networks

Petra Kudová

PH.D. THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Prague 2006

ADVISOR:
Mgr. Roman Neruda, CSc.

Abstract

In this work we study and develop learning algorithms for networks based on regularization theory. In particular, we focus on learning possibilities for a family of regularization networks and radial basis function networks (RBF networks). The framework above the basic algorithm derived from theory is designed. It includes an estimation of a regularization parameter and a kernel function by minimization of cross-validation error.

Two composite types of kernel functions are proposed — a sum kernel and a product kernel — in order to deal with heterogenous or large data.

Three learning approaches for the RBF networks — the gradient learning, three-step learning, and genetic learning — are discussed. Based on these, two hybrid approaches are proposed — the four-step learning and the hybrid genetic learning.

All learning algorithms for the regularization networks and the RBF networks are studied experimentally and thoroughly compared.

We claim that the regularization networks and the RBF networks are comparable in terms of generalization error, but they differ with respect to their model complexity. The regularization network approach usually leads to solutions with higher number of base units, thus, the RBF networks can be used as a ’cheaper’ alternative in terms of model size and learning time.

Acknowledgments

During my PhD study I was lucky to meet many people who did not hesitate to give me professional and personal help. I would hardly have finished this thesis if it had not been for them.

Roman Neruda, the advisor of my thesis, initiated this research and in numerous discussions provided me with many valuable pieces of advice. I would like to thank him for his great support and motivation.

Not less I am obliged to my colleagues from the Institute of Computer Science, Academy of Sciences of the Czech Republic. Namely, to Věra Kůrková, the head of our department, who gave me inspiring ideas, references to literature and papers related to the topic, and also read a part of the manuscript and suggested improvements. To Zdeněk Strakoš, who provided me with literature and discussions on numerical topics. To Petr Savický, who encouraged me with his lively interest in my work. To Terka Šámalová, who deserves thanks for proof-reading of the first chapters, discussions on mathematical issues, and cooperation. And finally, to Pavel Krušina, who had to react to my various complaints on the Bang system.

I am also grateful for the opportunity to spend three months at the Edinburgh Parallel Computing Centre (EPCC), University of Edinburgh, where I met many interesting people. Among them, Ben Paechter hosted me and enabled my visit. Catherine Inglis and Adam Carter, members of EPCC staff, provided me with friendly atmosphere and technical support. My friend Marek Kubiak, also a visitor, often spent his time in our discussions on genetic algorithms, as well as everyday troubles of PhD student's life.

Let me also thank to Martin Vidner for initial language proof-reading, and to Radka Šmahelová for supervising my English.

Not only the companionship of great people and scientists, but also financial support was an essential point. My work was supported by the Grant Agency of the Czech Republic under the grant no. 201/05/0557, by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Appplications", and partially by the European Commission's Research Infrastructures activity of the Stucturing the European Research Area program, contract no. RII3-CT-2003-506079 (HPC-Europa).

Very special thanks to Žabžulka for supplying me with chocolate and love.

Contents

Symbol Index

$\mathbb{N}$	natural numbers (non-negative integers)
$\mathbb{R}$	real numbers
$\mathbf{I}$	identity matrix
$\mathbf{K}$	kernel matrix, $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
N	number of data points
d	dimension of the input space
m	dimension of the output space
h	number of hidden units
b	width of the Gaussian function
$\mathbf{c}$	center (of kernel, RBF function)
γ	regularization parameter
$K(\cdot), G(\cdot)$	kernel (basis) function

List of Figures

List of Tables

Chapter 1

Introduction

It is a very sad thing that nowadays
there is so little useless information.
Oscar Wilde

1.1 Motivation

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 contain 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 able to handle the learning problem. The neural network research dates back to the early 1950s, when McCulloch and Pitts defined a formal model of neuron. Although their original motivation was to model neural systems of living organisms, neural networks are applicable 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 learning algorithms (e.g. [15]). The model, that is, a particular type of neural network, is usually chosen in advance and its parameters are tuned during learning in order 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 an estimate of it. 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 term and the 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 the form of feed-forward neural network with one hidden layer, called *regularization network*. Different types of stabilizers lead to different types of activation functions, i.e. *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 the 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 well established 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.

1.2 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 optimization 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 results in 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 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 are considered to be known in the theoretical assumptions. However, this is seldom 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 the creation of 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 over the input space 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, yet viable, 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.

1.3 Structure of the Work

Let us describe the structure of this work in order to help its reader navigate throughout the book. In general, the organization of this thesis reflects the course of development of our work. It starts with chapters explaining the underlying theory, continues with description of studied algorithms, and ends with an extensive chapter describing experimental results.

Chapter 2 presents the theory underlying the regularization networks. The whole chapter is a compilation of the results published in [12, 59, 73, 64], and others. It starts with the definition and formalization of the learning problem. Then derivation of regularization network is presented, both using the stabilizers based on the Fourier transform and Reproducing Kernel Hilbert Spaces (RKHSs). The latter derivation slightly differs from the one published in [59]. Our modification concerns scaling the data term of the minimized functional in order to avoid numerical unstability, but does not cause any important changes in derivation or solution. Finally, the regularization network is formally defined.

Though the chapter does not present any novel work, it is relevant for the further chapters and deeper understanding of the problems tackled in them. The definitions of learning problem and regularization networks will be used in the rest of the work and therefore should not be skipped.

In Chapter 3 we propose new solutions to the learning problem — *product kernel regularization network* and *sum kernel regularization network*. Aronszajn's derivation of RKHS product and RKHS sum [1] is presented and used to propose composite types of kernel functions — *product kernel* and *sum kernel*. The simple "Divide et Impera" approach is proposed to deal with bigger tasks. The whole chapter is based on the joint work with T. Šámalová [36, 35].

Chapter 4 is devoted to learning by using regularization networks. The basic regularization network learning algorithm is presented, as it results directly from the theory. The role of this algorithm parameters, i.e. the regularization parameter and kernel function, is discussed. The framework above this algorithm is proposed in order to create an autonomous learning procedure. Based on cross-validation, we introduce two methods for the regularization parameter and kernel function setup — the adaptive grid search and the genetic parameter search.

Chapter 5 deals with the learning algorithms for generalized regularization networks, particularly RBF networks. First, the notion of generalized regularization networks (as proposed in [12, 58]) is introduced, and RBF networks are described. Then, the three main approaches for RBF networks learning are presented — they are the gradient learning, three-step learning, and genetic learning. These have already been studied in our previous work [21] to some extent. Finally, two hybrid methods are proposed.

Chapter 6 presents the experimental part of the work. Since it is quite extensive, we describe its further structure. It is organized into seven subsections. The first two subsections state our motivation and goals, and describe the data sets and methodology used.

Subsection 6.3 is devoted to experiments with the regularization networks. They demonstrate the role of the regularization parameter and kernel function, test the algorithms for their setup, compare different types of kernel function, study the performance of product and sum kernels, and finally test the "Divide et Impera" approach.

Subsection 6.4 deals with different learning algorithms for the RBF networks. The gradient learning, three-step learning, genetic learning, and the hybrid methods — four-step learning and hybrid genetic learning — are demonstrated and compared on the experiments.

Subsection 6.5 compares the regularization networks and the RBF networks. In Subsection 6.6 the application of regularization networks and RBF networks to the prediction of river flow rate, made in cooperation with the University of J.E.Purkyně and the Czech Hydrometeorological Institute in Ústí nad Labem, is presented. The last subsection brings a summary of the experiments and draws conclusions.

The main results of the work are summarized in the last chapter, where several possible directions for future work are outlined.

1.4 Related Works by the Author

The results presented in this work have been already published in papers presented at both international and local conferences, and published in journals, or technical reports. Publications dealing with the regularization networks and their learning include [32, 31, 30, 29, 25, 26]. The results concerning product and sum kernel regularization networks were published in [36, 35, 71]. The comparison of regularization network and RBF network was studied in [33, 20, 28, 27]. Papers and reports dealing with the various learning approaches for RBF networks include [54, 52, 53, 24, 23, 22]. And finally, the results of application on prediction of river flow rate were presented in [40].

Other papers published by the author, but not directly relating to this work include: [70, 34] dealing with unsupervised learning, and [51, 19] presenting the multi-agent system Bang that was used as a platform for the implementation of the algorithms studied in this work and realization of the experimental part.

Chapter 2

Regularization Networks

A lack of information cannot be remedied
by any mathematical trickery.
Lanczos, 1964

In this chapter we deal with the problem of supervised learning by means of function approximation theory and regularization theory. First, we introduce the problem and formulate it in a formal way as a function approximation problem. In Section 2.2 we describe the main idea of regularization theory and show how the learning problem is handled. In the next two sections we show a derivation of the learning problem solution, a *regularization network* [12]. In Section 2.3 we present the derivation using stabilizers based on the Fourier transform [12]; in Section 2.4 the derivation taking advantage of Reproducing Kernel Hilbert Spaces [59]. In Section 2.5 we formally define the regularization network and describe its architecture.

2.1 Learning from Examples

The problem of *learning from examples* or *supervised learning* is a subject of great interest at present. In various applications, such as classification of handwritten digits, prediction of stock market share values, and weather forecasting, one encounters the problem when they are given a data sample of limited size and a concise description of the data has to be found.

In the case of supervised learning, the data is a sample of input-output patterns (called a *training sample* or *training set*), thus a concise description of the data is typically a function that can produce the output, when given the input. Then the task of learning is to find a deterministic function that maps any input to an output so that the disagreement with the future input-output observations is minimized.

Now we can formalize the problem of learning from examples as a function approximation problem.

Definition 2.1.1 *(Learning from Examples) We are given a set of examples (pairs)*

$$\{(\mathbf{x}_i, y_i) \in R^d \times R\}_{i=1}^N \tag{2.1}$$

that was obtained by random sampling of 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*).*

Definition 2.1.2 *(Training Set) The set of input-output pairs (2.1) is called* training set.

Clearly, whenever we are given an input vector that is present in the training set, we can return the corresponding output, or the output that appeared in the highest number of pairs together with the input vector, in case of duplicities. However, generalization to cases not present in the training set is difficult. In addition, the training set typically contains noise, so the above described procedure fails.

In other words, it is not necessary that the function exactly interpolates all the given data points, but what we need is a function with a good *generalization*, which is a function that gives relevant outputs also for the data not included in the training set.

It is easy to see that the problem is generally ill-posed [59, 18]. The concept of *well-posed* and *ill-posed* problems was introduced by Hadamard [14].

Definition 2.1.3 *(Well-Posed Problem) The mathematical problem is* well-posed *if it has the following properties:*

1. *A solution exists.*
2. *The solution is unique.*
3. *The solution depends continuously on the data, in some reasonable topology.*

Otherwise the problem is called ill-posed.

In case of the learning problem, the first criterion in Definition 2.1.3 may be not met if there is an input appearing together with different outputs in the training set. Still, if the solution exists, there are many functions interpolating the given data points (see Figure 2.1 for illustration). And finally, presence of noise may violate the continuity requirement.

In the rest of this chapter we will deal with the learning problem defined in Definition 2.1.1. Note that the one-dimensional output does not limit the general applicability of learning techniques presented in the following sections.

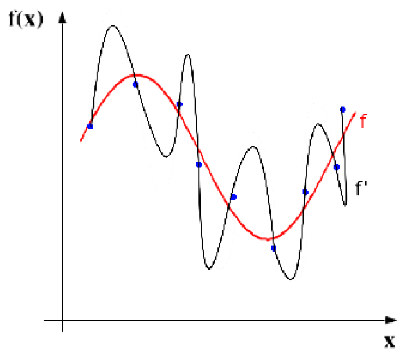

Figure 2.1: The problem of learning from examples: the unknown target function f, one of the wrong solutions f'.

2.2 Regularization Approach

A commonly used method that deals with the problem of learning from examples, defined in the previous section, is *empirical risk* minimization [68, 69]. It works with a space (or generally a set) of functions that are considered to be possible solutions. Such a space is called a *hypothesis space*.

Then, the empirical cost is minimized over the hypothesis space. It means that the function f^* is sought such that

$$f^* = \min_{f \in \mathcal{H}} \sum_{i=1}^{N} V(f(\mathbf{x}_i), y_i), \tag{2.2}$$

where V is a suitable *loss function*. Typically a square of difference is used

$$f^* = \min_{f \in \mathcal{H}} \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2. \tag{2.3}$$

As we have already discussed in the previous section, such a problem is generally ill--posed. However, we are interested only in such solutions that possess the generalization ability. There is no way to solve this problem unless some additional information about the unknown function is available.

Therefore some a priori knowledge about the function has to be considered. If such knowledge is not available, 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 some auxiliary nonnegative functional that embeds prior information about the solution [12]. Then the solution is sought as a function minimizing the functional:

$$H[f] = \sum_{i=1}^{N} V(f(\mathbf{x}_i), y_i) + \gamma \Phi[f], \tag{2.4}$$

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.

Minimizing simultaneously both the data term and the regularization term is the basic principle of *regularization*, widely known as *Tikhonov regularization* [67], — a general approach to ill-posed problems.

The problem of minimizing the functional (2.4) can be shown to have a unique solution for a wide class of stabilizers and loss functions V. Derivation of the solution shape is known as the Representer Theorem. It can be found in [12], [11], where the stabilizers based on the Fourier transform are used. In [59] the solution is derived with the help of Reproducing Kernel Hilbert Spaces (RKHSs). The existence and uniqueness of the solution was also proved in [72].

Now we show two ways of deriving the solution. In the next section we deal with the case of stabilizers based on the Fourier transform, and in Section 2.4 with the derivation using RKHSs.

2.3 Stabilizers Based on the Fourier Transform

In this section we present the derivation of solution for the stabilizers based on the Fourier transform, as it was proposed by Girosi, Jones and Poggio [12].

As we have already mentioned in the previous section, the typical assumption about a function with a good generalization is that the function does not oscillate too much. Therefore one can look for a stabilizer measuring the "oscillatory" behavior of the function.

Such a stabilizer may be constructed with the help of the Fourier transform. The Fourier transform of function produces a spectrum from which the original function can be reconstructed. In other words, it produces the representation of a function in the frequency domain.

If we consider functions in the frequency domain, a function oscillates less than another one if it has less energy in high frequencies. To measure the oscillary behavior of the function, a high pass filter, i.e. a filter passing only high frequencies, is first applied. Then the power (L_2 norm) of the result is taken. So the stabilizer has a form:

$$\Phi[f] = \int_{R^d} d\mathbf{s} \frac{|\tilde{f}(\mathbf{s})|^2}{\tilde{G}(\mathbf{s})}, \tag{2.5}$$

where $\tilde{f}$ indicates the Fourier transform of f , $\tilde{G}$ is a positive function that goes to zero for $||s|| \rightarrow \infty$, i.e. $1/\tilde{G}$ is a high-pass filter. The typical example of such a function $\tilde{G}$ is the Gaussian function.

Now the goal is to minimize the functional:

$$H[f] = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 + \gamma \int_{R^d} d\mathbf{s} \frac{|\tilde{f}(\mathbf{s})|^2}{\tilde{G}(\mathbf{s})}. \tag{2.6}$$

Under slight assumptions on $\tilde{G}$ it can be shown that the minimum of the functional (2.6) has the form of a linear combination of basis functions:

Theorem 2.3.1 *(Girosi, Jones, Poggio [12]) Let a function $\tilde{G} : \mathbb{R}^d \rightarrow \mathbb{R}$ be symmetric, such that its Fourier transform G is real and symmetric. Then the solution of minimization of functional (2.6) has the form:*

$$f(x) = \sum_{i=1}^{N} w_i G(\mathbf{x} - \mathbf{x}_i) + \sum_{\alpha=1}^{k} d_\alpha \psi_\alpha(\mathbf{x}), \tag{2.7}$$

where $\{\psi_\alpha\}_{\alpha=1}^{k}$ is a basis of the k-dimensional null space $\mathcal{N}$ of the functional Φ (in most cases a set of polynomials). Coefficients d_α and w_i depend on the data and satisfy the following linear system:

$$(\mathbf{G} + \gamma \mathbf{I})\mathbf{w} + \mathbf{\Psi}^T \mathbf{d} = \mathbf{y} \tag{2.8}$$

$$\mathbf{\Phi c} = 0 \tag{2.9}$$

where $\mathbf{I}$ is the identity matrix, and we have defined[1]*:*

$$\mathbf{y} = (y_1, \ldots, y_N),\ \mathbf{w} = (w_1, \ldots, w_N),\ \mathbf{d} = (d_1, \ldots, d_k) \tag{2.10}$$

$$\mathbf{G}_{ij} = G(\mathbf{x}_i - \mathbf{x}_j),\ \mathbf{\Psi}_{\alpha i} = \psi_\alpha(\mathbf{x}_i) \tag{2.11}$$

The proof is based on the fact that a minimum of a functional can exist in an interior point only if the first derivative equals zero. It can be found in [12], or in more detail in [73].

For $\gamma = 0$ we get a strict interpolation and the solution of the linear system depends on the properties of the basis function G [43].

The basis function G used in the approximation scheme is set by choice of the stabilizer, function $\tilde{G}$ in particular. So the type of the basis function reflects a prior assumption on the approximated function.

The important classes of functions suitable for the choice of $\tilde{G}$ are positive semi--definite, positive definite, and conditionally positive semi-definite functions.

[1] $\mathbf{G}_{ij}$ stands for an entry of the i-the row and j-the column of the matrix $\mathbf{G}$

Definition 2.3.2 *(Positive Semi-Definite Function) X is a nonempty set. Function $K : X \times X \to \mathbb{R}$ satisfying*

$$\sum_{i,j=0}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

for all $n \in \mathbb{N}$, for all $a_1, \ldots, a_n \in \mathbb{R}$ and for all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in X$ is called a positive semi-definite *function.*

Definition 2.3.3 *(Positive Definite Function) Positive semi-definite function $K : X \times X \to \mathbb{R}$ satisfying*

$$\sum_{i,j=0}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) = 0 \implies \forall i\, a_i = 0$$

for all $n \in \mathbb{N}$, for all $a_1, \ldots, a_n \in \mathbb{R}$ and for all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in X$ is called a (strictly) positive definite *function.*

Definition 2.3.4 *(Conditionally Positive Semi-Definite Function) X is a nonempty set. Function $K : X \times X \to \mathbb{R}$ satisfying*

$$\sum_{i,j=0}^{n} a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

for all $n \in \mathbb{N}$, for all $a_1, \ldots, a_n \in \mathbb{R}$ such that $\sum_{i=1}^{n} a_i = 0$, and for all $\mathbf{x}_1, \ldots, \mathbf{x}_n \in X$ is called a conditionally positive semi-definite *function.*

According to [12], for the class of positive semi-definite functions, the functional Φ is a norm, so its null space is empty and there is no polynomial term in (2.7). For the case of conditionally positive semi-definite functions, Φ is a semi-norm and the basis of its null space is a set of polynomials. In practical applications, the polynomial term is usually omitted.

2.3.1 Examples of Stabilizers

The choice of the basis function in an approximation schema (2.7) and the choice of the stabilizer (2.5) are equivalent. They both represent prior knowledge or an assumption about the approximated function. For better illustration, we will show three important groups of the stabilizers — *radial stabilizers*, *tensor product stabilizers*, and *additive stabilizers* and corresponding basis functions (all proposed in [12]).

- *Radial stabilizers* are a commonly used group of stabilizers. They are radially symmetric, which means that they satisfy

$$\Phi[f(\mathbf{x})] = \Phi[f(R\mathbf{x})], \tag{2.12}$$

Function	Name	Type	k
$G(r) = e^{-\frac{r^2}{b}}$	Gaussian function	p.s.d.	
$G(r) = \sqrt{r^2 + c^2}$	multi-quadratic	c.p.s.d.	1
$G(r) = \frac{1}{\sqrt{r^2+c^2}}$	inverse multi-quadratic	p.s.d.	
$G(r) = r^{2n+1}$	thin plate splines	c.p.s.d.	n
$G(r) = r^{2n} lnr$	thin plate splines	c.p.s.d.	n

Table 2.1: Examples of radial basis functions (p.s.d. stands for positive semi-definite, c.p.s.d. conditionally positive semi-definite, k stands for the order of the polynomial term if it is present in the solution).

where R is an arbitrary rotation matrix. Radial symmetry reflects the assumption that all the input variables have equal relevance, that is, there are no privileged directions. They lead to a *radial basis function* $G(||\mathbf{x}||)$ and the approximation model corresponds to the RBF networks. The class of feasible radial basis functions is the class of conditionally positive semi-definite functions, for which the functional (2.5) is a semi-norm, and the problem of minimizing (2.6) is well defined.

An important example is the *Gaussian function*, which corresponds to the stabilizer of the form

$$\Phi[f] = \int_{R^d} d\mathbf{s}\, e^{\frac{||\mathbf{s}||^2}{b}} |\tilde{f}(\mathbf{s})|^2, \tag{2.13}$$

where $b \in \mathbb{R}$ and $b > 0$. This stabilizer leads to the basis function

$$G(\mathbf{x}) = e^{-\frac{||\mathbf{x}||^2}{b}}. \tag{2.14}$$

As the Gaussian function is a positive definite function, $\Phi[f]$ is a norm, its null space contains only the zero element, and therefore the polynomial terms of equation (2.7) are not present.

Other Radial Basis Functions are listed in Table 2.1. The conditionally positive semi-definite functions need polynomial terms of order k in the solutions.

- *Tensor product stabilizer* can be used to derive the tensor product approximation schema. The tensor product stabilizer has the form

$$\Phi(f) = \int_{R^d} d\mathbf{s} \frac{|\tilde{f}(\mathbf{s})|^2}{\Pi_{j=1}^{d} \tilde{g}(s_j)}, \tag{2.15}$$

where $\tilde{g}$ is an appropriate one-dimensional function. For positive semi-definite functions g the functional $\Phi[f]$ is a norm and its null space is empty. For conditionally positive semi-definite functions g the null space can be more complicated, so these functions are typically not used.

The stabilizer leads to a tensor product basis function

$$G(\mathbf{x}) = \Pi_{j=1}^{d} g(x_j), \tag{2.16}$$

where g is the Fourier transform of $\tilde{g}$.

An interesting example is the choice of $\tilde{g}(s) = \frac{1}{1+s^2}$, which leads to the basis function:

$$G(\mathbf{x}) = \Pi_{j=1}^{d} e^{-|x_j|} = e^{-\sum_{j=1}^{d}|x_j|} = e^{-||\mathbf{x}||_{L1}}. \tag{2.17}$$

This basis function is interesting from the point of view of hardware implementation, since it requires only the computation of L_1 norm (instead of the usual Euclidean norm L_2).

- *Additive stabilizers* have the form

$$\Phi[f] = \sum_{\mu=1}^{d} \frac{1}{\theta_\mu} \int_R ds \frac{|\tilde{f}_\mu(s)|^2}{\tilde{g}(s)}, \tag{2.18}$$

where $\theta_\mu > 0$ are parameters allowing us to impose different degrees of smoothness on the concrete additive components.

By using such stabilizers it is possible to derive the class of *additive approximation schemes,* having a kernel function

$$G(\mathbf{x}) = \sum_{\mu=1}^{d} \theta_\mu g(x^\mu). \tag{2.19}$$

2.4 Kernel Based Approach

Kernels and kernel based approaches play a crucial role in the learning theory and the modern machine learning, which is dominated by kernel based algorithms. In this section we show the derivation of regularized learning problem solution in the kernel framework, with the help of Reproducing Kernel Hilbert Spaces according to Poggio and Smale [59].

The Reproducing Kernel Hilbert Space (RKHS) was first defined by Aronszajn [1].

Definition 2.4.1 *(Reproducing Kernel Hilbert Space) A* Reproducing Kernel Hilbert Space *is a Hilbert space $\mathcal{H}$ of pointwise defined functions on non-empty domain Ω with the property, that for each $\mathbf{x} \in \Omega$ the evaluation functional on $\mathcal{H}$ given by $\mathcal{F}_\mathbf{x} : f \mapsto f(\mathbf{x})$ is bounded.*

It can be shown that every RKHS can be associated with a positive semi-definite symmetric function — a *reproducing kernel*.

Theorem 2.4.2 *(Aronszajn [1]) Let $\mathcal{H}$ be an RKHS. Then there exists a unique positive semi-definite symmetric function $K : \Omega \times \Omega \to \mathcal{R}$ (called* reproducing kernel*) corresponding to $\mathcal{H}$ such that*

1. *For every $\mathbf{y} \in \Omega$, the function $K_{\mathbf{y}}(\mathbf{x}) = K(\mathbf{x}, \mathbf{y})$ is an element of $\mathcal{H}$.*
2. *For any $f \in \mathcal{H}$ and $\mathbf{y} \in \Omega$ the following reproducing property holds $f(\mathbf{y}) = \langle f, K_{\mathbf{y}} \rangle$, where $\langle ., . \rangle$ is scalar product in $\mathcal{H}$.*

On the other hand, every positive semi-definite symmetric function K is a reproducing kernel for exactly one RKHS.

Theorem 2.4.3 *(Aronszajn [1]) Let K be a positive semi-definite symmetric function. Then there exists exactly one RKHS $\mathcal{H}_K$, such that K is its reproducing kernel. $\mathcal{H}_K$ can be described as*

$$\mathcal{H}_K = \mathrm{compl}\left\{ \sum_{i=1}^{n} a_i K_{\mathbf{x}_i}; \mathbf{x}_i \in \Omega, a_i \in \mathcal{R}, n \in \mathbb{N} \right\}, \tag{2.20}$$

where compl *means completion of the normed linear space, $K_{\mathbf{x}_i}$ stands for kernel function K with the second variable fixed to $\mathbf{x}_i$.*

The proofs of Theorems 2.4.2 and 2.4.3 can be found in [1].

The norm in a RKHS is given by a scalar product

$$\langle K_{\mathbf{x_i}}, K_{\mathbf{x_j}} \rangle = K(\mathbf{x_i}, \mathbf{x_j}), \tag{2.21}$$

extending linearly to

$$\langle f, g \rangle = \langle \sum_{i=1}^{n} a_i K_{\mathbf{x}_i}, \sum_{j=1}^{n} b_j K_{\mathbf{x}_j} \rangle = \sum_{i,j=1}^{n} a_i b_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{2.22}$$

Poggio and Smale [59] used RKHSs to derive the solution of regularized learning problem in the following way.

Let $\mathcal{H}_K$ be an RKHS defined by a symmetric, positive-definite kernel function K. Then one can take the $\mathcal{H}_K$ as a hypothesis space and define the stabilizer by means of the norm in $\mathcal{H}_K$

$$\Phi[f] = ||f||_K^2. \tag{2.23}$$

Theorem 2.4.4 *Let $\mathcal{H}_K$ be a RKHS defined by a positive semi-definite symmetric kernel function K. Then the minimum of the functional $H[f]$:*

$$H[f] = \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 + \gamma ||f||_K^2 \tag{2.24}$$

over $\mathcal{H}_K$ is unique and has the form

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i K_{\mathbf{x}_i}(\mathbf{x}) \tag{2.25}$$

and the coefficients w_i satisfy

$$(\mathbf{K} + \gamma \mathbf{I})\mathbf{w} = \mathbf{y}, \tag{2.26}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{K}$ is the matrix $\mathbf{K}_{i,j} = K(\mathbf{x_i}, \mathbf{x_j})$, and $\mathbf{y} = (y_1, \ldots, y_N)$.

Proof: (Sketch of the proof) We apply the operator $\int dt \overline{f}(t) \frac{\delta}{\delta f}$ (integral of the functional derivative) to the functional (2.24) and set it equal to zero. We get:

$$\sum_{i=1}^{N} (y_i - f(\mathbf{x}_i)) \overline{f}(\mathbf{x_i}) + \gamma \langle f, \overline{f} \rangle = 0 \tag{2.27}$$

Since the equation is valid for any $\overline{f}$, it is valid also for $\overline{f} = K_{\mathbf{x}}$:

$$\sum_{i=1}^{N} (y_i - f(\mathbf{x}_i)) K_{\mathbf{x}}(\mathbf{x_i}) + \gamma \langle f, K_{\mathbf{x}} \rangle = 0 \tag{2.28}$$

Applying the reproducing property we get

$$\sum_{i=1}^{N} (y_i - f(\mathbf{x}_i)) K_{\mathbf{x}}(\mathbf{x_i}) + \gamma f(\mathbf{x}) = 0$$

$$\frac{1}{\gamma} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i)) K_{\mathbf{x}}(\mathbf{x_i}) = f(\mathbf{x})$$

and we can write

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i K_{\mathbf{x}_i}(\mathbf{x})$$

$$w_i = \frac{y_i - f(\mathbf{x}_i)}{\gamma}$$

□

Theorem 2.4.4 was presented in [59]. We have slightly changed the functional (2.24). The change concerns the scaling factor of the data term. Poggio and Smale in [59] scale the data term proportionally to the number of data points, i.e. they define the task as the minimization of functional

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

and obtain a linear system defining the weights:

$$(\mathbf{K} + N\gamma\mathbf{I})\mathbf{w} = \mathbf{y} \tag{2.29}$$

(compare to (2.26)).

Clearly, scaling the data term does not change the solution of the minimization problem. However, the solution (2.29) is not numerically feasible. For a big data set, i.e. big values of N, the diagonal part dominates and the information given by the data set (expressed by matrix $\mathbf{K}$) is suppressed. One can argue that this can be cured by adjusting the regularization parameter γ to be smaller. But the role of regularization parameter is to control the trade-off between the data term and the regularization term, and we do not want it to depend on N.

2.5 Regularization Network

In the previous sections we have described the derivation of the regularized learning problem solution, first using stabilizers based on the Fourier transform, second with the help of RKHSs. In both the cases the solution has a form of a linear combination of basis (kernel) functions (see Theorems 2.3.1 and 2.4.4)

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i K(\mathbf{x}, \mathbf{x}_i), \tag{2.30}$$

supposing that we are given a training set

$$\{(\mathbf{x}_i, y_i) \in R^d \times R\}_{i=1}^{N}. \tag{2.31}$$

The solution derived in Section 2.3 has in addition a polynomial term. When a positive semi-definite basis function is used, this term does not appear. In Section 2.4 only positive semi-definite functions were considered, so the solution (2.25) has no polynomial term. Further we will work mainly with positive semi-definite kernels, and in other cases we will omit the polynomial term.

The solution (2.30) can be represented as a feed-forward neural network with one hidden layer and a linear output layer (see Figure 2.2). The hidden layer consists of *kernel units* realizing the basis (kernel) functions.

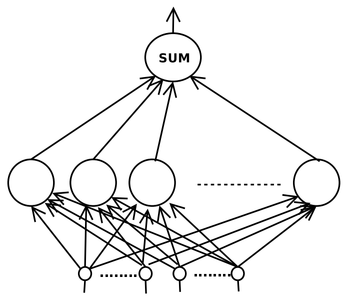

Figure 2.2: Regularization network scheme.

Definition 2.5.1 *(Kernel Unit) A* kernel unit *is a computational unit with d real inputs $\mathbf{x} = x_1, \ldots, x_d$ and one real output y. It evaluates the function*

$$y(\mathbf{x}) = K(\mathbf{x}, \mathbf{c}),$$

where K is a suitable kernel function, $\mathbf{c} \in \mathbb{R}^d$. The vector $\mathbf{c}$ is called a center.

Note that in the optimal solution the centers of the kernel units, i.e. the second arguments of the kernel functions, are fixed to the data points $\mathbf{x}_i$. Such a neural network is called a *regularization network* (RN) [12].

Definition 2.5.2 *(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*

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i K(\mathbf{x_i}, \mathbf{c_i}), \tag{2.32}$$

where N is a number of hidden neurons (i.e. number of basis functions), $w_i \in \mathbb{R}$, $\mathbf{c}_i \in \mathbb{R}^d, \mathbf{x}_i \in \mathbb{R}^d$, $K : \mathbb{R}^d \rightarrow \mathbb{R}$ 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.

The practical aspects of learning with the regularization networks and the corresponding algorithms will be discussed in detail in Chapter 4.

Chapter 3

Product and Sum Regularization Networks

The whole is more than the sum of its parts.
Aristotle, Metaphysica

In this chapter we propose new types of regularization networks — a *product kernel regularization network* and a *sum kernel regularization network*. These approximation schemata are based on the composite types of kernel functions, and were first introduced in [35, 36].

The next section describes our motivation for the composite kernels. In Section 3.2 the product of two RKHS, a mathematical justification for a *product kernel*, is constructed according to Aronszajn [1]. Then in Section 3.3 the product kernel and the product kernel regularization network are introduced. Section 3.4 deals with the sum of two RKHS [1] that is used in Section 3.5 to derive a *sum kernel* and the sum kernel regularization network.

3.1 Motivation

The kernel function used in a particular application of regularization network is typically supposed to be given in advance, for instance chosen by a user.

In fact, the choice of kernel function is equivalent to the choice of prior assumption about the problem at hand (see Sections 2.3–2.4). It reflects our prior knowledge or assumption about the problem and its solution. Therefore its choice is crucial for the quality of the solution and should be always done according to the given task.

As a kernel function we can use any symmetric, positive semi-definite function, possibly a conditionally positive semi-definite function. However, the most frequently used kernel function is the Gaussian function. Other common kernel functions are listed in Table 2.1. The procedure we use for choosing a suitable kernel function will be discussed in the next chapter.

Although only kernels defined on real numbers are considered throughout this work, in practical applications we often meet data containing attributes of different types, such as enumerations, sets, strings, texts, etc. Such data may be converted to real numbers by suitable preprocessing or the regularization network learning framework may be generalized so that it can work on such types. For such a generalization, sophisticated kernel functions defined on various types were created. The examples of kernel functions defined on objects including graphs, sets, texts, etc. can be found in [65].

To sum it up, when choosing the kernel function, two aspects have to be considered. They are the prior knowledge of the problem and type of the data domain.

However, the real data are often heterogenous. The heterogeneity refers either to attributes or parts of the input space, or both. By the former we mean that different attributes are of different types or differ in quality. By the latter we mean that the data have different qualities (such as density) 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, 64]. Based on the operations of sum and product we introduce two types of composite kernel units, namely a *product kernel* and a *sum kernel*.

3.2 Product of RKHSs

In this section we present a derivation of the product of two RKHSs and show that its reproducing kernel is a product of two kernels corresponding to the original RKHSs [1]. This will be used in the next section to derive a product kernel regularization network.

Let F_1 on Ω_1 and F_2 on Ω_2 be two different RKHSs, and let K_1 and K_2 be their reproducing kernels. The goal is to find a set of functions on $\Omega = \Omega_1 \times \Omega_2$ that forms an RKHS with the reproducing kernel given by $K((\mathbf{x}_1, \mathbf{x}_2), (\mathbf{y}_1, \mathbf{y}_2)) = K_1(\mathbf{x}_1, \mathbf{y}_1)K_2(\mathbf{x}_2, \mathbf{y}_2)$, where $\mathbf{x}_1, \mathbf{y}_1 \in \Omega_1$ and $\mathbf{x}_2, \mathbf{y}_2 \in \Omega_2$.

First consider the following set of functions on $\Omega = \Omega_1 \times \Omega_2$:

$$F' = \left\{ \sum_{k=1}^{n} f_{1,k}(\mathbf{x}_1) f_{2,k}(\mathbf{x}_2) \mid n \in \mathbb{N}, f_{1,k} \in F_1, f_{2,k} \in F_2, \mathbf{x}_1 \in \Omega_1, \mathbf{x}_2 \in \Omega_2 \right\} . \quad (3.1)$$

To transform F' into a Hilbert space, one needs to define a scalar product on it and make it complete with respect to the norm given by this scalar product. The scalar product is given by the following lemma.

Lemma 3.2.1 *(Aronszajn [1]) Let functions $f, g \in F'$ be expressed as*

$$f(\mathbf{x}_1, \mathbf{x}_2) = \sum_{k=1}^{n} f_{1,k}(\mathbf{x}_1) f_{2,k}(\mathbf{x}_2) \tag{3.2}$$

and

$$g(\mathbf{x}_1, \mathbf{x}_2) = \sum_{j=1}^{m} g_{1,j}(\mathbf{x}_1) g_{2,j}(\mathbf{x}_2). \tag{3.3}$$

Let $\langle f, g \rangle$ be defined as

$$\langle f, g \rangle = \sum_{k=1}^{n} \sum_{j=1}^{m} \langle f_{1,k}, g_{1,j} \rangle_1 \langle f_{2,k}, g_{2,j} \rangle_2, \tag{3.4}$$

where $\langle \cdot, \cdot \rangle_i$ denotes the scalar product on F_i. Then $\langle f, g \rangle$ is a scalar product on F'.

Now the norm on F' can be defined as usual.

Definition 3.2.2 *(Norm on F') Let F' be a function space (3.1). Let $\langle \cdot, \cdot \rangle$ defined by (3.4) be a scalar product on F'. Then the norm on F' is defined by*

$$\|f\|_{F'} = \sqrt{\langle f, f \rangle}. \tag{3.5}$$

To obtain a Hilbert space from a scalar product space F' (with scalar product (3.4)), one needs to make it complete. The following theorem constructs the completion of F' with respect to the norm $\| \cdot \|_{F'}$.

Theorem 3.2.3 *(Aronszajn [1]) Let F' be the set of functions defined in (3.1). Let $\{g_i^{(k)}\}_k$ be complete orthonormal sequence in the space F_i, for $i = 1, 2$. Then the class of functions F on Ω*

$$F = \{g' \mid g'(\mathbf{x}_1, \mathbf{x}_2) = \sum_{k=1}^{\infty} \sum_{l=1}^{\infty} a_{kl} g_1^{(k)}(\mathbf{x}_1) g_2^{(l)}(\mathbf{x}_2)\}, \tag{3.6}$$

$$\text{with } \langle g', g' \rangle = \sum_{k=1}^{\infty} \sum_{l=1}^{\infty} |a_{kl}|^2 < \infty, \tag{3.7}$$

forms a complete Hilbert space and is the completion of F' with respect to the norm $\| \cdot \|_{F'}$.

Note that any finite sum of type (3.6) is also of type (3.1) and the norm $\|\cdot\|_{F'}$ coincides with the norm defined by (3.7). For the proof see [1].

Definition 3.2.4 *(Product of RKHSs) Let F_1 be an RKHS on Ω_1 and F_2 an RKHS on Ω_2. Let*

$$F = \mathrm{compl}\left\{ \sum_{k=1}^{n} f_{1,k}(\mathbf{x}_1) f_{2,k}(\mathbf{x}_2) \mid n \in \mathbb{N}, f_{1,k} \in F_1, f_{2,k} \in F_2 \right\}, \tag{3.8}$$

where compl *means completion of the set with respect to the norm $\|\cdot\|_{F'}$. F is called the* product of F_1 and F_2 *and write*

$$F = F_1 \otimes F_2. \tag{3.9}$$

The last thing to be proved that F is an RKHS with the reproducing kernel given by the product of kernels K_1 and K_2. According to Aronszajn [1]:

Theorem 3.2.5 *(Aronszajn [1]) For $i = 1, 2$ let F_i be an RKHS on Ω_i with kernel K_i. Then the product $F = F_1 \otimes F_2$ on $\Omega_1 \times \Omega_2$ is an RKHS with kernel given by*

$$K((\mathbf{x}_1, \mathbf{x}_2), (\mathbf{y}_1, \mathbf{y}_2)) = K_1(\mathbf{x}_1, \mathbf{y}_1) K_2(\mathbf{x}_2, \mathbf{y}_2), \tag{3.10}$$

where $\mathbf{x}_1, \mathbf{y}_1 \in \Omega_1$, $\mathbf{x}_2, \mathbf{y}_2 \in \Omega_2$.

3.3 Product Kernels

In the previous section we showed a derivation of the product of two RKHSs, and that its reproducing kernel can be obtained as a product of reproducing kernels of original RKHSs. Such a kernel we call a *product kernel*. More generally:

Definition 3.3.1 *(Product Kernel) Let $K_1, \ldots, K_k$ be the 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*

$$K(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k) = K_1(\mathbf{x}_1, \mathbf{y}_1) K_2(\mathbf{x}_2, \mathbf{y}_2) \cdots K_k(\mathbf{x}_k, \mathbf{y}_k), \tag{3.11}$$

where $\mathbf{x}_i \in \Omega_i$, we call a product kernel.

A computational unit that realizes the product kernel function will be called a *product unit* (see Figure 3.1).

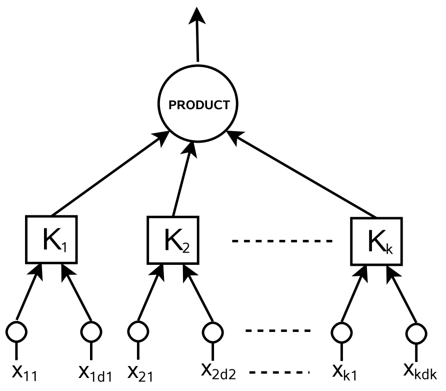

Figure 3.1: A unit realizing a product kernel.

Definition 3.3.2 *(Product Unit) A* product unit *is a computational unit with multiple inputs* $(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k})$*,* $\mathbf{x_i} \in \Omega_i$*, and one real output* y*, realizing a function*

$$y(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}) = K_1(\mathbf{x}_1, \mathbf{c}_1)K_2(\mathbf{x}_2, \mathbf{c}_2)\cdots K_k(\mathbf{x}_k, \mathbf{c}_k),$$

where K_i *are kernel functions. Vectors* $\mathbf{c}_i$ *are called* centers.

The regularization network with the hidden layer formed by product kernels is called a *product kernel regularization network* (PKRN).

Definition 3.3.3 *(Product Kernel Regularization Network) A* product kernel regularization network *(PKRN) is a regularization network realizing a function*

$$f(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}) = \sum_{i=1}^{N} w_i K_1(\mathbf{x}_1, \mathbf{c}_1)K_2(\mathbf{x}_2, \mathbf{c}_2)\cdots K_k(\mathbf{x}_k, \mathbf{c}_k), \tag{3.12}$$

where $\mathbf{x_i}, \mathbf{c_i} \in \Omega_i$*,* K_i *are kernel functions.*

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 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

$$\{(\mathbf{x}_1^i, \mathbf{x}_2^i, \ldots, \mathbf{x}_k^i, y^i) \in R^{d_1} \times R^{d_2} \times \ldots \times R^{d_k} \times R\}_{i=1}^N. \tag{3.13}$$

Using a product kernel on such training data enables us to process different $\mathbf{x}_i$, i.e. groups of attributes, separately by different kernel functions.

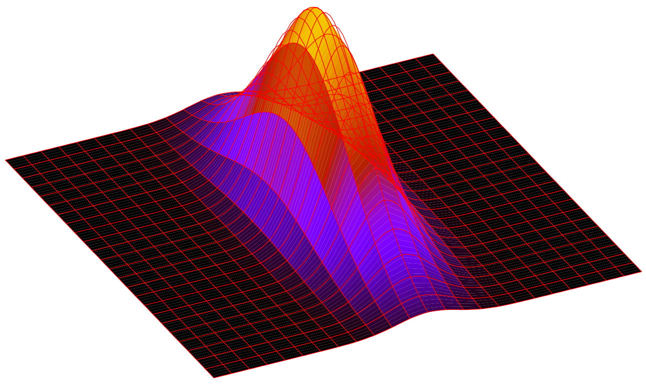

Figure 3.2: A product kernel from Example 3.3.4.

Though the theory is derived for kernels defined on real numbers ($\Omega \subset \mathbb{R}^d$), it is possible to combine kernel functions defined on different types. Then attributes in the input vector may be very diverse, for instance real numbers mixed with categorical attributes, strings, texts, or various objects. The individual attributes can be divided into groups of the same type, forming several input vectors. Each input vector, i.e. a part of the original input vector, is then processed by a kernel function suitable for its type.

Example 3.3.4 The simplest example of PKRN is the one using a product of two Gaussian kernels. Suppose that in one dimension the data are suitable for approximation via a narrow Gaussian kernel, in the second dimension the function is smooth, so we want to use a broader Gaussian kernel. Then we obtain an approximation schema

$$f(\mathbf{x}_1, \mathbf{x}_2) = \sum_{i=1}^{N} w_i e^{-(\frac{\|\mathbf{x}_1 - \mathbf{x}_1^i\|}{d_1})^2} \cdot e^{-(\frac{\|\mathbf{x}_2 - \mathbf{x}_2^i\|}{d_2})^2}, \tag{3.14}$$

where d_1 and d_2 are the widths of the Gaussians. See Figure 3.2 for the illustration of the resulting kernel.

3.4 Sum of RKHSs

In this section we construct a sum of two RKHSs and show that its reproducing kernel can be obtained as a sum of the reproducing kernels of the two original RKHSs.

Let F_1 and F_2 be RKHSs of functions on $\Omega \subset \mathbb{R}^d$. Let K_1 and K_2 be the corresponding kernels and $\|.\|_1$ and $\|.\|_2$ the corresponding norms.

Lemma 3.4.1 *(Aronszajn [1]) Let $K_1 : \Omega \times \Omega \to \mathbb{R}$ and $K_2 : \Omega \times \Omega \to \mathbb{R}$ be positive semi-definite functions. Then the function $K : \Omega \times \Omega \to \mathbb{R}$ defined as*

$$K(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y})$$

is also a positive semi-definite function.

Since the sum of two positive semi-definite functions is a positive definite function, it is a reproducing kernel for an RKHS. Now we show how to find the class of functions that form this RKHS.

Definition 3.4.2 *(H) Let H be the space of all couples $\{f_1, f_2\}$ on Ω such that*

$$H = \{\{f_1, f_2\} \mid f_1 \in F_1, f_2 \in F_2\}, \tag{3.15}$$

and let the metric on H be given by

$$\|\{f_1, f_2\}\|^2 = \|f_1\|_1^2 + \|f_2\|_2^2. \tag{3.16}$$

Let F_0 be a class of all functions f belonging to $F_1 \cap F_2$. We define H_0 as

$$H_0 := \{\{f, -f\}; f \in F_0\}. \tag{3.17}$$

H_0 is a closed subspace of H, thus we can write $H = H_0 \oplus H'$, where H' is the complementary subspace to H_0.

Definition 3.4.3 *(H') Let H be the space defined in Definition (3.4.2) and H_0 the space (3.17). Then we define H' as a subspace of H such that $H = H_0 \oplus H'$.*

To every element $\{f', f''\}$ of H there corresponds a function $f(\mathbf{x}) = f'(\mathbf{x}) + f''(\mathbf{x})$. So there is a linear correspondence transforming H into a linear class of functions on Ω.

Definition 3.4.4 *(F) Let H be the space defined in Definition (3.4.2). We define F as*

$$F = \{f \mid f(\mathbf{x}) = f'(\mathbf{x}) + f''(\mathbf{x}), \{f', f''\} \in H\}. \tag{3.18}$$

Elements of H_0 are precisely those transformed into the zero function and thus the correspondence between H' and F is one-to-one and has an inverse (for every $f \in F$ we obtain one $\{g'(f), g''(f)\}$).

So the metric on F can be defined by the following way:

Definition 3.4.5 *(Norm on F) Let F be the space defined in Definition 3.4.4. Then we define the norm on F as*

$$\|f\|^2 = \|\{g'(f), g''(f)\}\|^2 = \|g'(f)\|_1^2 + \|g''(f)\|_2^2.$$

The last thing to be proved is that the function $K(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y})$ is the reproducing kernel for the space F.

Theorem 3.4.6 *(Aronszajn [1]) Let F_1 and F_2 be the RKHSs and K_1, K_2 and $\|.\|_1$, $\|.\|_2$ the corresponding kernels and norms. Let F be defined as in Definition 3.4.4 with the norm defined in Definition 3.4.5. Then*

$$K(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y}) \tag{3.19}$$

is the kernel corresponding to F.

The claim holds also for F defined as a class of all functions $f = f_1 + f_2$ with $f_i \in F_i$ and norm $\|f\|^2 = \min(\|f_1\|_1^2 + \|f_2\|_2^2)$ with the minimum taken for all decompositions $f = f_1 + f_2$ with f_i in F_i.

3.5 Sum Kernels

The reproducing kernel of the sum of RKHSs shown in the previous section, which is the kernel that can be obtained as a sum of other kernel functions, is called a *sum kernel*.

Definition 3.5.1 *(Sum Kernel) The kernel function K that can be obtained as a sum of two or more other kernel functions $K_1, \ldots, K_k$*

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k} K_i(\mathbf{x}, \mathbf{y}), \tag{3.20}$$

is called a sum kernel.

The computational unit realizing the sum kernel is shown in Figure 3.3. We call it a *sum unit.*

Definition 3.5.2 *(Sum Unit) A* sum unit *is a unit with multiple inputs $\mathbf{x}$, $\mathbf{x} \in \Omega$, and one real output y, realizing a function*

$$y(\mathbf{x}) = K_1(\mathbf{x}, \mathbf{c}) + K_2(\mathbf{x}, \mathbf{c}) + \cdots + K_k(\mathbf{x}, \mathbf{c}),$$

where K_i are kernel functions. Vector $\mathbf{c} \in \Omega$ is called a center.

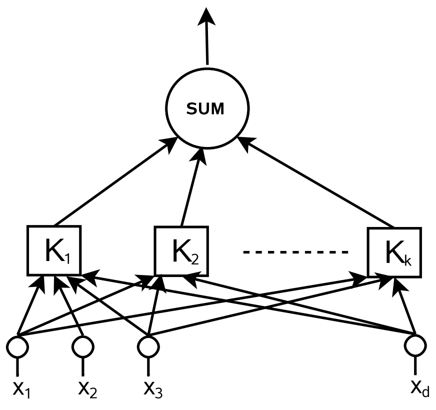

Figure 3.3: A unit realizing a sum kernel.

The regularization network that has sum units in its hidden layer is called a *sum kernel regularization network* (SKRN).

Definition 3.5.3 *(Sum Kernel Regularization Network) Let $K_1, \ldots, K_k$ be kernel functions on Ω. The* sum kernel regularization network *(SKRN) is an approximation schema that has the form*

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i (K_1(\mathbf{x}, \mathbf{c}_i) + \ldots + K_k(\mathbf{x}, \mathbf{c}_i)), \tag{3.21}$$

where $\mathbf{x} \in \Omega, \mathbf{c_i} \in \Omega$.

The sum kernel is intended for use in cases when a priori knowledge or analysis of data suggests looking for a solution being a sum of two or more functions. For example, when the data is generated from a function influenced by two sources of different frequencies. Then we can use a kernel obtained as a sum of two parts corresponding to high and low frequencies (see Figure 3.4).

In our experiments in Chapter 6, Subsection 6.3.4, we will show that the kernel obtained as a sum of two Gaussian functions has an interesting behavior. It enables us to achieve very low errors on the training data while preserving its generalization ability. Such an SKRN outperforms a standard regularization network in terms of approximation error on most tasks we have tested.

3.5.1 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*. We will take advantage of the following lemma.

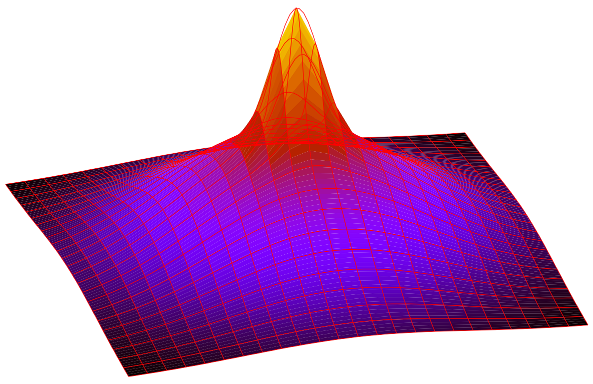

Figure 3.4: An example of a sum kernel formed by two Gaussian functions.

Lemma 3.5.4 *(Aronszajn [1]) Let F be an RKHS of real-valued functions on Ω with K as a kernel. Then function K_A defined by*

$$K_A(\mathbf{x},\mathbf{y}) = \begin{cases} K(\mathbf{x},\mathbf{y}) & \textit{if } \mathbf{x},\mathbf{y} \in A, \\ 0 & \textit{otherwise;} \end{cases} \tag{3.22}$$

is a kernel for the space $F_A = \{f_A, f \in F\}$, where $f_A(\mathbf{x}) = f(\mathbf{x})$ if $\mathbf{x} \in A$ and $f_A(\mathbf{x}) = 0$ otherwise.

Definition 3.5.5 *(Restricted Sum Kernel) The kernel $K_A(\mathbf{x},\mathbf{y})$ (3.22) 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 disjunct subsets $A_1, \dots, A_k$ 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:

$$K(\mathbf{x},\mathbf{y}) = \begin{cases} K_i(\mathbf{x},\mathbf{y}) & \text{if } \mathbf{x},\mathbf{y} \in A_i \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

3.5.2 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

$$f(\mathbf{x}) = \sum_{\mathbf{x}_i \in A_1} w_i K_1(\mathbf{x}, \mathbf{x}_i) + \ldots + \sum_{\mathbf{x}_i \in A_k} w_i K_k(\mathbf{x}, \mathbf{x}_i), \tag{3.24}$$

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

In addition, for the case of disjunct sets A_s, always exactly one member of (3.24) 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 a nonzero output. So we can write

$$f(\mathbf{x}) = \begin{cases} \sum_{\mathbf{x}_i \in A_s} w_{si} K_s(\mathbf{x}, \mathbf{x}_i) & \text{if } \exists s : \mathbf{x} \in A_s \\ 0 & \text{if } \forall s : \mathbf{x} \notin A_s. \end{cases} \tag{3.25}$$

Conversely, to determine the value of w_{si}, 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 of smaller linear systems (2.26) instead of a big one.

Replacing one linear system by k smaller ones reduces the space requirements of learning[1]. In addition the time requirements decrease, which will be shown later in Section 6.3.5. The drawback of this approach is a slightly bigger approximation error. It is caused by the lack of information on the borders of the input space areas, i.e. sets A_i.

[1]The space requirements of learning are given by the amount of space needed to store and solve the linear system (2.26). For more details on the learning algorithm see Section 4.1.

Chapter 4

Learning with Regularization Networks

Prediction is difficult,
especially of the future.
Niels Bohr

In this chapter we deal with the learning using the regularization networks introduced in Chapter 2. The following section describes the basic learning algorithm for a regularization network. In Section 4.2 we discuss the role of the regularization parameter and the kernel function. In Section 4.3 we propose a framework above the RN learning algorithm that realizes the whole learning procedure including the setup. The Section 4.4 describes cross-validation, a standard technique for the estimation of neural network generalization ability that is used in Section 4.5, where we propose the *adaptive grid search* algorithm for the estimation of the optimal regularization parameter and kernel function, and in Section 4.6, where the *genetic search* algorithm is proposed.

4.1 The Regularization Network Learning Algorithm

The basic learning algorithm for a regularization network, which is sketched in Algorithm 4.1.1, follows directly from the Representer Theorem (see Theorem 2.3.1 and 2.4.2). It consists of two steps. First, we set the centers of kernel functions to the given data points, and then we compute the weights solving the linear system (4.1). The first step is trivial, but the second step involves linear optimization and forms a crucial part of the algorithm.

```
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
    $w_1, \ldots, w_N$:
        $(\mathbf{K} + \gamma\mathbf{I})\mathbf{w} = \mathbf{y}, \qquad (4.1)$
    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 4.1.1. The RN learning algorithm.

The strength of the algorithm stems from the fact that the linear system we are solving is well-posed for positive semi-definite kernel function K, i.e. it has a unique solution and the solution exists[1] [12, 11, 59, 72].

However, as it will be shown later experimentally (Chapter 6, Subsection 6.3.1), the real performance of the algorithm significantly depends on the choice of regularization parameter and kernel function. These parameters are supposed to be given in advance. We will call them *metaparameters* to distinguish them from the parameters of the regularization network itself (weights, centers).

Another aspect of the successful application of the algorithm is the choice of method for solving the linear system. This problem is well studied by numerical mathematics and a variety of algorithms exists [4, 75, 62]. The choice of the method should depend on the size of the linear system, i.e. the size of the training set.

For the data sets of small and medium size, the linear system can be solved by direct methods, such as RQ decomposition [62]. Then the algorithm is simple and effective.

The tasks with huge data sets are more difficult to solve, and they lead to solutions of unreasonable size as well. Other algorithms should be used in such cases. One option is represented by the RBF networks belonging to the family of *generalized regularization networks*, discussed later in Chapter 5. Alternatively, in Section 3.5.2 a simple "Divide et Impera" approach has been proposed, dividing the task to several smaller subtasks.

[1]It has N variables, N equations, $\mathbf{K}$ is positive semidefinite and $(\gamma\mathbf{I} + \mathbf{K})$ is strictly positive.

We can apply the basic algorithm on these subtasks and then obtain the resulting network as a sum of the sub-results.

4.2 Role of Regularization Parameter and Kernel Function

In the previous section the RN learning algorithm (Algorithm 4.1.1) was described. Now we will deal with its metaparameters, the regularization parameter and the kernel function, in more detail. In particular, we will discuss how these metaparameters influence the solution and also the algorithm numerical stability.

Recall that the regularization network found by the RN learning algorithm (Algorithm 4.1.1) is the solution of the minimization problem

$$H[f] = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 + \gamma \Phi[f], \tag{4.2}$$

where Φ is a *stabilizer* and $\gamma > 0$ is *the regularization parameter*.

The regularization parameter γ controls the trade-off between the data term and the regularization term, i.e. the trade-off between the closeness to data and the solution smoothness. The non-zero regularization parameter prevents over-fitting and should always reflect the noise level. Therefore it has to be set up according to the given task, there is no universal value for it.

The second metaparameter is the kernel function K. In general, the choice of the kernel function corresponds to

1. Choice of the stabilizer: When using the stabilizers based on the Fourier transform (2.5), the particular form of this stabilizer is given by the choice of a high--pass filter (that is the choice of $\tilde{G}$). This choice determines the kernel function used in the solution (see Section 2.3).

2. Choice of a function space for learning: In derivation of RN with the help of RKHSs (see Section 2.4), the choice of the kernel function is equivalent to the choice of an RKHS that is used as the hypothesis space.

In both the cases the kernel function represents our knowledge or assumption about the problem and its solution. Wolpert [76] introduced the *no-free-lunch theorem* stating that there is no general purpose learning algorithm, and the only way one strategy can outperform another is if it is specialized in the specific problem under consideration. Such specialization requires prior knowledge of the given problem. Similarly no kernel can outperform other kernels in all possible problems [64] and so the kernel function should be chosen according to our knowledge of the problem at hand.

The choice of a value of both metaparameters influences also the numerical stability of the linear system (4.1) that has to be solved in the RN learning algorithm (Algorithm 4.1.1).

We have learned that the system is well-posed. But we also have to ask the question whether the system is numerically well-posed, i.e. insensitive to small perturbations of the data.

A rough measure of the problem feasibility to digital (numerical) computation is a *condition number* [9]. The problem with a low condition number is said to be *well--conditioned*, while the problem with a high condition number is said to be *ill-conditioned*.

Definition 4.2.1 *(Condition Number of a Matrix) Let* $\mathbf{A}$ *be a real matrix. Then the number* $\kappa(\mathbf{A})$ *given by*

$$\kappa(\mathbf{A}) = ||\mathbf{A}^{-1}|| \cdot ||\mathbf{A}||, \tag{4.3}$$

where $||\cdot||$ *is any consistent norm, is called a* condition number of matrix. *If* $||\cdot||$ *is the* L_2 *norm, the condition number can be computed as*

$$\kappa(\mathbf{A}) = \frac{\sigma_{max}(\mathbf{A})}{\sigma_{min}(\mathbf{A})}, \tag{4.4}$$

where $\sigma_{max}(\mathbf{A})$ *and* $\sigma_{min}(\mathbf{A})$ *are the maximal and minimal singular values of* $\mathbf{A}$ *respectively.*

The condition number associated with the linear equation $\mathbf{Ax} = \mathbf{b}$ gives a bound on how inaccurate the solution $\mathbf{x}$ will be after an approximate solution. In fact, the condition number effectively amplifies the error present in $\mathbf{b}$.

If the parameter γ is large, the matrix $\mathbf{K} + \gamma\mathbf{I}$ has a dominant diagonal and the condition number is small [9]. So the choice of γ has a direct influence on the numerical properties of the linear system (4.1). However, the choice of γ should, at the first place, reflect the level of noise in our problem, not to cure the ill-conditioning of the problem. If we choose γ too large, the linear system may be easy to solve, but the solution will not fit our data at all. So the optimal value of γ should balance the well-conditioning and the closeness to the data.

Naturally, the numerical properties of the linear system does not depend only on the value of γ, but also on the properties of the matrix $\mathbf{K}$ that is given by the kernel function choice.

As the most common kernel function is the Gaussian function

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{||\mathbf{x}-\mathbf{y}||^2}{b}}, \tag{4.5}$$

we will discuss the properties of the matrix $\mathbf{K}$ corresponding to the Gaussian kernel function in more detail.

Clearly the values of entries of the matrix $\mathbf{K}$ depend both on the distance between individual data points and the width b. The entries on the main diagonal are always equal to one, all other elements are from the interval $(0, 1\rangle$.

$$\begin{aligned} \mathbf{K}_{ii} &= 1 \\ \mathbf{K}_{ij} &\in (0,1\rangle, i \neq j \end{aligned}$$

It is easy to see that by decreasing the width we make the matrix entries smaller, and vice versa. Considering a digital representation with finite precision, both the extremes result in losing information. Too small width leads to a diagonal identity matrix $\mathbf{K}$ and a trivial linear system; too wide width to matrix $\mathbf{K}$ with all entries close to one, which makes the system (4.1) difficult to solve.

Considering the condition number, smaller widths result in matrices with a dominant diagonal, i.e. a smaller condition number; wider widths on the other hand lead to matrices with a high condition number. Again, there is a trade-off between avoiding the ill-conditioning and not losing the information. Note that what we mean by small width and wide width is always relative and depends on the density of data points. The width should be always chosen according to the given data.

The case of the Gaussian kernel function was also studied in [42, 49]. Narcowich, Sivakumar and Ward defined the *separation radius* as the minimal distance between two data points. They have shown that for the Gaussian kernel function the condition number of matrix $\mathbf{K}$ depends only on two parameters: the dimension d of the input space and $t = \frac{q}{b^2}$, where q is the separation radius and b is the width of the Gaussian. It justifies the intuitive idea that the denser data we have, the narrower Gaussian kernels are suitable and vice versa.

So in the choice of both the regularization parameter γ and kernel function type, there is always a trade-off between making the problem easier to solve and not losing relevant information, and a trade-off between fitting the training data and making the solution smooth enough to generalize. Both metaparameters reflect our prior knowledge about the problem, the regularization parameter corresponds to the level of noise, whereas the kernel function express a general knowledge or assumption.

Later, in Chapter 6, Section 6.3.1, the role of metaparameters is illustrated on experiments, and it is shown that a wrong choice of kernel function may lead to the failure of the RN learning algorithm.

4.3 Learning Framework

The discussion in the previous section indicates that the real performance of the algorithm depends significantly on the metaparameters choice. The metaparameters are a part of formulation of the problem we are solving. Clearly, if we formulate our problem improperly, the obtained solution may be useless.

Ideally, these parameters should be selected by the user based on their knowledge of the problem given. 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, etc. (For sum and product kernels it is necessary to determine the type for all kernels used in the sum, resp. product).
    (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 4.1.1)

The proposed autonomous learning procedure consists of two parts. In the first part, we set up the metaparameters of the learning algorithm, in the second part we run the algorithm on the given data. The setup includes choosing the kernel function, tuning its additional parameters and choosing the regularization parameter.

To search for optimal metaparameters, we need to be able to say whether one particular choice is better than another one. In the next section, standard techniques that can be used to measure the "quality" of a solution will be described.

4.4 Cross-Validation

In this section we describe various cross-validation techniques, which are standard statistical techniques used to estimate the real performance of neural networks [66, 15].

We say that a network is well-trained or that it performs well if it learns enough about the past to be able to generalize to the future.

The past is represented by the given training set. However, we do not typically know anything about the future. Without any prior knowledge about the problem we cannot say if the network is a good solution of the given problem or if it generalizes well. But we can estimate its generalization ability using the available data.

The cross-validation techniques are based on the idea that we split the data into two parts, called a *training set* and a *validation set*. The network is trained on the training set and then the error on the validation set is evaluated. The learning algorithm is then run on the training set with different setups (metaparameters). For each resulting network we evaluate the error on the validation set. The network with the lowest error on the validation set is then picked as a one with the best generalization ability.

There are several types of the cross-validation. In order to describe them, we introduce the following notation:

Definition 4.4.1 *(RN Trained on Data Set S) Let $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subset \mathbb{R}^n \times \mathbb{R}$ be a given data set. Then the regularization network found by the RN learning algorithm (Algorithm 4.1.1) run on the data set S is called* regularization network trained on the data set S *and its function denoted as f^S.*

Definition 4.4.2 *(Error on Data Set S) Let $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subset \mathbb{R}^n \times \mathbb{R}$ be a given data set and f be a regularization network. Then the quantity*

$$E(f, S) = \sum_{i=1}^{N} (f(\mathbf{x}_i) - y_i)^2 \tag{4.6}$$

is called the error on the data set S.

The easiest cross-validation approach is known as the *hold out cross-validation* (see Algorithm 4.4.1). The data points are chosen randomly from the initial sample to form the validation data S_{val} (typically, less than a third of the initial sample), and the remaining observations are retained as the training data S_{train}. The estimate of the generalization error is given by $E_{holdout}$:

$$E_{holdout} = E(f^{S_{train}}, S_{val}). \tag{4.7}$$

```
Input:   Data set $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}$
Output:  The estimate of generalization error $E_{holdout}$

  1. Split the data randomly to two subsets $S_{train}$
     and $S_{val}$:   $S = S_{train} \cup S_{val}$ and $S_{train} \cap S_{val} = \emptyset$

  2. Run the RN learning algorithm on the data set
     $S_{train}$ to obtain $f^{S_{train}}$

  3. $E_{holdout} \leftarrow E(f^{S_{train}}, S_{val})$
```

Algorithm 4.4.1. The hold out cross-validation.

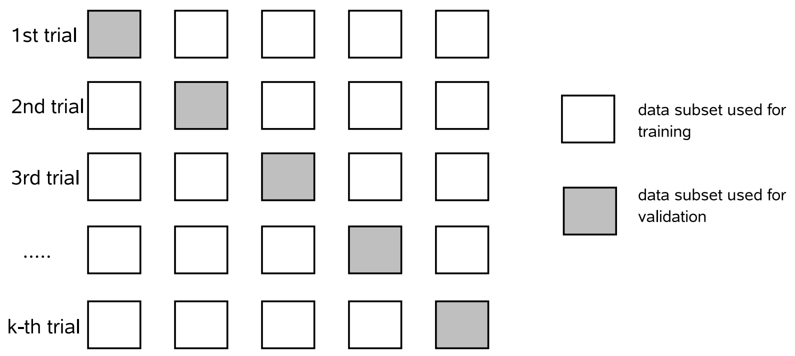
1st trial
2nd trial
3rd trial
.....
k-th trial
data subset used for training
data subset used for validation

Figure 4.1: Schema of the k fold cross-validation procedure.

Another variant, known as the *k-fold cross-validation*, uses partitioning of the original data set S into k subsets $S_1, \ldots, S_k$, so that $\bigcup_i S_i = S$ and $\forall\ i \neq j : S_i \bigcap S_j = \emptyset$. The cross-validation process consists of k trials. In each trial, a single subset is retained as the validation set, and the remaining $k-1$ subsets are used as the training set, so that each of the k subsamples is used exactly once as the validation set. See Figure 4.1 for illustration. Then the k results from the folds can be averaged (or otherwise combined) to produce a single estimation E_{kfolds}:

$$E_{kfolds} = \frac{1}{k} \sum_{i=1}^{k} E(f^{\bigcup_{i \neq j} S_j}, S_i). \tag{4.8}$$

The special case of k-fold cross-validation is the *leave-one-out cross-validation*, where the k is equal to the number of data points in the original data set. In each trial, a single observation from the original set is used as the validation data, and the remaining points as the training set.

The hold out cross-validation is very sensitive to the partitioning of the data into training and validation subsets, and therefore gives us only a very rough estimate of the real generalization ability. On the other hand, the leave-one-out cross-validation may be too time-consuming, except for very small data sets. The k-fold cross-validation represents a compromise between these two approaches and is frequently used.

In the rest of our work, we will use the k-fold cross-validation; and the estimate of generalization error E_{kfold} will be called a *cross-validation error*.

Definition 4.4.3 *(Cross-validation Error) Let $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subset \mathbb{R}^n \times \mathbb{R}$ be a given data set with partitioning $S_1, \ldots, S_k : S = \bigcup_{i=1}^k S_i$ and $\forall\ i \neq j : S_i \bigcap S_j = \emptyset$. We call the quantity*

$$E_{cross}(S) = \frac{1}{k} \sum_{i=1}^{k} E(f^{\bigcup_{i \neq j} S_j}, S_i) \tag{4.9}$$

a cross-validation error.

```
Input:   Data set $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subseteq \mathbb{R}^d \times \mathbb{R}$
         $k \in \mathbb{N}, k > 0$
Output:  The estimate of generalization error $E_{kfold}$

1. Split the data randomly into $k$ subsets $S_1, \dots, S_k$:
       $S = \bigcup_{i=1}^k S_i$ and $\forall\ i \neq j : S_i \bigcap S_j = \emptyset$
2. $E_{kfold} \leftarrow 0$, $i \leftarrow 1$
3. $TS \leftarrow \bigcup_{j \neq i} S_j$
4. Run the RN learning algorithm on the data set
   $TS$ to obtain $f^{TS}$
5. $E_{kfold} \leftarrow E_{kfold} + E(f^{TS}, S_i)$
6. $i \leftarrow i + 1$, if $i <= k$ go to 3
7. $E_{kfold} \leftarrow \frac{1}{k} E_{kfold}$
```

Algorithm 4.4.2. The k-fold cross-validation.

4.5 Adaptive Grid Search

In this section we introduce the algorithm we use for the setup of the RN learning algorithm (Algorithm 4.1.1).

We suppose that the kernel type is given by the user. Then our algorithm searches for the optimal value of the regularization parameter and of additional kernel parameters. Without the loss of generality, we suppose that the kernel has one real parameter p. The application of the algorithm on the cases where the kernel has two or more parameters is straightforward.

We will use the following notation:

Definition 4.5.1 *($f^S_{\gamma,p}$) Let $S = \{\mathbf{x}_i, y_i\} \subset \mathbb{R}^n \times \mathbb{R}$ be a given data set. Then $f^S_{\gamma,p}$ denotes the regularization network found by the RN learning algorithm (Algorithm 4.1.1) with the regularization parameter γ and the kernel parameter p on the data set S.*

Definition 4.5.2 *($E_{cross}(\gamma, p, S)$) Let $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subset \mathbb{R}^n \times \mathbb{R}$ be a given data set with partitioning $S_1, \dots, S_k : S = \bigcup_{i=1}^k S_i$ and $\forall\ i \neq j : S_i \bigcap S_j = \emptyset$. By $E_{cross}(\gamma, p, S)$ we* *denote the cross-validation error*

$$E_{cross}(\gamma, p, S) = \frac{1}{k}\sum_{i=1}^{k} E(f_{\gamma,p}^{\bigcup_{i\neq j} S_j}, S_i). \tag{4.10}$$

We will search for such metaparameters that minimize the cross-validation error (4.10). It means we will choose a solution $f^S_{\gamma^*,p^*}$ such that

$$[\gamma^*, p^*] = \text{argmin}_{\gamma,p} E_{cross}(\gamma, p, S). \tag{4.11}$$

Clearly, it is not possible to go through all possible values of these parameters. Therefore we create a grid of couples $[\gamma, p]$ 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 (Algorithm 4.5.1). It starts with a coarse grid, i.e. sparse sampling, and then creates a finer grid around the point with the minimum.

The winning values of parameters found by the Algorithm 4.5.1 are then used to run the RN learning algorithm (Algorithm 4.1.1) on the whole training set.

```
Input:  Data set $S = \{\mathbf{x}_i, y_i\}_{i=1}^N \subseteq \mathbb{R}^n \times \mathbb{R}$
Output:  Parameters $\gamma$ and $p$.

  1. Create a set of couples $\{[\gamma, p]_i, i = 1, \dots, K\}$,
     uniformly distributed in $\langle\gamma_{min}, \gamma_{max}\rangle \times \langle p_{min}, p_{max}\rangle$.
  2. For each $[\gamma, p]_i$ for $i = 1, \dots, K$ and for each
     couple evaluate the cross-validation error
     $E^i_{cross} \leftarrow E_{cross}(\gamma_i, p_i, S)$.
  3. Select the $i$ with the lowest $E^i_{cross}$.
  4. If the couple $[\gamma, p]_i$ is at the border of the
     grid, move the grid (see Figure 4.2a).
  5. If the couple $[\gamma, p]_i$ is inside the grid,
     create finer grid around this couple (see
     Figure 4.2b).
  6. Go to 2 and iterate until the
     cross-validation error stops decreasing.
```

Algorithm 4.5.1. Adaptive grid search.

The disadvantage of this approach is the high number of evaluations of the Algorithm 4.1.1 needed during the search. Nevertheless, these evaluations are completely independent, so they can be performed in parallel.

We have also observed over-fitting with respect to a particular partitioning to the k parts. By the over-fitting we mean that it often happens that the real generalization ability of the network obtained by using parameters with a lower cross-validation error on the particular partitioning to the k parts may be worse than the one of the network obtained by using parameters with a higher cross-validation error. However, this problem is not crucial, since such an increase in error is typically not significant. It rather justifies that a few adaptive grid search iterations are sufficient.

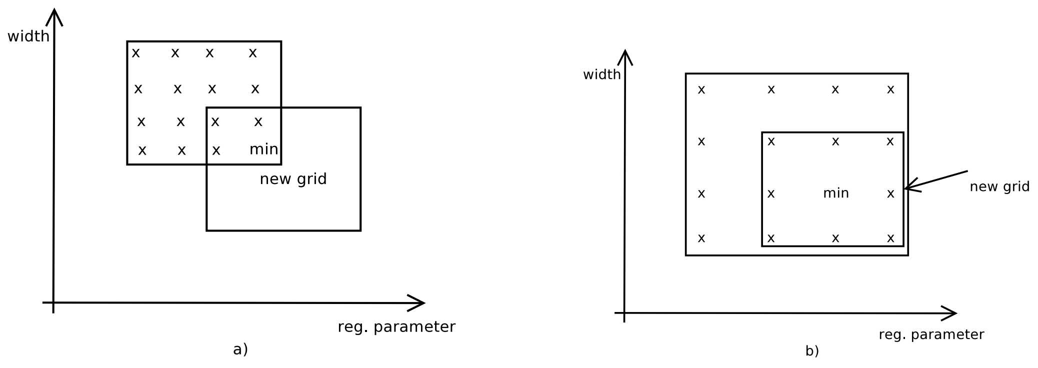

Figure 4.2: a) Move the grid, b) Create a finer grid.

4.6 Genetic Parameter Search

Because the simple adaptive grid search algorithm has several drawbacks (the high number of evaluations needed, danger of over-fitting), we introduce a simple genetic algorithm to our search — the *genetic parameter search* (Algorithm 4.6.1).

The genetic algorithms (GAs) [45, 44] represent a stochastic search technique used to find approximate solutions to optimization and search problems. They belong to the family of evolutionary algorithms that use techniques inspired by evolutionary biology such as mutation, selection, and crossover.

The genetic algorithms typically work with a population of *individuals* representing abstract representations of feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. The better the solution is, the higher the fitness value it gets.

The population evolves towards better solutions. The evolution starts from a population of completely random individuals and iterates in generations. In each generation, the fitness of each individual is evaluated. Individuals are stochastically selected from

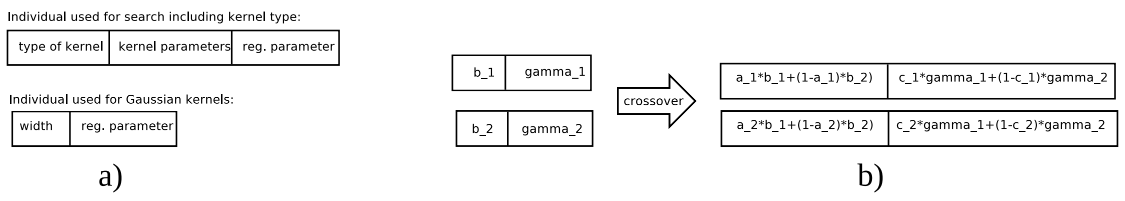
Individual used for search including kernel type:
type of kernel
kernel parameters
reg. parameter
Individual used for Gaussian kernels:
width
reg. parameter
a)
b_1
gamma_1
b_2
gamma_2
crossover
a_1*b_1+(1-a_1)*b_2)
c_1*gamma_1+(1-c_1)*gamma_2
a_2*b_1+(1-a_2)*b_2)
c_2*gamma_1+(1-c_2)*gamma_2
b)

Figure 4.3: a) Individuals, b) Crossover for Gaussian kernels.

the current population (based on their fitness), and modified by means of operators *mutation* and *crossover* to form a new population. The new population is then used in the next iteration of the algorithm.

We work with individuals coding the parameters of the RN learning algorithm. They are the kernel function type, its additional parameters, and the regularization parameter, see Figure 4.3a. When the kernel function type is known in advance, the individual consists only of the kernel parameter and regularization parameter.

Since we want to minimize the cross-validation error, the fitness should reflect it. So the lower the cross-validation error is, the higher the fitness value is.

Because the evaluation of fitness is very expensive in terms of time requirements (many evaluations are needed to compute the cross-validation error), we use the *lazy evaluations* [6]. By the lazy evaluations we mean that each time the fitness of the individual is being computed, only one part is evaluated (i.e. one trial of k-fold cross--validation is performed). In other words, instead of the cross-validation error we compute only its estimate. Each time the fitness is evaluated, another part is computed and this estimate is made more accurate. In addition, this enables us to avoid the over-fitting by selecting a new random partitioning each time the fitness is evaluated.

New generations of individuals are created by using the operators of *selection*, *crossover* and *mutation*. Mutation introduces a small random perturbation to the existing individuals. The crossover (Figure 4.3b) creates two new individuals from two existing individuals by choosing new parameter values randomly in the interval formed by the old values. Classical roulette-wheel selection is used (the higher the fitness is, the higher the probability of being selected is).

When we also search for the kernel function type, then the population consists of different types of individuals (species) and we have two possibilities of *crossover*. The former one works as it was described, only with the individuals of the same type, so we have to ensure that individuals of the same kind are always selected. The latter one combines different kinds of kernels together by using the operator product and sum as described in the previous chapter. We recommend to use the first type of crossover or allow simple combinations of kernels only, otherwise the search time-requirements may increase significantly.

```
Input:   Data set $\{\mathbf{x}_i, y_i\}_{i=1}^N \subseteq \mathbb{R}^n \times \mathbb{R}$
Output:  Parameters $\gamma$ and $p$.

 1. Create randomly an initial population $P_0$ of $M$
    individuals.

 2. Reset individuals' counters.
    for $i = 1, \ldots, M : c_i = 0, err_i = 0$

 3. $i \leftarrow 0$

 4. $P_{i+1} \leftarrow$ empty set

 5. $I_1 \leftarrow selection(P_i); I_2 \leftarrow selection(P_i)$

 6. with probability $p_{cross}$:   $(I_1, I_2) \leftarrow crossover(I_1, I_2)$

 7. with probability $p_{mutate}$:   $I_k \leftarrow mutate(I_k), k = 1, 2$

 8. insert $I_1, I_2$ into $P_{i+1}$

 9. if $P_{i+1}$ has less then $M$ individuals goto 5

10. for $j = 1, \ldots M : I_j \in P_{i+1}$
    divide the data set randomly to $T_{train}$ and $T_{val}$
    let $\gamma_j, p_j$ be the parameter values defined by $I_j$   :
    $err_j \leftarrow err_j + E(f_{\gamma_j, p_j}^{T_{train}}, T_{val})$
    $c_j \leftarrow c_j + 1$
    fitness($I_j$) $\leftarrow C - \frac{1}{c_j} err_j$

11. $i \leftarrow i + 1$

12. goto 4 and iterate until the fitness stops
    increasing
```

Algorithm 4.6.1. Genetic parameter search.

Chapter 5

Radial Basis Function Networks

Ask not what mathematics can do for biology,
but what biology can do for mathematics.
Stanislaw Ulam

In this chapter we deal with a more universal variant of regularization networks, known as *generalized regularization networks*. Particularly, we will focus on one subclass of generalized regularization networks — *radial basis function networks* (RBF networks), and their learning algorithms.

In the next section, a generalized regularization network is defined. In Section 5.2 RBF networks, a subclass of generalized regularization networks, are introduced. Sections 5.3, 5.4, and 5.5 describe the RBF network learning algorithms: *gradient learning*, *three-step learning*, and *genetic learning*, respectively. Finally, in Section 5.6 hybrid learning methods are discussed.

5.1 Generalized Regularization Networks

Throughout the previous chapters we were dealing with the regularization networks, whose architecture represents an exact solution of the regularized learning problem. The regularization network has a form of a linear combination of kernel functions, where the number of kernel functions corresponds to the number of the data points in the corresponding training set, and the centers of the kernel functions are fixed to these data points.

Such an approach benefits from a straightforward learning algorithm, since only the weights of linear combinations have to be estimated. However, the constraint on the number of kernel functions limits the algorithm applicability. Large data sets lead to the solutions of unreasonable size and a time-consuming learning phase, which makes this approach unfeasible.

What size of network is acceptable always depends on the particular application. In some applications, the goal is to obtain a correct approximation and the size of the solution is not important. On the other hand, in many applications the goal is to replace a large data set by its model of a reasonably small size.

In the situations where the size of the regularization network representing the optimal solution is too large, we can search for an approximate solution within the set of networks with a limited number of kernel functions.

Poggio, Girosi, and Jones [12, 58] proposed to use the term a *generalized regularization network* for a wide class of functions representable by a feed-forward neural network with one hidden layer and a linear output layer.

Definition 5.1.1 *(Generalized Regularization Network) A* generalized regularization network *is a feed-forward neural network with one hidden layer containing kernel units and a linear output layer. It represents a function*

$$f(\mathbf{x}) = \sum_{i=1}^{k} w_i K_{p_i}(\mathbf{x}, \mathbf{c_i}), \tag{5.1}$$

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

A generalized regularization network has typically much fewer hidden units than the corresponding data set size. On the other hand, each hidden unit has its own parameter p_i modifying the kernel function. For instance, if the kernel function is the Gaussian, different hidden units realize the Gaussian functions with different widths.

In the case of a regularization network, the optimal values of network parameters were given by the data set and the corresponding linear system. In the case of a generalized regularization network not only weights have to be estimated, but also the centers and kernel parameters. Typically, the values for those additional parameters are found by various heuristics.

The generalized regularization networks cover a wide range of function classes corresponding to the different classes of prior assumptions and corresponding stabilizers. The most known classes are RBF networks, tensor product splines and additive splines (see the examples of stabilizers in Section 2.3.1).

5.2 Radial Basis Function Networks

The history of *radial basis function networks* can be traced back to the 1980s, particularly to the study of interpolation problems in numerical analysis. It is where the radial basis functions were first introduced, in the solution of the real multivariate interpolation problem [60, 39].

A *radial function* is a function that is determined by its *center* and its output depends only on the distance of the argument from this center. In a 2-dimensional space with the Euclidean metric, the points with the same output values lay on circles.

Definition 5.2.1 *(Radial Function) Let $f : \mathbb{R}^d \to \mathbb{R}$ be a function*

$$f(\mathbf{x}) = \varphi(||\mathbf{x} - \mathbf{c}||^2),$$

where $\varphi : \mathbb{R} \to \mathbb{R}$ and $|| \cdot ||$ is a suitable norm (typically the Euclidean norm). Then f is called a radial function *and* $\mathbf{c}$ *is called a* center.

The study of radial basis functions was followed by the introduction of a new type of neural network — an *RBF network* [58, 46, 5]. The RBF network is realized as a linear combination of basis functions and represents an alternative to the classical models, such as multilayer perceptrons. Besides its motivation coming from numerical analysis, it was inspired by the presence of many local response units in human brain.

Both the biological and numerical motivation meet with the regularization theory that created the theoretical background for the RBF network architecture. The regularization approach with radial stabilizers leads to the regularization networks with radial basis functions in their hidden layer (see Section 2.3.1). In general, the RBF network belongs to the family of generalized regularization networks.

The hidden layer of an RBF network consists of *RBF units* realizing a particular radial basis function. We consider the radial basis function using a general weighted norm [15].

Definition 5.2.2 *(Weighted Norm) Let $\mathbf{C}$ be a $d \times d$ matrix. Then the norm defined as*

$$\parallel \mathbf{x} \parallel^2_{\mathbf{C}} = (\mathbf{C}\mathbf{x})^T(\mathbf{C}\mathbf{x}) = \mathbf{x}^T\mathbf{C}^T\mathbf{C}\mathbf{x} \tag{5.2}$$

is called a weighted norm *determined by the matrix* $\mathbf{C}$.

It can be seen that the Euclidean norm is a special case of a weighted norm determined by the identity matrix. For the sake of simplicity, we will use the symbol $\mathbf{\Sigma^{-1}}$ instead of $\mathbf{C}^T\mathbf{C}$.[1] In order to use a weighted norm each RBF unit has another additional parameter, a matrix $\mathbf{C}$.

[1]The reason for such notation is that it is inverse of covariance matrix Σ of a multivariate Gaussian distribution represented by the corresponding hidden unit.

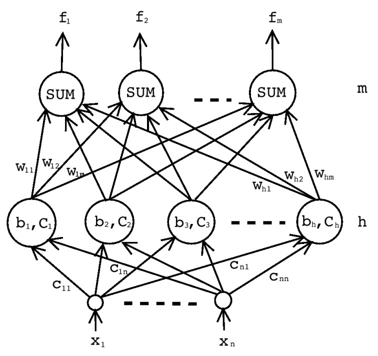

Figure 5.1: An RBF network.

Definition 5.2.3 *(RBF Unit) An* RBF unit *is a neuron with multiple real inputs* $\mathbf{x} = (x_1, \ldots, x_d)$ *and one real output* o*, realizing a function*

$$o(\mathbf{x}) = \varphi(\xi); \qquad \xi = \frac{\| \mathbf{x} - \mathbf{c} \|_{\mathbf{C}}}{b},$$

where $\varphi : \mathbb{R} \to \mathbb{R}$ *is a radial basis function,* $\mathbf{c} \in \mathbb{R}^d$ *is a* center*,* $b \in \mathbb{R}$ *is a* width*, and* $\mathbf{C}$ *is a matrix defining the weighted norm.*

Definition 5.2.4 *(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:* $\mathbf{f} = (f_1, \ldots, f_s, \ldots, f_m) : \mathbb{R}^d \to \mathbb{R}^m$:

$$f_s(\mathbf{x}) = \sum_{j=1}^{h} w_{js} \varphi \left(\frac{\| \mathbf{x} - \mathbf{c}_j \|_{\mathbf{C}_j}}{b_j} \right), \tag{5.3}$$

where $w_{ji} \in \mathbb{R}$ *and* φ *is a radial basis function (see Figure 5.1).*

The RBF networks benefit from a rich spectrum of learning possibilities. In the following sections, we will describe three main learning approaches — the gradient learning, three-step learning, and genetic learning. The study of these algorithms together with experimental results was also published in our papers [54, 53, 21].

All considered learning algorithms suppose that the number of hidden units is given in advance. According to the Cover's theorem [8], a classification problem cast in a high-dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space. Therefore it is recommended for the number of hidden units to be higher than the dimension of the input space.

Since the RBF networks have generally multiple outputs, we consider the training set in the form

$$T = \{(\mathbf{x}_i, \mathbf{y}_i); i = 1, \ldots, N, \mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \mathbb{R}^m\}, \tag{5.4}$$

and work with the following error function, called a *training error*:

$$E = \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{m} (f_j(\mathbf{x}_i) - y_{ij})^2. \tag{5.5}$$

Minimizing the error function on the training set is always accompanied with the danger of over-fitting. Different kinds of regularization can be used to prevent the over--fitting, such as penalizing the networks with large values of weights or large second derivatives [3].

Another possibility is to keep a part of the data apart, as an *evaluation set*, and stop the learning when the error on the evaluation set starts increasing [15]. This is applicable for iterative algorithms, such as gradient learning and genetic learning.

In our experiments, we will also show that the danger of over-fitting increases with the number of hidden units. So the control of network size is a very easy option to prevent the over-fitting.

5.3 Gradient Learning

The most straightforward approach to the RBF network learning is based on the well--known back-propagation algorithm for the multilayer perceptron (MLP) [63, 74, 15]. 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 formally similar structure as the MLP, it can be trained by the modification of the back-propagation algorithm, see Algorithm 5.3.1.

As the RBF network has only one hidden layer, evaluating the derivatives (5.6)–(5.9) is rather simple:

$$\frac{\partial E}{\partial w_{kq}} = \sum_{i=1}^{N} e_q^i \varphi_k(\mathbf{x}_i) \tag{5.6}$$

$$\frac{\partial E}{\partial \mathbf{c}_k} = -\frac{\Sigma_k^{-1}}{b_k} \sum_{i=1}^{N} \frac{[\mathbf{x}_i - \mathbf{c}_k]}{\| \mathbf{x}_i - \mathbf{c}_k \|_{\mathbf{C}_k}} \varphi_k'(\mathbf{x}_i) \sum_{s=1}^{m} e_s^i w_{ks} \tag{5.7}$$

$$\frac{\partial E}{\partial b_k} = -\frac{1}{b_k^2} \sum_{i=1}^{N} \| \mathbf{x}_i - \mathbf{c}_k \|_{\mathbf{C}_k} \varphi_k'(\mathbf{x}_i) \sum_{s=1}^{m} e_s^i w_{ks} \tag{5.8}$$

$$\frac{\partial E}{\partial \Sigma_k^{-1}} = \frac{1}{2b_k} \sum_{i=1}^{N} \frac{[\mathbf{x}_i - \mathbf{c}_k][\mathbf{x}_i - \mathbf{c}_k]^T}{\| \mathbf{x}_i - \mathbf{c}_k \|_{\mathbf{C}_k}} \varphi_k'(\mathbf{x}_i) \sum_{s=1}^{m} e_s^i w_{ks}, \tag{5.9}$$

where $e_q^i = f_q(\mathbf{x}_i) - y_{iq}$, $\varphi_k(\mathbf{x}_i) = \varphi\left(\frac{\|\mathbf{x}_i - \mathbf{c}_k\|_{\mathbf{C_k}}}{b_k}\right)$, $\varphi'_k(\mathbf{x}_i) = \varphi'\left(\frac{\|\mathbf{x}_i - \mathbf{c}_k\|_{\mathbf{C_k}}}{b_k}\right)$, and φ' denotes a derivative of φ.

The gradient descent algorithm is a basic optimization method, nowadays usually more sophisticated algorithms are used. We use a gradient descent enhanced with a momentum term [57] (see the step 3 of Algorithm 5.3.1) for the stepwise parameter modifications.

Note that the gradient descent algorithm is a local search algorithm. It depends on the random initialization and suffers from local minima. Therefore several different initializations should be tried and the best solution picked.

```
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:  $s = 1, \cdots, m$ a $k = 1, \cdots, h$

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

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

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

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

Algorithm 5.3.1. Gradient learning.

More details on the gradient learning algorithm and a detailed description of our algorithm for the computation of derivatives (5.6)–(5.9) can be found in [21].

5.4 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 ([16], [46]).

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 units parameters (widths, weighted norm matrices) are estimated. During the third step the output weights are determined. The algorithm is listed in Algorithm 5.4.1.

The goal of the first step is to distribute the hidden units in the input space so that the positions of the centers reflect the density of the data points. The centers $\mathbf{c}_i$ are set up so that they minimize the quality

$$E_{VQ} = \sum_{i=1}^{N} ||\mathbf{x}_i - \mathbf{c}_{j_{x_i}}||^2, \text{ where } j_{x_i} = \text{argmin}_j ||\mathbf{x}_i - \mathbf{c}_j||^2, \tag{5.10}$$

where $\mathbf{x}_i$ are the data points. This can be done by using various clustering or vector quantization techniques, such as the k-means algorithm [13].

The second step sets up the widths and weighted norm matrices — if they are present. These parameters determine the size and the shape of the area controlled by the unit.

The suitable parameter values can be found by the gradient minimization of the function

$$E(b_1 \cdots b_h; \Sigma_1^{-1} \cdots \Sigma_h^{-1}) = \frac{1}{2} \sum_{r=1}^{h} \left[\sum_{s=1}^{h} \varphi\left(\xi_{sr}\right) \xi_{sr}^2 - P \right]^2 \tag{5.11}$$

$$\xi_{sr} = \frac{\parallel \mathbf{c}_s - \mathbf{c}_r \parallel_{\mathbf{C}_r}}{b_r},$$

where P is the parameter controlling the overlap between the areas of importance belonging to the particular units [47].

In case of widths we can get around the minimization by using simple heuristics. The one used most often is called the q-neighbors rule, and it simply sets the width proportionally to the average distance of q nearest neighboring units (q is typically small number, such as 2 or 3).

The third step is supervised learning known from the multilayer perceptron networks reduced to a linear regression task. The only parameters to be set are the weights between the hidden and the output layer, which represent the coefficients of the linear combinations of RBF units outputs. Our goal is to minimize the overall error function (5.5) with respect to the weights w_{ij}.

It can be achieved by using gradient minimization or directly solving the linear system:

$$\mathbf{PW} = \mathbf{Y}, \tag{5.12}$$

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

As we typically have more training samples than hidden units, the system (5.12) is overdetermined. So any of various least square methods is used to find the solution.

The weights can be computed as

$$\mathbf{W} = \mathbf{P}^{+}\mathbf{Y}, \tag{5.13}$$

where $\mathbf{P}^{+} = (\mathbf{P}^{\mathbf{T}}\mathbf{P})^{-1}\mathbf{P}^{\mathbf{T}}$ is a *Moore-Penrose pseudo-inverse* [48, 56]. Alternatively, the methods based on SVD or QR decomposition [62] can be used.

It is true, however, that the success of this learning step depends on the previous steps.

```
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 vector
   quantization
2. Set up widths $b_i$ and matrices $\Sigma_i^{-1}$ for $i = 1, \ldots, h$ by
   minimization of (5.11)
3. Find the values for $w_{js}$ for $j = 1, \ldots, h$ and $s = 1, \ldots, m$
   by solving linear system (5.12) using a least square
   algorithm.
```

Algorithm 5.4.1. Three-step learning.

5.5 Genetic Learning

The third learning method presented here is based on the genetic algorithms (GAs). A reader who is not familiar with GAs is kindly ask to read Section 4.6.

To apply the GAs to RBF network learning, one has to devise a suitable way of encoding the parameters and adopt the genetic operators to work on corresponding individuals.

Unlike the traditional GAs approaches, we use a direct float encoding for the RBF network parameters. An individual is formed by a sequence of blocks. Each block contains a vector of values of one RBF unit parameters. See Figure 5.2.

The 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[2]. The 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. In our algorithm we use the standard *roulette-wheel* selection.

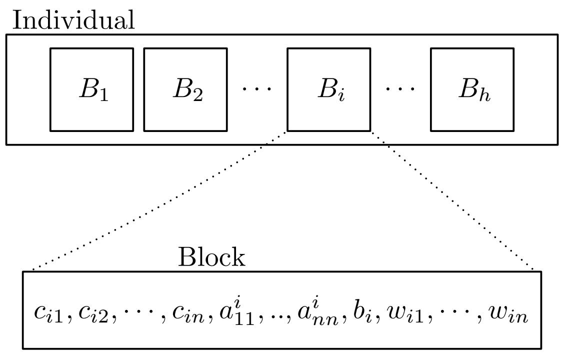
Individual
B_1
B_2
$\cdots$
B_i
$\cdots$
B_h
Block
$c_{i1}, c_{i2}, \cdots, c_{in}, a^i_{11}, .., a^i_{nn}, b_i, w_{i1}, \cdots, w_{in}$

Figure 5.2: An individual representing an RBF network.

The crossover operator composes a pair of new individuals combining parts of two old individuals. First, a crossover point is randomly chosen in the both individuals, and then the corresponding parts of individuals are swapped (see Algorithm 5.5.1). The positive effect of the crossover is the creation of new solutions recombining the current individuals.

```
Input:   $I_1 = \{B^1_1, \ldots, B^1_h\}$
         $I_2 = \{B^2_1, \ldots, B^2_h\}$
Output:  $I^*_1, I^*_2$

1. $k_{cross} \leftarrow random(h)$
2. $I^*_1 \leftarrow \{B^1_1, \ldots, B^1_{k_{cross}}, B^2_{k_{cross}+1}, \ldots, B^2_h\}$
3. $I^*_2 \leftarrow \{B^2_1, \ldots, B^2_{k_{cross}}, B^1_{k_{cross}+1}, \ldots, B^1_h\}$
```

Algorithm 5.5.1. Crossover.

[2]In the context of the GAs we often speak about the fitness of the individual. The error plays the same role, with the difference that a low error corresponds to a high fitness, and vice versa.

Finally, the mutation operator represents small local random changes of an individual (see Algorithm 5.5.2). Both the crossover and mutation are applied with certain probabilities only.

```
Input:   $I = \{B_1, \ldots, B_h\}$
Output:  $I^*$

1. $k \leftarrow random(h)$
2. $B_k^* \leftarrow B_k$
3. for $p$ in $B_k^* = \{c_{k1}, \ldots, c_{kd}, a_{11}^k, \ldots, a_{dd}^k, b_k, w_{k1}, \ldots, w_{km}\}$
   do
       $\delta \leftarrow random(-1.0, 1.0)$
       $p \leftarrow p + \delta$
   done
4. $I^* \leftarrow \{B_1, \ldots, B_k^*, \ldots, B_h\}$
```

Algorithm 5.5.2. Mutation.

The sketch of the genetic learning is listed in Algorithm 5.5.3. The GAs are 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 the problems with bigger individuals resulting in a huge search space.

Besides the standard GAs tailored to the RBF networks we have also implemented the canonical version of genetic learning described in [50]. This algorithm modifies the crossover and mutation operators in such a way that they operate on a minimal search space.

For more details on the implementation of the genetic learning and its variants see [21, 54, 24, 23].

5.6 Hybrid Methods

The three described algorithms represent three main branches of the wide range of RBF learning algorithms. For each approach, many variants and various modifications exist.

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. The hybrid approaches based on combinations of the well-known algorithms may achieve a synergy effect and thus over--perform the single algorithms.

```
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 initial 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 current population
   is small enough, stop and return the parameters
   coded in the individual with the minimal error.
4. Create 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 5.5.3. Genetic learning.

In this section, we introduce two hybrid approaches — the *hybrid genetic learning* and the *four-step learning algorithm*.

The genetic learning can be combined with the other algorithms in various ways. In particular, the GAs can be used to perform the first one or two steps in the three-step learning.

In the former case, GAs are applied to solve the vector quantization problem of the first step, i.e. to find the centers minimizing the error (5.10). The application of the GAs is similar as in the genetic learning, but the individual codes only the network centers and the corresponding error is evaluated according to (5.10).

The latter case replaces both the first and second step by the GAs. The third step setting the output weights is performed by a linear optimization technique. 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.

Such an approach is called *hybrid genetic learning*. It uses the same encoding as the genetic learning, except that the individual encodes only the hidden layer, not the output weights. To evaluate the error associated with the individual, we first have to find the weights optimal for the corresponding hidden layer, and then evaluate the error of the obtained network. See Algorithm 5.6.1. More details on the hybrid genetic learning can be found in [54].

The second hybrid approach 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. This algorithm is called *four-step learning*. See Algorithm 5.6.2. More details can be found in [53].

```
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 5.6.1. 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. $\tau := 0$
   Run the Algorithm 5.4.1 to setup $\mathbf{c}_k(0)$, $b_k(0)$,
   $\Sigma_k^{-1}(0)$, $w_{ks}(0)$ 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:  $s = 1, \cdots, m$ a $k = 1, \cdots, h$

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

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

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

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

Algorithm 5.6.2. Four-step learning.

Chapter 6

Experiments

Errors using inadequate data are much
less than those using no data at all.
Charles Babbage

In this, chapter our experimental study of methods and algorithms described in the previous chapters is represented. In the next section we explain our motivation and set our goals. Section 6.2 describes data sets and methodology we have used in our experiments. In Section 6.3 we present experiments illustrating regularization network behavior. Section 6.4 describes experimental results concerning various learning methods for RBF networks. Then, Section 6.5 compares the regularization network and RBF network approach. A real-life problem, the prediction of river flow rate, handled by both the regularization networks and RBF networks, is presented in Section 6.6. Finally, Section 6.7 summarizes the experimental results.

6.1 Experimental Study of Learning Algorithms

The study of machine learning and neural networks has both theoretical and empirical aspects. In general, the goal of experimental study is deeper understanding of behaviors and the conditions under which they occur. In the case of machine learning, the behavior is an ability to learn and generalize, and the conditions are learning algorithms and domain knowledge [37].

The learning algorithms described in this work benefit from a very good theoretical background, since they are based on the regularization theory. Regularization networks possess a rigorous derivation, while RBF networks as generalized regularization networks are based also on heuristical approaches. We believe that for the both approaches the experimental study can further improve our understanding of these algorithms and their behavior. The goal of our experiments is to verify the theoretical results and fill in the gap between the theory and practice.

No matter how strong the theoretical background of a particular learning algorithm is, in practice we always meet numerical inaccuracies, round-off errors and other constraints given by the hardware limits of contemporary computers. The experiments should provide an additional source of information that can be used together with the theoretical results before applying the individual learning algorithm to a real-life problem.

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 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‘.

The results answering the tasks stated by our goals can be found in the following sections. The Section 6.3 deals with Goal 1. Experiments regarding Goal 2 are presented in Subsection 6.3.1; and Goal 3 is tackled in Subsection 6.3.3. Subsection 6.3.4 and Subsection 6.3.5 are devoted to Goal 4. Results concerning Goal 5 are presented in Section 6.4. Finally, Goal 6 is studied in Section 6.5. In addition, in Section 6.6 the regularization networks and RBF networks will be applied on the prediction of flow rate on the Czech river Ploučnice as an example of a real-life application.

6.2 Methodology and Data

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.

Task name	n	m	N_{train}	N_{test}	**Type**
cancer	9	2	525	174	class
card	51	2	518	172	class
diabetes	8	2	576	192	class
flare	24	3	800	266	approx
glass	9	6	161	53	class
heartac	35	1	228	75	approx
hearta	35	1	690	230	approx
heartc	35	2	228	75	class
heart	35	2	690	230	class
horse	58	3	273	91	class
soybean	82	19	513	170	class

Table 6.1: Overview of Proben1 tasks. Number of inputs (n), number of outputs (m), number of samples in training and testing sets (N_{train},N_{test}). Type of task: approximation or classification.

PROBEN1 [61] is a repository of benchmark data sets intended for experiments with neural networks. It contains approximation as well as classification tasks. Most of the tasks are also available in the UCI machine learning repository [10].

Table 6.1 gives a summary of the tasks from PROBEN1. Each task is present in three variants, three different partitioning into training and testing data. We refer to this variants with suffix 1,2, or 3 (e.g. CANCER1, CANCER2, CANCER3). More details of the individual data sets, their source, and qualities can be found in [61].

Two spirals is an artificial two-dimensional classification problem of two intertwined spirals. The training set contains 194 data points (see Figure 6.1).

The *Lenna* image [17] was used as an approximation task. Our training set contains 2500 samples forming the image of 50×50 pixels, see Figure 6.2.

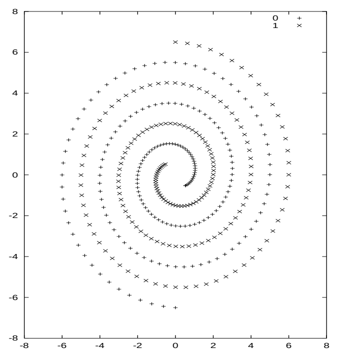

Figure 6.1: Two spirals problem.

Figure 6.2: Lenna.

The real-life tasks are represented by river flow rate prediction. The data set contains samples for each day consisting of the present river flow rate and total rainfall on the Czech river Ploučnice. The goal is to predict the current flow rate from the previous values of flow rate. Several variants of the task were created by preprocessing. They are described in Section 6.6.

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 error of the resulting network.

If not stated otherwise, the following procedure is used for experiments with RNs:

1. find the values for γ and the kernel's parameters with the lowest cross-validation error on the training set,
2. use the whole training set and the parameters found by Step 1 to estimate the weights of the RN,
3. evaluate the error on the testing set.

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:

$$E = 100 \frac{1}{Nm} \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i)||^2, \tag{6.1}$$

where $|| \cdot ||$ denotes the Euclidean norm.

In the following text, we will use the notation E_{train} and E_{test} for the error computed over the training seta and test set, respectively.

The experiments have been performed in our distributed multi-agent system called Bang [51, 2], the standard numerical library LAPACK [38] was used to solve linear least square problems (step 2 in Algorithm 4.1.1).

For the experimental study of learning algorithms, more than one single evaluation of the learning algorithm is needed. Each algorithm has to be evaluated under different conditions, i.e. on different data sets and with different setups.

Most experiments were run on the computer clusters *Lomond* [41], *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 refers to an Intel Xeon 2.80 GHz processor. The times are in $[h] : m : s$ format, where h stands for hours, m for minutes and s for seconds; and are they are rounded up to seconds.

6.3 Regularization Networks

In this section we present the results of our experiments with regularization networks. First we demonstrate the role of the regularization parameter and the kernel function, and show that their choice is crucial for the performance of the RN learning algorithm. Then, in Subsection 6.3.2, we demonstrate a behavior of algorithms for the setup of these metaparameters. In Subsection 6.3.3, the most common kernel functions are compared. Subsection 6.3.4 describes the results of experiments with product and sum kernels. Subsection 6.3.5 demonstrates the advantages of the *Divide et Impera* approach. Finally, Subsection 6.3.6 gives the brief summary of results presented throughout this section.

6.3.1 Role of Regularization Parameter and Kernel Function

The aim of this experiment was to demonstrate the role of metaparameters in the RN learning and to illustrate how they influence the results.

First, we tried to run the RN learning algorithm (Algorithm 4.1.1) on the tasks from PROBEN1. A regularization network with an Gaussian kernel was used. The setup was done manually and different values of the regularization parameter and width were used.

Figure 6.3 shows the dependency of errors in the training set and the test set on the value of the regularization parameter γ. In addition, the value of the condition number of the corresponding linear system is displayed. This particular image was made on the data set HEARTA1.

We can see that the error on the training set increases with the increasing value of the regularization parameter γ. On the other hand, the error on the test set is high when the regularization parameter is close to zero; then it decreases with the increasing regularization parameter and at some point starts to increase again. This behavior corresponds to the trade-off between the data term and the regularization term in the minimized functional (see Section 2.2). If the regularization parameter is too small, we observe over-fitting; if it is too high, the data term has no influence on the solution and both the training and test errors are increasing.

Note also the condition number. It decreases with the increasing regularization parameter. That is a simple consequence of the increasing regularization term that makes the diagonal dominant.

Figure 6.4 illustrates the test error dependency both on the regularization parameter γ and the width of the Gaussian kernel b. It uses the two-dimensional plot with contours. The darker colors (red) correspond to the lowest values, the brighter colors (yellow, white) to higher values. So we can see that the optimal parameters are of width 0.3 and the regularization parameter about 0.001.

The lowest values of the error function are obtained with the widths between 0.2 and 0.4. For the widths of 0.1 and smaller, and 0.7 and higher, the error is significantly

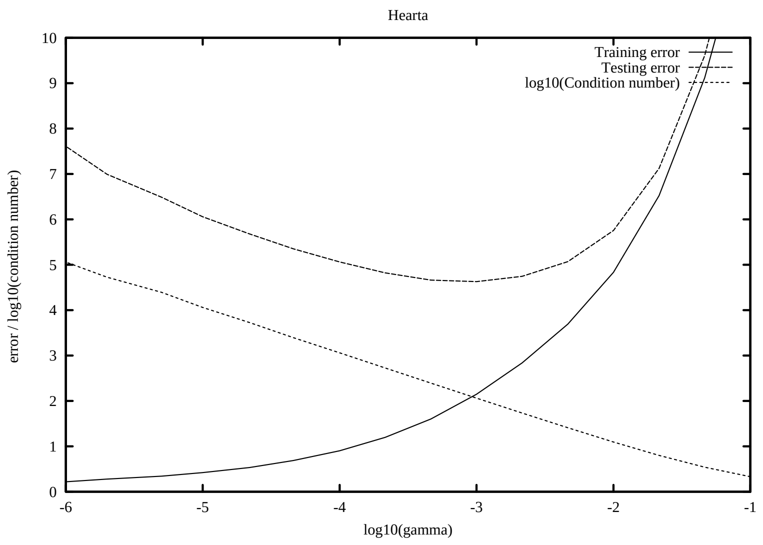

Figure 6.3: Dependency of errors (on the training and test data sets) and the condition number of the corresponding linear system on the regularization parameter γ. Obtained by a regularization network with the Gaussian kernel on the task HEARTA1.

higher. It shows that the width of the Gaussian function influences the results as well as the regularization parameter.

The plot is generated from the results obtained on the data set GLASS1.

Second, we have chosen an approximation of the Lenna image. The training set LENNA contains 2500 samples representing the image of 50×50 pixels. Again, we have run the RN learning algorithm with different setups. The obtained regularization networks were used to generate a 100×100 image.

Figure 6.5 displays the resulting images. Again, the higher the regularization parameter is, the smoother the result is. For too high regularization parameters, we get black images, as in the lowest row.

The choice of the Gaussian kernel width also plays its role. Note the leftmost column. The widths are too small, so for the inputs not present in the training set we get almost zero output from the hidden layer, which leads to black strips. Clearly, such a problem cannot be cured by the regularization parameter.

Both the experiments illustrate the influence of the regularization parameter and the kernel function on the regularization network performance. They show that the choice of these parameters is crucial for successful learning; one cannot choose arbitrary values and some kind of search for optimal metaparameters is necessary. In addition, a wrong choice of the kernel function (such as narrow Gaussians for the approximation of Lenna image) cannot be cured by changing regularization parameter, and might result in completely useless solution.

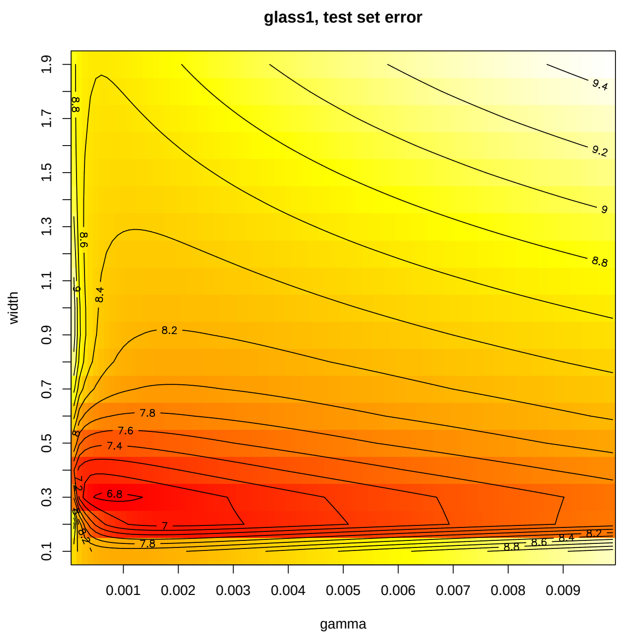

Figure 6.4: Dependency of error on the test data set on the choice of regularization parameter γ and width b. Obtained by a regularization network with the Gaussian kernel on the task GLASS1.

6.3.2 Setup of the Metaparameters

This experiment demonstrates a behavior of the setup of the metaparameters, namely of the adaptive grid search algorithm (see Algorithm 4.5.1) and the genetic parameter search algorithm (see Algorithm 4.6.1).

These algorithms were tested on the tasks from PROBEN1 repository, the task GLASS1 was chosen randomly for illustration of the results. Both algorithms were run on this task. After each iteration the computation was stopped and the test error was evaluated using the actual best metaparameters.

A run of the adaptive grid search algorithm on the GLASS1 task is displayed in Figure 6.6. The cross-validation error decreased significantly during the first two iterations of the algorithm (i.e. about 40 evaluations of cross-validation error), then the decrease is very small. The test error also decreased during the first 40 evaluations, then it oscillates.

The genetic parameter search is illustrated in Figure 6.7, which was also generated from a run on the GLASS1 task. The population had only 5 individuals. The 100 generations were computed, but the cross-validation error and test error decrease is not significant during the last 30 generations.

The lazy evaluations mentioned in Section 4.6 were not used. We have tested them on the tasks from PROBEN1 repository, and they typically cause divergence of the al-

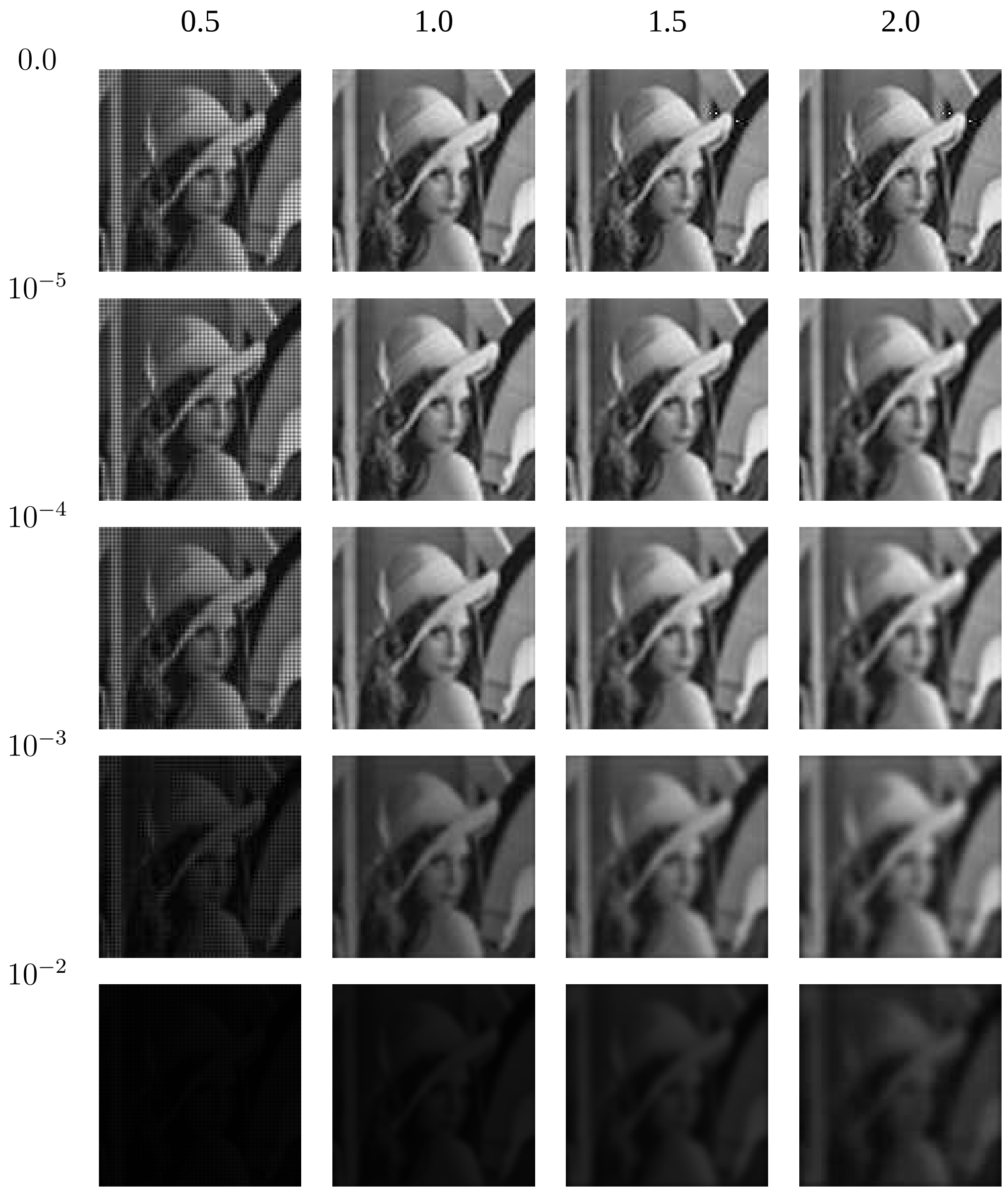

Figure 6.5: Images generated by the regularization network learned on the Lenna image (50×50 pixels) using Gaussian kernels with the widths from 0.5 to 2.0 and the regularization parameters from 0.0 to 0.01.

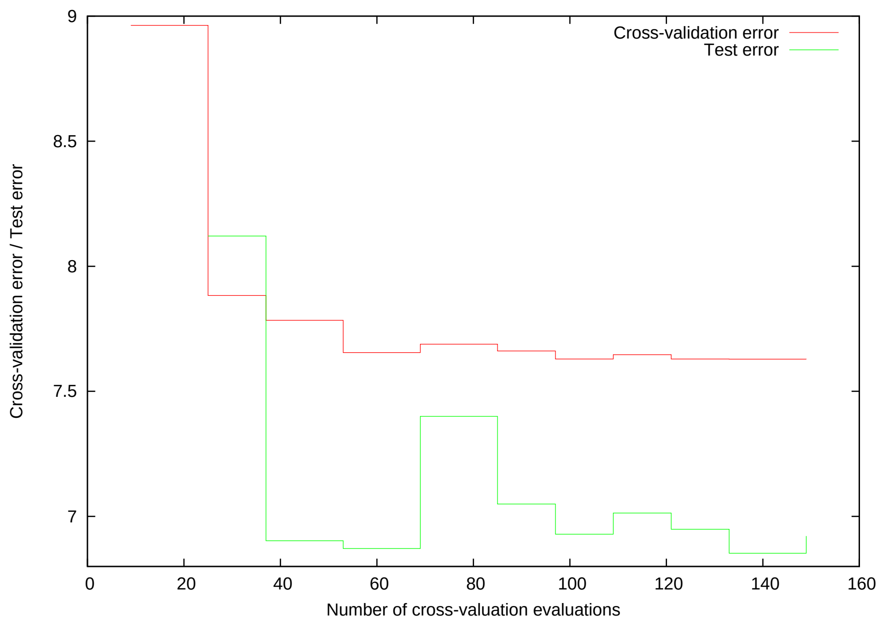

Figure 6.6: The cross-validation error and the corresponding test error during a run of the adaptive grid search algorithm (Algorithm 4.5.1) on the GLASS1 task.

gorithm. It is probably caused by the fact that all these tasks are rather small so the differences between the individual partitioning in the k-fold cross-validation procedure are too high.

Both algorithms achieved the similar test error (around 6.9). However, the genetic parameter search needed much more evaluations of cross-validation error (100 generations corresponds to 500 evaluations, while the adaptive grid search needed around 60).

The time requirements needed for 10 evaluations of the cross-validation error on tasks from PROBEN1 repository are listed in Table 6.2. The table also contains the time requirements of the final run[1] of the RN learning algorithm (Algorithm 4.1.1).

In all our following experiments, we use three iterations (54 evaluations) of the adaptive grid search algorithm for the setup of the metaparameters. The genetic parameter search might be useful for tasks where more parameters are sought (i.e. more complicated composite kernels with several parameters), so the grid has more dimensions and the adaptive grid search would need more evaluations.

[1]i.e. one run of the algorithm on the whole training set

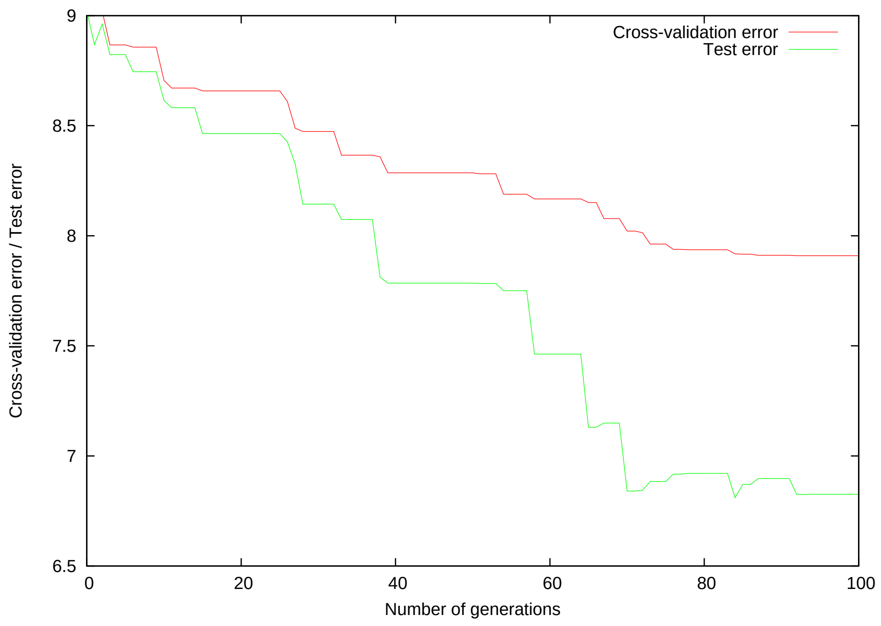

Figure 6.7: The cross-validation error and the corresponding test error during a run of the genetic parameter search algorithm (Algorithm 4.6.1) on the GLASS1 task.

Task	**Cross-validation (10 evals.)**	**RN learning**
cancer	0:23	0:01
card	1:08	0:01
diabetes	0:28	0:01
flare	4:41	0:03
glass	0:03	0:01
heart	2:19	0:02
hearta	2:20	0:02
heartac	0:10	0:01
heartc	0:10	0:01
horse	0:11	0:01
soybean	1:05	0:02

Table 6.2: Time requirements of the regularization network learning. Times needed for 10 evaluations of cross-validation error and times for the final run of the RN learning algorithm (Algorithm 4.1.1) are listed.

6.3.3 Comparison of Kernel Functions

The goal of this experiment was to compare regularization networks with different kernel functions.

We know that the kernel function represents prior knowledge of the given problem, and so it has to be chosen according to the given task. Typically, such prior knowledge is not available and one has to try several kernel functions.

In our experiments, we try to answer the question, whether there is a kernel function that is a better first choice than another and whether there is any class of kernel functions that are suitable for most tasks. We do not expect to find a kernel function that outperforms the others in all situations, simply because it is not possible. We rather expect that the experimental study will help us to understand better how the choice of kernel influences the solution, and give us clues for its optimal choice.

For this purpose we have chosen the data collection PROBEN1 (see Table 6.1). The regularization networks with the kernel functions listed in Table 6.3 (and also shown in Figure 6.8) were used. They represent the most common kernel and activation functions used in neural networks and learning methods.

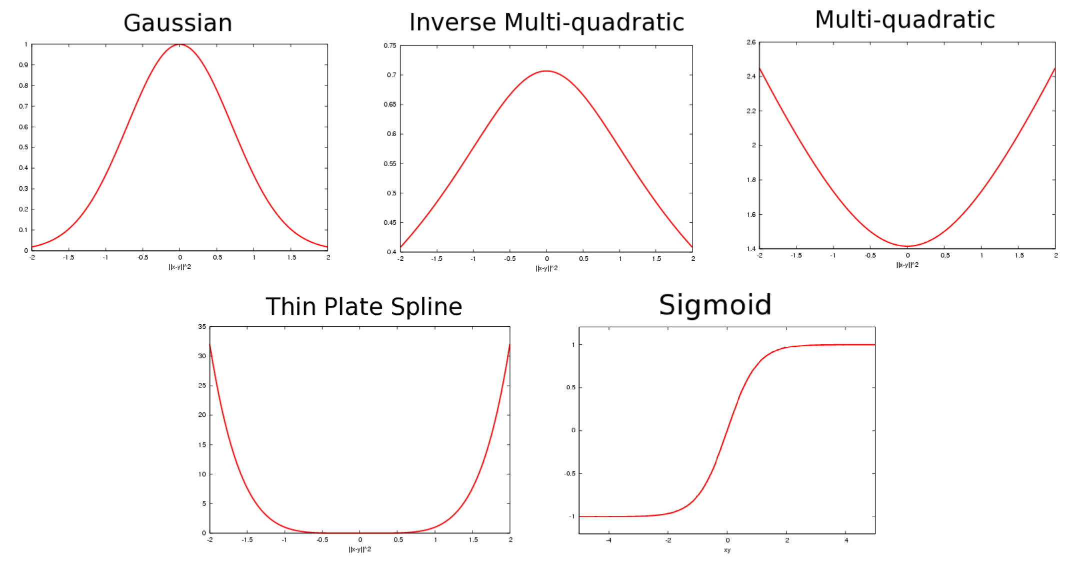

Figure 6.8: Kernel functions.

Table 6.4 compares the training and test errors achieved with these kernels on the data tasks from PROBEN1. For a better illustration, the results are also summarized in Figure 6.10.

In addition, Figure 6.9 compares the overall error (error summed over all the data sets) on the training set and the test set.

In Table 6.4 the lowest errors on the test set in each row are highlighted. In most cases, the lowest error was achieved by the RN with the inverse multi-quadratic kernel

Gaussian	$K(x,y) = e^{-\|\|x-y\|\|^2}$
Inverse Multi-quadratic	$K(x,y) = (\|\|x-y\|\|^2 + c^2)^{-1/2}$
Multi-quadratic	$K(x,y) = (\|\|x-y\|\|^2 + c^2)^{1/2}$
Thin Plate Spline	$K(x,y) = \|\|x-y\|\|^{2n+1}$
Sigmoid	$K(x,y) = tanh(xy - \theta)$

Table 6.3: Kernel functions.

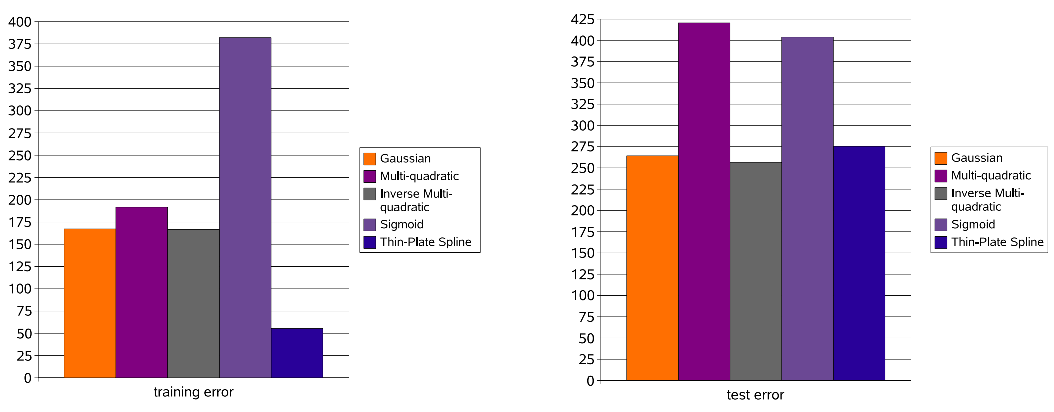

Figure 6.9: Comparison of overall training error (left) and test error (right) for different kernels.

function. For many cases, the Gaussian function achieves the second lowest test error. It can be seen also from Figure 6.10 and from the comparison of overall test errors at Figure 6.9. 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 the usage of local functions, including the Gaussian function, and show that the commonly used Gaussian function is a good first choice.

Comparing the training errors in Table 6.4, we can see that the lowest error on the training set was achieved by the RN with the thin plate spline kernel function and multi--quadratic function. The multi-quadratic function, however, failed completely on the GLASS tasks, therefore the overall training error in Figure 6.9 is higher. The almost zero training error on many tasks is caused by the fact that the zero regularization parameter was chosen by the setup procedure.

The useful property of these kernels is that even without the regularization term, they preserve the generalization ability (their test errors are not high). Such kernels are suitable for tasks without noise, for which the close fitting of training data is desirable.

	Gaussian		**Multi-quadratic**		**Inv. Multi-quadratic**		**Sigmoid**		**Thin-Plate Spline**	
	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}	E_{train}	E_{test}
cancer1	2.38	1.79	0.00	1.61	1.79	**1.49**	3.02	1.83	0.00	**1.49**
cancer2	1.86	3.01	0.00	3.03	1.46	**2.88**	2.54	3.58	0.00	**2.88**
cancer3	2.07	2.79	0.00	3.25	1.89	**2.59**	2.66	2.84	0.00	2.74
card1	7.71	**10.00**	0.00	22.35	8.69	10.01	24.72	24.98	0.00	11.47
card2	6.79	12.75	0.00	15.21	7.31	12.56	26.17	26.45	0.00	14.06
card3	7.10	**12.32**	0.84	14.70	6.00	12.36	11.51	15.39	0.00	14.15
diabetes1	14.17	16.22	15.81	17.25	13.13	16.12	13.77	16.73	11.79	17.07
diabetes2	13.95	16.85	15.88	17.11	14.33	**16.80**	13.09	18.35	13.63	16.82
diabetes3	13.75	15.99	15.92	16.32	13.63	**15.93**	13.97	16.69	11.85	17.18
flare1	0.36	0.55	0.19	0.64	0.35	**0.54**	0.38	0.55	0.26	0.58
flare2	0.42	0.27	0.21	0.42	0.43	**0.27**	0.45	0.30	0.31	0.34
flare3	0.40	0.34	0.20	0.47	0.41	0.34	0.41	0.35	0.29	0.41
glass1	3.90	7.33	84.91	70.75	2.20	**6.12**	6.76	8.58	0.00	6.41
glass2	3.58	7.78	31.56	27.56	1.88	**6.79**	6.90	8.92	0.00	7.29
glass3	3.87	7.25	24.75	36.83	2.24	**6.14**	6.74	9.09	0.00	6.20
heartac1	3.80	3.13	0.00	4.08	4.16	**2.82**	9.55	8.80	0.00	3.51
heartac2	2.75	3.95	0.00	5.00	3.38	**3.84**	8.15	7.27	0.26	4.77
heartac3	3.12	5.17	0.00	4.92	3.34	5.08	5.43	6.14	0.00	**4.72**
hearta1	3.46	4.46	0.00	5.73	3.17	**4.31**	8.44	8.40	2.47	5.14
hearta2	3.48	4.26	0.00	5.79	3.18	4.13	8.11	8.31	0.00	4.86
hearta3	3.39	4.49	0.00	5.52	3.12	**4.40**	8.53	8.43	0.00	4.96
heartc1	8.78	15.93	0.00	15.93	9.46	15.94	18.86	21.64	0.00	**15.65**
heartc2	11.55	6.52	0.00	7.33	12.38	**6.16**	24.80	22.71	0.00	6.65
heartc3	6.54	13.66	0.00	14.23	8.03	12.72	20.34	18.68	0.00	13.82
heart1	9.76	13.69	0.00	16.95	9.58	**13.59**	26.92	27.69	7.23	13.83
heart2	9.48	13.86	0.00	19.29	9.29	**13.74**	14.34	16.80	6.82	14.69
heart3	8.92	16.01	0.00	21.84	8.03	16.15	26.27	27.40	0.00	18.86
horse1	4.51	12.47	0.16	12.33	7.16	**11.69**	18.43	16.75	0.16	12.13
horse2	4.14	15.38	1.08	17.72	5.70	15.34	18.08	17.48	0.18	16.72
horse3	0.82	14.26	0.18	14.52	0.37	**14.00**	18.36	18.20	0.18	14.23
soybean1	0.12	0.67	0.00	0.70	0.10	**0.66**	4.80	4.83	0.00	0.68
soybean2	0.17	0.49	0.01	**0.48**	0.22	0.49	4.78	4.88	0.01	0.50
soybean3	0.15	0.61	0.01	0.67	0.23	0.58	4.80	4.82	0.01	0.66

Table 6.4: Comparison of errors on the training and test sets obtained by regularization networks with different kernel functions. For each task, the lowest error on the test set is highlighted.

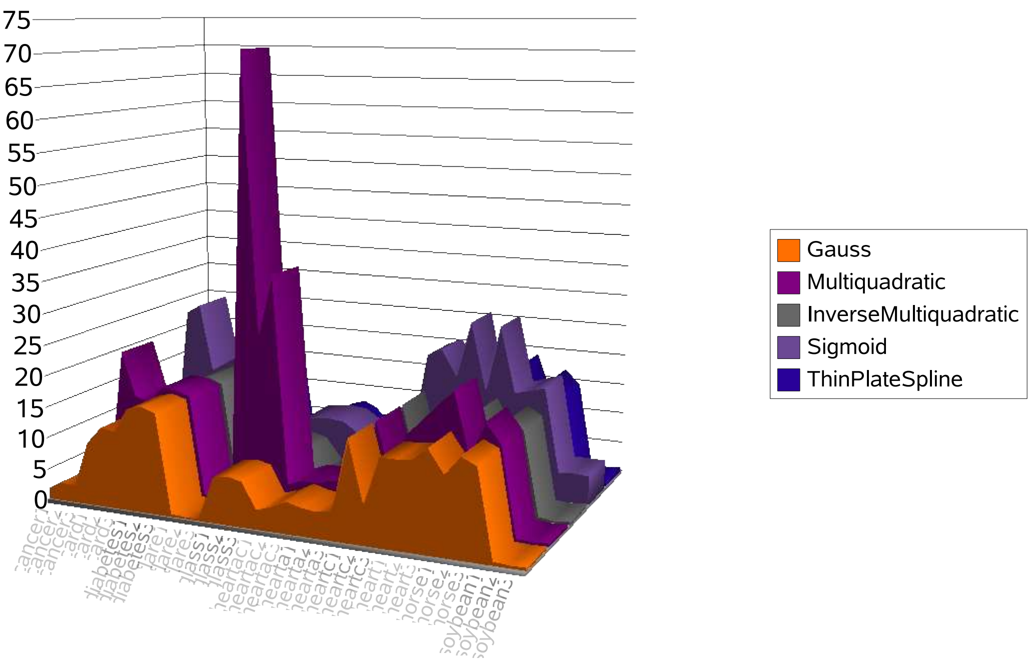

Figure 6.10: Comparison of test errors for RNs with different kernel functions on the tasks from PROBEN1 repository.

6.3.4 Product and Sum Kernels

The following experiment should demonstrate the feasibility of product and sum kernels proposed in Chapter 3.

For this purpose we have chosen regularization networks with the Gaussian kernels, since the Gaussian function is the most common kernel function. The product and sum kernels were composed as products of two or three Gaussian functions of different widths, and sums of two Gaussian functions of different widths, respectively. In case of the product kernels, the input attributes were split into two, resp. three input vectors randomly.

The resulting errors achieved by the RNs, PKRNs, and two SKRNs on the tasks from PROBEN1 are compared in Table 6.5. Figure 6.11 compares the RN with the Gaussian kernels and sum kernels.

The lowest test error in each row of Table 6.5 is highlighted. 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.

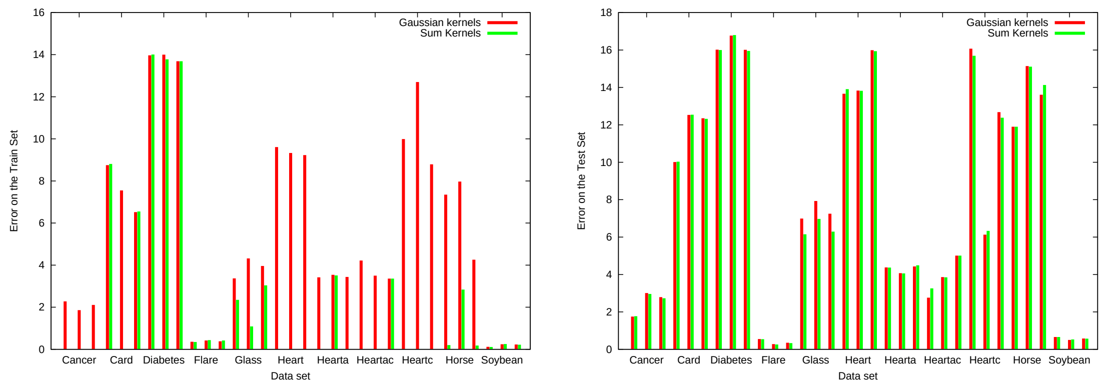

Figure 6.11: Comparison of the sum kernel (green) and the Gaussian kernel (red) on the tasks from PROBEN1. Training error (left) and test error (right).

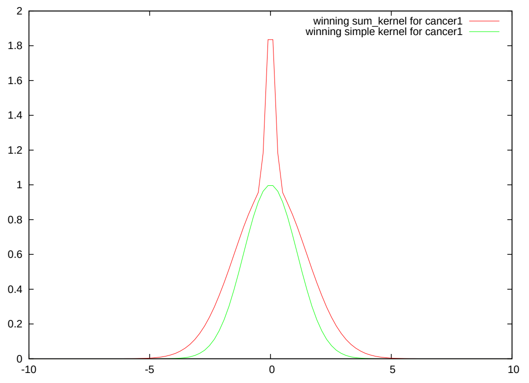

Figure 6.12: Chosen kernels for the CANCER1 task. The Gaussian kernel (green) and the sum kernel (red).

The SKRN showed an interesting behavior on several data sets. In Table 6.5 and Figure 6.11 we can see that the training error on the CANCER tasks and almost all variants of HEART is almost zero (rounded to zero). Still the corresponding test errors are not high, but comparable to those achieved by the RN and the PKRN; in many cases even the lowest ones. In these cases, the regularization parameter chosen by the setup is close to zero, therefore the training error is very low. Still the SKRNs possess the generalization ability.

This is caused by the kernel shape. In Figure 6.12 there is a kernel function found by the setup on CANCER1 tasks. It consists of two Gaussians, a wider one and very narrow one. The behavior of such a kernel has a clear explanation. The narrow Gaussian emphasizes the strict interpolation of the training sample, since for the inputs from the training set, the corresponding hidden unit outweighs the other units. On the other hand, the generalization property is assured by the wider Gaussian function. In addition, such a kernel leads to the linear system with a dominant diagonal, so from this point of view, the regularization member is not needed.

6.3.5 Divide et Impera

In Section 3.5.2 we proposed a *Divide et Impera* technique for dealing with bigger data sets. The main idea is to divide the tasks into several independent subtasks, that is done by partitioning of the data set into several subsets. Then we run the RN learning algorithm for each of these subsets, and get a result as a sum of the networks obtained.

Since the time complexity needed for solving of linear system[2] is $O(n^3)$, replacing one bigger linear system by several smaller one does not only reduce the space complexity, but also the time complexity. However, we expect that such an approach would lead to worse results, i.e. higher errors.

Therefore the goal of another experiment was to compare the standard algorithm with the *Divide et Impera* approach. Again we used the repository PROBEN1 and the regularization networks with Gaussian kernels.

For the *Divide et Impera* approach, each data set was divided into two subsets, with the exception of CANCER, DIABETES, FLARE, and SOYBEAN, where we used the partitioning into three subsets.

Table 6.6 lists the errors obtained by the single RN learning algorithm and the Divide et Impera approach on the tasks from PROBEN1. The lowest test error for each task is highlighted. As expected, in most cases they were achieved by the basic algorithm. However, the test errors achieved by the Divide et Impera approach are comparable, even in 8 cases equal to or lower than the errors obtained by the basic algorithm. Three of these cases (FLARE2,FLARE2, SOYBEAN1) are the tasks divided into 3 subsets.

Figure 6.13 compares the time needed by the learning algorithm. The reduction of time complexity is apparent.

The *Divide et Impera* approach represents one alternative to 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.

6.3.6 Summary

In this section, we presented the results of our experiments with the regularization networks. We demonstrated the role of metaparameters and showed that their choice significantly influences the solution quality. Algorithms performing the setup of these metaparameters were demonstrated.

[2]Considering a linear system $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is a square matrix: n is the number of rows resp. columns of the matrix A.

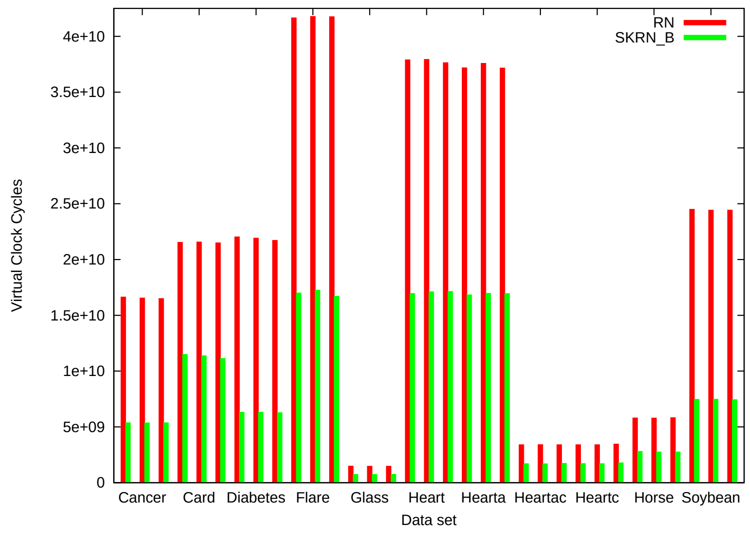

Figure 6.13: Time (in clock cycles, using the PAPI library [55]) needed for one run of the RN learning algorithm and the Divide et Impera.

Then several common kernel functions have been compared. On the collection of benchmarks PROBEN1, the best results were obtained with the kernel functions with a local response, i.e. the Gaussian function and inverse multi-quadratic function.

The product and sum kernels proposed in Chapter 3 were applied on the tasks from PROBEN1. The results illustrate that they represent a vital alternative to the simple Gaussian kernels. To benefit from them, prior knowledge of the problem suggesting the use of a composed kernel is needed.

Yet, the sum kernels outperform the simple Gaussian on most tasks. The sum of two Gaussian functions exhibits a good behavior. When formed by one narrow and one wide Gaussian function, it possess the ability to generalize, so the regularization term is not needed for the tasks with low level of noise. Similar behavior has also been achieved with the thin-plate spline and multi-quadratic functions, with a suitable parameter setting.

Also the *Divide et Impera* approach was demonstrated. We have shown that it can significantly reduce the time complexity, thus it represents an alternative, which can be applied to bigger data sets.

	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
diabetes1	13.97	16.02	14.01	**16.00**	16.44	16.75
diabetes2	14.00	**16.77**	13.78	16.80	15.87	18.14
diabetes3	13.69	16.01	13.69	**15.95**	16.31	16.62
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
heart1	9.61	**13.66**	0.00	13.91	9.56	13.67
heart2	9.33	13.83	0.00	**13.82**	9.43	13.86
heart3	9.23	15.99	0.00	**15.94**	9.15	16.06
hearta1	3.42	4.38	0.00	**4.37**	3.47	4.39
hearta2	3.54	4.07	3.51	**4.06**	3.28	4.29
hearta3	3.44	**4.43**	0.00	4.49	3.40	4.44
heartac1	4.22	**2.76**	0.00	3.26	4.22	**2.76**
heartac2	3.50	3.86	0.00	**3.85**	3.49	3.87
heartac3	3.36	**5.01**	3.36	**5.01**	3.26	5.18
heartc1	9.99	16.07	0.00	**15.69**	10.00	16.08
heartc2	12.70	**6.13**	0.00	6.33	12.37	6.29
heartc3	8.79	12.68	0.00	**12.38**	8.71	12.65
horse1	7.35	**11.90**	0.20	**11.90**	14.25	12.45
horse2	7.97	15.14	2.84	**15.11**	12.24	15.97
horse3	4.26	**13.61**	0.18	14.13	9.63	15.88
soybean1	0.12	**0.66**	0.11	**0.66**	0.13	0.86
soybean2	0.24	**0.50**	0.25	0.53	0.23	0.71
soybean3	0.23	0.58	0.22	**0.57**	0.21	0.78

Table 6.5: Comparisons of errors on the training and test set for the RN with the Gaussian kernels, the SKRN, and the PKRN.

	Single algorithm		**Divide et Impera**	
Task	E_{train}	E_{test}	E_{train}	E_{test}
cancer1	2.28	**1.75**	2.11	1.93
cancer2	1.86	**3.01**	1.68	3.37
cancer3	2.11	**2.79**	1.68	2.95
card1	8.75	**10.01**	8.55	10.58
card2	7.55	**12.53**	7.22	13.03
card3	6.52	**12.35**	6.22	12.86
diabetes1	13.97	**16.02**	12.92	16.66
diabetes2	14.00	**16.77**	13.64	17.33
diabetes3	13.69	**16.01**	12.85	16.34
flare1	0.36	**0.55**	0.35	0.59
flare2	0.42	**0.28**	0.41	**0.28**
flare3	0.38	**0.35**	0.38	**0.34**
glass1	3.37	**6.99**	2.56	6.78
glass2	4.32	7.93	3.27	**7.29**
glass3	3.96	7.25	3.48	**6.44**
heart1	9.61	**13.66**	9.51	13.79
heart2	9.33	**13.83**	8.52	14.31
heart3	9.23	**15.99**	8.30	16.75
hearta1	3.42	**4.38**	3.20	4.45
hearta2	3.54	**4.07**	3.17	4.34
hearta3	3.44	4.43	3.37	**4.40**
heartac1	4.22	**2.76**	3.68	3.37
heartac2	3.50	**3.86**	2.99	3.97
heartac3	3.36	**5.01**	3.14	5.13
heartc1	9.99	**16.07**	6.50	**16.07**
heartc2	12.70	**6.13**	11.06	6.69
heartc3	8.79	12.68	9.91	**11.74**
horse1	7.35	**11.90**	7.66	12.62
horse2	7.97	**15.14**	6.84	15.70
horse3	4.26	**13.61**	8.56	15.24
soybean1	0.12	0.66	0.12	**0.64**
soybean2	0.24	**0.50**	0.19	0.54
soybean3	0.23	**0.58**	0.15	0.72

Table 6.6: Comparisons of errors on the training and test set for the RNs learned by Algorithm 4.1.1 (single algorithm) and the RNs learned by the Divide et Impera approach.

6.4 RBF Networks

This section presents the results of experiments with RBF networks. We demonstrate the individual learning algorithms on the benchmark data sets and compare them to each other and to multi-layer perceptrons.

First, in the next subsection, we illustrate the advantages of using the hidden units with weighted norms. In the following three subsections, the gradient learning, three--step learning, and genetic learning are studied. Then in Subsection 6.4.5, the experiments with hybrid approaches are presented. Finally, in Subsection 6.4.6 we present a comparison of the learning algorithms studied and draw conclusions.

6.4.1 RBF Units with Weighted Norms

The goal of the first experiment is to demonstrate advantages of using the RBF units with weighted norms. For this aim we have selected an two-dimensional classification problem, known as *two spirals*. Its objective is to classify two intertwined spirals formed by points in the plane (see Figure 6.15a). The geometry of the problem is suitable for the adaptation of shape of the areas controlled by individual units.

We tested the RBF networks with 100 units using the Euclidean norm, and 50 and 70 units using weighted norms. The gradient learning was used as a learning algorithm.

Table 6.7 compares the numbers of iterations that were necessary to achieve the given error threshold. The 70 unit weighted norm network outperforms the 100 unit network with the Euclidean norm by the factor of 2–3, for threshold 1.0 and smaller.

	100 Euclidean			**70 weighted**			**50 weighted**		
	iteration		time	iteration		time	iteration		time
ϵ	avg	stddev		avg	stddev		avg	stddev	
10.0	1188.4	275.2	0:42	830.1	208.7	0:36	1854.9	525.1	1:06
5.0	1944.4	568.3	1:09	1115.8	340.5	0:48	2991.8	1061.8	1:47
1.0	3739.1	1559.1	2:14	1993.6	769.0	1:26	4965.3	1955.4	2:57
0.5	5362.6	3599.4	3:11	2475.1	967.8	1:46	7277.9	3088.8	4:20
0.1	15842.7	10783.7	9:26	4923.0	1864.9	3:31	24424.6	10819.1	14:32

Table 6.7: Two spirals: The number of iterations and time needed to achieve the error equal or less than ϵ. Gradient learning with 100 hidden units using the Euclidean norm, 50 and 70 hidden units using weighted norms.

The solution quality in Figure 6.14, which illustrates the classification of points in the plane by the learned RBF network. We can see that the middle plot, corresponding to the classification performance of the 70 weighted norm unit network, is clearly the most faithful one. The smaller network with weighted norms (right) can still achieve good performance compared to the Euclidean norm network double in size (left).

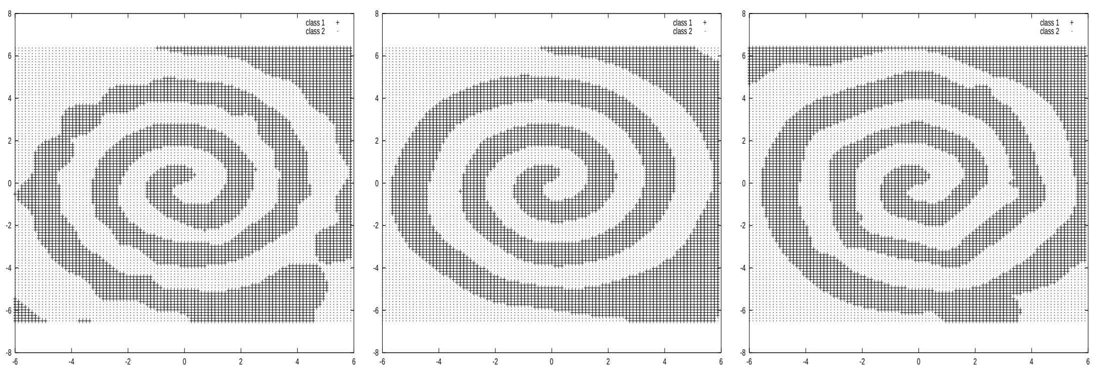

Figure 6.14: Two spirals: The classification of the input space by the learned RBF network. a) 100 units with the Euclidean norm (left) b) 70 units with weighted norms (center) c) 50 units with weighted norms (right).

Finally, Figure 6.15 shows the position and shape of network units in the input space. Contour lines of a given threshold for the individual RBF units are drawn. It illustrates an intuition that covering the input points by ovals is easier than by circles.

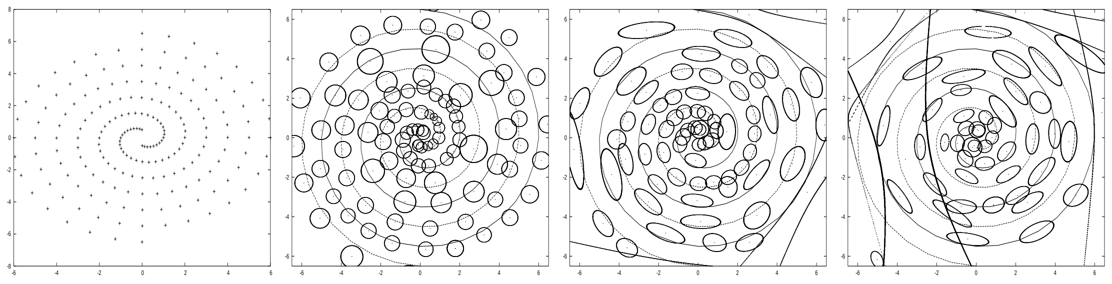

Figure 6.15: Two spirals: a) The training set. b)-d) The shape and position of hidden units: b) 100 units with the Euclidean norm c) 70 units with weighted norms d) 50 units with weighted norms. a) - d) goes from the left to the right.

In the *two spirals* problem, the RBF units with weighted norms outperform the classical units. They benefit from their flexibility and ability to cover the input space better. On the other hand, an RBF unit with a weighted norm has another $d(d+1)/2$ parameters to estimate (the entries of the diagonal matrix $\mathbf{\Sigma^{-1}}$, see Section 5.2). So there is the time overhead for performing one iteration of gradient learning with the RBF units with weighted norms. In our experiment, the approximate times for 100 training iterations are 3.5 s, 4 s, or 3.5 s for the RBF networks with 100 Euclidean, 70 weighted, or 50 weighted units, respectively. However, this overhead is reasonable, and the RBF network with 70 weighted norms needed a much smaller number of iterations to achieve the given error thresholds; so its learning was faster in the end.

6.4.2 Gradient Learning

The goal of this experiment was to study and demonstrate the behavior of gradient learning (see Section 5.3).

We selected the tasks CANCER, GLASS, HEARTA from the PROBEN1 repository. The first two represent classification problems, while the third one an approximation problem.

For the CANCER tasks we used an RBF network with 5 units, for the GLASS tasks one with 10, 15 units, and for the HEARTA tasks one with 30, 40, 50 units. Since the gradient learning starts with random initialization, all computations were run 10 times and the means and standard deviations were computed.

Table 6.8 lists the training and test errors achieved by the gradient learning. On the GLASS tasks, the best results were achieved by the RBF network with 15 units, on the HEARTA tasks by the RBF network with 50 units. So in both the cases the network having more hidden units performed better.

Task	h	E_{train}		E_{test}	
		avg	stddev	avg	stddev
cancer1		2.49	0.10	1.40	0.23
cancer2	5	2.03	0.89	3.44	0.53
cancer3		2.04	0.50	3.44	0.51
glass1		4.14	0.36	7.39	0.59
glass2	10	3.72	0.24	8.26	0.47
glass3		4.27	0.38	7.41	0.43
glass1		3.35	0.33	6.59	0.32
glass2	15	2.94	0.19	7.85	0.43
glass3		3.47	0.31	6.95	0.26
hearta1		2.67	0.08	4.35	0.15
hearta2	30	2.56	0.10	4.39	0.10
hearta3		2.65	0.13	4.23	0.12
hearta1		2.72	0.00	4.10	0.12
hearta2	40	2.57	0.08	4.19	0.12
hearta3		2.62	0.09	4.04	0.08
hearta1		2.70	0.13	4.06	0.10
hearta2	50	2.56	0.06	4.13	0.10
hearta3		2.67	0.07	3.95	0.09

Table 6.8: Training and test error of the RBF network found by the gradient learning algorithm.

Tables 6.9, 6.10, 6.11, and Figures 6.16, 6.17 and 6.18 show the number of iterations and time needed to achieve a given error threshold on different tasks. We can see that the speed of learning is quite high at the beginning of the computation, but gets smaller with the number of iterations.

The time needed for 100 iterations of the algorithm was 1 s for the CANCER task and the network with 5 units, 3 s and 6.5 s for GLASS and 10, resp. 15 units, 52.9 s, 70 s, and 88.6 s for HEARTA and 30, 40, 50 units, respectively.

		Iterations		Time
		avg	stddev	avg
10.0	cancer1	8.9	12.06	0:01
	cancer2	360.2	612.43	0:04
	cancer3	4.1	1.22	0:01
5.0	cancer1	11.9	12.98	0:01
	cancer2	366.0	615.46	0:04
	cancer3	8.1	1.51	0:01
4.0	cancer1	16.0	13.70	0:01
	cancer2	368.6	616.25	0:04
	cancer3	11.4	1.11	0:01
3.0	cancer1	194.1	40.78	0:02
	cancer2	388.5	611.22	0:04
	cancer3	52.2	15.16	0:01
2.5	cancer1	2274.1	1104.02	0:24
	cancer2	521.1	562.05	0:06
	cancer3	386.2	83.73	0:04

Table 6.9: The number of iterations and time needed to achieve the given value of the training error on the CANCER task.

The gradient learning (Algorithm 5.3.1) is based on the gradient descent algorithm, that is a local optimization algorithm, which starts with random initialization. Therefore there sometimes appear large differences between the results achieved by different runs of the algorithm (see the values of standard deviations in Table 6.8 and intervals at Figures 6.17 and 6.18). For real applications, it is recommended to run the algorithm for more different initializations, possibly for a smaller number of iterations, and pick the best computation.

The rate of error decrease can be used as an indicator of the algorithm stop criterion.

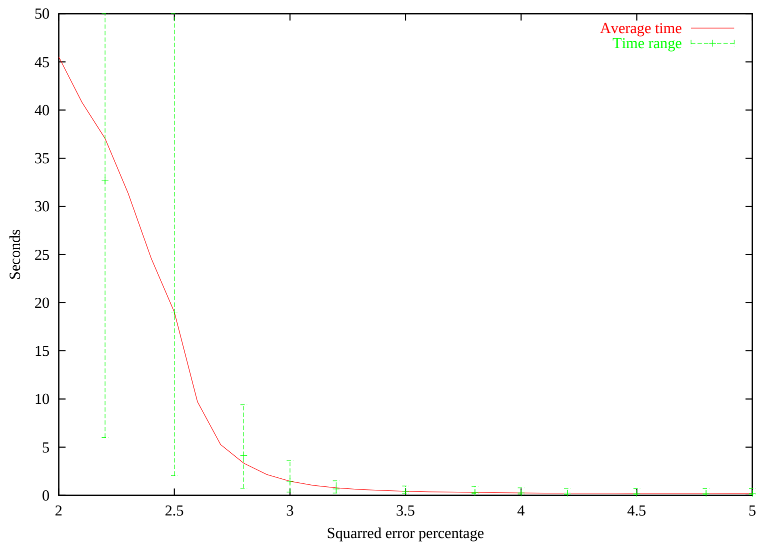

Figure 6.16: The number of iterations and time needed to achieve the given value of the training error on the CANCER task.

		10 units			**15 units**		
		Iterations		Time	Iterations		Time
ϵ	Data	avg	stddev	avg	avg	stddev	avg
	glass1	1.0	0.00	0:01	1.0	0.00	0:01
15.0	glass2	1.0	0.00	0:01	1.7	0.46	0:01
	glass3	1.0	0.00	0:01	1.0	0.00	0:01
	glass1	96.2	33.44	0:03	90.7	26.42	0:06
10.0	glass2	92.2	24.30	0:03	71.5	10.93	0:04
	glass3	77.9	13.60	0:02	77.9	13.60	0:05
	glass1	1123.7	160.01	0:34	950.4	116.17	1:01
7.0	glass2	703.3	125.37	0:21	611.6	31.17	0:40
	glass3	953.1	52.80	0:29	953.1	52.80	1:02
	glass1	2454.6	405.77	1:16	1910.3	199.25	2:04
6.0	glass2	1270.6	257.06	0:39	1066.6	105.12	1:09
	glass3	2043.8	121.60	1:03	2043.8	121.60	2:13
	glass1	4806.8	1022.57	2:08	3368.7	363.25	3:39
5.0	glass2	2523.1	385.37	1:07	2130.4	258.38	2:19
	glass3	4226.0	702.44	2:00	4226.0	702.44	4:35

Table 6.10: The number of iterations and time needed to achieve the given value of the training error on the GLASS task.

		30 units			**40 units**			**50 units**		
		Iterations		Time	Iterations		Time	Iterations		Time
ϵ	Data	avg	stddev	avg	avg	stddev	avg	avg	stddev	avg
	hearta1	16.8	2.86	0:09	16.8	2.36	0:11	18.2	1.78	0:16
10.0	hearta2	19.3	2.41	0:10	18.8	3.76	0:13	23.0	4.34	0:21
	hearta3	16.5	3.47	0:09	17.3	3.95	0:12	18.2	4.07	0:16
	hearta1	24.2	3.82	0:13	24.6	4.10	0:17	25.8	3.37	0:22
7.5	hearta2	27.8	6.79	0:14	24.4	4.96	0:17	32.4	8.26	0:29
	hearta3	22.4	3.72	0:11	23.5	4.96	0:16	24.1	4.66	0:21
	hearta1	64.2	13.42	0:34	64.3	7.72	0:45	61.8	8.72	0:54
5.0	hearta2	68.7	10.22	0:36	56.3	9.64	0:39	70.1	9.75	1:02
	hearta3	54.1	7.08	0:29	53.8	7.87	0:37	56.8	6.48	0:49
	hearta1	190.1	38.34	1:41	186.9	9.10	2:10	164.7	17.58	2:25
4.0	hearta2	197.0	31.54	1:44	173.4	35.60	2:01	171.6	23.09	2:31
	hearta3	171.1	21.76	1:31	160.9	23.84	1:52	160.7	21.25	2:21
	hearta1	1217.5	217.38	10:44	1151.3	76.36	13:26	1022.2	211.26	15:05
3.0	hearta2	1021.0	182.17	9:01	926.6	141.73	10:48	843.6	100.84	12:26
	hearta3	1199.4	289.31	10:34	993.6	179.36	11:35	160.7	21.25	2:21

Table 6.11: The number of iterations and time needed to achieve the given value of the training error on the HEARTA task.

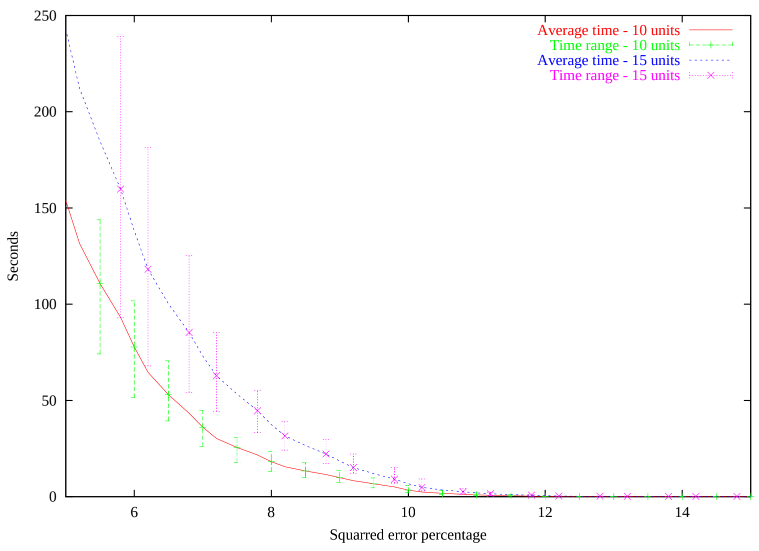

Figure 6.17: Time needed to achieve the given value of the training error on the GLASS task.

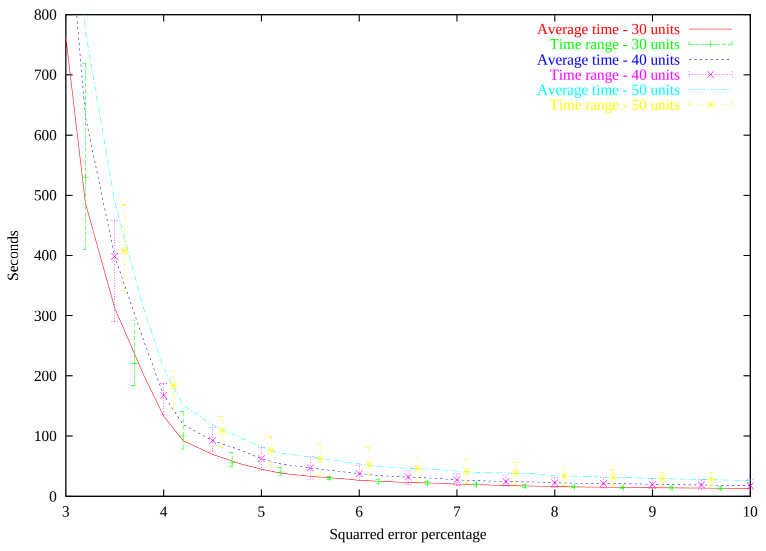

Figure 6.18: Time needed to achieve the given value of the training error on the HEARTA task.

6.4.3 Three-Step Learning

Now we will present the results of our experiments with the three-step learning.

Again, we have chosen the tasks CANCER, GLASS, and HEARTA from the PROBEN1 repository. On the CANCER task we applied networks with 5, 10, 20, and 50 units, on the GLASS tasks with 15, 30, and 50 units, and on the HEARTA task 30, 40, and 50 units.

The training and test errors of RBF networks obtained by the three-step learning are summarized in Table 6.12. Figure 6.19 compares the training and test errors, including classification error (percentage of correctly classified samples), for the RBF networks of different size on CANCER task.

At Figure 6.19 we observe the decrease of the training error with the increase of the number of hidden units. However, for 50 units we obtain higher test errors. The test error for 50 units on CANCER3 (see Table 6.12) is even higher than for 20 units.

The training error on the GLASS and HEARTA tasks (Table 6.12) also decreases with the increasing number of hidden units. However, this does not apply on the test errors. On the GLASS tasks we get higher test errors for 30 and 50 hidden units than for 15 units. On GLASS1 and GLASS3 the test errors are very high, in case of 50 units the network has not learned the task at all. On HEARTA, this increase of test error is not so significant; on HEARTA2, the test error even decreases with the increasing number of units. On HEARTA1 and HEARTA3 the increase is slow.

The observed behavior, the increase of test errors with decreasing train errors, is caused by over-fitting and losing the generalization ability. With the higher number of hidden units, there is a higher danger of over-fitting. For different tasks, a different number of units is suitable, so it is recommended to try several variants.

The time needed by the algorithm is 1 s (4 s, 16 s, 1 m 33 s) on CANCER for the network with 5 units (10,20, 50units, respectively), 17 s (1 m 7 s, 2 m 59 s) on GLASS with 15 units (resp. 30, 50 units), and 4 m 38 s (7 m 45 s, 11 m 42 s) on HEARTA with 30 units (resp. 40, 50 units).

Task	**5 units**				**20 units**				**50 units**			
	E_{train}		E_{test}		E_{train}		E_{test}		E_{train}		E_{test}	
	avg	stddev	avg	stddev	avg	stddev	avg	stddev	avg	stddev	avg	stddev
cancer1	3.63	0.06	2.34	0.09	3.24	0.42	2.31	0.52	2.45	0.12	2.21	0.28
cancer2	3.34	0.36	4.35	0.53	2.82	0.42	3.89	0.61	2.12	0.10	3.57	0.27
cancer3	4.03	1.51	4.01	1.75	2.74	0.38	2.91	0.48	2.15	0.06	3.42	0.37

Task	**15 units**				**30 units**				**50 units**			
	E_{train}		E_{test}		E_{train}		E_{test}		E_{train}		E_{test}	
	avg	stddev	avg	stddev	avg	stddev	avg	stddev	avg	stddev	avg	stddev
glass1	7.55	0.18	10.28	0.90	6.02	0.10	15.86	4.68	4.61	0.16	194.19	104.49
glass2	7.41	0.15	9.33	0.25	5.90	0.16	9.76	0.68	4.48	0.14	14.81	2.49
glass3	7.54	0.25	10.08	0.84	5.91	0.13	13.54	2.37	4.74	0.13	129.86	170.14

Task	**30 units**				**40 units**				**50 units**			
	E_{train}		E_{test}		E_{train}		E_{test}		E_{train}		E_{test}	
	avg	stddev	avg	stddev	avg	stddev	avg	stddev	avg	stddev	avg	stddev
hearta1	4.35	0.15	4.64	0.13	4.10	0.12	4.67	0.20	4.06	0.10	4.87	0.21
hearta2	4.39	0.10	4.50	0.16	4.19	0.12	4.46	0.21	4.13	0.10	4.44	0.11
hearta3	4.23	0.12	5.07	0.13	4.04	0.08	4.86	0.19	3.95	0.09	4.95	0.19

Table 6.12: Training error and test errors for the networks found by the three-step learning.

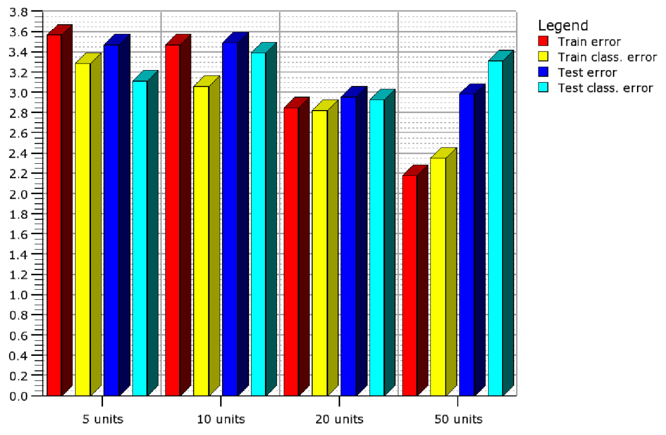

Figure 6.19: Error and classification error of networks obtained by the three-step learning on the CANCER tasks.

6.4.4 Genetic Learning

The goal of the following experiment was to demonstrate the behavior of the genetic learning and compare the classical GAs to the canonical GAs (the canonical GAs is a variant of the GAs proposed in [50]).

An RBF network with 5 units was trained on the CANCER1, CANCER2, and CANCER3 tasks. All experiments were run 10 times and the mean and standard deviations of the resulting error values were computed.

The resulting training and test errors are listed in Table 6.13. The canonical algorithm obtained better results on the CANCER2 and CANCER3 tasks, the classical algorithm on CANCER1.

Table 6.14 and Figure 6.20 show the number of iterations and the corresponding time that were necessary to obtain the given error thresholds. We can see that initially the error rate decrease is very high, and then drops down.

The time requirements of 100 generations was 66 s for both the classical and canonical version of the genetic learning.

Though the time requirements of the genetic learning are quite high, they can be reduced by parallelization. But still, the genetic algorithms are more useful if used in combination with other methods (see the following subsection).

Task	**Canonical algorithm**				**Classical algorithm**			
	E_{train}		E_{test}		E_{train}		E_{test}	
	avg	stddev	avg	stddev	avg	stddev	avg	stddev
cancer1	4.97	0.79	4.20	1.17	**4.80**	0.68	**3.87**	0.95
cancer2	**4.74**	0.57	**5.09**	0.52	4.88	1.12	5.71	1.35
cancer3	**4.35**	0.54	**4.51**	0.69	4.45	0.83	4.72	0.77

Table 6.13: The results achieved by the classical and canonical genetic learning.

		Canonical algorithm			**Classical algorithm**		
		Iterations		Time	Iterations		Time
ϵ	Data	avg	stddev	avg	avg	stddev	avg
10.0	cancer1	933.8	1019.82	0:10:13	2110.0	2734.89	0:23:06
	cancer2	1100.0	949.39	0:12:03	496.2	385.74	0:05:26
	cancer3	501.2	560.01	0:05:29	515.0	375.50	0:05:39
5.0	cancer1	33992.5	17454.90	6:12:18	25431.2	18095.75	4:38:32
	cancer2	36893.8	18238.51	6:44:04	20492.5	18929.86	3:44:26
	cancer3	7191.2	7557.69	1:18:46	24046.2	12981.90	4:23:21
4.0	cancer1	46107.5	10253.58	8:24:59	46610.0	9022.01	8:30:29
	cancer2	49765.0	3320.42	9:05:03	47267.5	5223.70	8:37:41
	cancer3	36428.8	18210.84	6:38:59	40005.0	12522.37	7:18:09

Table 6.14: The number of iterations and time needed to achieve the given error threshold by the genetic learning.

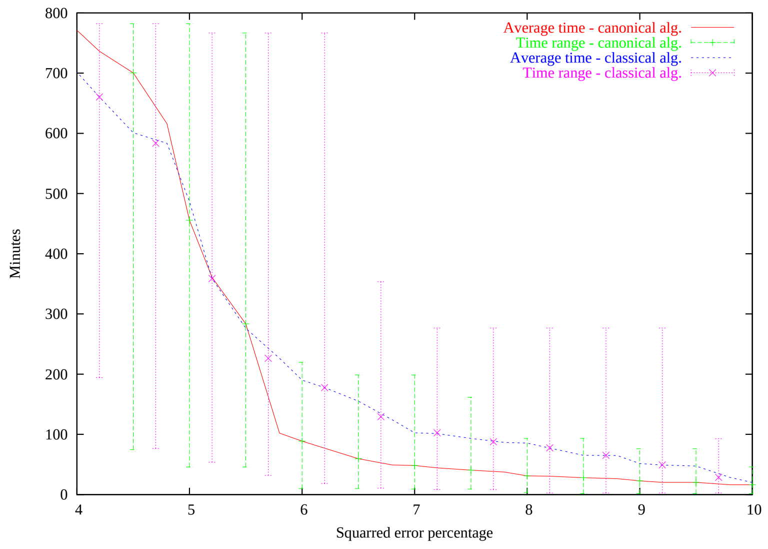

Figure 6.20: Time needed to achieve the given error threshold by the classical and canonical genetic learning (CANCER task).

6.4.5 Hybrid Methods

In this section we present the results obtained by the hybrid methods — the four-step learning and the hybrid genetic learning. The both algorithms were described in Section 5.6.

First, the four-step learning was applied on the CANCER and GLASS tasks. For the CANCER we used the networks with 5 and 20 units, for GLASS the networks with 15 and 30 units.

The results achieved by the four-step learning are summarized in Table 6.15. Comparing the test errors to the test errors obtained with the three-step learning (Table 6.12), the fourth step further improves the results in rather all cases. The only exception are the GLASS2 and GLASS3 tasks, for which only the training error decreased, while the test error is the same.

The method combines the advantages of the two approaches; the gradient fourth step further improves the network obtained by the three-step learning, while the first part, formed by the three-step learning, decreases the time requirements by creating a good starting point for gradient descent search.

Task	h	E_{train}		E_{test}		Task	h	E_{train}		E_{test}	
		avg	stddev	avg	stddev			avg	stddev	avg	stddev
cancer1		2.40	0.10	1.81	0.06	glass1		7.06	0.14	10.02	0.77
cancer2	5	1.98	0.06	3.15	0.04	glass2	15	6.97	0.12	9.08	0.23
cancer3		2.21	0.09	2.70	0.05	glass3		7.10	0.27	9.56	0.51
cancer1		1.70	0.12	1.64	0.16	glass1		5.98	0.09	15.81	4.68
cancer2	20	1.41	0.11	2.89	0.07	glass2	30	5.89	0.14	9.76	0.67
cancer3		1.43	0.13	2.74	0.20	glass3		5.86	0.13	13.54	2.41

Table 6.15: Training and test errors achieved by the four-step learning.

The second experiment tests the hybrid genetic learning. A network with 5 units was applied on the CANCER tasks. Both the classical and the canonical versions of the algorithm were tested.

The training and test errors achieved by the hybrid genetic learning are listed in Table 6.16. We can see that on all the three tasks, the canonical algorithm finds better solutions than the classical algorithm. If we compare the results to the genetic learning (Table 6.13), we see that the hybrid genetic learning clearly outperforms the standard genetic learning. Moreover, it achieves comparable results with the gradient learning (Table 6.8).

In addition, Figure 6.21 shows the time needed to achieve given error thresholds. The time required by 100 iterations of hybrid genetic learning was 134 s.

The experiment shows that the hybrid genetic search is competitive to other methods and may achieve better solutions. However, the time requirements are quite high (note that for each fitness evaluation, linear optimization has to be solved). For a real application, parallelization is desirable.

Task	**Canonical algorithm**				**Classical algorithm**			
	E_{train}		E_{test}		E_{train}		E_{test}	
	avg	stddev	avg	stddev	avg	stddev	avg	stddev
cancer1	**2.44**	0.11	**1.80**	0.55	2.51	0.07	1.95	0.27
cancer2	**1.70**	0.58	**3.52**	1.39	1.85	0.14	3.74	0.54
cancer3	**2.12**	0.11	**2.92**	0.28	2.12	0.18	3.23	0.75

Table 6.16: The results achieved by the hybrid genetic learning.

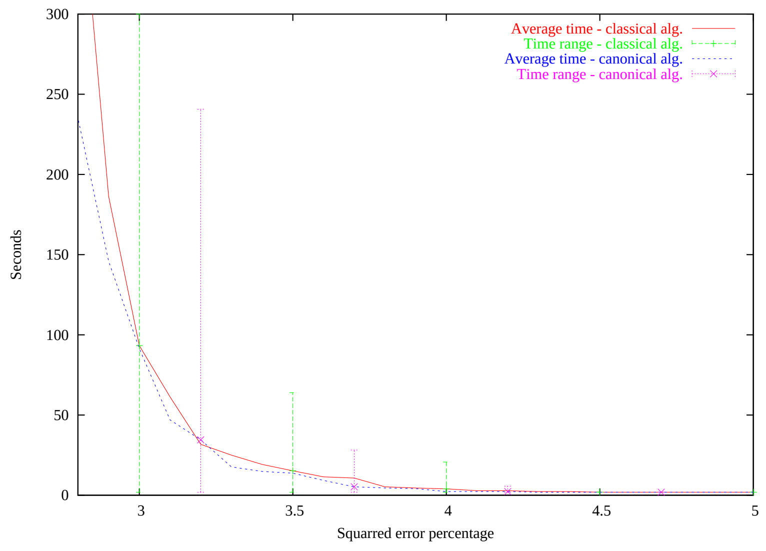

Figure 6.21: Time needed by the hybrid genetic learning to achieve the given error threshold.

The both presented hybrid approaches improve the behavior of single learning algorithms. Clever combinations of the available learning algorithms is a promising way of obtaining novel superior methods.

6.4.6 Conclusions

Let us generalize the observations from the previous experiments with the RBF networks.

Table 6.17 summarizes the results presented in the previous sections and compares different learning algorithms on the CANCER task with 5 unit networks, and the GLASS task with 15 unit networks. It contains means of training and test errors over the three

	Cancer (5 units)			Glass (15 units)		
Used method	E_{train}	E_{test}	Time h:m:s	E_{train}	E_{test}	Time m:s
Grad. learn.	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
GAs (can.)	4.69	4.60	07:24:16	–	–	–
Hybrid GAs	2.09	2.75	02:30:31	–	–	–

Table 6.17: Comparison of learning methods on the CANCER data set for the network with 5 hidden units and on the glass data set for the network with 15 hidden units. Average training and test error.

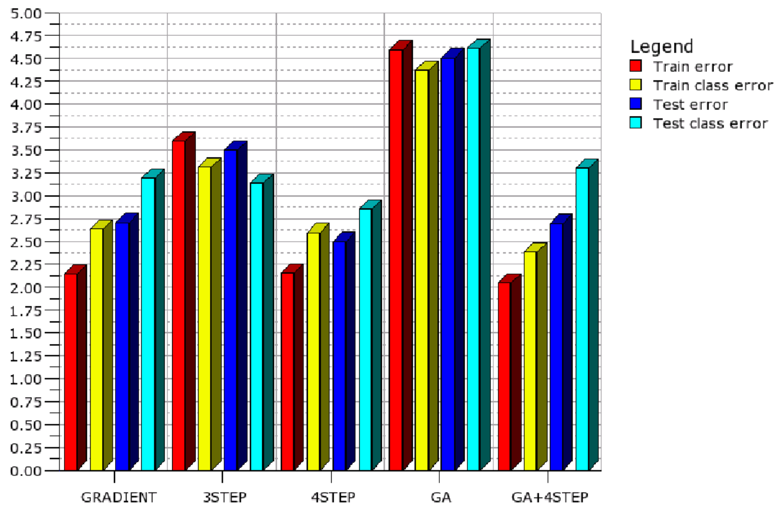

Figure 6.22: Comparison of learning methods on the CANCER data set for the network with 5 hidden units.

variants of given tasks, and time requirements. In addition, Figure 6.22 compares the training and test errors (including the classification errors) for the CANCER task and network with 5 units.

First, consider the main three approaches, the gradient learning, the three-step learning, and the genetic learning. The gradient-based algorithm 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 of 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 GAs to converge to the desired values. The results are still not as good

Data	**RBF network**					**MLP**				
	Error		Class.		Arch.	Error		Class.		Arch.
	avg	stddev	avg	stddev		avg	stddev	avg	stddev	
cancer1	1.64	0.16	1.26	0.43	20	1.60	0.41	1.47	0.60	4+2
cancer2	2.89	0.07	3.16	0.39	20	3.40	0.33	4.52	0.70	8+4
cancer3	2.74	0.20	3.05	0.37	20	2.57	0.24	3.37	0.71	16+8
glass1	6.59	0.32	27.55	2.56	15	9.75	0.41	39.03	8.14	16+8
glass2	7.85	0.43	35.47	3.46	15	10.27	0.40	55.60	2.83	16+8
glass3	6.95	0.26	27.74	1.47	15	10.91	0.48	59.25	7.83	16+8
hearta1	4.84	0.25	–		30	4.76	1.14	–		32+0
hearta2	4.66	0.08	–		30	4.52	1.10	–		16+0
hearta3	4.54	0.06	–		30	4.81	0.87	–		32+0

Table 6.18: Summary: Comparison of learning results on the testing set achieved by the RBF networks and multilayer perceptrons trained by RPROP algorithm ([61]). Architecture (Arch.) is the number of hidden units in case of RBF networks, and the number of nodes in the first and second hidden layer in case of MLPs. Error stands for E_{test}, Class. stands for the classification error on the test set.

as with the gradient learning. Nevertheless, the GAs — as a general learning procedure — has its potential in learning the networks with heterogeneous units; and it is suitable for parallelization.

Then 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.

Table 6.18 compares our results with the multi-layer perceptron networks (MLP). The results for MLP are taken from [61]. One can see that the RBF networks achieved about the same performance on the CANCER and HEART tasks, while they rather outperform the MLP networks on the GLASS data, both in terms of the errors and number of units.

6.5 Regularization Networks vs. RBF Networks

The main aim of the experimental part of this work was to assess the relative performance of RNs and RBF networks. This section presents the results of the experiments comparing these two approaches.

The benchmark data collection PROBEN1 was used to perform the comparison.

The regularization networks have been trained by the RN learning algorithm (Algorithm 4.1.1) with the metaparameters set up by the adaptive grid search (Algorithm 4.5.1).

The RBF networks have been trained by the gradient learning; the statistics are always computed from 10 repetitions of the runs. A very simple procedure has been applied to determine the best architecture for the RBF networks: a few reasonable sizes of hidden layer (10, 15, 20, 30 units) have been tried and the best one selected for the comparison. It is possible that cross-validation might further improve these settings; nevertheless, the current results are already competitive.

Table 6.19 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 MLP. Figure 6.23 brings the comparison of the training and test errors of regularization networks and RBF networks.

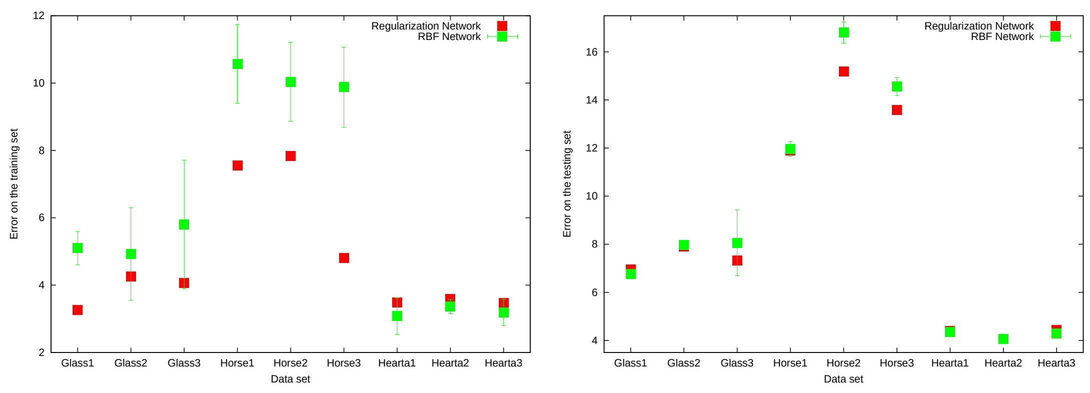

Figure 6.23: Comparison of error values of RBF vs. RN results on the training and test set.

In terms of the test error, the regularization networks achieved the best results on 23 tasks; the RBF networks on 8 tasks (see Table 6.19). One can see that both the training and test 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[3] 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 possess-

[3]Note that the RBF unit has more parameters (center, width, and posibly weigthed norm matrix) than the RN unit (only center). In this case, RBF units without norm matrices are used, so there is only one extra parameter in each unit.

	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
heartac1	**2.78**	228	3.69	0.07	10	2.82	0.22	2+0
heartac2	**3.86**	228	4.98	0.03	10	4.54	0.87	8+4
heartac3	**5.01**	228	5.81	0.00	10	5.37	0.56	16+8
hearta1	4.40	690	**4.36**	0.00	15	4.76	1.14	32+0
hearta2	**4.05**	690	**4.05**	0.00	10	4.52	1.10	16+0
hearta3	4.43	690	**4.29**	0.00	10	4.81	0.87	32+0
heartc1	**16.02**	228	16.17	0.06	10	17.18	0.79	16+8
heartc2	**6.10**	228	6.49	0.03	10	6.47	2.86	8+8
heartc3	**12.66**	228	14.35	0.37	10	14.57	2.82	32+0
heart1	**13.65**	690	14.05	0.15	10	14.33	1.26	32+0
heart2	13.80	690	**11.67**	0.46	20	14.43	3.29	32+0
heart3	15.99	690	**12.02**	0.50	15	16.58	0.39	32+0
horse1	**11.90**	273	11.96	0.04	10	13.95	0.60	16+8
horse2	**15.18**	273	16.80	0.10	10	18.99	1.21	16+8
horse3	**13.58**	273	14.56	0.07	10	17.79	2.45	32+0
soybean1	**0.66**	513	0.73	0.00	30	1.03	0.05	16+8
soybean2	**0.49**	513	0.60	0.14	30	0.90	0.08	32+0
soybean3	**0.58**	513	0.72	0.01	30	1.05	0.09	16+0

Table 6.19: Comparison of E_{test} of RN, RBF, and MLP.

ing large data amounts, “cheaper” alternatives represented by generalized regularization networks, such as RBF networks, are more competent.

To show that the RN networks and RBF networks represent competitive learning methods not only to the MLP, but also to modern learning algorithms, we picked the comparison to the SVM (Support Vector Machine).

The comparison was made on the classification tasks CANCER and GLASS. The SVM was trained using the available library [7], which represents a current standard of SVM learning.

Table 6.20 compares the RN, RBF network, and SVM in terms of classification accuracy on the test set, i.e. the percentage of correctly classified samples. The regularization networks achieved the highest accuracy on 3 tasks (CANCER1, GLASS1, GLASS2), RBF network on 2 tasks (CANCER2, CANCER3), and the SVM on one task (GLASS3). In general, 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 worthy 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 6.20: Comparison of classification accuracy of RN, RBF network, and support vector machines (SVM).

6.6 Rainfall-Runoff Modeling

In this section we describe an application of regularization networks and RBF networks to the 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. The Ploučnice River springs in the southwest slope of Ještěd hill (1012 meters above sea level) in the altitude of 654 meters above sea level, and it flows to the Elbe River in the town of Děčín in 122 meters above sea level. For our experiments, we have chosen the Ploučnice valley from its beginning to the town of Mimoň. The catchment area is 1193,9 km^2 in total, from the beginning to Mimoň it is 267,9 km^2 only, the flow length is 106,2 km.

The historical data including daily flow and rainfall values for the Ploučnice valley between November 1994 and April 2003 have been collected. The data are provided by the Czech Hydrometeorological Institute in Ústí nad Labem.

Name	History	Current flow rate	Inputs	Outputs	Training samples	Test samples
plouc1	1 day	no	2	1	1000	367
plouc1s	1 day	yes	3	1	1000	367
plouc2	2 days	no	4	1	1000	366
plouc2s	2 days	yes	5	1	1000	366

Table 6.21: Overview of data sets for the flow rate forecast on the river Ploučnice.

The data set has been split into training data — 1000 days between January 1999 and September 2001, and testing data — between October 2001 and April 2003.

We have performed two series of experiments, in the first one we worked with one--day history models, in the second one with two-day history ones. In one-day history models we tried to predict tomorrow's flow from the today's values of rainfall amount and flow. This is handled as fitting the function $f : \mathbb{R}^2 \to \mathbb{R}$, which for the values of flow rate and rainfall from the previous day returns the value of flow rate for the current day. The corresponding data set is labeled as PLOUC1.

For two-day history we took the rain falls and flows from two previous days into consideration. Thus, the approximated function is $f : \mathbb{R}^4 \to \mathbb{R}$. The data set is referred to as PLOUC2.

In both the cases, we consider the prediction using the input vector enhanced by the current rain fall value. In practical applications, the prediction of such a value is used. The corresponding data sets are labeled as PLOUC1S and PLOUC2S. All the data sets are summarized in Table 6.21.

Besides the training and test error (6.1) we evaluate the *efficiency coefficient* (EC) that is used to quantitatively measure the performance of rainfall-runoff models. The efficiency coefficient is defined as

$$EC = 1 - \frac{\sum_{i=1}^{N}(Q_{mi} - Q_{pi})^2}{\sum_{i=1}^{N}(Q_{mi} - \overline{Q})^2}, \tag{6.2}$$

where N is the number of samples, Q_{mi} are the daily measured flows, Q_{pi} the corresponding predicted flows, and $\overline{Q}$ is the average measured flow value. $0 < EC < 1$, the bigger the value is, the better performance.

First we applied a regularization network and an RBF network on all variants of the task. Table 6.22 and Figure 6.24 show the results obtained by the RBF network with

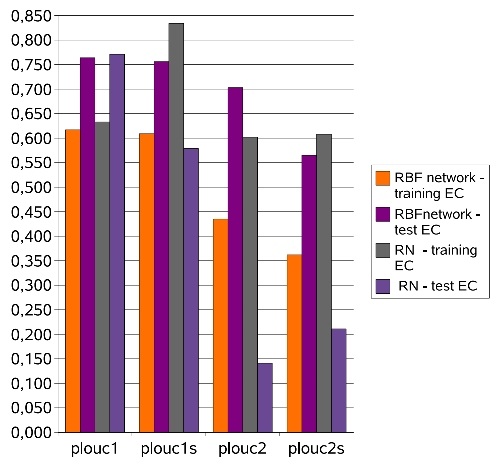

Figure 6.24: The efficiency coefficients for different data sets.

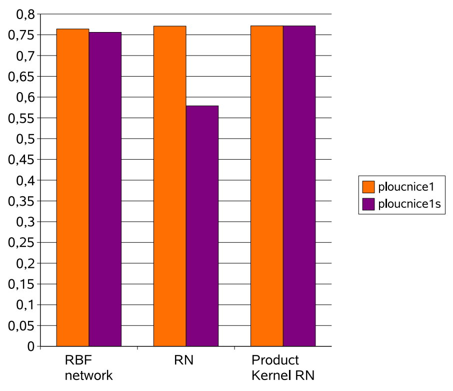

Figure 6.25: The efficiency coefficient obtained by the RBF network, the RN and the PKRN.

	RBF network				**RN**			
Task	E_{train}	E_{test}	EC_{train}	EC_{test}	E_{train}	E_{test}	EC_{train}	EC_{test}
plouc1	0.059	0.049	0.617	0.764	0.057	0.048	0.633	0.771
plouc1s	0.061	0.051	0.609	0.756	0.025	0.089	0.834	0.579
plouc2	0.088	0.062	0.435	0.703	0.062	0.182	0.602	0.141
plouc2s	0.099	0.092	0.362	0.565	0.061	0.167	0.608	0.211

Table 6.22: Training and test errors and efficiency coefficients obtained by the RBF network and the regularization network on the individual data sets.

30 hidden units and the regularization network with Gaussian kernels. We can see that the longer history (2 days) has not brought any improvement. The efficiency achieved on PLOUC2 and PLOUC2S is even smaller than on the tasks PLOUC1 and PLOUC1S. We think that this is caused by the sparse sampling of the observations. The Ploučnice River is rather small and it is possible that two-day old information has no influence on the current flow rate.

In the next experiment, we applied the regularization network with product kernels on PLOUC1 and PLOUC1S. The flow rate values and the rainfall values were treated separately, by the Gaussians of different widths. The results are compared to the RBF network and the RN in Table 6.23 and Figure 6.25. We can see that on the PLOUC1

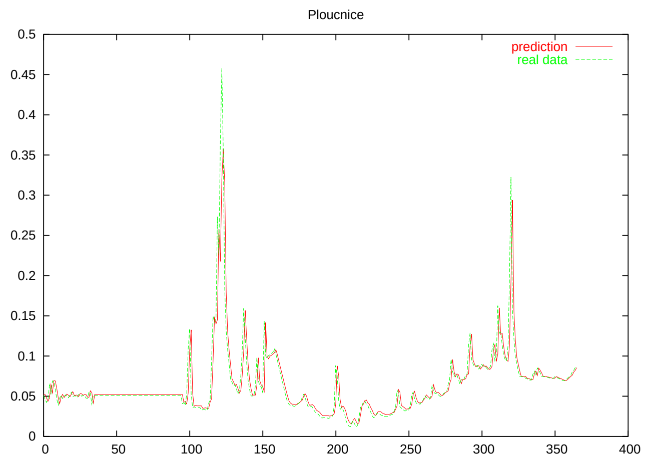

Figure 6.26: Prediction of flow rate by the regularization network.

tasks, the PKRN obtained almost the same results[4] as the simple RN. However, on the PLOUC1S it obtained better results in term of the efficiency coefficient than the simple RN, and slightly better than the RBF network. Finally, Figure 6.26 shows the real and predicted flow rate.

Task	**RBF network**		**RN**		**PKRN**	
	E_{test}	EC_{test}	E_{test}	EC_{test}	E_{test}	EC_{test}
plouc1	0.049	0.764	0.048	0.771	0.048	0.771
plouc1s	0.051	0.756	0.089	0.579	0.048	0.771

Table 6.23: Test errors and efficiency coefficients obtained by the RBF networks, RN, and PKRN.

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.

Better results were obtained using only one-day history. It is probable that the results may be further improved by using more frequent sampling [5].

[4]after round-off
[5]for example measurements at least every 12 hours

6.7 Summary

In the experimental results presented in this chapter, we demonstrated the real behavior of algorithms based on the regularization theory presented in the previous chapters.

In our experiments with the regularization networks, we illustrated the role of meta-parameters, i.e. the regularization parameter and the kernel function, and showed that their choice significantly influences the quality of the solution.

The most common kernel functions were compared on the PROBEN1 benchmark repository. The best results were achieved by the inverse multi-quadratic and Gaussian kernel functions, which are the representatives of the family of kernels with local response. Such kernels represent a good first choice for the setup of kernel function.

Also product and sum kernels were demonstrated. The sum kernels outperform the simple Gaussian kernel on most tasks from PROBEN1. In addition, the sum of two Gaussians of different widths exhibits the ability to achieve a very small training error while preserving generalization.

The *Divide et Impera* approach was applied on the benchmark data. It significantly decreases the time and space requirements, while the results obtained are comparable with the standard algorithm. Such an approach represents an option for large data sets that are not suitable for the processing by the standard algorithm.

In addition to the regularization networks, the RBF networks representing the family of generalized regularization networks were tested. Different learning approaches were compared, and we also showed that the individual methods may be further improved by hybrid approaches.

Finally, the regularization networks and RBF networks were compared in order to illustrate the difference between an 'exact' and an 'approximate' solution of the learning problem. We showed that the RBF networks obtained results competitive to the regularization networks. They achieved in average only about 6% higher error than regularization networks, while they have up to 50 times less hidden units. So they can be used as their cheaper alternative, especially for tasks with large data sets.

As an example of a real-life application we presented the prediction of river flow rate. The prediction was made on the Ploučnice River. Both the regularization networks and RBF networks were applied. Both the approaches can be successfully used for creating small rainfall-runoff models.

Chapter 7

Conclusion

Science is always wrong. It never solves a problem without creating ten more.
George Bernard Shaw

This chapter summarizes the work achieved in the thesis and its main results. In addition, several possible directions for the further research are discussed.

7.1 Main Results

In general, 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 (Algorithm 4.1.1) derived directly from the theory, and various learning algorithms for the RBF networks were investigated.

The basic RN learning algorithm represents an incomplete tool for learning, since it requires a non-trivial setup of metaparameters. It was shown in the experiments (Section 6.3.1) that these metaparameters, the regularization parameter and the kernel function, significantly influence the quality of the solution.

Therefore the framework above the basic RN learning algorithm was created (Section 4.3), including the estimation of the metaparameters. The metaparameters with the lowest cross-validation value are sought by the setup phase. Two techniques for this setup were introduced — the adaptive grid search (Algorithm 4.5) and the genetic parameter search (Algorithm 4.6.1).

Since the choice of the kernel function plays a crucial role in learning, we decided that this part of regularization network deserves more attention. It resulted in proposing the composite types of kernel functions — a product kernel and sum kernel. In the experiments (Section 6.3.4) we demonstrated their feasibility and showed that they are a vital alternative to the classical (i.e. simple) kernels. These kernels are especially useful on the tasks that are heterogenous in some sense, either varying in attributes or different parts of the input space.

A 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 wide. Such a kernel function obtained good results even without the regularization term[1]. Almost zero training errors were achieved[2], while the generalization property was preserved due to the wider Gaussian. Such kernel functions may be very useful for the tasks with a low level of noise.

Inspired by the concept of restricted sum kernels, we proposed the "Divide et Impera" approach (Section 3.5.2). It is a simple procedure that splits the tasks into several disjunct subtasks. The learning algorithm is applied on each of these subtasks, possibly in parallel. The solution is then computed as a sum of the networks obtained. Such an approach does not only save the space, but also significantly reduces the time requirements (Section 6.3.5).

Despite the thorough theoretical background, the regularization network may be not feasible in some situations. Particularly, the solution is too large for the 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 in Section 5.2. These approaches were compared in the experiments (Section 6.4). 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 in terms of error.

Inspired by these results, the two hybrid approaches were proposed — the four-step learning (Algorithm 5.6.2) and the hybrid genetic learning (Algorithm 5.6.1). Their behavior was demonstrated in experiments (Section 6.4.5) and it was shown, that they in some aspects, improve the original algorithms.

The four-step learning adds a gradient optimization to the three-step learning. It achieves lower errors than the three-step learning and still has lower time requirements than the gradient learning. The hybrid genetic learning represents a combination of the genetic learning and the third part of the three-step learning. It achieved very good results, outperforming the other approaches; however, it suffers from very high time requirements.

[1]i.e. with zero regularization parameter
[2]because of the zero regularization parameter and the narrow Gaussian

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 (Section 6.5), the regularization networks and RBF networks achieved comparable results. So we claim that the RBF networks represent a cheaper alternative to the regularization networks, in terms of model size and learning time.

Finally, we presented an application to a real-life problem. Both the regularization networks and RBF networks were successfully applied on the prediction of the river flow rate.

7.2 Future Work

The thesis would be incomplete if we did not mention the possible directions of further research. Some of them, namely the control of learning parameters and composite kernels, have already been in the state of work in progress, the others are just ideas that appeared during work on this thesis.

- **Parallelization:** parallelization may bring significant speed-up to the learning. The learning techniques based on the GAs and the setup techniques based on cross-validation are suitable for straightforward parallelization, since they contain many independent evaluations of subtasks. On the other hand, the learning algorithms based on the gradient descent are known to be difficult to parallelize.

- **Pruning**: though the number of hidden units of regularization network is usually high, not all of them are necessarily needed. Since similar results were obtained by much smaller RBF networks on rather all tasks, it is probable that the network may be pruned in some way (for example by examining the output weights) and its size reduced.

- **Regularization**: the regularization presented in this works corresponds to the well-known Tikhonov regularization. The ill-posed problems are well-studied by numerical mathematics, and various approaches and techniques exist. The possibilities of application of other approaches (e.g. non-linear regularization) in the context of machine learning deserve more attention.

- **Composite Kernels:** more complicated composite kernel functions should be investigated, especially for the application to diverse data, i.e. data containing not only numbers, but objects like strings, trees, documents, etc. One should search for new ways of designing kernel functions and create the composite ones.

- **Large Data:** nowadays rather real problems work with very large data sets. It is more than necessary to study the ways of time and space complexity reductions, making the algorithms more effective and applicable on such tasks.

- **Control of the Learning Parameters:** the performance of learning algorithms based on the gradient descent algorithm (back-propagation, gradient learning for RBF networks, etc.) depends on the setup of learning parameters, especially the learning rate. Developing automatic adaptive controllers that will tune the learning parameters during learning may speed up the learning.

Bibliography

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

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

[3] Ch. Bishop. Improving the generalization properties of RBF neural networks. *Neural Computation*, 3:579–588, 1991.

[4] E. Björk. *Numerical methods for least square problems*. SIAM, Philadelphia, 1996.

[5] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[6] O. Buchtala, M. Klimek, and B. Sick. Evolutionary optimization of radial basis function classifiers for data mining applications. *IEEE Transactions on Systems, Man and Cybernetics*, 35(5):928–947, Oct. 2005.

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

[8] T.M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, pages 326–334, 1965.

[9] J. W. Demmel. The geometry of ill-conditioning. *J. Complex.*, 3(2):201–229, 1987.

[10] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases, 1998.

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

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

[13] R. M. Gray. Vector quantization. *IEEE ASP Magazine*, 1:4–29, 1984.

[14] J. Hadamard. Sur les problèmès aux dèrivèes partielles et leur signification physique. In *Princeton University Bulletin*, pages 49–52. 1902.

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

[16] K. Hlaváčková and R. Neruda. Radial basis function networks. *Neural Network World*, 3(1):93–101, 1993.

[17] Lenna Image. `http://www.lenna.org`.

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

[19] P. Krušina, P. Kudová, Z. Petrová, and R. Neruda. Hybrid AI models using bang. Technical Report V-879, Institute of Computer Science, AS CR, 2002.

[20] P. Kudová. Comparison of kernel based regularization networks and RBF networks. Abstract. In Antoch J., editor, COMPSTAT'2004. Book of abstracts, page 191. Prague, 2004.

[21] P. Kudová. Neuronové sítě typu RBF pro analýzu dat. Master's thesis, MFF UK, 2001.

[22] P. Kudová. Učení neuronových sítí typu RBF. Technical Report V-846, Institute of Computer Science, AS CR, 2001.

[23] P. Kudová. Genetic and eugenetic learning of RBF networks. Technical report, Institute of Computer Science, AS CR, 2002.

[24] P. Kudová. Hybrid learning of RBF networks. In Šafránková J., editor, *WDS03 Proceedings of Contributed Papers*, volume 1. Mathematics and Computer Sciences, pages 102–107. Praha, MATFYZPRESS, 2003.

[25] P. Kudová. Learning of generalized regularization networks. Technical Report V-851, Institute of Computer Science, AS CR, 2003.

[26] P. Kudová. Learning of generalized regularization networks. In Hakl F., editor, *Doktorandský den 03*, pages 60–66. Praha, MATFYZPRESS, 2003.

[27] P. Kudová. Comparison of kernel based regularization networks and RBF networks. In Vojtáš P., editor, *Information Technologies - Applications and Theory*. Prrodovedeck fakulta Univerzity Pavla Jozefa afrika, 2004.

[28] P. Kudová. Kernel based regularization networks and RBF networks. In Hakl F., editor, *Doktorandský Den 2004, Paseky nad Jizerou*, pages 59–65. Praha, MATFYZPRESS, 2004.

[29] P. Kudová. Kernel based regularization networks. In Hakl F., editor, *Doktorandský den 05*, pages 65–74. Praha, MATFYZPRESS, 2005.

[30] P. Kudová. Learning with kernel based regularization networks. In *Information Technologies - Applications and Theory*, pages 83–92. Košice, Prírodovedecká fakulta Univerzity Pavla Jozefa Šafárika, 2005.

[31] P. Kudová. Learning with regularization networks in Bang. In TAM'06, `http://www.bsc.es/TAM200/talks.htm`, 6 2006.

[32] P. Kudová. The role of kernel function in regularization network. In *In proceedings of conference ITAT'2006*, 2006. IN PRINT.

[33] P. Kudová and R. Neruda. Kernel based learning methods: Regularization networks and RBF networks. In Lawrence N. Winkler J., Niranjan M., editor, *Deterministic and Statistical Methods in Machine Learning*, pages 124–136. Berlin, Springer-Verlag, 2005.

[34] P. Kudová, H. Řezanková, D. Húsek, and V. Snášel. Categorical data clustering using statistical methods and neural networks. In E. Barashev, S. Kuznetsov, P. Velikhov, and Novikov B., editors, *SYRCoDIS'2006, Spring Young Researchers Colloquium on Database and Information Systems*, 3, pages 19–23, Moscow, Russia, 2006.

[35] P. Kudová and T. Šámalová. Product kernel regularization networks. In Ribeiro B., Albrecht R.F., Dobnikar A., Pearson D.W., and Steele N.C., editors, *Adaptive and Natural Computing Algorithms*, pages 433–436. Wien, SpringerVerlag, 2005.

[36] P. Kudová and T. Šámalová. Sum and product kernel regularization networks. In L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, and J. Zurada, editors, *Artificial Intelligence and Soft Computing*, Lecture Notes in Artificial Intelligence, pages 56–65. Berlin, Springer-Verlag, 2006.

[37] P. Langley. Machine learning as an experimental science. *Machine Learning*, 3(1):5–8, 1988.

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

[39] W.A. Light. Some aspects of radial basis function approximation. In *Approximation Theory, Spline Functions and Applications*, pages 163–190. Kluwer Academic Publishers, 1992.

[40] Neruda M., R. Neruda, and P. Kudová. Forecasting runoff with artificial neural networks. In Arattano M. Maraga F., editor, *Progress in Surface and Subsurface Water Studies at Plot and Small BAsin Scale*, pages 65–69. Paris, UNESCO, 2005.

[41] 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`.

[42] H. N. Mhaskar. When is approximation by gaussian networks necessarily a linear process? *Neural Networks*, 17(7):989–1001, 2004.

[43] C. A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 1986.

[44] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.

[45] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[46] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:289–303, 1989.

[47] J. Moody and C. Darken. Fast adaptive k-means clustering: some empirical results. In *Proceedings of the IJCNN'90*, volume 2, pages 233–238, San Diego, 1990.

[48] E. H. Moore. On the reciprocal of the general algebraic matrix. In *Bulletin of the American Mathematical Society*, volume 26, pages 394–395. 1920.

[49] F.J. Narcowich, N. Sivakumar, and J.D. Ward. On condition numbers associated with radial-function interpolation. *Journal of Mathematical Analysis and Applications*, 186:457–485, 1994.

[50] R. Neruda. *Functional Equivalence and Genetic Learning of RBF networks*. PhD thesis, MFF UK, Charles University, Prague., 1997.

[51] R. Neruda, P. Krušina, P. Kudová, P. Rydvan, and G. Beuster. Bang 3: A computational multi-agent system. In Zhong N., Bradshaw J., Pal S.K., Talia D., Liu J., and Cerrone N., editors, *Intelligent Agent Technology*, pages 563–564. Piscataway, IEEE, 2004.

[52] R. Neruda and P. Kudová. Hybrid learning of RBF networks. In Sloot P.M.A., Tan C.J.K., Dongarra J.J., and Hoekstra A.G., editors, *Computational Science*, pages 594–603. Berlin, Springer, 2002.

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

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

[55] PAPI. Performance application programming interface, `http://icl.cs.utk.edu/papi/`.

[56] R. Penrose. A generalized inverse for matrices. In *Proceedings of the Cambridge Philosophical Society*, volume 51, pages 406–413. 1955.

[57] 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.

[58] T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical report, Cambridge, MA, USA, 1989. A. I. Memo No. 1140, C.B.I.P. Paper No. 31.

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

[60] M.J.D Powel. Radial basis functions for multivariable interpolation: A review. In *IMA Conference on Algorithms for the Approximation of Functions and Data*, pages 143–167, RMCS, Shrivenham, England, 1985.

[61] L. Prechelt. PROBEN1 – a set of benchmarks and benchmarking rules for neural network training algorithms. Technical Report 21/94, Universitaet Karlsruhe, 9 1994.

[62] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1986.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. pages 318–362, 1986.

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

[65] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[66] M. Stone. Cross-validation: A review. *Mathematics, Operations and Statistics.*, 9:127 – 140, 1978.

[67] A.N. Tikhonov and V.Y. Arsenin. *Solutions of Ill-posed Problems.* W.H. Winston, Washington, D.C, 1977.

[68] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, Berlin, 1982.

[69] V. N. Vapnik. *Statistical Learning Theory*. Wiley, New-York, 1998.

[70] H. Řezanková, D. Húsek, P. Kudová, and V. Snášel. Comparison of some approaches to clustering categorical data. In *COMPSTAT 2006 [CD-ROM]*, pages 607–613, Roma : Universita degli Studi di Roma La Sapienza, 2006.

[71] T. Šámalová and P. Kudová. Sum and product kernel networks. Technical Report V-935, Institute of Computer Science, AS CR, 2005.

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

[73] G. Wahba. Spline models for observational data. *Series in Applied Mathematics*, 59, 1990.

[74] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[75] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation*, volume 2 of *Linear Algebra*. Springer, Berlin, 1971.

[76] D.H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computations*, 8(7):1341–1390, 1996.

Appendix A

Obtaining the thesis electronically

The thesis in postscript and PDF format and additional materials can be found on the CD enclosed. The directory structure of the CD is following:

thesis/	contains the text of the thesis in postscript and PDF format
other/	contains the extended thesis abstract, author's CV and list of publications
data/	contains the data sets used in the experimental part of our work

Alternatively, the thesis, as well as the other materials, can be download from:

http://www.cs.cas.cz/˜petra/phd/

The experiments presented in the thesis were performed using our implementation of the studied algorithms. The implementation was realized as a part of the multi-agent system Bang that is developed at the Institute of Computer Science, Academy of Sciences of the Czech Republic. The Bang system can be obtained via CVS:

```
cvs -d:pserver:anonymous@bang.cvs.sourceforge.net:/cvsroot/bang login
cvs -z3 -d:pserver:anonymous@bang.cvs.sourceforge.net:/cvsroot/bang
 checkout -P bang3
```

You may also consult the home page of the project:

http://bang.sf.net/

To compile and run the Bang system, one needs a POSIX standards conforming operating system. For example, Linux, OpenBSD, FreeBSD and Irix should work. A recent C++ compiler conforming to ISO C++ standards is also necessary. GNU C++ 3.x is a good choice, but other compilers should work as well.

In case of any difficulties, contact the author by e-mail: petra@cs.cas.cz.