Learning Errors by Radial Basis Function Neural Networks and Regularization Networks

Roman Neruda, Petra Vidnerová
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, Prague 8, Czech Republic
roman@cs.cas.cz

Abstract

Regularization theory presents a sound framework to solving supervised learning problems. However, there is a gap between the theoretical results and practical suitability of regularization networks (RN). Radial basis function networks (RBF) that can be seen as a special case of regularization networks have a rich selection of learning algorithms. In this work we study a relationship between RN and RBF, and show that theoretical estimates for RN hold for a concrete RBF applied to real-world data, to a certain degree. This can provide several recommendations for strategies on choosing number of units in RBF network.

1 Introduction

The problem of *learning from examples* (also called *supervised learning*) is a subject of great interest. The problem can be formulated as follows. We are given a set of examples $\{(\vec{x}_i, y_i) \in R^d \times R\}_{i=1}^N$ that was obtained by random sampling of some real function f, generally in presence of noise. To this set we refer as *a training set*. The goal is to recover the function f from data, or find the best estimate of it. It is not necessary that the function exactly interpolates all the given data points, but we need a function with good generalization. That is a function that gives relevant outputs also for the data not included in the training set.

The supervised learning of neural networks is often studied as a function approximation problem [2]. Given the data set, we are looking for the function that approximate the unknown function f. It is usually done by *empirical risk minimization*, i.e. minimizing the functional $H[f] = \frac{1}{N}\sum_{i=1}^N (f(\vec{x}_i) - y_i)^2$ over a chosen *hypothesis space*, i.e. over a set of functions represented by a chosen type of neural network. In section 2 we will study the problem of learning from examples as a function approximation problem and show how regularization network (RN) is derived from regularization theory. In section 3 we will describe one type of neural network—an RBF network that can be seen as a special case of RN. Learning methods based on regularization approach have in general very good theoretical background. Also the relation between the number of hidden units and approximation accuracy was extensively studied and bounds on convergence rate of solutions with limited number of hidden units to optimal solution (e.g. [10, 1, 3]) derived. In the section 4 we demonstrate that the theoretical estimates for RN to some degree holds for RBF networks and derive several recommendations for choosing number of units.

2 Approximation via regularization network

In this section we will study the problem of learning from examples by means of regularization theory. We are given a set of examples $\{(\vec{x}_i, y_i) \in R^d \times R\}_{i=1}^N$ obtained by random sampling of some real function f and we would like to find this function. Since this problem is ill-posed, we have to add some a priori knowledge about the function. We usually assume that the function is *smooth*, in the sense that two similar inputs corresponds to two similar outputs and the function does not oscillate too much. This is the main idea of the regularization theory, where the solution is found by minimizing the functional (1) containing both the data and smoothness information.

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

where Φ is called a *stabilizer* and $\gamma > 0$ is *the regularization parameter* controlling the trade off between the closeness to data and the smoothness of the solution. The regularization scheme (1) was first introduced by Tikhonov [9] and therefore it is called a Tikhonov regularization. The regularization approach has good theoretical background, it was shown that for a wide class of stabilizers the solution has a form of feed-forward neural network with one hidden layer, called *regularization network*, and that different types of stabilizers lead to different types of regularization networks [5, 6].

Poggio and Smale in [6] proposed a learning algorithm 3.1 derived from the regularization scheme (1). They choose the hypothesis space as a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_K$ defined by an explicitly chosen, symmetric, positive-definite kernel function $K_{\vec{x}}(\vec{x}') = K(\vec{x}, \vec{x}')$. The stabilizer is defined by means of norm in $\mathcal{H}_K$, so the problem is formulated as follows:

$$\min_{f \in \mathcal{H}_K} H[f], \text{where } H[f] = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\vec{x}_i))^2 + \gamma||f||_K^2. \tag{2}$$

The solution of minimization (2) is unique and has the form

$$f(\vec{x}) = \sum_{i=1}^{N} c_i K_{\vec{x}_i}(\vec{x}), \qquad (N\gamma I + K)\vec{c} = \vec{y}, \tag{3}$$

where I is the identity matrix, K is the matrix $K_{i,j} = K(\vec{x_i}, \vec{x_j})$, and $\vec{y} = (y_1, \ldots, y_N)$. The most commonly used kernel function is Gaussian $K(\vec{x}, \vec{x}') = e^{-\left(\frac{\|\vec{x}-\vec{x}'\|}{b}\right)^2}$.

Input: Data set $\{\vec{x}_i, y_i\}_{i=1}^N \subseteq X \times Y$ **Output:** Function f.

1. Choose a symmetric, positive-definite function $K_{\vec{x}}(\vec{x}')$, continuous on $X \times X$.
2. Create $f : X \to Y$ as $f(\vec{x}) = \sum_{i=1}^N c_i K_{\vec{x}_i}(\vec{x})$ and compute $\vec{c} = (c_1, \ldots, c_N)$ by solving

$$(N\gamma I + K)\vec{c} = \vec{y}, \tag{4}$$

where I is the identity matrix, K is the matrix $K_{i,j} = K(\vec{x_i}, \vec{x_j})$, and $\vec{y} = (y_1, \ldots, y_N)$, $\gamma > 0$ is real number.

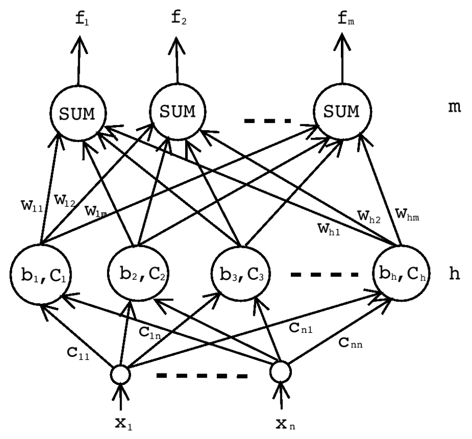
Algorithm 2.1

The power of the algorithm (2.1) is in its simplicity and effectiveness. On the other hand, it has also big drawbacks. First of all, the size of the model (that is a number of kernel functions) corresponds to the size of the training set and so the tasks with huge data sets lead to solutions of implausible size. Then there are the parameters γ and in case of Gaussian kernel also width b, which are supposed to be fixed. Once they are fixed, the algorithm reduces to the problem of solving linear system of equations (4).

Since the system has N variables, N equations, K is positive-definite and $(N\gamma I + K)$ is strictly positive, it is well-posed, i.e. is has a unique solution and the solution exists. But we would also like it to be well-conditioned, i.e. insensitive to small perturbations of the data. In other words, we would like the condition number of the matrix $(N\gamma I + K)$ to be small, which is fulfilled if $N\gamma$ is large. Note that we are not entirely free to choose γ, because with too large γ we loose the closeness to data. The second parameter b determines the width of the Gaussians. Suppose that the distances between the data points are high or the widths are small, than the matrix K has 1s on diagonal and small numbers everywhere else and therefore is well-conditioned. If the widths are too large, all elements of the matrix K are close to 1 and its condition number tends to be high. The real performance of the algorithm depends significantly on the choice of parameters γ and b. The optimal choice of these parameters depends on a particular data set and there is no general heuristics for setting them.

3 RBF neural networks

An RBF neural network (RBF network) represents a relatively new model of neural network. On the contrary to classical models (multilayer perceptrons, etc.) it is a network with local units which was motivated by the presence of many local response units in human brain. Other motivation came from numerical mathematics, radial basis functions (RBF) were first introduced in the solution of real multivariate problems [7].



$$y(\vec{x}) = \varphi\left(\frac{\| \vec{x} - \vec{c} \|_C}{b}\right) \qquad (5)$$

$$f_s(\vec{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\| \vec{x} - \vec{c}_j \|_{C_j}}{b_j}\right) \qquad (6)$$

Figure 1. a) RBF network architecture b) RBF network function

An RBF network is a standard feed-forward neural network with one hidden layer of RBF units and linear output layer (fig. 1). By an RBF unit we mean a neuron with n real inputs and one real output, realizing a radial basis function (5), usually Gaussian. Instead of the Euclidean norm we use the *weighted norm* $\| \cdot \|_C$, where $\|\vec{x}\|^2_C = (C\vec{x})^T(C\vec{x}) = \vec{x}^T C^T C\vec{x}$. The network computes a function $\vec{f} = (f_1, \ldots, f_m)$ as linear combination of outputs of the hidden layer (see (6)).

The goal of RBF network learning is to find the parameters (i.e. centers $\vec{c}$, widths b, norm matrices C and weights w) so as the network function approximates the function given by the training set $\{(\vec{x}_i, \vec{y}_i) \in R^n \times R^m\}_{i=1}^N$. There is a variety of algorithms for RBF network learning, in our past work we studied their behavior and possibilities of their combinations [4]. The two most significant algorithms,*Three step learning* and *Gradient learning*, are sketched in Algorithm 2.1 and Algorithm 2.2. See [4] for details.

```
Input: Data set $\{\vec{x}_i, \vec{y}_i\}_{i=1}^N$                    Output: $\{\vec{c}_i, b_i, C_i, w_{ij}\}_{i=1..h}^{j=1..m}$
1. Set the centers $\vec{c}_i$ by a k-means clustering.
2. Set the widths $b_i$ and matrices $C_i$.
3. Set the weights $w_{ij}$ by solving $\Phi W = D$.
```

$$D_{ij} = \sum_{t=1}^{N} y_{tj} e^{-\left(\frac{\|\vec{x}_t - \vec{c}_i\|_{C_i}}{b_i}\right)^2}, \Phi_{qr} = \sum_{t=1}^{N} e^{-\left(\frac{\|\vec{x}_t - \vec{c}_q\|_{C_q}}{b_q}\right)^2} e^{-\left(\frac{\|\vec{x}_t - \vec{c}_r\|_{C_r}}{b_r}\right)^2}$$

Algorithm 3.1

```
Input: Data set $\{\vec{x}_i, \vec{y}_i\}_{i=1}^N$                    Output: $\{\vec{c}_i, b_i, C_i, w_{ij}\}_{i=1..h}^{j=1..m}$
1. Put the small part of data aside as an evaluation set $ES$,
   keep the rest as a training set $TS$ .
2. $\forall j\ \vec{c}_j(i) \leftarrow$ random sample from $TS_1$, $\forall j\ b_j(i), \Sigma_j^{-1}(i) \leftarrow$ small
   random value, $i \leftarrow 0$
3. $\forall j, p(i)$ in $\vec{c}_j(i), b_j(i), \Sigma_j^{-1}(i)$:
   $\Delta p(i) \leftarrow -\epsilon \frac{\delta E_1}{\delta p} + \alpha \Delta p(i-1)$,        $p(i) \leftarrow p(i) + \Delta p(i)$
4. $E_1 \leftarrow \sum_{\vec{x} \in TS_1} (f(\vec{x}) - y_i)^2$, $E_2 \leftarrow \sum_{\vec{x} \in TS_2} (f(\vec{x}) - y_i)^2$
5. If $E_1$ and $E_2$ are decreasing, $i \leftarrow i + 1$, go to 3, else STOP. If $E_2$ started to increase, STOP.
```

Algorithm 3.2

4 Error estimates

The relation between the number of hidden units and approximation accuracy was extensively studied and bounds on convergence rate of solutions with limited number of hidden units to optimal solution (3) (e.g. [10, 1, 3]) derived.

Most of the results agree on convergence rate close to $\frac{1}{\sqrt{h}}$, where h is the number of hidden units. In [3, Theorems 4.2–6.3], upper bounds are derived on the convergence rate of suboptimal solutions to the optimal solution achievable without restrictions on the model complexity. The bounds are of the form $\frac{1}{\sqrt{h}}$ multiplied by a term depending on the data set size, the output vector, the Gram matrix of the kernel function with the respect to the input data (matrix obtained by applying kernel function on all couples of data points), and the regularization parameter.

In this section, we study the relation between the network size (i.e. number of hidden units) and approximation accuracy and generalization by experimental means. With respect to theoretical results, we expect the approximation accuracy to improve with increasing number of hidden units. On the other hand, reasonable approximation accuracy should be achieved already with small networks. Also high number of hidden units makes the learning task more difficult, which can influence the results. The situation with generalization ability is different. Networks with higher number of free parameters tends to be overtrained and exhibit poor generalization ability.

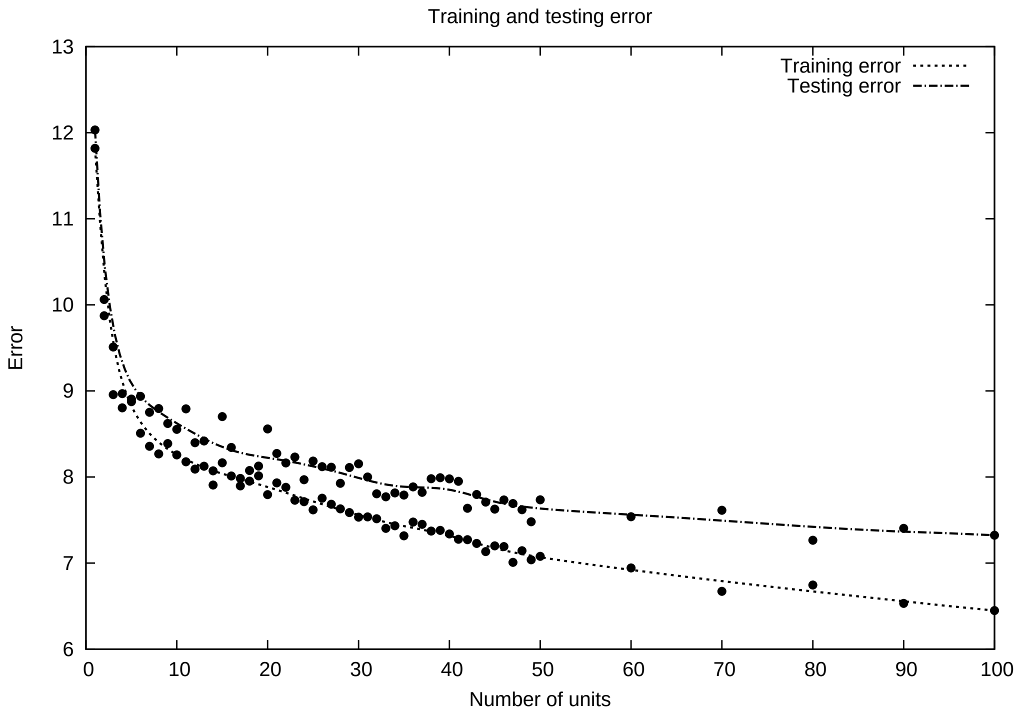


Figure 2. Cancer data set: Testing and training errors depending on the number of network units.

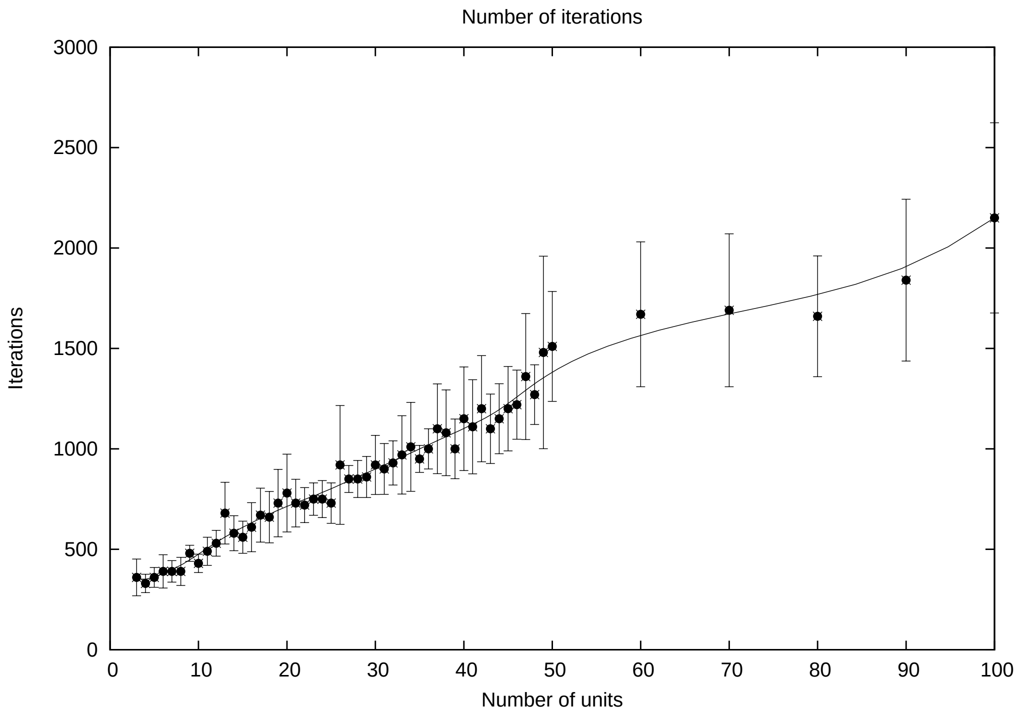


Figure 3. Cancer data set: Number of iterations depending on the number of network units.

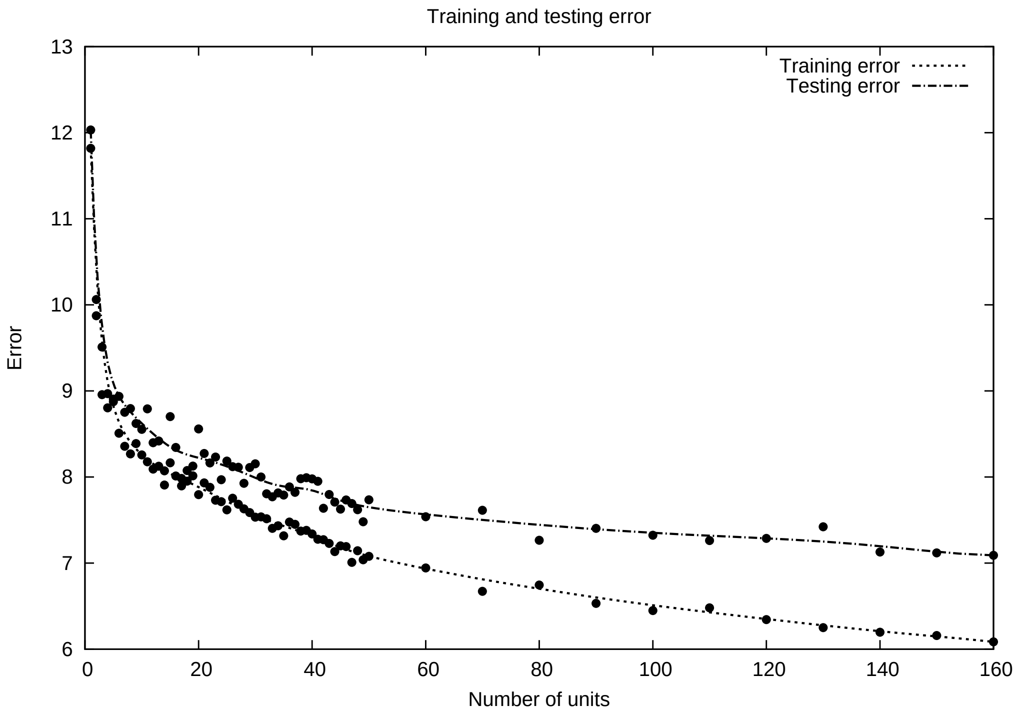


Figure 4. Glass data set: Testing and training errors depending on the number of network units.

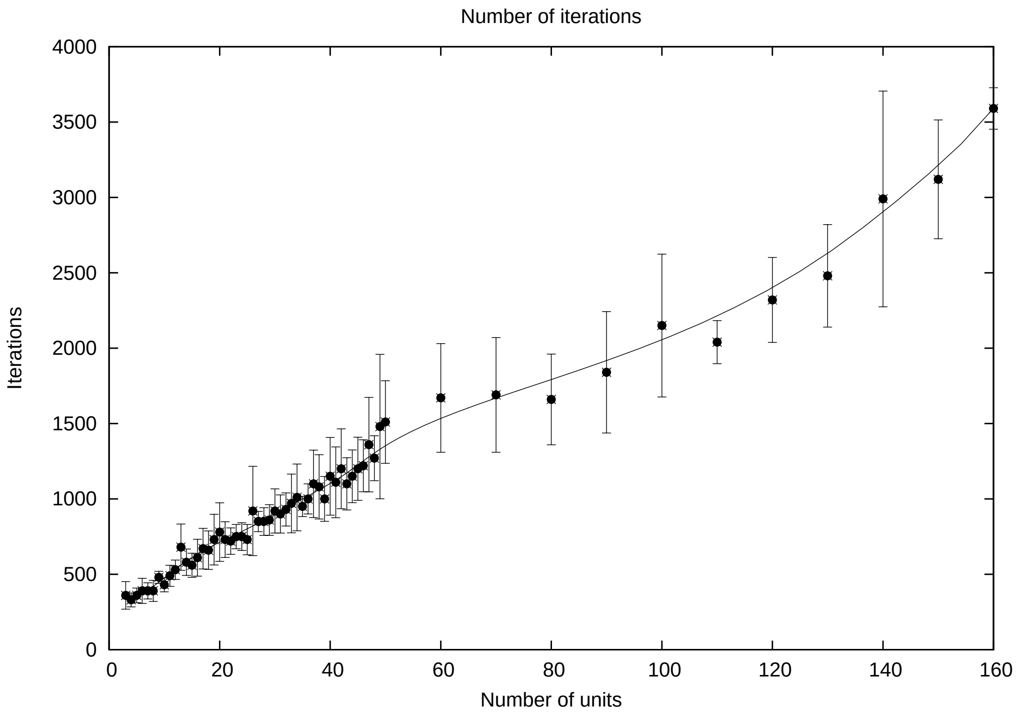


Figure 5. Glass data set: Number of iterations depending on the number of network units.

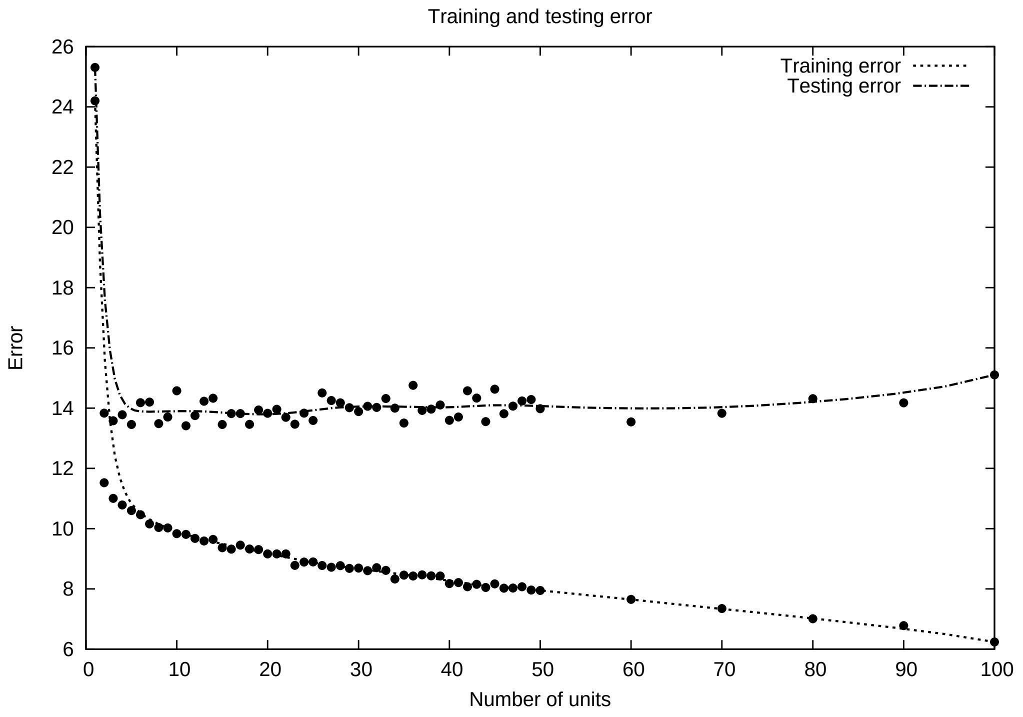


Figure 6. Heart data set: Testing and training errors depending on the number of network units.

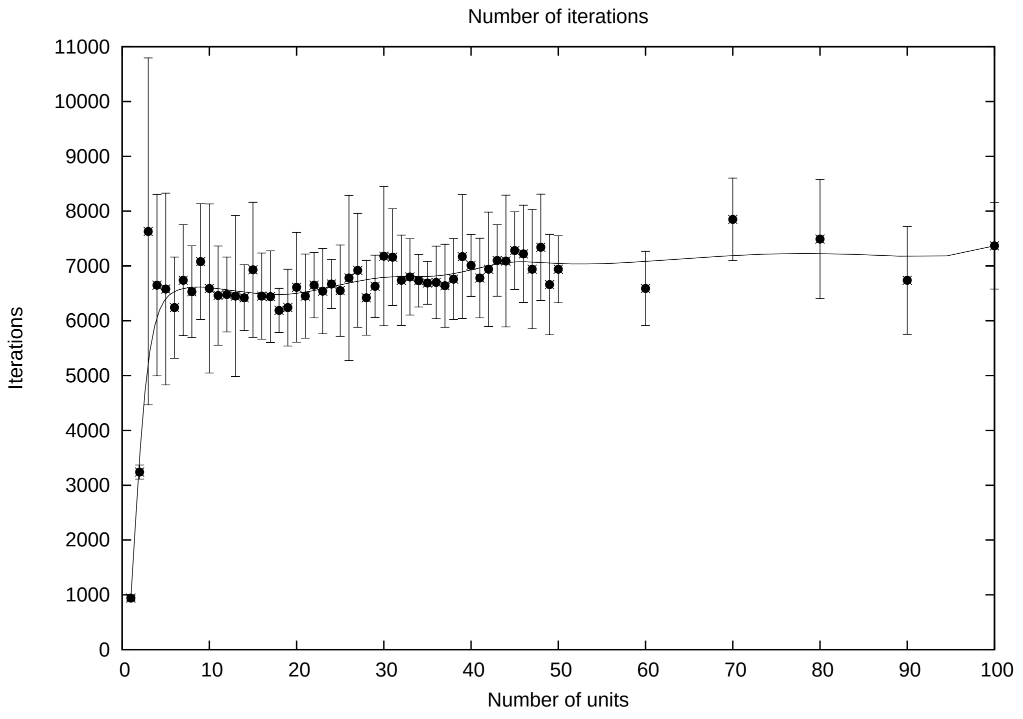


Figure 7. Heart data set: Number of iterations depending on the number of network units.

The above mentioned behavior has been demonstrated during our case study where we applied gradient based learning algorithm 2 to RBF network learning on three data sets from the Proben1 set [8] — cancer, glass and heart. In order to achieve statistically relevant results and to avoid influences of the stochastic nature of the gradient algorithm, all experiments have been performed ten times and the figure shows median values.

Figures 2, 4, and 6 show graphs of training and testing error depending on number of hidden units in the RBF networks. For the first (smallest) data set, it can be seen that the training error roughly follows the $\frac{1}{\sqrt{h}}$ curve between 1–70 units, then, the learning algorithm performs poorer, probably due to suboptimal ϵ choice. The testing error on the other hand shows that around 10 hidden units there is a good generalization ability that is again achieved for networks of 50 units, and after that a trend showing overspecification is quite clear. Two remaining sets do not exhibit this behavior, probably due to the larger size of tasks.

Figures 3, 5, and 7 show the number of iterations needed to achieve a solution depending on number of hidden units. These graphs illustrate the complexity of the task reflected for different network sizes. It is interesting to note that the curve slightly differs for the cancer and glass data sets on one hand, and the heart data set on the other hand, where the steeper growth might indicate bigger complexity of the problem.

Since the convergence is quite fast, we can suggest that small networks provide sufficiently good solutions. The theoretically estimated convergence rates justify using network of smaller complexity in real-life applications. Smaller networks have also smaller number of parameters that has to be tuned during the training process. Therefore, they are more easily trained.

5 Conclusion

Most of the learning algorithms work with networks of fixed architectures. Those optimizing also the number of hidden units can be divided into two groups – incremental and pruning. Pruning algorithm starts with large networks and tries to eliminate the irrelevant units, while incremental algorithms start with small network and add units as long as the network performance improves. The mentioned theoretical results speaks in favour of incremental algorithms. First, learning of small networks is fast since small numbers of parameters has to be optimized. Second, it is quite probable that reasonable solution will be found among smaller networks. Based on our experiments, we recommend to start with small number of hidden units and increase the network size only as long as also generalization ability improves.

There are several issues that remain to be solved in our future work. The behavior of learning algorithm is influenced by a good choice of learning parameters. In our case, an optimal selection of the learning rate ϵ of the gradient algorithm had crucial effect on the performance. Some way of automatic adaptive change of learning parameters should be tested. Moreover, we plan to perform the same experiments with a three-step learning algorithm for RBF that is closer to the RN approach and usually provides faster, if not always better solutions.

Acknowledgements

This work has been supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567).

References

[1] Valentina Corradi and Halbert White. Regularized neural networks: some convergence rate results. *Neural Computation*, 7:1225 – 1244, 1995.

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

[3] Věra Kuková and Marcello Sanguineti. Learning with generalization capability by kernal methods of bounded complexity. *J. Complex.*, 21(3):350–367, 2005.

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

[5] 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.

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

[7] 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.

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

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

[10] Lei Xu, Adam Krzyżak, and Alan Yuille. On radial basis function nets and kernel regression: statistical consistency, convergence rates, and receptive field size. *Neural Netw.*, 7(4):609–628, 1994.