Applied and
Computational
Mathematics
Division

NISTIR 5168

Computing and Applied Mathematics Laboratory

Computational Experience with Radial Basis Function Networks

James L. Blue

May, 1993

Technology Administration
U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Gaithersburg, MD 20899

Computational Experience with Radial Basis Function Networks*

James L. Blue
Applied and Computational Mathematics Division
National Institute of Standards and Technology
Gaithersburg, MD 20899

Abstract

We discuss the use of Radial Basis Functions for use in neural networks for hand-printed character recognition. The results are expected to apply to other applications of neural networks for classifying input patterns.

1. Introduction

Several types of Radial Basis Functions (RBFs) and RBF networks have been suggested. The general idea is simple: the activation of a node should be large for an input pattern close to a particular pattern defined as the node's *center*, and small for an input pattern far from the center. For an n-component input vector $\mathbf{x}$, one simple RBF has a Gaussian activation depending on the scaled Euclidean distance of $\mathbf{x}$ from center j, $\mathbf{c}_j$:

$$\begin{aligned} d_j^2 &= \sum_{i=1}^{n}(c_{ij} - x_i)^2/\sigma_j^2 \\ \phi(d_j) &= \exp(-d_j^2) \end{aligned} \tag{1}$$

However, any distance function d that is (mathematically) a norm could be used, and any activation function ϕ that is nonnnegative and decreases monotonically to zero with increasing distance could be used. (Some appropriate RBFs are discussed by Poggio and Girosi.[6]) The most general quadratic distance function is, using matrix notation,

$$d_j^2 = (\mathbf{x} - \mathbf{c}_j)^{\mathrm{T}}\mathbf{A}(\mathbf{x} - \mathbf{c}_j)$$

where $\mathbf{A}$ is any symmetric, positive-definite matrix. The easiest generalization, and the one used in this paper, is to let the scaling width, σ_j, differ for each i.

*Contribution of the National Institute of Standards and Technology. Not subject to copyright.

This is equivalent to making **A** a diagonal matrix.

$$d_j^2 = \sum_{i=1}^{n} (x_i - c_{ij})^2 / \sigma_{ij}^2 \tag{2}$$

We consider two activation functions, called types 1 and 2; type 2 includes a bias term for each RBF:

$$\begin{aligned} \phi^{(1)}(d_j) &= \exp(-d_j^2) \\ \phi^{(2)}(d_j) &= \frac{1}{1 + \exp(\theta_j + d_j^2)} \end{aligned} \tag{3}$$

Other generalizations have also been used (for example, see Musavi, et al.[5]), but are difficult to manage for large n. For function approximation, a network composed of a layer of RBFs followed by a linear output layer is commonly used; for classification, a standard output layer of sigmoidal nodes may also be used. The RBF in (1) is radially symmetric in n-dimensional space; more general ones are not, but the name has stuck.

Some early papers suggested choosing the centers and widths *a priori* and using supervised learning for selecting the output layer weights; later papers suggest learning some combination of the centers, widths, and weights.

2. Training

The current work uses training and test sets, each containing 10,000 48-component patterns selected randomly from NIST Special Database 3.[1] Each pattern is a truncated Karhunen-Loève expansion of a 32 by 32 pixel binary image of a hand-printed digit. The RBF networks considered have 24 to 48 inputs, a hidden layer consisting of from one to four RBFs per digit, and an output layer of 10 linear or sigmoidal nodes. Also considered are some standard Multi-Layer Perceptron (MLP) networks, with sigmoidal hidden and output layers.

The supervised learning minimized the standard objective function, the sum of squares of the output errors. For networks with a sigmoidal output layer, a small constant times the sum of squares of the output layer weights, was added to the objective function as a regularization, i.e. to minimize over-training. In order to simplify the gradient calculation, the inverses of the widths, $s_{ij} = 1/\sigma_{ij}$, were used as variables.

The optimization (training) was done using a combination of scaled conjugate gradients[4] and a limited-memory quasi-Newton algorithm.[3] The program was written to allow any combination of the centers, widths, and weights to be learned, and the remainder to stay fixed. Training was done with varying training set sizes, from 156 patterns to the entire 10,000 patterns; testing was done on the entire 10,000 pattern testing set.

2.1. Initial Values

The initial values for the RBF centers were obtained from a K-means algorithm.[2] The widths produced by the K-means algorithm were not directly useful. Instead, uniform widths, several times the typical widths from the K-means algorithm, were used. It proved better to make the Gaussians much too broad than too narrow; the exact value used is unimportant as long as it is large enough. The importance of large widths may be understood by the following argument.

Suppose a pattern $\mathbf{x}$ has all its d_j values so large that their activations are essentially zero. Then the contribution from $\mathbf{x}$ to the gradient of the objective function will be essentially zero for all centers, and the pattern will not influence the training at all. An extreme case of this behavior can be seen by taking all widths much too small. Then all RBFs produce zero for all patterns, and all the optimization can do is to adjust the bias terms in the output layer; the process converges rapidly to a poor local minimum.

For the same reason, random output layer weights do not work well for RBF networks. In the work reported here, the initial weights used were "sensible": positive for the weights from the centers to their corresponding output nodes, zero for the remaining weights and for the biases.

2.2. Comments on Training

The number of free parameters in the experiments reported here ranged from 570 to 4250. The objective function has multiple local minima and is sensitive to details of the initial values; a relatively small change in the initial values for the parameters generally results in finding a different local minimum. For each network, ten different sets of initial conditions were used; for RBFs, it proved adequate to use a random $\pm 5\%$ variation on the widths. For MLPs, initial weights were chosen from a uniform random distribution in $(-0.5, +0.5)$.

Two strategies were used in training. The first is to train on successively larger subsets of the 10,000 pattern training set: 156, 312, 625, 1250, 2500, 5000, and finally 10,000 patterns. Training on the smallest sets goes quickly, and each set of parameters is a good initial guess for training on the next larger training set, but there is a possibility of wasting some work. The second strategy is to train only on the full training set.

The first strategy was, on average, faster, but not drastically faster. It has the added advantage of providing extra information, as seen in Figures 1 and 2. Especially for larger networks, the first strategy, on average, provides better training and testing.

3. Computational Experience

The following comments refer only to experience obtained on NIST Special Database 3.

Keeping the centers and widths fixed and learning appropriate weights resulted in poor training and poor testing in networks with only one or a few RBFs per class. Accordingly, centers and widths were also learned. Using initial widths from the K-means algorithm also resulted in poor training and testing; the optimization got stuck in a poor local minimum. Accordingly, all initial widths were then set to the same reasonably large value, with a small random variation.

In general, RBF networks with sigmoidal output layers trained significantly more slowly than RBF networks with linear output layers, and gave somewhat worse training and testing errors.

In general, RBF type 2 networks trained more slowly than RBF type 1 networks and gave slightly poorer training and testing errors. However, RBF type 2 networks with sigmoidal output layers have the useful property that the output weights can be fixed at reasonable values, rather than learned, with little or no worsening of the training and testing error.

RBF networks are self-pruning to some degree. Unimportant connections are effectively pruned away by the training process learning a large width, σ_ij; each large width effectively deletes one connection from an input to one RBF and reduces the number of active parameters by two. More pruning is done with small training sets than with large ones, and more with large networks than with small ones. Some results are shown in Table 1.

The remainder of the paper refers only to RBF type 1 networks with linear output layers.

Compared to MLP networks of a similar size (i.e., similar numbers of free parameters to optimize), RBF networks in general train at about the same rate. Their behavior versus training set size is different, though. Figure 1 gives results for a small (24-16-10, 570 parameters) and a large (48-36-10, 2130 parameters) MLP network. The large network gives quite accurate training results, much better than the small one, but the testing error is not much different for large training set sizes.

For comparison, Figure 2 gives results for a small (24-10-10, 590 parameters) and a medium (24-30-10, 1750 parameters) RBF network. The large network does not train as accurately, but there is much less difference in training and testing accuracy than for the MLP networks. The RBF networks are less likely to overfit the training data.

Figure 3 summarizes many hours of computation for MLP and RBF networks. Training and testing results from ten random starts are shown for each each network.

The RBF results are closer to the diagonal, the diagonal being as good as one could ever expect. The smaller networks are closer to the diagonal than the larger ones; 10,000 training patterns are sufficient to train the small networks as well as they can be trained, but more patterns are needed for the larger networks.

The MLP results are farther from the diagonal. Increasing the network size gives better training error, but no better testing error. Many more training patterns are needed.

Figure 4 shows testing error versus number of free parameters. For a small number of free parameters, MLP networks do better.

Figure 5 shows the testing error versus the percent of the testing patterns rejected as inconclusive. Rejection was based solely on the activation level of the highest output node. There is no apparent advantage either to MLP or RBF networks.

4. Comment

RBF networks are not limited to using equal numbers of centers per class. This can be useful for digits, for example, where the digits '0' and '1' are easy to recognize, but '5' and '8' are more difficult; more RBFs can easily be assigned to the harder classes.

References

[1] M. D. Garris and R. A. Wilkinson. Handwritten segmented characters database. Technical Report Special Database 3, **HWSC**, National Institute of Standards and Technology, February 1992.

[2] J. A. Hartigan. *Clustering Algorithms.* Wiley, New York, 1975.

[3] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[4] M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. Technical Report PB-339, University of Aarhus, November 1990.

[5] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and K. M. Hummels. On the training of radial basis function classifiers. *Neural Networks*, 5:595–603, 1992.

[6] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

Table 1: Networks used and free parameters for each; for RBF networks, active number used in best solution found training on the full training set.

RBF/MLP	Structure	Parameters	Active
R	24-10-10	590	566
R	24-20-10	1170	1014
R	24-30-10	1750	1448
R	24-40-10	2330	1972
R	36-10-10	830	766
R	36-20-10	1650	1364
R	48-10-10	1070	982
R	48-20-10	2130	1650
R	48-30-10	3190	2122
R	48-40-10	4250	2870
M	24-16-10	570	
M	24-24-10	850	
M	24-36-10	1270	
M	48-18-10	1072	
M	48-36-10	2134	

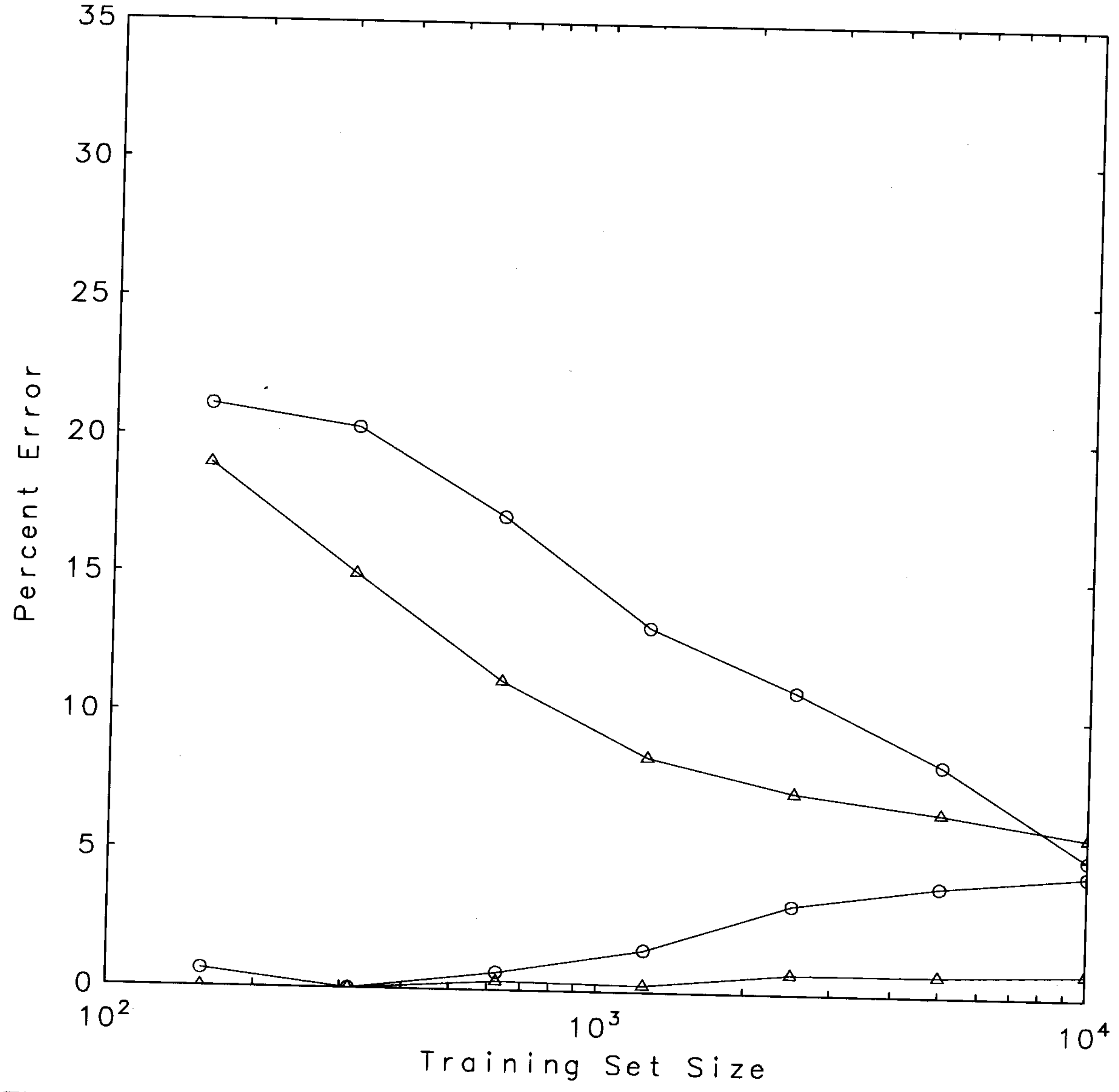

Figure 1: Testing error (top curves) and training error (bottom curves) versus training set size for MLP networks. The results are shown for a representative random start when trained with the full training set: 24-16-10 (○), 48-36-10 (△).

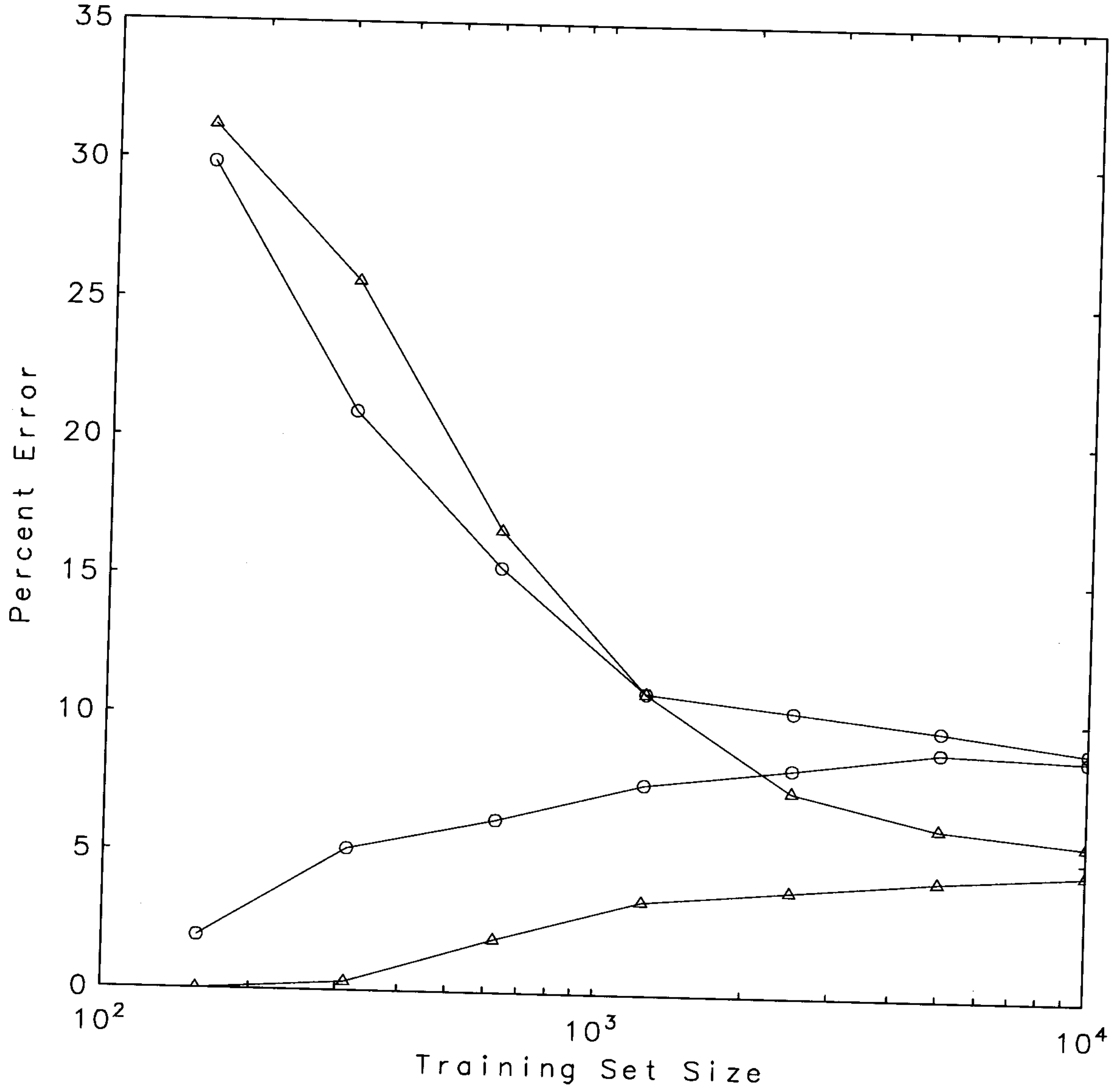

Figure 2: Testing error (top curves) and training error (bottom curves) versus training set size for RBF type 1 networks with a linear output layer. The results are shown for a representative random start when trained with the full training set: 24-10-10 (○), 48-40-10 (△).

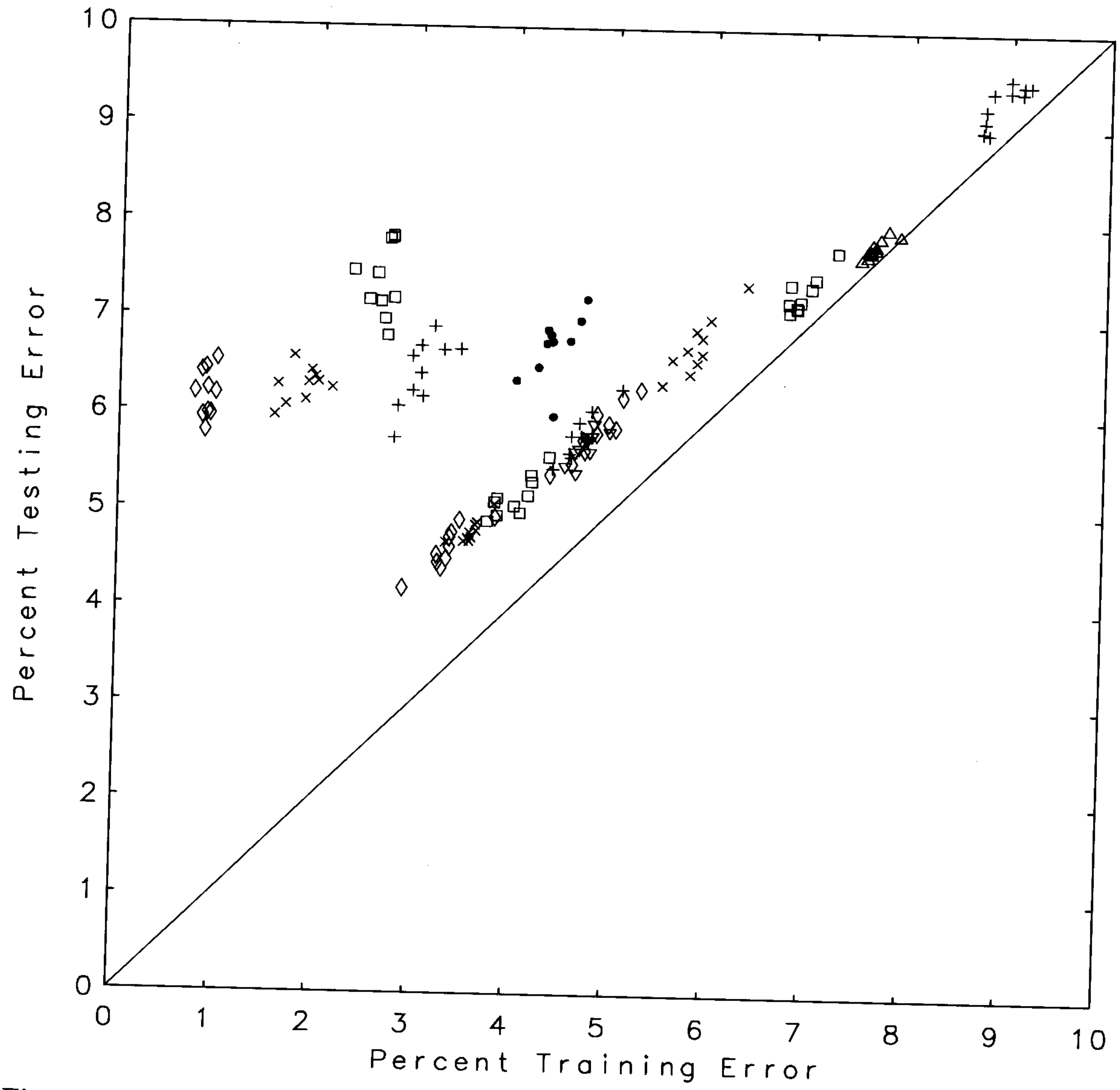

Figure 3: Training error versus testing error for different networks. All training is on the full training set. The symbols near the upper left of the figure are MLP networks: 24-16-10 (•), 24-24-10 (+), 24-36-10 (×), 48-18-10 (□), and 48-36-10 (◇). The symbols nearer the diagonal line are RBF type 1 networks with a linear output layer: 24-10-10 (+), 24-20-10 (×), 24-30-10 (lower group of +), 24-40-10 (lower group of ×), 36-10-10 (△), 36-20-10 (▽), 48-10-10 (□), 48-20-10 (◇), 48-30-10 (lower group of □), and 48-40-10 (lower group of ◇).

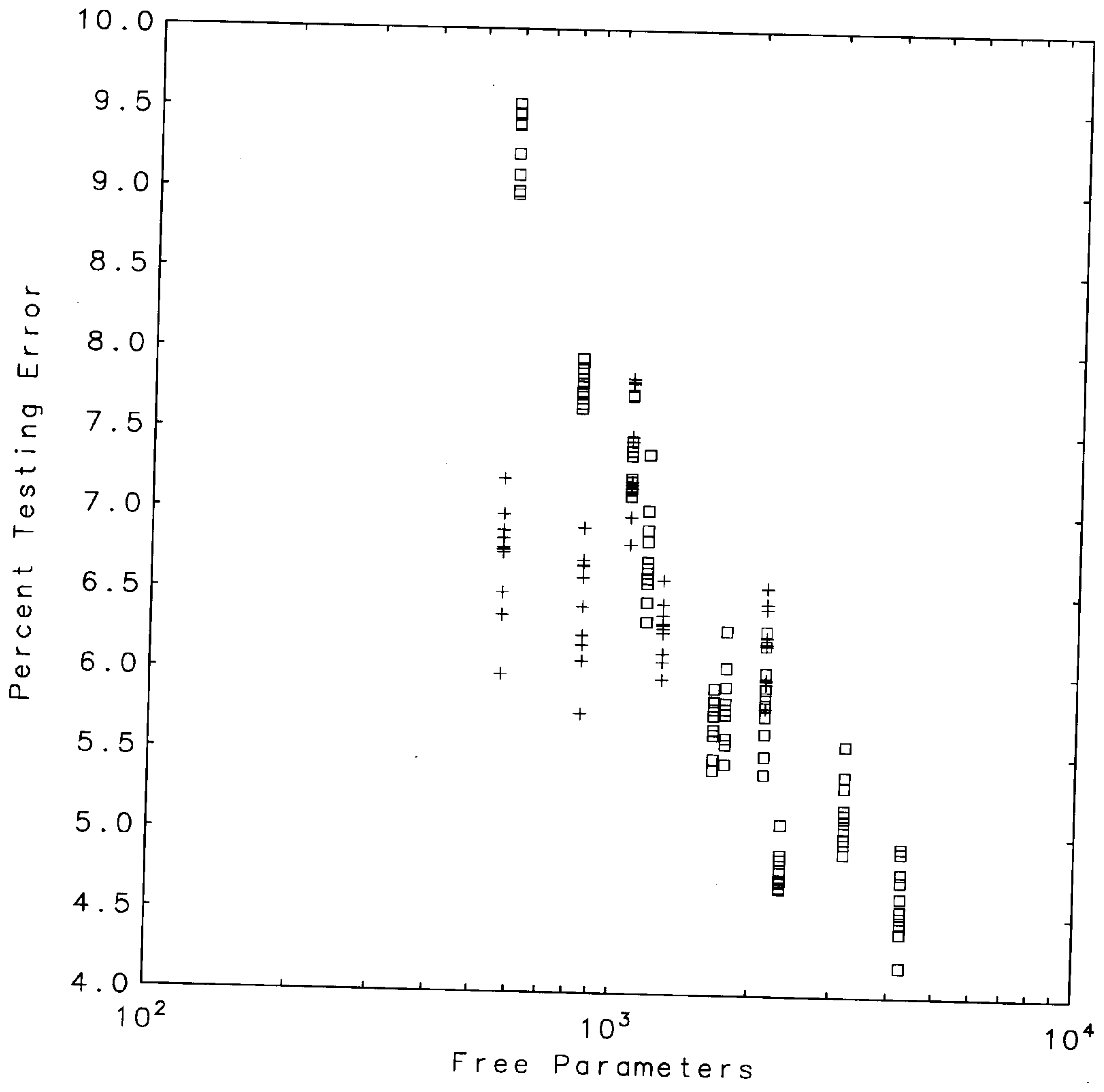

Figure 4: Testing error versus number of free parameters for MLP networks (+) and for RBF networks (□).

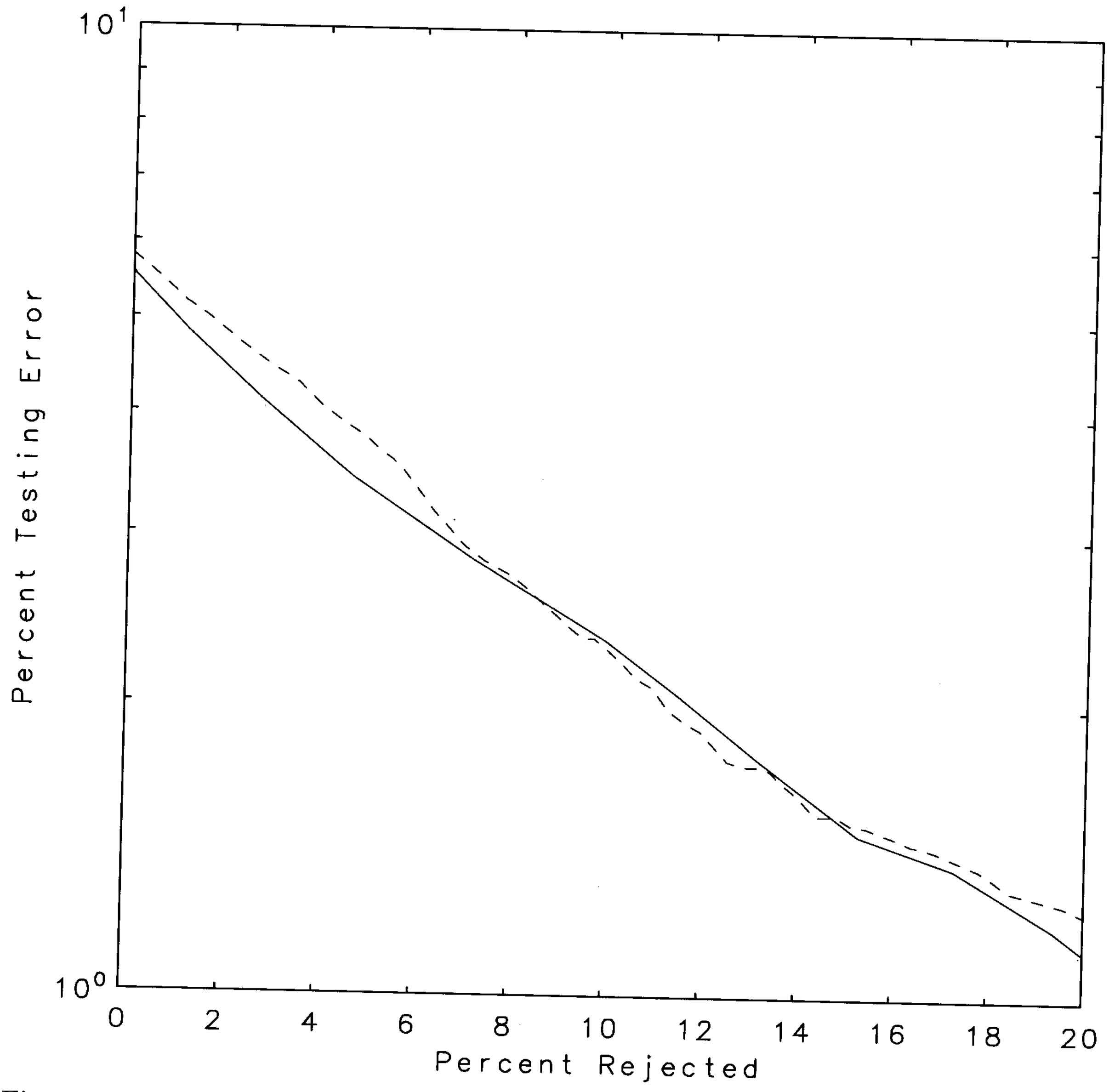

Figure 5: Testing error versus percent rejected for representative networks trained with the full training set. Solid line: RBF type 1, 48-40-10; dashed line: MLP, 48-36-10.