Local Training for Radial Basis Function Networks: Towards Solving the Hidden Unit Problem

Tyler Holcomb and Manfred Morari*

Chemical Engineering 210-41
California Institute of Technology
Pasadena CA 91125

Abstract

This work examines training methods for radial basis function networks (RBFNs). First, the theoretical and practical motivation for RBFNs is reviewed, as are two currently popular training methods. Next a new training method is developed using well known results from functional analysis. This method trains each hidden unit individually, and is thus called the local training method. The structure of the method allows analysis of individual hidden units; moreover a covariance-related quantity is defined that gives insight into how many hidden units to employ. Two examples illustrate the usefulness of the method. Lastly, an *ad hoc* method to further improve RBFN performance is demonstrated.

1 Introduction and Background

For problems in which one would like to use historical knowledge to interpret current data without doing extensive modeling or investigative work, attention is increasingly focusing on methods that "learn from example." One technique that shows considerable promise is the radial basis function network (RBFN). RBFNs have both biological and mathematical motivation, are highly parallel, are capable of recognizing current sensor data that differ in an important and definable way from the historical data, and have been demonstrated to be successful on an interesting array of problems [3] [4] [5]. However, the paradigm suffers from a lack of methods for determining how many processing units are needed for any given problem, and some of the leading training methods have harsh limitations on the nature of the processing units.

RBFNs are feedforward networks, and thus form mappings from an input vector $\hat{x}$ to an output y of the form

$$y = \sum_{i=1}^{h} c_i g_i(\hat{x}) \tag{1}$$

where c_i are network parameters to be determined. The defining aspect of a RBFN is that second layer consists of units of the functional form:

$$g_i(\hat{x}) = g(\|\hat{x} - \hat{t}_i\|_W) \tag{2}$$

where $\|\cdot\|_W$ is a weighted Euclidean norm with weight matrix W, $\hat{t}_i \in \Re^d$ is a fixed vector, $\hat{x} \in \Re^d$ is the input vector, and g is a scalar function that is completely monotonic on $(0, \infty)$ [5, page 17]. Throughout this paper, matrices will be denoted by upper case (W), column vectors by lower case with hat ($\hat{x}$), and scalars by plain lower case. The $\hat{t}_i$ associated with a single hidden unit is called the center of the hidden unit. The weights between the hidden layer and the output layer, c_i, are multiplicative, and the output unit is linear.

While such a network architecture is interesting in its own right, Poggio [5] has shown that this architecture can be derived from regularization techniques. Following Poggio's development, let $S = \{(\hat{x}_i, y_i) \in \Re^d \times \Re | i = 1, ..., n\}$ be a set of data to be approximated by a function y. The regularization approach selects the function $y(\hat{x})$ that minimizes the functional $h[y(\hat{x})]$ where

$$h[y(\hat{x})] = \sum_{i=1}^{n} (y_i - y(\hat{x}_i))^2 + \lambda \|py(\hat{x})\|^2, \tag{3}$$

p is an operator (usually a differential operator), $\|\cdot\|$ is a norm on the function space to whom py belongs, and λ is a positive real number. The operator p represents prior knowledge about the problem and strongly influences the structure of the solution; depending on the choice of p, many well-known techniques result. For one dimensional problems using the L^2 norm and the constraint operator $p = \frac{d^2}{dx^2}$, the solution to (3) produces the cubic spline method. More generally, if p is radially symmetric, the solution to equation (3) is

$$y(x) = \sum_{i=1}^{N} c_i g(\|\hat{x} - \hat{x}_i\|_W^2) \tag{4}$$

where g is a Green's function related to p. Equation (4) is a description of a RBFN where there is one hidden unit per data sample, the center of each unit ($\hat{t}_i$) is one data sample ($\hat{x}_i$), and the output layer weights, c_i, are set by least squares linear regression between the output of the hidden units and the output training data, y_i.

Radially symmetric operators of special interest are those that act on all derivatives to insure a very smooth solution, namely

$$\|py\|^2 = \int_{\Re^d} \sum_{m=0}^{\infty} a_m (\nabla^m y(\hat{x}))^2 d\hat{x}. \tag{5}$$

where the a_m are constants, ∇ is the gradient operator, ∇^2 is the Laplacian operator, *etc.* Various values for a_m give rise to bell shaped radial basis functions and make W the identity matrix (I). In particular, setting $a_m = \sigma^{2m}/m!2^m$ yields radial basis functions that are Gaussians with variance σ^2.

Thus, the RBFN architecture has a firm theoretical motivation. Even more encouraging, however, is that the RBFN architecture was arrived at by other researchers based primarily on practical experience. Particular interest has been generated by the success of an algorithm developed by Moody and Darken. Moody and Darken's algorithm [4] uses hidden units of the form

$$g_i(\hat{x}) = \exp\left(\frac{\|\hat{x} - \hat{t}_i\|^2}{\sigma_i^2}\right), \tag{6}$$

where σ_i is analogous to the standard deviation for a Gaussian and is referred to radius of the hidden unit. Equation (6) is a special case of equation (2) wherein $W = \sigma_i^{-1} I$. One should note that σ_i varies with each unit; in the development by Poggio, W was the same for all hidden units. This change is necessary because the Poggio development required one hidden unit per training sample, while Moody and Darken's algorithm explicitly treats problems with more training samples than hidden units. The training procedure is divided into an unsupervised learning phase and a supervised learning phase. Initially, one chooses the number of hidden units, h. Next, a k-means clustering algorithm performs unsupervised clustering analysis to set the centers, $\hat{t}_i$, of each hidden unit. The k-means clustering operates by initially setting the centers of the h hidden units to $\hat{t}_i = \hat{x}_i$ for $i = 1, \ldots, h$. Next, each input datum is assigned to the nearest center (in the Euclidean sense). When all of the input data have been so assigned, each center is reset to the mean of all the data currently assigned to that center. Using the new values for the centers, the process is repeated until none of the centers move. With the centers (t_i's) determined, the σ_i of each hidden unit is computed by $\sigma_i = \sqrt{d_1 d_2}$ where d_1 and d_2 are the Euclidean distances from the ith center to the two nearest centers. This rule is *ad hoc*; other heuristics could be used. Lastly, the output layer weights c_i are determined by performing standard linear least squares regression between the outputs of the hidden layer and the output values for the training samples (y_i).

Moody and Darken emphasized the speed advantages of their method relative to gradient descent and other global optimization approaches. Leonard and Kramer [3] compared RBFNs using Moody and Darken's algorithm to backpropagation networks and k-nearest neighbor classifiers and determined that RBFNs possessed other advantages as well. These advantages were creating decision surfaces in a more sensible manner, detecting novel inputs that were far away (in a Euclidean sense) from the data used for training, and performing better on the studied examples.

2 Local Training Method

The successes discussed above encourage further investigation. The result of Poggio only applies when there is one hidden unit per data sample; however one would normally like to use fewer units than this. Poggio suggests choosing a number of hidden units fewer than the number of samples and setting the RBFN parameters via gradient descent in a manner analogous to backpropagation. As Poggio notes, the rigorous result he derived no longer

*Author to whom correspondence should be addressed: phone (818)356-4186, fax (818)568-8743, e-mail mm@imc.caltech.edu

holds. Moreover, this approach forces the practitioner to choose the number of hidden units; no good guidelines exist for doing this other than trial and error. Interestingly, Moody and Darken considered such an approach and rejected it in favor of their algorithm because their method has less demanding computational requirements.

While faster than global optimization methods, Moody and Darken's method has significant limitations. First, the number of hidden units must also be determined by trial and error. Additionally, the Moody and Darken method only applies to RBFNs using (6). Geometrically, this means that the local receptive field (the subset of inputs to which the hidden unit responds) for each hidden unit must be spherical, regardless of how appropriate this is for the task at hand. Moreover, for classification tasks, the unsupervised clustering step can be misleading. The clustering occurs without regard to which class a given input belongs. Thus, inputs that are close together but of different classes are often included in the same cluster (the same receptive field for a hidden unit). This will tend to make the classification task for the output layer more difficult than if the each hidden unit had a distinct preference for a particular class. This will be made clearer in an example below.

The above methods, hereafter referred to as global training methods, focus on doing global training to produce "locally tuned" hidden units. While effective, the entire approach seems to cry out for "local tuning" of the "locally tuned" units. By drawing a direct analogy with functional approximation, such "local tuning" is in fact direct and simple.

Consider the standard L^2 approximation problem of determining the c_i in

$$y(x) = \sum_{i=1}^{\infty} c_i g_i(x), \quad \text{where } y(x) \in L^2[-a,a], \tag{7}$$

$$g_i(x) \in L^2[-a,a], \quad < g_i(x), g_j(x) > = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}, \tag{8}$$

and

$$< g_i(x), g_j(x) > = \int_{-a}^{a} g_i(x) g_j(x) dx. \tag{9}$$

The $g_i(x)$ are orthonormal basis functions, such as Legendre polynomials or sinusoids. Drawing from well known analysis results, the answer is $c_i = < y(x), g_i(x) >$. For practical use, one normally desires an expression with a finite number of terms. Naturally, one would like to truncate the expansion in a manner that minimizes the error. Defining the square of the error resulting from using of the first h terms of the expansion in equation (7) as e_h^2, one finds

$$e_h^2 = < y(x) - \sum_{i=1}^{h} c_i g_i(x), y(x) - \sum_{i=1}^{h} c_i g_i(x) > = \sum_{i=h+1}^{\infty} c_i^2. \tag{10}$$

Clearly one should reorder the bases so that the c_i^2 associated with each basis $g_i(x)$ decreases monotonically. In the case of a Fourier expansion, this corresponds to using the frequencies (bases) with the greatest "energy". More descriptive for the general case, the quantity c_i^2 will be called a covariance because it measures the covariance of the basis function with the output. Labeling the reordered bases $\tilde{g}_i(x)$, one algorithm that performs the desired reordering is:

```
$y_1(x)$     $= y(x)$
FOR $i$      $= 1$ to $\infty$
             find the $j$ that maximizes $< y_i(x), g_j(x) >^2$        (11)
             $\tilde{g}_i(x)$       $= g_j(x)$
             $c_i$           $= < y(x), \tilde{g}_i(x) >$
             $y_{i+1}(x)$    $= y_i(x) - c_i \tilde{g}_i(x)$
NEXT $i$.
```

Next consider the slightly different case where $j \in \Re$ parametrizes the orthonormal basis functions. Even though j is now continuous, one would still use the ordering technique above to choose the first h bases so as to minimize the error. Thinking of each basis as a hidden unit in a RBFN, this points out how to train the hidden units one at a time, and thus "locally tune" each individual unit. Instead of a single continuous parameter j, the parameters of each hidden unit would be optimized in turn to maximize $< y_i(x), g(x; hidden\_unit\_parameters) >^2$

Unfortunately, radial basis functions do not satisfy the orthogonality condition in equation (8). However, with modification the above algorithm can still be applied. Since the radial basis functions are no longer orthogonal, the c_i can no longer be determined independently; after the determination of each basis function, all of the c_i must be recomputed. This is not a cause for concern, however, since the c_i are determined via computationally inexpensive linear least squares regression, just as in the Moody and Darken algorithm. The loss of orthogonality has a more serious implication, however: the derivation for minimizing the error, equation (10), breaks down. Thus, one can not be certain that using the algorithm will be optimal. One can, however, put forth a heuristic argument to support using the approach.

The justification developed in equation (10) is possible because the basis functions are orthogonal. That is, the basis functions are non-interacting in the well defined sense that their inner product is zero, and the contribution that each basis function makes to the total approximation is not affected by any of the other basis functions. Since radial basis functions are positive everywhere, the hidden units are clearly interacting. However, a hallmark of radial basis functions is that they are "locally tuned;" that is, the outputs are small everywhere except near the center of the basis function (hidden unit). If the centers of the hidden units are far apart relative to their receptive fields, then the inner product between radial basis functions will be small and in this sense the radial basis functions will be approximately orthogonal. This argument also points out a potential problem. If the centers of the hidden units are relatively close, then the radial basis functions will be interacting (have overlapping receptive fields), and the optimality argument will break down. While this in and of itself does not mean the method will fail if the centers are close, one should be aware that the local training approach has this potential difficulty.

Expression (2) described the general form for the radial basis function. For the remainder of the paper, a specific form will be used for clarity and concreteness. Some of the developments below apply only to this particular function, but similar developments can be followed for other radial basis functions. The radial basis function used for the remainder of the paper is the multivariable Gaussian,

$$g(\hat{x}; W_i, \hat{t}_i) = \frac{|W_i|}{\pi^{\frac{d}{2}}} \exp\left(-\frac{1}{2}(\hat{x} - \hat{t}_i)^T W_i^2 (\hat{x} - \hat{t}_i)\right) \tag{12}$$

where d is the dimensionality of the input vector $\hat{x}$, $W_i \in \Re^{d \times d}$ is symmetric and non-singular, $|\cdot|$ is the determinant, and $\hat{t}_i$ is the center for the hidden unit. The covariance matrix for the multivariable Gaussian is W_i^{-2}. Equations (2) and (12) look superficially different, but (12) is in fact a special case of (2). First, $\hat{t}_i$ is identical in both expressions. Moreover, so are the W_i. In (2), W_i had the interpretation of the weight for the weighted Euclidean norm. In (12) W_i has the additional constraint of being symmetric and non-singular, but otherwise performs the identical mathematical function. However, in the context of the multivariable Gaussian, W_i has another meaning. For the one dimensional case, W_i is simply the inverse of the standard deviation; for higher dimensions, W_i scales and rotates the hidden unit's receptive field. Thus, the local training method provides for a much richer class of basis functions than the method of Moody and Darken, allowing each of the hidden units to be more "locally tuned."

The story is not yet complete, however. If one uses the local training algorithm with the multivariable Gaussian radial basis function, the method will fail to generalize. This is because as $W_i \to \infty$ and $\hat{t}_i \to$ any $\hat{x}_i$, $< y_i(\hat{x}), g(\hat{x}; W_i, \hat{t}_i) > \to \infty$. Thus, the local training method produces one hidden unit per training sample, makes each hidden unit (basis function) a unit impulse centered on one training sample, orthogonalizes all of the radial basis functions in the sense of (8), and reproduces the training set with zero error. The resulting RBFN in effect memorizes the training data. However, the entire point of the endeavor is to produce a procedure that processes novel inputs in a manner consistent with the historical data; one would like the RBFN to generalize. Put another way, one would like to force the hidden units to "spread out" their receptive fields.

Thus, generalization requires an artificial method to force the receptive field to "spread out". One would like to keep matters simple and control the "degree of generalization" for each unit in a clear and simple manner. This can be done by adding a penalty to the objective function (11) that meets the following conditions:

1. The penalty function use only one penalty parameter. This user-determined parameter will be called the generalization parameter, p.

2. As $p \to \infty$, $W_i \to 0$. That is, as the generalization parameter is made large, the optimization of (11) plus the penalty function drives the hidden unit's receptive field to cover the entire training set, making the hidden unit "over-generalize."

3. When $p = 1$, the optimization of (11) plus the penalty function drives the hidden units to the unit impulse and the "memorizing" behavior is recovered.

4. The receptive field varies smoothly as p is varied over $(1, \infty)$.

A penalty function which meets these goals is $\pi^{\frac{-d}{2}}\left(|W_i|^{1-p} - |W_i|^p\right)$. Consistent with the first condition, only one penalty parameter, p, is introduced. The first term of the penalty function, $|W_i|^{1-p}$, addresses the second condition. As p becomes large, this term dominates the objective function and encourages W_i to be small. As p nears unity, the first term becomes unimportant and the second term discourages W from being small. The penalty function does not rigorously meet the third condition; when $p = 1$, the optimization of (11) plus the penalty function does not drive the hidden unit to the unit impulse. However, the penalty function does approximate the desired behavior. In numerical experiments, as $p \to \infty$, values of $|W_i|$ increase monotonically until numerical overflows result. The penalty function is continuous for $p \in (1, \infty)$. Moreover, the quantity $|W_i|$ determines directly the area within contours of constant probability density for the multivariable Gaussian. [2, page 24] Thus, the penalty function acts directly on the measure of the area of the receptive field and meets the fourth condition.

The full algorithm below will treat multivariable problems. The only change needed is the addition of more linear output units; the hidden units are unaffected. Before presenting the local training algorithm, a few terms will be defined. The number of hidden units, h; the number of training samples, n; the dimension of the inputs, d; and the dimension the outputs, l; define the dimensionality of all quantities used.

As described above, the parameters for the ith hidden unit are its center($\hat{t}_i \in \Re^d$) and its symmetric non-singular weight matrix ($W_i \in \Re^{d \times d}$). The response of the ith hidden unit to the input training data $X \in \Re^{n \times d}$ is $\hat{f}_i = \hat{f}_i(X; \hat{t}_i, W_i) \in \Re^n$, and the matrix of all of the hidden unit responses to all of the training data is $F \in \Re^{n \times h}$, where $F = [\hat{f}_1|\hat{f}_2|\dots|\hat{f}_h]$. The matrix of output weights from the hidden units to the linear output units is $C \in \Re^{h \times l}$. Lastly, the output training data is $Y \in \Re^{n \times l}$. Since the radial basis functions are in fact not orthogonal, Y must be modified after each hidden unit is trained. Thus, $Y^{(i)}$ is used to train the ith hidden unit. Moreover, the training data for the ith hidden unit and the jth output is $\hat{y}_j^{(i)} \in \Re^n$. That is $Y^{(i)} = [\hat{y}_1^{(i)}|\hat{y}_2^{(i)}|\dots|\hat{y}_l^{(i)}]$. The local training algorithm is then:

```
$Y^{(1)}$    $= Y$
$i$          $= 0,$        $h$   $= 0$
LABEL: train_next_hidden_unit
$i$          $= i + 1,$    $h$   $= h + 1$
find the $W_i, \hat{t}_i$ and $j$ that maximize $z_i$, where
             $z_i = (\hat{f}_i^T\ \hat{y}_j^{(i)})^2 + \pi^{\frac{-d}{2}}\left(|W_i|^{1-p} - |W_i|^p\right)$        (13)
$C$          $= (F^T F)^{-1} F^T Y$
$Y^{(i+1)}$  $= Y - FC$
IF another hidden unit desired GOTO train_next_hidden_unit
```

Several points are worthy of note. Since this algorithm deals with a finite set of training data, the functional inner product, equation (9), has been replaced with Euclidean inner product (dot product) in expression (13). Also, the quantity h is incremented but never used directly in the algorithm. This extra step is included to emphasize that many of the matrices used in the algorithm change dimension as more hidden units are used. One should note that the value of objective function, z_i, is no longer a covariance because of the addition of the penalty function. After each unit is trained, the network output is subtracted from the original output training data. Because of this, additional units with receptive fields in the same region as prior hidden units are not be favored. This helps the reduce the effect of the loss of orthogonality. Most important, however, is that the training of a given hidden unit is well defined, but the criterion for deciding whether or not to train another hidden is not.

This lack of a clear termination criterion is not unique to the local training method. With global training methods, one first chooses a parameter (the number of hidden units) that bears no clear connection to the performance and then produces a "solution." If this solution is "unsatisfactory," one then chooses a different number of hidden units and tries again. Thus even though the training procedure for a given number of hidden units is well defined, the overall procedure by which one builds a desired approximator is very much a black-box, "hit-or-miss" approach. Thus both global training methods and the local training method lack a clear termination criterion.

The local method does have an advantage over the global training methods for determining the number of hidden units. Using global methods, one must rely solely on training and testing error for an entire network to compare different networks. With the local training method, one has another important quantity that allows one to judge each hidden unit individually: the value of the objective function (z_i) for each unit. When choosing among units trained with the same value of p, the selection is very simple: choose the hidden unit with the largest z_i. Moreover, z_i can provide insight into when one has "enough" hidden units. As mentioned above, the local training method tends to keep the hidden units' receptive fields distinct. When the existing receptive fields cover the input data "well," the z_i's for any further units tend to be small. Thus, if z_i decreases significantly when training additional hidden units, one should stop adding hidden units. This loose heuristic is not a hard and fast rule; as shown below, however, the objective function value is useful for determining if additional hidden units are beneficial.

There are other benefits to the local training method. Unlike most of the global training methods, the role of each hidden unit is easier to assess. In the case of multiple outputs (*e.g.* classification problems), each hidden unit is associated with one particular output (class). While the linear output layer, C, uses all of the hidden units to determine the output of each linear output unit, the local training method associates each hidden unit with a class. The output (class) associated with each hidden unit is represented by the j that achieves the maximum in expression (13). This aspect of the local training method has two advantages. First, the difficulty of inputs of different classes being in the same receptive field that can bedevil Moody and Darken's method is dealt with directly. Secondly, knowing which class each hidden unit is trying to describe yields information about both the nature of the classes and the utility of adding more hidden units. Additionally, one can "locally tune" individual hidden units on an *ad hoc* basis by adjusting the generalization parameter p. Approaches for evaluating the class association of each hidden unit and adjusting the generalization parameter are wholly empirical at the moment. However, these techniques are useful, as will be shown below.

One issue remains. In much neural research, great attention is focused on the "learning rule," that is the optimization technique employed. Following the development of Poggio [6, page 33], one could employ gradient descent. However, this investigation does not focus on the various optimization techniques that might be used . Instead, off-the-shelf optimization software is used. This saves the work of developing such software and allows the use of more advanced algorithms.

3 Examples

The preceding sections presented the motivation for local training and developed the local training method. In this section two examples are developed. RBFNs are approximators by design; however, for the purpose of clear illustration, both examples are two dimensional classification problems. For these classifications, there is one output per class. For each class, if an input is a member of that class, the corresponding training output is 1; otherwise the training output is 0. Thus, for each training input sample, the training output sample (a vector) has precisely a single 1 and the remaining training outputs are zero. The RBFN performs classification by assigning each input to the class corresponding to the linear output unit with the largest value.

For the purpose of graphical illustration, the receptive field of the ith hidden unit is the set of input vectors described by $\{\hat{x} \in \Re^d | (\hat{x} - \hat{t}_i)^T W_i^2 (\hat{x} - \hat{t}_i) \leq 1\}$. That is, points within unit distance (in the Mahalanbolis metric) of the center of the hidden unit constitute the receptive field. In the one dimensional case, this would be the region within one standard deviation of the center.

All of the networks were evaluated by using testing data drawn from the same stochastic distributions as the training data. The error rate was determined by computing the percentage of samples of a given class that were assigned to some other class by that classifier. Thus, the statement "the error for class B was 10%" means that for the testing data, 10% of the samples from class B were determined by the classifier to belong to some other class. The overall error rate is the percentage of all input samples that were misclassified. Lastly, GAMS/MINOS [1] was used to perform the maximization of expression (13) for all of the examples.

3.1 Binary example

The first example involves determining the state of a binary variable. In particularly, consider a process that, when operating correctly, has a normally distributed output

$$\hat{x} = \begin{bmatrix} s_1 + s_2 \\ s_1 - s_2 \end{bmatrix} \tag{14}$$

Method	Hidden units	Class A error %	Class B error %	Overall error %
Moody	2	46.7	44.2	45.5
Moody	3	7.9	26.7	17.4
Local, $p = 1.7$	2	3.1	4.4	3.7
Theoretical minimum		2.2	2.2	2.2

Table 1: Performance of different methods for binary example

hidden unit	z_i	class
1	107	A
2	98	B
3	57	A

Table 2: Local training results for binary example

where $s_1 \sim \mathcal{N}(0.1, 0.05)$ and $s_2 \sim \mathcal{N}(0.1, 0.2)$. Moreover, this process is subject to an upset condition that does not affect variance of the outputs but translates the mean of the $\hat{x}$ from $[0.1 \quad 0.1]^T$ to $[-0.1 \quad -0.1]^T$. The problem is to train a RBFN to determine if the process is operating correctly (deemed Class A) or if the process is in the upset condition (deemed Class B).

For training, 50 samples were drawn from each class, while 1,500 samples from each class were used for testing. Networks with two and three hidden units were trained with the Moody and Darken method. The receptive fields for these networks are shown in Figure 1, while the performance is described in Table 1.

Next, the local training method was used, wherein one adds one hidden unit at time. All hidden units were trained using $p = 1.7$. The training summary is given in Table 2. Notice that no errors are given; the training was done using solely z_i. The objective function value for the third unit (z_3) is dramatically lower than for the previous two units. Thus, two hidden units were deemed to be the "correct" number of hidden units for this problem. The receptive fields for the locally trained RBFN are shown in Figure 2. Table 1 gives the performance of this RBFN and the theoretical minimum error rate. The theoretical minimum is based on the minimum error rate decision boundaries [2]; these can be computed because the distributions are known.

3.2 Continuous example

The second example is drawn from the work of Leonard and Kramer [3]. That work describes a fault diagnosis problem wherein the two continuous variables s_1 and s_2 determine three fault conditions. These variables cannot be measured directly, but measurements consisting of linear combinations of these variables corrupted by Gaussian noise ($\hat{x}$) do exist. The three classes (fault conditions) are defined as:

$$\begin{array}{rcl} |s_1| \text{ and } |s_2| < 0.05 & \Rightarrow & \text{Class A}, \\ |s_1| \geq 0.05 & \Rightarrow & \text{Class B}, \\ |s_2| \geq 0.05 & \Rightarrow & \text{Class C}. \end{array}$$

The problem is constrained so that $|s_1|$ and $|s_2|$ never exceed 0.05 simultaneously. Moreover, there is an equal *a priori* probability for each class, namely one-third. To meet the above conditions, the data were generated from two different distributions. For both distributions,

$$s \sim \mathcal{N}(0, 0.1163), \quad v_1 \sim \mathcal{N}(0, 0.015), \text{ and } v_2 \sim \mathcal{N}(0, 0.015). \tag{15}$$

The first half of the data (45 samples for training, 1,500 for testing) were produced by

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} s + v_1 \\ s + v_2 \end{bmatrix}. \tag{16}$$

A given sample $\hat{x} = [x_1 \quad x_2]^T$ was assigned to class A if $|s| < 0.05$, and assigned to class B otherwise. The second half of the data was generated from

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} s + v_1 \\ -s + v_2 \end{bmatrix}. \tag{17}$$

Method	Hidden units	Class A error %	Class B error %	Class C error %	Overall error %
Moody	5	43.4	0.8	8.6	17.7
Moody	8	33.9	30.1	8.1	24.1
Local, $p = 1.3$	5	23.2	0.8	1.7	9.1
Local, *ad hoc*	5	15.1	1.7	3.2	6.8
Theoretical minimum		7.4	4.0	4.0	5.1

Table 3: Performance of different methods for continuous example

Hidden Unit	z_i	Class
1	1566	A
2	665	B
3	459	C
4	220	B
5	133	C
6	3	A

Table 4: Local training results for continuous example

A given sample $\hat{x} = [x_1 \quad x_2]^T$ was assigned to class A if $|s| < 0.05$, and assigned to class C otherwise. One should take careful note that the structure for this problem is identical to that of Leonard and Kramer, but that the distributions for s_1 and s_2 are different.

RBFNS with 5 and 8 hidden units were trained with the Moody and Darken algorithm using the ninety sample training set. The receptive fields for these networks are shown in Figures 3 and 4, while the performance is described in Table 3.

Next, a 5 hidden unit RBFN was trained using the local training method with $p = 1.3$ for all hidden units. The training summary is given in Table 4. Due to the dramatic drop-off in z_i from hidden unit 5 to hidden unit 6, 5 hidden units were deemed to be the "correct" number for this problem. The performance for the 5 hidden unit locally trained network, whose hidden unit receptive fields are shown in Figure 5, is shown in Table 3. Once again, the underlying distributions are known so the theoretical minimum error rates can be computed; these are shown in Table 3. The values given as the theoretical minimum for a given class are those that minimize the overall error. Clearly, one could assign all inputs to one class, thereby insuring the error for that class was 0.

3.3 Computational performance

The Moody and Darken method and local training method were performed using markedly different software running on different computers. Thus, a direct comparison of the computational effort required for the two approaches is not possible. However, by comparing benchmarks, it appears that training a full network via Moody and Darken requires similar, though probably less, CPU time than training one hidden unit with the local training method. In the binary example, both the k-means clustering step and locally training one hidden unit took on the order of ten CPU seconds on a μ -vax II. These tasks required on the order of a half of a minute of μ -vax II CPU for the continuous example.

4 Discussion

The two examples presented above clearly illustrated some advantages of the local training algorithm. The first example was developed because it exploited a known deficiency in k-means clustering: if the clusters are not spherical, k-means clustering can require many units to distinguish the clusters. On the other hand, the local training method is suited to rotated elliptical clusters, of which spherical clusters are a subset. The advantage that the local training method gains over Moody and Darken method is clear from a comparison of Figure 1 to Figure 2. The receptive fields of the locally trained units are more reflective of the shape of the clusters, and this leads directly to better performance. In the case of the same number of hidden units (two), the locally trained network had less than 1/10th the error rate. This comparison can be viewed as unfair, since the locally trained method had access to more adjustable parameters. However, the three hidden unit

RBFN trained via Moody and Darken had ten adjustable parameters, the same as the two hidden unit locally trained RBFN. Still, the Moody and Darken method had more than four times the error rate (17.4% vs. 3.7%).

The first example also illuminated the manner in which the z_i help determine the number of hidden units. The first two hidden units had similar objective function values. Since, as seen in Figure 2, these two receptive fields covered the data well, one would intuitively expect that a third hidden unit would have little effect. Indeed, z_3 is roughly half of both z_1 and z_2. In this problem, the error information just serves to corroborate the conclusion driven by the z_i's that two hidden units are adequate.

Similar results were observed with the second problem. Comparison of Figures 3 and 4 to Figure 5 shown once again that the local training method does a better job of "locally tuning" the hidden units. When the two methods use the same number of hidden units, both methods assign hidden units to data clusters in an intuitively reasonable manner; the center (class A) cluster, and each cluster of error conditions (classes B and C) is assigned one hidden unit. However, the local training method produces receptive fields that much more accurately reflect the data. Since one of the advantages of RBFNs is that they can detect inputs far away from the training data, this better "tuning" is important. For the locally trained network, very little of the receptive field lies outside the convex hull of the training data. This is not so with the Moody and Darken trained network. Moreover, because of the objective function values, one has indications that such coverage has occurred. With the Moody and Darken method, no such intuition is possible if pictures are not available. Since many problems have more than two inputs, the difference is significant.

Once again, the above comparison is somewhat unfair because the locally trained network had 30 adjustable parameters, while the Moody and Darken network had only 20. However, the locally trained network maintains the advantage against the 8 hidden unit network, which has 32 adjustable parameters. Surprisingly, the error rate actually increased for this network. Moreover, the receptive fields of the the hidden units lost much of their physical meaning, as shown in Figure 4.

Choosing values for p was not discussed above. While this clearly influences the outcome of training, both examples performed satisfactorily using the first values of p attempted. In the binary examples, $p = 1.5$ was also attempted, and the overall error increased to 6.9%. This work focused on determining the viability of the local training method; future work clearly needs to be done gain a better understanding of how varying values of p affects the method.

Also, the local training method's need for CPU time relative to other methods is not well understood. Solving a series of small, independent problems, as the local method does, should have advantages over performing a single global optimization of the entire network. This advantage has not been clearly established. Likewise, the advantages relative to Moody and Darken are also poorly understood. A hidden unit with d inputs and a spherical receptive field (equation (6)) has $d + 1$ parameters, while a multivariable Gaussian hidden unit (equation (12)) has $(d^2 + d)/2$ parameters. Thus, as the number of inputs is increased, this scaling would seem to favor the RBFN using spherical receptive fields. However the binary example demonstrated that spherical receptive fields may be inappropriate for some problems and may lead to a networks that require more hidden units than the local training method would require to achieve a similar level of performance. This open issue of relative advantages in training times of the various methods is the subject of ongoing study.

Using an *ad hoc* approach, the locally trained RBFN can be improved still further. Most of the error occurred as the result of class A inputs being placed in a different category. There are two likely causes for this: the distribution for class A is multi-modal, and more hidden units are needed to reflect this, or the current hidden unit is not "well tuned." Evaluation of the z_i's indicates that an additional hidden unit would probably not help matters much; the objective function value for the next hidden unit trained for class A dropped by more than two orders of magnitude ($z_1 = 1566$ and $z_6 = 3$). Thus, retraining the first hidden unit with different values of p seems worth trying. Before attempting retraining, one needs to consider how such tweaking affects the overall training method.

Strictly speaking, when one retrains a given hidden unit, all hidden units trained subsequent to the retrained unit should also be retrained. However, if there is little overlap of receptive fields for a given class (the hidden units remain "approximately orthogonal"), retraining one hidden unit will have only small effect on the other hidden units, and only the output layer, C, need be recomputed. Using the inner product, this degree of interaction can be evaluated quantitatively. Since Figure 5 clearly shows that receptive fields for each class have little overlap, the degree of interaction between the hidden units was deemed small by fiat and the other hidden units were not retrained.

The hidden unit associated with class A was retrained with $p = 1.5$ One cannot compare the z_i for units trained with different values of p, since, as stated above, the value z_i is affected by p. The receptive field for the retrained hidden unit, along with the receptive fields of the other hidden units (trained with $p = 1.3$) is shown in Figure 6. This modified network's performance is shown in Table 3. The error rate for Class A was decreased by almost a third (from 23.2% to 15.1%), while the error rates for Classes A and B rose slightly. The overall error rate was reduced by more than a quarter (from 9.1% to 6.8%), and the optimal error rate of 5.1% was approached much more closely. A search to determine the optimal value of p was not attempted; the only value of p attempted was 1.5.

5 Conclusions

This work has examined training methods for radial basis function networks (RBFNs). Past RBFN training approaches have generally fixed the number of hidden units, set the network parameters via clustering and/or global optimization methods, and used the training and testing error to evaluate the entire network. This work further developed the idea that the success of RBFNs is due in part to the "local tuning" of the hidden units. Using a direct development from functional analysis, a local training method was developed whereby each hidden unit is trained individually. This approach allows the use of richer classes of radial basis functions than can be tolerated by the Moody and Darken algorithm.

The key advantage to the local training method is that more information is available about each hidden unit. This makes possible the determination of the number and nature of the hidden units on more than just error data. In particular, the values of the objective function make possible direct comparison of various hidden units. Additionally, each hidden unit has a generalization parameter associated with it that determines the broadness of the hidden unit's receptive field. Two examples illustrated the local training method and the manner in which hidden units can be evaluated.

Significant work remains to be done. Foremost, the penalty function used to force generalization needs more investigation. The current penalty function does not rigorously recover the memorizing behavior as $p \to 1$. Also, the relationship between the area of the receptive field and p is not well understood. As future investigation reveals more about what penalty functions are appropriate for the local training method, the current penalty function will quite likely be superceded.

As discussed above, the local training method rests on the assumption that the receptive fields are not overlapping. For classification problems, where one is typically trying to isolate various groups of data, this can be a good assumption. For approximation problems, where one is typically trying a build a "smooth" surface, this assumption may break down. Thus, the local training method needs to be tested on approximation problems.

The local training method still has many important unanswered questions. However, this method represents an interesting approach for designing the hidden units directly and for gaining insight into how may hidden units are needed. Moreover, the current method has demonstrated clear advantages over the Moody and Darken method on several examples.

Acknowledgments *Tyler Holcomb is a recipient of a National Science Foundation Graduate Fellowship. This research was supported by the Caltech Consortium in Chemistry and Chemical Engineering. Founding members of the Consortium are E. I. du Pont de Nemours and Company, inc., Eastman Kodak Company, Minnesota Mining and Manufacturing Company, and Shell Oil Company Foundation.*

References

[1] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide.* 1988.

[2] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* 1973.

[3] J. Leonard and M. Kramer. Classifying process behavior with neural newtorks: Strategies for improved training and generalization. pages 2478–2483. 1990 American Control Conference, 1990.

[4] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[5] T.Poggio and F. Girosi. A theory of networks for approximation and learning. Technical Report A.I. Memo No. 1140, M.I.T., 1989.

[6] T.Poggio and F. Girosi. Extensions of a theory of networks for approximation and learning: dimensionality reduction and clustering. Technical Report A.I. Memo No. 1167, M.I.T., 1990.

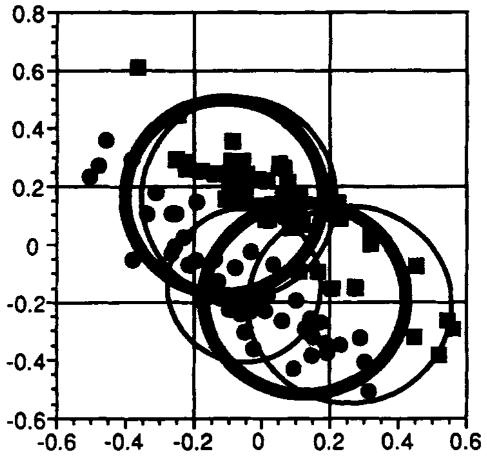

Figure 1: Binary example. Receptive fields for the Moody and Darken trained 2 hidden unit RBFN (thick lines) and 3 hidden unit RBFN (thin lines).

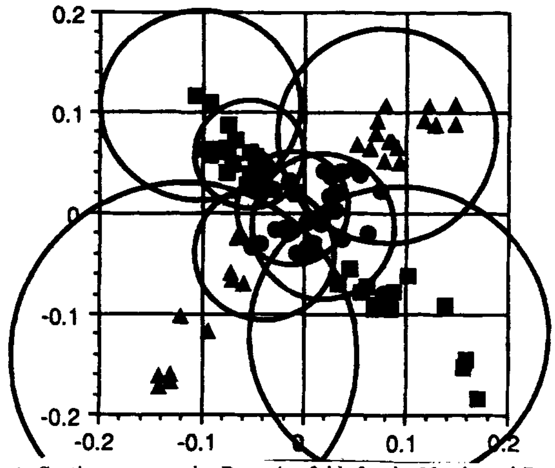

Figure 4: Continuous example. Receptive fields for the Moody and Darken trained 8 hidden unit RBFN.

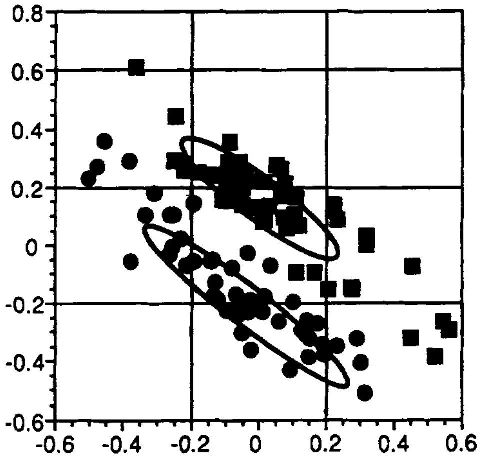

Figure 2: Binary example. Receptive fields for the locally trained 2 hidden unit RBFN.

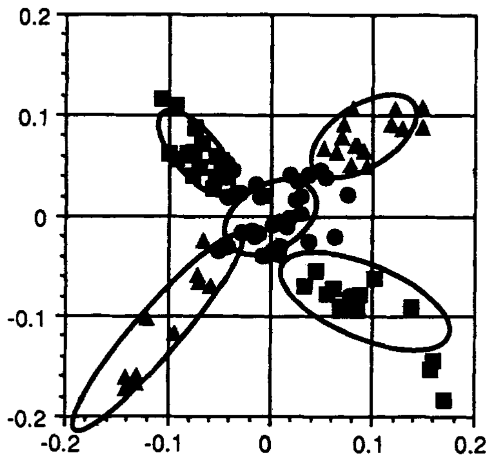

Figure 5: Continuous example. Receptive fields for the locally trained 5 hidden unit RBFN.

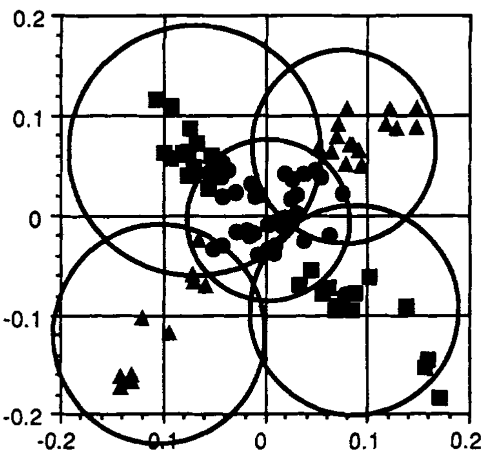

Figure 3: Continuous example. Receptive fields for the Moody and Darken trained 5 hidden unit RBFN.

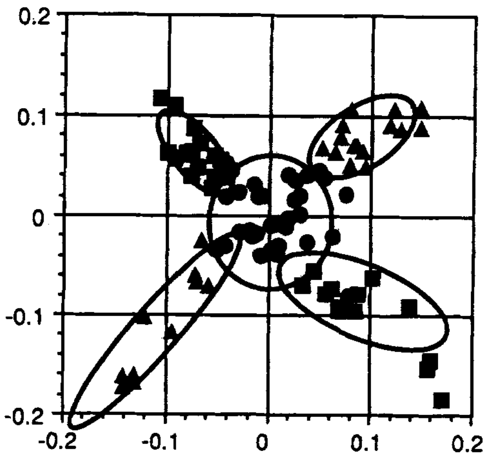

Figure 6: Continuous example. Receptive fields for the locally *ad hoc* trained 5 hidden unit RBFN.