Machine Learning, 27, 173–200 (1997)

Representing Probabilistic Rules with Networks of Gaussian Basis Functions

VOLKER TRESP volker.tresp@mchp.siemens.de
Siemens AG, Central Research, 81730 München, Germany

JÜRGEN HOLLATZ juergen.hollatz@mchp.siemens.de
Siemens AG, Central Research, 81730 München, Germany

SUBUTAI AHMAD ahmad@interval.com
Interval Research Corporation, 1801-C Page Mill Rd., Palo Alto, CA 94304

Editor: Jude W. Shavlik

Abstract. There is great interest in understanding the intrinsic knowledge neural networks have acquired during training. Most work in this direction is focussed on the multi-layer perceptron architecture. The topic of this paper is networks of Gaussian basis functions which are used extensively as learning systems in neural computation. We show that networks of Gaussian basis functions can be generated from simple probabilistic rules. Also, if appropriate learning rules are used, probabilistic rules can be extracted from trained networks. We present methods for the reduction of network complexity with the goal of obtaining concise and meaningful rules. We show how prior knowledge can be refined or supplemented using data by employing either a Bayesian approach, by a weighted combination of knowledge bases, or by generating artificial training data representing the prior knowledge. We validate our approach using a standard statistical data set.

Keywords: Neural networks, theory refinement, knowledge-based neural networks, probability density estimation, knowledge extraction, mixture densities, combining knowledge bases, Bayesian learning

1. Introduction

Many systems which were developed in the field of machine learning are rule-based, i.e., they provide an explicit representation of the acquired knowledge in the form of a set of rules. A rule-based representation has a number of advantages: rules are compact, modular, and explicit, plus they can be analyzed by domain experts and can be checked for plausibility. If it is felt that the represented knowledge is incomplete, informative additional experiments can be designed by carefully analyzing the available rule base. Over the last decade, neural networks (more precisely, artificial neural networks) are being used increasingly as learning systems (Rumelhart & McClelland, 1986; Hertz, Krogh, & Palmer, 1991). In neural networks, the acquired knowledge is only implicitly represented in the network architecture and weight values. It is therefore in general difficult to obtain explicit understanding of what the neural network has learned which in many cases might be highly desirably. Consider the rather spectacular case of Tesauro's TD-Gammon network (1992). TD-Gammon is a neural network that learned to play championship-level Backgammon by playing against itself without a supervisor. TD-Gammon's weights contain a tremendous amount of useful information. Currently there are basically only two ways to understand the functionality of the network: by plotting patterns of weight values or by gathering statistics of the network output through extensive play. The former method provides no more than a general impression; the latter forces the human to redo the entire learning process. It would be extremely helpful if it was possible to automatically construct readable higher level descriptions of the stored network knowledge.

So far we only discussed the *extraction* of learned knowledge from a neural network. For many reasons the"reverse" process, by which we mean the incorporation of prior high-level rule-based knowledge into the structuring and training of a neural network, is of great importance as well. First, a network that has been pre-initialized with domain knowledge — even if it is approximate knowledge— may learn faster (i.e., converge in fewer learning steps to an acceptable solution) than a network learning from scratch (Shavlik & Towell, 1989; Gallant, 1988). A second reason is that in many domains it is difficult to get a significant number of training examples. In this case we clearly want to utilize any prior knowledge we may possess about the domain. A third reason is that the data distribution over input space is often highly nonuniform. Thus even if we have access to a large training corpus, it may contain very few, if any, examples in some regions of input space. However the system's response to those areas may be critical. As an example consider the diagnosis of rare fatal diseases. In such situations, even though the training set may contain few examples of the disease, a domain theory may exist and it is desirable to exploit this knowledge.

The topic of this paper is the two-way relationship — i.e., the extraction of learned knowledge from a trained neural network and the inclusion of prior rule-based knowledge into the structuring and training of neural networks — between network-based representations and higher level, rule-based representations. Previous work in this area has concentrated on the popular multi-layer perceptron architecture either for incorporating rule-based knowledge into network training (Fu, 1989; Towell & Shavlik, 1994) or for extracting knowledge out of a trained network (Fu, 1991; Towell & Shavlik, 1993; Thrun, 1995). In this paper we consider normalized Gaussian basis function (NGBF) networks which represent another commonly used learning system in the neural network community. In Tresp, Hollatz and Ahmad (1993) it was shown that there is a certain equivalence between NGBF-networks and probabilistic rules if appropriate learning rules for the NGBF-networks are used. This approach will be explored in detail in this paper. We will demonstrate that the probabilistic setting has unique advantages. In particular, it is straightforward to calculate inverse models, conditional probability densities, and optimal responses with missing or noisy features. In a non-probabilistic setting these calculations are either impossible or involve either complex numerical integrations or heuristic solutions.

The models described in this paper are based on the mixtures of Gaussians which are commonly used as probability density estimators (Duda & Hart, 1973). Cheeseman *et al.* (1988) use mixtures of Gaussians in their AutoClass system to discover unlabeled classes or clusters in data sets which can be considered as a form of data analysis. The novel aspect of this paper is to develop and exploit the three-way relationship between probabilistic rule-bases, networks of Gaussian basis functions which are commonly used in the neural network community, and statistical Gaussian mixtures models.

Section 2 introduces networks of normalized Gaussian basis functions (NGBFs). Section 3 shows how networks of NGBFs can be constructed using a set of probabilistic rules and demonstrates how they can be used for inference in classification and regression. Section 4 shows how a rule base can be generated from data by training an NGBF-network using appropriate learning rules and by extracting rules after training. Section 5 shows how prior rule-based knowledge can be combined with learning from training data. In Section 6 we present experimental results using a widely used statistical data set and describe methods for optimizing the network structure. Section 7 presents modifications and discusses related work and in Section 8 we present conclusions.

2. Gaussian Basis Function Networks

In this section we introduce Gaussian basis function (GBF) networks and networks of normalized Gaussian basis functions (NGBFs), then discuss the most common algorithms to train NGBF-networks.

Gaussian basis function (GBF) networks are commonly used as predictors and classifiers in the neural network community (Moody & Darken, 1989; Poggio & Girosi, 1990). The output of a GBF-network is the weighted superposition of the responses of N Gaussian basis functions

$$y = GBF(x) = \sum_{i=1}^{N} w_i \exp[-\frac{1}{2}\sum_{j=1}^{M}\frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}]$$

with $x = (x_1, x_2, \ldots, x_M)' \in \Re^M$ and $y \in \Re$; the prime $()'$ indicates the transpose of a vector or matrix. The GBFs are parameterized by the locations of their centers $c_i = (c_{i1}, \ldots, c_{iM})'$, and the vectors of scaling parameters $\sigma_i = (\sigma_{i1}, \ldots, \sigma_{iM})'$ where σ_{ij} is a measure of the width of the ith Gaussian in the jth dimension. Additional parameters are the output weights $w = (w_1, \ldots, w_N)'$. Moody and Darken (1989) also introduced networks of normalized Gaussian basis functions (NGBFs) whose responses are mathematically described as[1]

$$y = NGBF(x) = \frac{\sum_{i=1}^{N} w_i b_i(x)}{\sum_{k=1}^{N} b_k(x)} = \sum_{i=1}^{N} w_i n_i(x) \qquad (1)$$

with

$$b_i(x) = \kappa_i \exp[-\frac{1}{2}\sum_{j=1}^{M}\frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}] \qquad \text{and} \qquad n_i(x) = \frac{b_i(x)}{\sum_{k=1}^{N} b_k(x)}.$$

In this paper, we are only concerned with NGBF-networks.

Typically, network parameters are determined using a training data set $\{(x^k, y^k)\}_{k=1}^{K}$. A number of training methods for NGBF-networks were suggested in the literature. In the method proposed by Moody and Darken, the centers c_i are cluster centers obtained using N-means clustering of the input data distribution. The scaling parameters σ_{ij} are determined using a heuristic, typically they are set to a constant multiple of the average distance between cluster centers. In the soft-clustering algorithm introduced by Nowlan (1991), a Gaussian mixture[2] model of the input data is formed and the centers and standard deviations of the Gaussian mixtures are used for the centers and scale parameters in the NGBF-networks. If a Gaussian unit is placed on each each data point, we obtain architectures proposed by Specht (1990, 1991) and Smyth (1994). The parameters κ_i in the NBGF-network determine the overall weight of a Gaussian in the network response (Tresp, Hollatz, & Ahmad, 1993) and are often set to one.

With centers and scaling parameters fixed, the output weights w which minimize the mean squared error of the NGBF-network on the training data can be determined using

$$w^{ls} = (n'n)^{-1}\, n'\, t^y \tag{2}$$

where $t^y = (y^1, \ldots, y^K)'$ is the vector of targets and n is an $K \times N$ matrix with elements $(n)_{ki} = n_i(x^k)$ (Sen & Srivastava, 1990).

In the learning algorithms just described, centers and widths of the GBFs are determined solely based on the input data and only the output weights are determined using information about the targets y^k. Alternatively, all parameters can be adjusted to minimize the training error defined as

$$\sum_{k=1}^{K} (NGBF(x^k) - y^k)^2 \tag{3}$$

using an optimization routine such as gradient descent (Röscheisen, Hofmann, & Tresp, 1992; Wettscherek & Dietterich, 1992).

In particular in classification problems, it often makes sense to also work with multi-dimensional outputs $y = (y_1, \ldots, y_l, \ldots, y_C)'$, with

$$y_l = \sum_{i=1}^{N} w_{il}\, n_i(x) \tag{4}$$

and where C denotes the number of classes. In classification, x^k corresponds to the feature vector of the $k-$th training pattern and

$$y_l^k = \begin{cases} 1 & \text{if } j = l \\ 0 & \text{otherwise} \end{cases}$$

indicates that the j-th class is the correct class for the k-th training pattern. During recall, a pattern is assigned to that class whose corresponding network output has maximum activity.

The responses of the Gaussian basis functions tend to have a local character in the sense that each basis function contributes to the response of the network in only a local region of the input space close to its center. The extend of this region of influence is determined by the scaling parameters. This indicates that we might be able to formulate approximate rules of the form: If $x \approx c_i$ THEN $y \approx w_i$. Figure 1 shows that this rule is much more consistent with the response of the NGBF-network than with the response of the GBF-network. The reason is that in the calculation of the response of the GBF-network, the contributions of the individual basis functions are additive. The total response of the network of NGBF-functions on the other hand is a weighted average of the responses of the individual units,

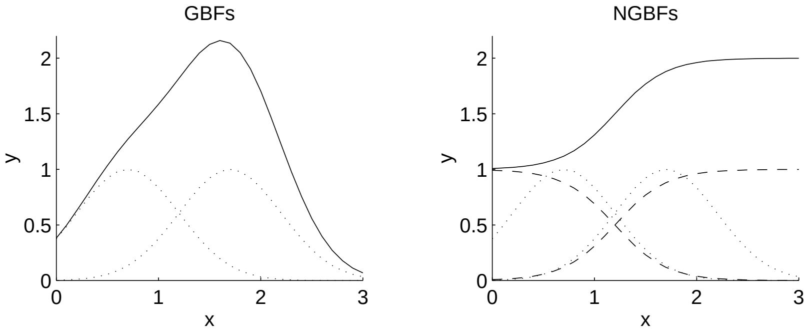

Figure 1. The figure compares the response of a network of Gaussian basis functions (GBFs, left) with the response of a network of normalized Gaussian basis functions (NGBFs, right). On the left, we see two Gaussian basis functions (dotted). The network response (continuous line) is a superposition of the two Gaussians weighted by the output weights 1.0 (for the left GBF) and 2.0 (for the right Gaussian). The right graph displays the responses of two normalized Gaussian basis functions with identical output weights (1.0, 2.0). Shown are also the two basis functions (dotted) and the normalized Gaussian basis functions (dashed). We used $\kappa_1 = \kappa_2 = 1$. It is apparent that the network response of the NGBF-network corresponds to the intuitive notion that close to the center of the left Gaussian the output should be close to the output weight of the left Gaussian (i.e., 1.0) and close to the center of the right Gaussian, the output should be close to the output weight of the right Gaussian (i.e., 2.0).

where the weighting function of each individual basis function is proportional to the activity of the corresponding Gaussian. This averaging results in a compromise between the output weights of the active Gaussians. In the following sections we pursue this observation further. We will show that by formulating probabilistic rules we can construct NGBF-networks and that probabilistic rules can be extracted from NGBF-networks trained on data. In addition, we will show how rule-based knowledge can be used in several ways in combination with trained NGBF-networks.

3. Constructing NGBF-networks from Rules

In this section we show how simple probabilistic rules can be used to incorporate prior knowledge of a domain expert. Then we show that if we perform inference using those probabilistic rules we obtain an NGBF-architecture. The premises of the rules make statements about the state of a discrete variable. In classification applications that variable typically has a real world meaning (i.e., the class). We show that this need not be the case and one novel aspect of this paper is to demonstrate how rules with premises which have no obvious real world meaning can be used to generate rule bases which are particularly useful for making inferences about continuous quantities.

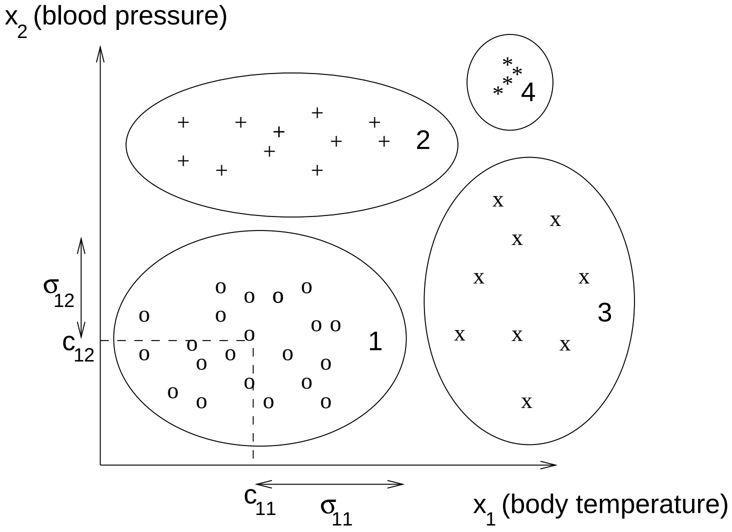

Figure 2. Shown are the distributions of the features (body temperature, blood pressure) for a healthy patient (1, o), for a patient with disease A (2, +), for a patient with disease B (3, x) and for a patient with both diseases (4, *). The ovals indicate the extend of Gaussians modeling the respective data distributions and the symbols o, +, x, and * indicate samples.

3.1. Classifiers

Classification can be considered as the problem of estimating the state of a discrete C-state random variable s (i.e., the class) given a feature vector $x = (x_1, \ldots, x_M)'$ where C is the number of classes and, in general, $x_i \in \Re$. Given the feature vector we can calculate the posterior probability of a class using Bayes' rule as

$$P(s = i|x) = \frac{P(x|s = i)P(s = i)}{\sum_{j=1}^{C} P(x|s = j)P(s = j)}. \qquad (5)$$

Here, $P(s = i)$ is the prior probability of class i and $P(x|s = i)$ is the probability density of feature vector x given class i. A simple example is shown in Figure 2. The different classes represent a healthy patient, a patient with disease A, a patient with disease B and a patient with both diseases. The two features are $x_1 =$ body temperature and $x_2 =$ blood pressure. We assume that the conditional densities of the features given the classes can be represented by normal densities

$$P(x|s = i) = G(x; c_i, \sigma_i) = \frac{1}{(2\pi)^{M/2} \prod_{j=1}^{M} \sigma_{ij}} \exp[-\frac{1}{2} \sum_{j=1}^{M} \frac{(x_j - c_{ij})^2}{\sigma_{ij}^2}]. \qquad (6)$$

Here, only axis-parallel Gaussians (i.e., with diagonal covariance matrices) are used which means that the individual features are independent if the true class is known. Note that

$G(x; c_i, \sigma_i)$ is our notation for a normal density centered at c_i and with a vector of scaling parameters σ_i. If the patient is healthy, which is true with probability $P(s = 1)$, body temperature and blood pressure are in a normal range. Similarly, the second Gaussian models disease A which results in high blood pressure and normal body temperature, the third Gaussian models disease B which results in high body temperature and normal blood pressure and the fourth Gaussian models patients with both diseases which results in high blood pressure and high body temperature. In terms of probabilistic networks (Figure 3A) (Pearl, 1988), the variable s which has C different states can be considered a parent node and the x_i are children which are independent if the state of s is known.

For easier interpretation, the human expert might find it convenient to put his or her knowledge in form of probabilistic rules (Table 1).

Table 1. A classification rule. For each class i define:

IF:		class i is true (which is the case with prior probability $P(s = i)$)
THEN:		(the features are independently Gaussian distributed and)
		the expected value of x_1 is c_{i1} and the standard deviation of x_1 is σ_{i1}
	AND	the expected value of x_2 is c_{i2} and the standard deviation of x_2 is σ_{i2}
	. . .	
	AND	the expected value of x_M is c_{iM} and the standard deviation of x_M is σ_{iM}.

Since the discrete variable s appears in the premise of the rules we will denote s in the following as the premise variable. The (typically real-valued) variables $\{x_j\}_{j=1}^M$ will be denoted interchangeably as conclusion variables or feature variables. If the expert defines a set of rules in this form we can use Equation 5 to perform inference, i.e., to classify a novel pattern. But note that if we use centers c_i and scaling parameters σ_i from the rule base, and set

$$\kappa_i = P(s = i) \times \frac{1}{(2\pi)^{M/2} \prod_{j=1}^{M} \sigma_{ij}}$$

and $w_{il} = 1$ if the i-th Gaussian is assigned to class l and $w_{ij} = 0$ otherwise we obtain an NGBF-classifier (Equation 4).

In the way just described we can build or prestructure an NGBF-classifier using a set of probabilistic rules.

3.2. More Complex Classifiers

If there are correlations between the features for each class or, more general, if $P(x|s = i)$ cannot be described by a single Gaussian, the classifier which was described in the last section is too simplistic. There are two obvious ways more complex classifiers can be built. As described by Ghahramani and Jordan (1993), linear correlations between features can be modeled by Gaussians with full covariance matrices. Alternatively, we can allow for more than one Gaussian to approximate the class conditional density. Let N be the total number of Gaussians and let $I(i)$ denote the set of indices of the Gaussians which are assigned to class i. Then

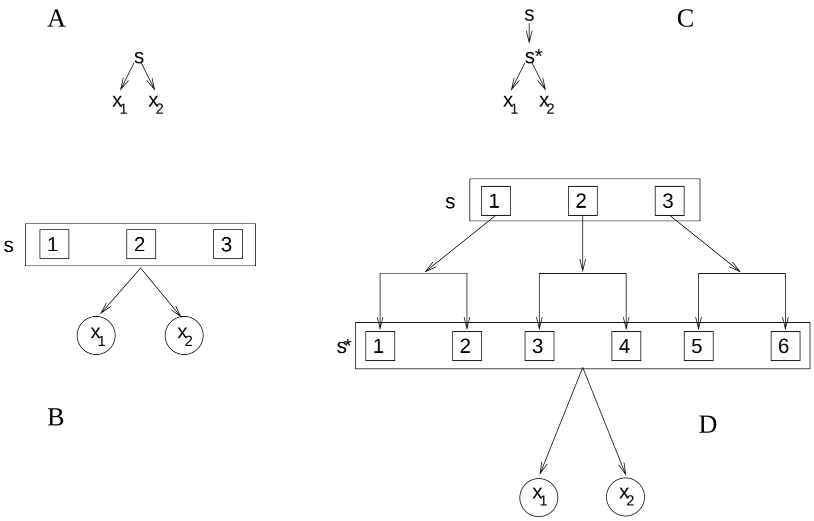

Figure 3. A: The dependency structure of a classification problem. The directed arcs indicate that the features x_1 and x_2 are independent if the state of s is known. B: As in A, with the internal states of s shown, indicating that there are three classes. C: Dependency structure of a hierarchical system. D: As in C, with the internal states of s and s^* shown. The arrows between the states of s and s^* indicate that pairs of Gaussians model the class-specific feature distributions.

$$P(x|s=i) = \sum_{j \in I(i)} P(s^* = j|s=i)\, G(x; c_j, \sigma_j) \tag{7}$$

where the state of $s \in \{1, \ldots, C\}$ indicates the class and the state of $s^* \in \{1, \ldots, N\}$ indicates the Gaussian unit with the constraint that $\sum_{j \in I(i)} P(s^* = j|s=i) = 1$ and with $P(s^* = j|s=i) = 0$ if $j \notin I(i)$.

If we substitute Equation 7 into Equation 5 we obtain again an NGBF-classifier

$$P(s=i|x) = \frac{P(s=i) \sum_{j \in I(i)} P(s^* = j|s=i)\, G(x; c_j, \sigma_j)}{\sum_{k=1}^{C} P(s=k) \sum_{j \in I(k)} P(s^* = j|s=k)\, G(x; c_j, \sigma_j)}. \tag{8}$$

In Figure 3 on the left, we indicate graphically the relationship between s (with 3 states) and the features x_1 and x_2 of the simple classifier described in the last section. On the right side we show the structure of a classifier where each class-conditional density is modeled by more than one Gaussian. In some cases the states of s^* might also have a real world meaning.

The corresponding rule-based formulation is shown in Table 2. Here, the expert also has to specify $P(s^* = j|s=i)$ for all classes i and Gaussians j. Note, that the hierarchy can be extended to an arbitrary number of layers.[3]

Table 2. A hierarchical rule-base. For each class i and each unit $j \in I(i)$, define:

IF:		class i is true (which is the case with prior probability $P(s = i)$)
THEN:		$s^* = j$ is true with probability $P(s^* = j \mid s = i)$.
IF:		$s^* = j$ is true
THEN		the expected value of x_1 is c_{j1} and the standard deviation of x_1 is σ_{j1}
	AND	the expected value of x_2 is c_{j2} and the standard deviation of x_2 is σ_{j2}
	. . .	
	AND	the expected value of x_M is c_{jM} and the standard deviation of x_M is σ_{jM}.

3.3. Modeling the Relationship between Continuous Variables

In many applications we are interested in making inference about a continuous quantity. It is not obvious how a discrete set of rules can be used to describe the structure in continuous variables. The basic idea presented here is to interpret the premise variable s as a hidden variable with no obvious real-world meaning. Consider Figure 4. Here we plot a (made up) distribution of the height and weight of a population. We cannot really claim that weight is the cause for height or vice versa. Also there are no obvious underlying causes (genetic factors, race, gender: we do not really know) which explain the data. Rather, to explain the data we "invent" hidden causes which are represented by the state of the discrete hidden variable s. In this example, there are only two hidden states $s = 1$ and $s = 2$ and a domain expert might specify rules as in Table 3.

Table 3. A modeling rule. For each hidden state i:

IF:		$s = i$ (which is the case with prior probability $P(s = i)$)
THEN:		the expected value of x_1 is c_{i1} and the standard deviation of x_1 is σ_{i1}
	AND	the expected value of x_2 is c_{i2} and the standard deviation of x_2 is σ_{i2}

The use of hidden random variables (in our sense variables without a real-world meaning) has a long tradition both in neural networks and probability theory (consider, for example, hidden Markov models in speech recognition and the hidden states in Boltzmann machines). Pearl (1988) argues that humans have a strong desire to "invent" hidden causes to explain observations. They are simply convenient vehicles for describing the observed world. The advantage is, as demonstrated, that a complex uncertain relationship between variables can be concisely summarized using a small number of simple rules. As in classification (Section 3.2) we can introduce hierarchies which enables us to describe the relationship between the variables at difference scales of resolution. In the next section, we will describe how this model can be used for prediction.

3.4. Inference and Prediction

A rule base might contain a mixture of rules with premises with or without a real-world meaning. In this section we show how we can use such a rule base to infer the states of

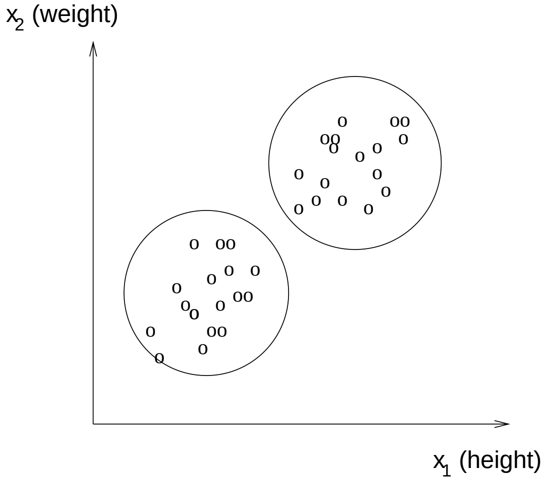

Figure 4. The graph shows two Gaussians (ovals) which model the joint distribution between two features (height, weight). Samples are indicated by small circles.

variables based on knowledge about the states of some other set of variables. We will show that inference rules can be realized by NGBF-networks. We already presented two inference rules: Equations 5 and 8 showed how we can infer the state of premise variable s if the feature vector is complete (i.e., all the states of x are known) for a simple classifier and for a hierarchical classifier. Here, we show how inference is performed if the feature vector is incomplete (we only have partial knowledge) or if we want to infer the state of one of the real-valued components of the feature vector.

Let us assume that only some of the components of the feature vector x are known. In classification we are then faced with the problem of estimating the correct class from incomplete features. Or, as in the example depicted in Figure 4, we might be interested in predicting x_2 (i.e., the weight which is unknown or missing) from x_1 (i.e., the height which can be measured) or vice versa. Let $x^m \subset \{x_1, \ldots, x_M\}$ denote the known feature variables, let $x^u = \{x_1, \ldots, x_M\} \setminus x^m$denote the unknown feature variables, and let c_i^m and σ_i^m consist of the components of c_i and σ_i in the dimensions of x^m. The probability of $s = i$ given x^m can be easily calculated as

$$P(s = i|x^m) = \frac{P(x^m|s = i)\ P(s = i)}{\sum_{j=1}^{N} P(x^m|s = j)\ P(s = j)} \tag{9}$$

where,

$$P(x^m|s = i) = \int G(x; c_i, \sigma_i)\ dx^u = G(x^m; c_i^m, \sigma_i^m). \tag{10}$$

The last equality demonstrates that the marginal distribution of a Gaussian is again a Gaussian: it is simply the projection of the Gaussian onto the dimensions of x^m. This is the reason why our model handles missing variables so easily (see also Ahmad & Tresp, 1993).

We can also predict any of the unknown feature variables $y \in x^u$ from the known feature variables x^m. Let c_i^y and σ_i^y denote center and width of the i-th Gaussian in y-dimension. The conditional density of y is

$$P(y|x^m) = \frac{\sum_{i=1}^N G(y; c_i^y, \sigma_i^y)\ G(x^m; c_i^m, \sigma_i^m)\ P(s=i)}{\sum_{j=1}^N G(x^m; c_j^m, \sigma_j^m)\ P(s=j)}. \tag{11}$$

For prediction we are typically interested in the expected value[4] of y given x^m, which can also be easily be calculated

$$E(y|x^m) = \frac{\sum_{i=1}^N w_i\ G(x^m; c_i^m, \sigma_i^m) P(s=i)}{\sum_{j=1}^N G(x^m; c_j^m, \sigma_j^m)\ P(s=j)} \tag{12}$$

where $w_i = c_i^y$. Note that that last equation can be realized by an NGBF-network (Equation 1). This means that NGBF-networks for estimating continuous variables can be constructed from probabilistic rules in a similar way as NGBF-networks for classification.

We want to emphasize again, that by using a probabilistic model, we can predict *any* feature variable $y \in \{x_1, \ldots, x_M\}$ from any set of measured feature variables $x^m \subset \{x_1, \ldots, x_M\}$. [5]

In Section 3.2 we showed how the class can be estimated in hierarchical models (Equation 8). Here, we derive Equations for estimating an unknown feature variable in a hierarchical model. For the expected value of an unknown variable y, we obtain using Bayes' rule

$$E(y|x^m) = \frac{\sum_{i=1}^C P(s=i) \sum_{j \in I(i)} w_j\ P(s^* = j | s = i)\ G(x^m; c_j^m, \sigma_j^m)}{\sum_{i=1}^C P(s=i) \sum_{j \in I(i)} P(s^* = j | s = i)\ G(x^m; c_j^m, \sigma_j^m)} \tag{13}$$

with $w_j = c_j^u$. This can also be written as

$$E(y|x^m) = \sum_{i=1}^C [g_i(x^m) \sum_{j \in I(i)} w_j\ g_j^*(x^m)] \tag{14}$$

where

$$g_i(x^m) = P(s=i|x^m) = \frac{P(s=i) \sum_{j \in I(i)} P(s^* = j | s = i) G(x^m; c_j^m, \sigma_j^m)}{\sum_{i=1}^C P(s=i) \sum_{j \in I(i)} P(s^* = j | s = i) G(x^m; c_j^m, \sigma_j^m)}$$

and

$$g_j^*(x^m) = P(s^* = j | x^m, s = i) = \frac{P(s^* = j | s = i) G(x^m; c_j^m, \sigma_j^m)}{\sum_{j \in I(i)} P(s^* = j | s = i) G(x^m; c_j^m, \sigma_j^m)}$$

Note that Equation 14 describes a hierarchical mixture of expert model (Jordan & Jacobs, 1993) with gating networks $g_i(x^m)$ and $g_j^*(x^m)$ and simple expert networks with constants outputs w_i (see discussion in Section 7).

4. Learning: Generating Rules out of a Data Set

So far we only considered that NGBF-networks are constructed based on probabilistic rules defined by a domain expert. In this section we show how we can *generate* rules from data by first training NGBF-networks with the appropriate probabilistic learning rule and by then extracting probabilistic rules to be analyzed by an expert.

We assume that the network structure is given, i.e., we know how many Gaussian basis functions are required for modeling in Section 3.3 or for approximating the class-specific density in Section 3.2. Model selection is discussed in Section 6.

We present learning rules for the simple non-hierarchical model; the learning rules for the hierarchical model can be found in Appendix A. We assume that we have K training data $\{x^k, s^k\}_{k=1}^K$. First we consider the case that the state of s^k is unknown. In this case the log-likelihood function of the model[6] is

$$L = \sum_{k=1}^{K} \log[\sum_{i=1}^{N} \hat{P}(s=i)\ G(x^k; \hat{c}_i, \hat{\sigma}_i)].$$

This is simply the log-likelihood of the Gaussian mixture model and we can use the well-known EM (expectation maximization) algorithm for learning (Dempster, Laird, & Rubin, 1977) which converges to a local maximum of the log-likelihood function. The EM algorithm consists of the repeated application of the E-step and the M-step. In the E-step, we estimate the states of the missing variables using our current parameter estimates. More precisely, the E-step estimates the probability that x^k was generated by component $s = i$

$$\hat{P}(s=i|x^k) = \frac{\hat{P}(s=i)\ G(x^k; \hat{c}_i, \hat{\sigma}_i)}{\sum_{l=1}^{N} \hat{P}(s=l)\ G(x^k; \hat{c}_l, \hat{\sigma}_l)}.$$

The M-step updates the parameter estimates based on the estimate $\hat{P}(s=i|x^k)$

$$\hat{P}(s=i) = \frac{1}{K} \sum_{k=1}^{K} \hat{P}(s=i|x^k), \quad (15)$$

$$\hat{c}_{ij} = \frac{\sum_{k=1}^{K} \hat{P}(s=i|x^k)\ x_j^k}{\sum_{k=1}^{K} \hat{P}(s=i|x^k)}, \quad (16)$$

$$\hat{\sigma}_{ij}^2 = \frac{\sum_{k=1}^{K} \hat{P}(s=i|x^k)\ (\hat{c}_{ij} - x_j^k)^2}{\sum_{k=1}^{K} \hat{P}(s=i|x^k)}. \quad (17)$$

The EM algorithm is used off-line although approximate on-line versions also exist (Nowlan, 1990; Neal & Hinton, 1993; Ghahramani & Jordan, 1993). The EM algorithm can also be used if some of the features in x^k are unknown or uncertain as shown in Tresp, Ahmad and Neuneier (1994) and Ghahramani and Jordan (1993).

Alternatively, we can use a number of unsupervised learning rules, out of the extensive literature on that topic, to determine the parameters in the Gaussian mixture network such as learning vector quantization, Kohonen feature maps, or adaptive resonance theory (see Hertz, Krogh, & Palmer, 1991). We prefer the EM learning rules mainly because they have a sound statistical foundation by optimizing the log-likelihood function.

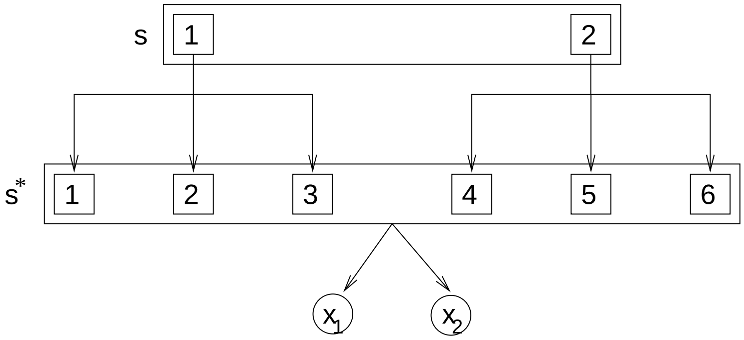

Figure 5. The left part of the model ($s^* \in \{1, 2, 3\}$) is trained on data and the right part ($s^* \in \{4, 5, 6\}$) is constructed from rules defined by a domain expert.

5. Combining Knowledge Bases

In Section 3 we constructed networks of Gaussian basis functions using prior knowledge and in the last section we trained mixture models from data to generate NGBF-networks. In many applications we might have both training data and domain expert knowledge available and in this section we will show how both can be combined (i.e., how the rule-based knowledge can be refined).

5.1. Incremental Mixture Density Models

The simple idea pursued in this section is to build one probabilistic model using the rules defined by the domain expert and to build a second model using the training data set and then to combine the two models to form a combined model containing both sub-models. Let's consider the example shown in Figure 5. The left part of the model ($s^* \in \{1, 2, 3\}$) is trained on data yielding $P(s^* = 1|s = 1), P(s^* = 3|s = 1), P(s^* = 3|s = 1), G(x; c_1, \sigma_1), G(x; c_2, \sigma_2)$, and $G(x; c_3, \sigma_3)$. The right portion ($s^* \in \{4, 5, 6\}$) is constructed from rules defined by a domain expert yielding $P(s^* = 4|s = 2), P(s^* = 5|s = 2), P(s^* = 6|s = 2), G(x; c_4, \sigma_4), G(x; c_5, \sigma_5)$, and $G(x; c_6, \sigma_6)$. The domain expert also has to define K^E which defines the equivalent number of training data the expert knowledge is worth, i.e., the certainty of the expert knowledge. If K is the number of data used for training we obtain $P(s = 1) = K/(K + K^E)$ and $P(s = 2) = K^E/(K + K^E)$. For inference, we can then use Equations 8 and 13. If we obtain more data or more rules we can add models in an obvious way and build up our knowledge base incrementally.

We have obtained a way of combining different knowledge bases or experts which forms a solution by "voting" or "mixing," which is distinct from standard Bayesian approaches to learning where prior knowledge is incorporated in priors on network parameters and network complexity (see the next section).

In analogy to Bayesian learning, we can add a default expert to the model who represents our knowledge prior to the availability of a domain expert and prior to the availability of data. Such a default expert might consist of one rule represented by a Gaussian centered at $c_d = 0$. The *a priori* weight of the default expert represented by K_d should be a small number. If later other experts are added they will dominate the prediction of the system where they are certain. In regions where no other expert is “active” the default expert will dominate.

5.2. Fine-Tuning: Bayesian Learning

As in the last section, we assume that a network was constructed from the probabilistic rules defined by a domain expert. If only a relatively small number of training samples are available, adding another network might add too much variance to the model and one might get better results by simply fine-tuning the network built from the domain expert. This is the basic idea in a Bayesian approach (Bernardo & Smith, 1993; Buntine & Weigend, 1991; MacKay, 1992). Let $P_W^M(x)$ denote a model of the probability density of x with parameter vector W (i.e., $\{c_i, \sigma_i, P(s = i)\}_{i=1}^N$). In an Bayesian approach the expert has to define $P(W)$ which is the prior distribution of the parameters. The predictive posterior probability is then

$$P^M(x|Data) = \int P^M(x|W) P^M(W|Data)\, dW \tag{18}$$

with

$$P^M(W|Data) = \frac{P^M(Data|W) P^M(W)}{P^M(Data)}.$$

Here, $P^M(Data|W) = \prod_{k=1}^K P^M(x^k|W)$ is the likelihood of the model. A commonly used approximation is

$$P^M(x|Data) \approx P^M(x|W^{MAP})$$

where

$$W^{MAP} = \arg\max_W P^M(Data|W) P^M(W)$$

i.e., one substitutes the parameters with the maximum a posterior (MAP) probability.

In a Bayesian approach the expert has to specify an a priori parameter *distribution* $P^M(W)$. When the expert can formulate her or his knowledge in terms of *conjugate* priors the EM update rules can be modified to converge to the MAP parameter estimates (Buntine, 1994; Ormoneit & Tresp, 1996).

A similar combination of prior knowledge and learning can be achieved by a procedure which is known as *early stopping* in the neural network literature (Bishop, 1995). Early stopping refers to a procedure where training is terminated before the minimum of the cost function is reached to obtain a regularized solution. We can use early stopping here in the following way. First, we build a network using prior rules. This network is used as the initialization for learning (using EM). If we train to convergence we completely eliminate the initialization (although we still influence which local optimum of the log-likelihood function is found) and if we do not train at all we ignore the data. If we train only a few iterations (*early stopping*) the resulting network will still contain a bias towards the initialization i.e., the prior knowledge.

5.3. Teacher-Provided Examples

Finally, we would like to present a third alternative. This approach is particularly interesting when it is not possible to formulate the domain knowledge in the form of probabilistic rules, but a domain expert is available who can be queried to provide typical examples. We want to penalize a model if the examples provided by the domain expert $\{x^{t,l}\}_{l=1}^{L}$ are not well represented by the model with parameter vector W. This can be put into a Bayesian framework if we define

$$P^M(W) \propto \prod_{l=1}^{L} P^M(x^{t,l}|W).$$

The MAP estimate then maximizes

$$P^M(W|Data) \propto \prod_{l=1}^{L} P^M(x^{t,l}|W) \prod_{k=1}^{K} P^M(x^k|W).$$

Since the prior has the same form as the likelihood, the examples provided by the expert can be treated as additional data (see Röscheisen, Hofmann, & Tresp, 1992). The extended data set is now $\{x^k\}_{k=1}^{K} \cup \{x^{t,l}\}_{l=1}^{L}$.

6. Network Optimization and Experiments

6.1. The Test Bed

As a test bed we used the Boston housing data. The data set consists of 506 samples with 14 variables. The variables are the median value of homes in Boston neighborhoods and 13 variables which potentially influence the housing prices (Harrison & Rubinfeld, 1978). The variables are described in Appendix B. We selected this data set because all variables have an easily understandable real-world meaning. All variables were normalized to zero mean and a standard deviation of one. Unless stated otherwise we divided the data into 10 equally sized sets. Ten times we trained on nine of the sets and tested on the left out set. The performance on the test sets was used to derive error bars for the generalization performance (Figures 6, 7 and 8) and for the two-tailed paired t-test in Section 6.3's Figure 9. The nine sets used for training are each equally divided into a training set and a validation set. The training set is used for learning the parameters in the networks and the validation set is used for determining optimal network structures in the following experiments.

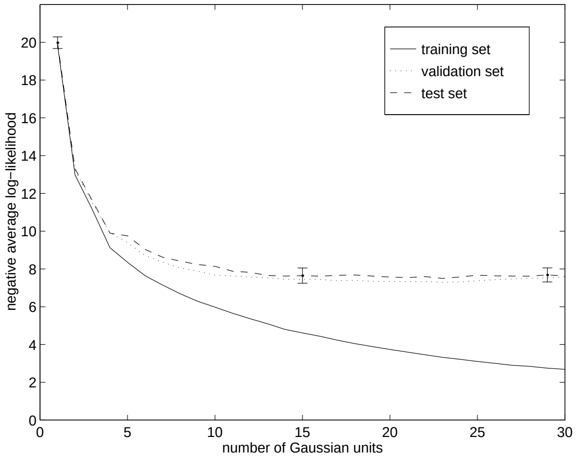

Figure 6. The graph shows the negative average log-likelihood for training set, validation set and test set as a function of the number of Gaussian units. Displayed are averages over ten experiments with different separations into training set, validation set and test set. The error bars indicate the variation in performance on the test set.

6.2. Network Optimization and Rule-extraction

Our first goal is to extract meaningful rules out of the data set. In this data set it is not known *a priori* how many Gaussians units are required to model the data. Our strategy is to start with a network with a rather large number of units and then to remove units by observing the network performance on the validation data set.[7] After a Gaussian is removed the network is retrained using the EM learning rules of Section 4. To select good candidate units to be pruned one might want to select a unit with a small probability of being chosen, i.e., with a small $P(s = i)$. But this unit might represent data points far away from the centers of the remaining Gaussians and those data points would then be represented badly by the remaining network after that unit is eliminated. A better way to prune is therefore to tentatively remove a unit and then recalculate the likelihood of the model on the training data set (without retraining), finally, pruning units whose removal decreases the likelihood the least (pruning procedure 1). In some cases this procedure might be too expensive (e.g. if there are too many training data or units) or the data set might not be available any more (as in on-line learning). In those cases we might decide to remove a unit which is well represented by the other units. Consider that we want to estimate the effect of the removal of the j-th unit. After removal, the data which were modeled by the j-th unit are now modeled by the remaining Gaussian units. The contribution of these data points to the log-likelihood function after removal of the j-th unit can be estimated as

$$L(j) \approx K \times P(s = j) \; \log \sum_{i, i \neq j} P(s = i) P(c_j | s = i). \tag{19}$$

$K \times P(s = j)$ is an estimate of the number of data points modeled by the j-th unit and the sum in the logarithm is equal to the probability density at $x = c_j$ after the removal of unit j. The procedure consists of removing the unit with smallest $L(j)$ (pruning procedure 2). Our experiments showed that both pruning procedure 1 and pruning procedure 2 almost always decide on the same order of units to prune.

In the experiments we started with 30 units and trained a Gaussian mixture model using the EM algorithm (Section 4). We then proceeded to prune units following pruning procedure 2. Each time a unit was removed, the network was retrained (using the EM algorithm).

In Figure 6 we plot the negative average log-likelihood as a function of the number of units in the network for the training data set, for the validation set and for the test data set. A good model has a small negative average log-likelihood. The large difference between test set and training set with a large number of units can be explained by the large variance in the network due to the large number of units. Based on the performance on the validation set we can conclude that between 8 and 10 units are necessary for a good model. In the following experiments we used networks with three units since three units are sufficient for acceptable performance (Figure 6) and the extracted rules are easily interpretable (see the following section). If more units are used the performance is better but the larger number of rules is more difficult to interpret.

6.2.1. *Simplifying Conclusions*

We can attempt to further simplify the model. To motivate this step, consider an example from medical diagnosis. To diagnose diseases we might consider 100 features or symptoms. Certainly, all symptoms are important but in most cases the diagnosis of a disease is only dependent on a small number of features and ignores the remaining ones. Therefore, rules of the form: IF *the patient has disease A* THEN *feature one (fever) is high but all other features (here: 99) are normal* seems reasonable. In this spirit, we bias the model to set as many of the conclusions as possible to “normal” to obtain very parsimonious rules.

In this medical example it might be clear *a priori* or from the data set what exactly a “normal” feature distribution means. In our data set this is not so obvious. Therefore, we simply calculate mean $mean_j$ and standard deviation std_j of each variable x_j in the network and define a normal feature j as one which is distributed as $P(x_j) \approx G(x_j; mean_j, std_j)$. Remember that in our model $G(x_j; c_{ij}, \sigma_{ij})$ represents the j-th conclusion of the i-th rule. In the first step we find conclusions which are close to “normal” which means that

$$G(x_j; c_{ij}, \sigma_{ij}) \approx G(x_j; mean_j, std_j)$$

holds. A useful measure for the difference between two distributions is the Kullback-Leibler distance (Cover & Thomas, 1991). The Kullback-Leibler distance between the two continuous probability densities $P_1(x), P_2(x)$ is defined as

$$KL(P_1(x), P_2(x)) = \int P_1(x) \log[\frac{P_1(x)}{P_2(x)}] \, dx.$$

Applied to our problem, and exploiting the assumption that features are Gaussian distributed we obtain

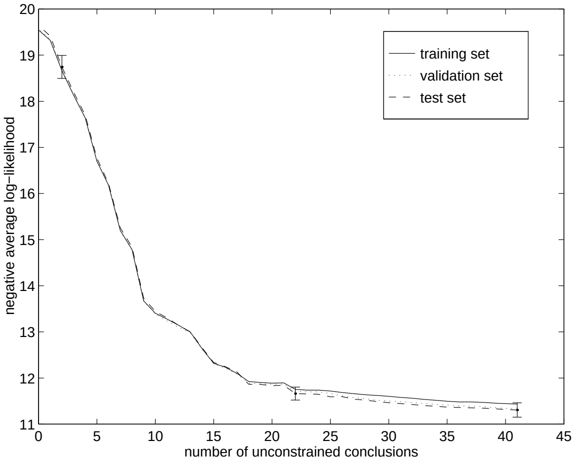

Figure 7. The graph shows the negative average log-likelihood of for training set, validation set and test set as a function of the number of unconstrained conclusions using a network with three Gaussian units. Displayed are averages over ten experiments with different separations into training set, validation set and test set. The error bars indicate the variation in performance on the test set.

$$KL(G(x_j; c_{ij}, \sigma_{ij}), G(x_j; mean_j, std_j)) \tag{20}$$

$$= \log[\frac{\sigma_{ij}}{std_j}] - \frac{1}{2} + \frac{std_j^2 + (c_{ij} - mean_j)^2}{2\sigma_{ij}^2}.$$

In the experiments we rank each conclusion of each Gaussian according to the distance measure in Equation 20. Assuming that for the j-th conclusion of the i-th Gaussian the Kullback-Leibler distance in Equation 20 is smallest, we then set $c_{ij} \rightarrow mean_j$ and $\sigma_{ij} \rightarrow std_j$. Figure 7 shows the negative average log-likelihood for training set, validation set and test set as a function of the number of conclusions which are not set to “normal” using a system with three units. We see that approximately 10 features can be set to normal without any significant reduction in performance (leaving 32 unconstraint conclusions). Tables 4 and 5 summarize an example of a resulting network.

The first rule or Gaussian unit ($i = 1$), for example, can be interpreted as a rule which is associated with a high housing price (feature 14). It translates into the rule shown in Table 6. According to the rule a low crime rate (feature 1) and a low percentage of lower status population (feature 13) are associated with a high house price. Similarly, Gaussian unit ($i = 2$) can be interpreted as a rule which is associated with a low housing price and Gaussian unit ($i = 3$) can be interpreted as a rule which is associated with an average housing price.

Table 4. Centers and scaling parameters in the trained network with three Gaussian units.

feature (j)		c_{ij}			σ_{ij}	
	$i = 1$	$i = 2$	$i = 3$	$i = 1$	$i = 2$	$i = 3$
1	-0.40	1.20	-0.28	0.20	1.76	0.20
2	normal	-0.49	-0.41	normal	0.20	0.33
3	-0.84	1.02	0.61	0.41	0.20	0.94
4	-0.25	-0.27	0.62	0.31	0.20	1.65
5	-0.69	1.02	normal	0.52	0.50	normal
6	normal	normal	normal	normal	normal	normal
7	-0.65	0.75	0.63	0.90	0.46	0.61
8	0.66	-0.83	-0.58	0.96	0.28	0.49
9	-0.59	1.66	-0.42	0.20	0.20	0.57
10	-0.68	1.53	-0.10	0.33	0.20	0.70
11	normal	0.81	normal	normal	0.20	normal
12	0.37	-0.83	normal	0.20	1.63	normal
13	-0.62	0.94	normal	0.54	0.90	normal
14	0.47	-0.82	normal	0.92	0.71	normal

Table 5. Prior probabilities for the units ($P(s = i)$) of the trained network with three Gaussian units.

i:	1	2	3
$P(s = i)$:	0.48	0.25	0.27

Table 6. Extracted rule for $i = 1$.

IF:		$s = 1$ (which is the case with prior probability 0.48)
THEN:		the expected value of $crime$ is -0.40 and the standard deviation of $crime$ is 0.20
	AND	zn is normal
	. . .	
	AND	the expected value of mv is 0.47 and the standard deviation of mv is 0.92

6.2.2. *Removing Variables (Input Pruning)*

The Gaussian units model the relationship among the 14 variables. More precisely, they model their joint probability density which allows the calculation of many quantities of interest such as conditional densities and expected values (i.e., inference, Equation 11 and 12). As a drawback, the Gaussian mixture model does not provide any information about independencies between variables as, for example, Bayesian networks are capable of doing (Pearl, 1988; Heckerman, 1995; Heckerman, Geiger, & Chickering, 1995; Buntine, 1994; Hofmann & Tresp, 1996). Here, we want to address the simpler question of which variables are required to predict one particular variable. It is well known that the removal of variables which are not relevant for the prediction often improves the performance of a predictor. Variables can be removed if they are either completely independent of the variable to be predicted or if the information which is contained in a variable is already

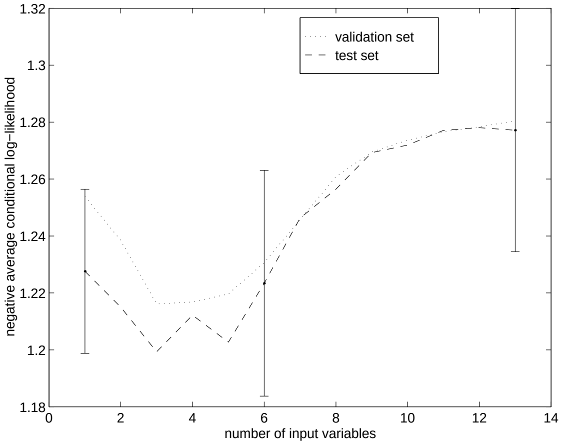

Figure 8. The graph shows the negative average *conditional* log-likelihood of validation set and test set as a function of the number of input variables using a network with three Gaussian units. Displayed are averages over ten experiments with different separations into training set, validation set and test set. The error bars indicate the variation in performance on the test set.

represented in the remaining variables: as an example consider that an input variable is a linear or a nonlinear function of the remaining input variables.

Let y be the variable to be predicted; y is independent of an input variable, say x_j, conditioned that we know the remaining variables if

$$P(y|\{x_1, \ldots, x_M\}) = P(y|\{x_1, \ldots, x_M\} \setminus x_j).$$

Since the true underlying model is unknown we have to base our decision on the available data. We evaluate the conditional log-likelihood which is defined as

$$L_C = \sum_{k=1}^{K} \log P^M(y^k|x_1^k, \ldots, x_M^k) \tag{21}$$

where $P^M(.)$ is calculated according to the model (Equation 11).

Our procedure consists of removing one variable and by calculating the conditional log-likelihood with that variable removed. We remove that variable for which the conditional log-likelihood decreased the least. We selected the housing price as the variable to be predicted. Figure 8 shows the negative average conditional log-likelihood for the validation set and the test set as a function of the number of (input) variables. At approximately three variables, the conditional likelihood is optimal. We can conclude that for the prediction of the housing price the information in the removed variables is either redundant or already contained in these three variables. Table 7 shows the order in which variables are pruned.

From our model (with three inputs and one output) we can now predict the housing price using Equation 12.

Table 7. The order of removal of variables.

no:	1	2	3	4	5	6	7	8	9	10	11	12	13
variable:	chas	rad	dis	crim	indus	nox	age	tax	p/t	zn	b	rm	lstat

6.3. Experiments with Rule-based Bias

In our second set of experiments we compared the various approaches for combining prior knowledge and learning from data. We designed our own "naive" rule base. The rule base consists of three rules. In rule 1 we tried to capture the concept of a normal neighborhood. Here we set the centers to the mean of the features, i.e., 0. In the second rule we tried to capture the properties of a wealthy neighborhood. To indicate a high value for feature x_j we set the center in that dimension to the standard deviations of the features $+std_j$, i.e., 1, and to indicate a low value we set the corresponding value to $-std_j$, i.e., -1. Our third rule captures the properties of a poor neighborhood which contains conclusions opposite to rule 2. The scaling parameters of all rules are set to $+std_j$. Table 8 summarizes the three rules. The network generated out of these rules is the expert network.

Table 8. Prior rules. The scaling parameters are always equal to one. A plus ($+$) indicates that the Gaussian of the conclusion is centered at $+std_j$, a minus ($-$) indicates that the conclusion is centered at $-std_j$ and a zero (0) indicates that the conclusion is centered at 0. The prior probabilities of each rule i is set to $P(s = i) = 1/3$.

j :	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$i = 1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$i = 2$	-	+	-	+	-	+	-	+	-	+	-	-	-	+
$i = 3$	+	-	+	-	+	-	+	-	+	-	+	+	+	-

In the first experiment, we trained a second network (the data network) to model the data using EM. We trained the data network using a varying number of training data randomly drawn out of the training data set. If more than 10 samples were used for training, the network consisted of ten units. If less than ten samples were available, the network consisted of as many units as samples. The dotted line in Figure 9 (top) shows the negative average log-likelihood as a function of the number of training data.

In the second experiment we investigated the *incremental mixture density approach* of Section 5.1. We set $K_{prior} = 50$ which indicates that our prior knowledge is worth 50 data points. We used the data network of the previous experiment in combination with the expert network as described in Section 5.1. The dashed line in Figure 9 (top) shows the negative average log-likelihood as a function of the number of training data.

In the third experiment we studied the Bayesian approach. We designed a network in which each rule in Table 8 is represented 3 times (so we obtain 9 rules). In this way we give the network sufficient resources to form a good model (with 9 instead of 3 units). We then fine-tuned the rules using EM update rules to find the MAP weights (Section 5.2). The

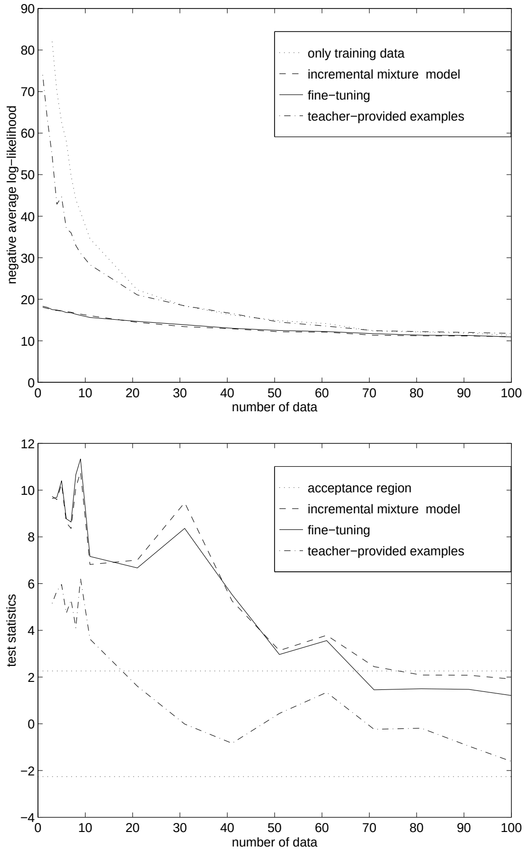

Figure 9. Top: The graph shows the negative average log-likelihood of the test set as a function of the number of training samples. Displayed are averages over ten experiments with different separations into training set, validation set, and test set. The dotted line shows the performance of a network trained only with training data. The dashed line shows the performance of the network using the incremental mixture density model (Section 5.1), the continuous line shows the performance of the fine-tuned network (Section 5.2), and the dotted-dashes line shows the performance of a network trained with a mixture of training data and data supplied by the domain expert (Section 5.3). Bottom: Shown are the test statistics for matched pairs between the network trained only on data and the approaches using prior knowledge (two-tailed paired t-test, Mendenhall & Sincich, 1992) based on the performance on the ten test sets. The null hypothesis is that there is no difference in performance between the network trained only on data and the approaches using prior knowledge. Outside of the dotted region, the null hypothesis is rejected (based on a 95% confidence interval). The incremental mixture density model and the fine-tuned network are significantly better than the network trained only on data up to approximately 70 training samples. The network trained with a mixture of training data and data supplied by the domain expert is only significantly better up to approximately 19 training samples.

continuous line in Figure 9 (top) shows the negative average log-likelihood of the fine-tuned network as a function of the number of training data.

In the final experiment we used *teacher-provided examples*, Section 5.3, by generating 100 samples according to the network defined by the expert following the probabilistic model. These data were supplemented by real training data and a network of 10 units was trained using EM. The dash-dotted line in Figure 9 (top) shows the negative average log-likelihood as a function of the number of training data.

Figure 9 (bottom) shows the test statistics for the two-tailed paired t-test to decide if including prior knowledge is helpful. Outside of the region defined by the two dotted lines, the methods including prior knowledge are significantly better than the network trained only on data. The results indicate clearly that with only a small number of training data available, prior knowledge can be very beneficial. The Bayesian approach and the incremental mixture density approach consistently outperforms the network which was trained solely on data up to a training set of up to approximately 70 samples. This indicates that the network structures defined by the rules are appropriate for this problem. The approach using teacher-provided examples is only better in comparison to the network which was trained solely on data up to a training set size of up to approximately 19 samples. The reason for the relatively bad performance of the latter approach is that only 100 artificial samples were generated and these cover only a small region of interest in the input space. To obtain better performance, many more teacher-supplied examples need to be used.

7. Modifications and Related Work

7.1. Mixtures of Experts

There is a strong connection of our approach with the mixtures of experts networks and their variations (Hampshire & Waibel, 1989; Jacobs *et al.*, 1991; Jordan & Jacobs, 1993). The output of a mixtures of experts network is

$$y(x) = \sum_i g_i(x)\ o_i(x), \tag{22}$$

where $o_i(x)$ is the output of the ith expert network, typically a feedforward neural network. $g_i(x) = P(i|x)$, the ith output of the gating network, stands for the probability of choosing expert i given the input x. The similarity to our approach becomes apparent if we identify g_i as n_i and o_i as w_i in Equation 1. In this interpretation each component in our model (i.e., Equation 1) is a — pretty boring — expert who always concludes w_i. On a further note, our incremental mixture model resembles the hierarchies of experts networks of Jordan and Jacobs (1993). The output of that model is

$$y(x) = \sum_m [g_m(x) \sum_i g_{im}(x)\ o_{im}(x)].$$

The relationship to Equation 14 is apparent. The main difference between the mixture of experts model and our approach is that we model the joint distribution of all variables whereas the mixture of experts model models the conditional distribution of the output variable given the input variables.

In this context we can interpret the learning rules in Appendix A to be a way of training hierarchical mixtures of experts using EM where both the E-step and the M-step can be calculated in closed form.

7.2. Discrete Variables

So far we only considered continuous features and Gaussian densities. As pointed out by Ghahramani and Jordan (1993) the mixture formalism as well as the efficient EM update algorithm extends readily to any component density from the exponential family. In particular for discrete features, binomial and multinomial distributions are more appropriate than Gaussian densities. For more detail, see Ghahramani and Jordan (1993) and Bernardo and Smith (1993).

7.3. Supervised Training of the Network

In many applications it is known a priori which variables are the input variables and which variable is the output variable and instead of training the model to predict the distribution of the joint input/output space using a mixture of Gaussian model we can directly train it to predict the conditional expected value $E(y|x)$. This can be achieved by optimizing all network parameters to minimize the mean squared training error Equation 3 as indicated in Section 2 (supervised learning). It is often advantageous to initialize the network to form a probabilistic model of the joint density and only perform supervised learning as post-processing: The probabilistic model gives useful initial values for the parameters in supervised learning. If we adapt all network parameters to minimize the prediction error, we cannot interpret $b_i(x)$ in Equation 1 as a conditional input density; we might rather think of $b_i(x)$ as the weight (or the certainty) of the conclusion w_i given the input.[8] During supervised training, the centers and output weights — unless special care is taken — wander outside of the range covered by the data and it becomes more difficult to extract meaningful rules. In Tresp, Hollatz and Ahmad (1993) rule-extraction and rule-prestructuring for networks trained with supervised learning are described.

8. Conclusions

The presented work is based on the three-way relationship between networks of NGBFs, Gaussian mixture models, and simple probabilistic rules. We discussed four aspects. First we showed how probabilistic rules can be used for describing structure between variables. If we perform inference using those rules we obtain a networks of NGBFs. Second, we showed that it is possible to extract probabilistic rules out of a networks of NGBFs which were trained on data using the EM algorithm. Third, we presented ways to optimize the network architecture, i.e., the number of Gaussian units and we presented ways to constrain the number of free parameters of the network. Finally we described several ways prior knowledge, formulated in probabilistic rules, can be combined with learning from data. The experiments show that with only few or no training data available prior knowledge can be used efficiently by the proposed methods. In particular, the *incremental mixture density approach* and the Bayesian approach gave good results. One of the main advantages of our approach is that it is based on probabilistic models. This allows us to obtain insight into the structure of the data by being able to extract probabilistically correct rules. Also, in our approach the joint probability distribution of all variables involved are modeled which provides much more information than is available in standard supervised learning. For example, we can handle missing data very elegantly and also can produce inverse models without any difficulty, which is not possible in networks trained using supervised learning algorithms.

Acknowledgments

Our special thanks go to Jude Shavlik for his help with this manuscript. The comments of two anonymous reviewers were very valuable. We acknowledge helpful discussions with Michael Jordan, Zoubin Ghahramani, Reimar Hofmann, Dirk Ormoneit, and Ralph Neuneier.

Appendix A

Learning Rules for the Hierarchical Model

We derive learning rules for the hierarchical model. We assume that we have K training data $\{x^k\}_{k=1}^K$. We consider the incomplete data case in which neither s^k or s^{*k} are known. In this case the log-likelihood function is

$$L = \sum_{k=1}^{K} \log[\sum_{l=1}^{C} \hat{P}(s=l) \sum_{m=1}^{N} \hat{P}(s^*=m|s=l) G(x^k; \hat{c}_m, \hat{\sigma}_m)].$$

The EM algorithm consists of the repeated application of the E-step and the M-step. In the E-step, we estimate the states of the missing variables using our current parameter estimates. More precisely, the E-step estimates the probability that x^k was generated by component $s^* = j$. Assuming that $j \in I(i)$

$$\hat{P}(s^*=j|x^k) = \frac{\hat{P}(s=i)\ \hat{P}(s^*=j|s=i)\ G(x^k; \hat{c}_j, \hat{\sigma}_j)}{\sum_{l=1}^{C} \hat{P}(s=l)\ \sum_{m=1}^{N} \hat{P}(s^*=m|s=l) G(x^k; \hat{c}_m, \hat{\sigma}_m)}.$$

Note that for complete patterns, $\hat{P}(s^*=j|x^k)$ is equal to one if $s^{*k}=j$ and is equal to zero otherwise.

The M-step updates the parameter estimates based on the estimate $\hat{P}(s^*=j|x^k)$

$$\hat{P}(s=i) \ = \ \frac{1}{K} \sum_{k=1}^{K} \sum_{j \in I(i)} \hat{P}(s^*=j|x^k), \tag{A.1}$$

$$\hat{P}(s^* = j|s = i) = \frac{1}{K}\sum_{k=1}^{K} \frac{\hat{P}(s^* = j|x^k)}{\sum_{m \in I(i)} \hat{P}(s^* = m|x^k)}, \quad (\forall j \in I(i)), \tag{A.2}$$

$$\hat{c}_{jl} = \frac{\sum_{k=1}^{K} \hat{P}(s^* = j|x^k)\, x_l^k}{\sum_{k=1}^{K} \hat{P}(s^* = j|x^k)}, \tag{A.3}$$

$$\hat{\sigma}_{jl}^2 = \frac{\sum_{k=1}^{K} \hat{P}(s^* = j|x^k)\, (\hat{c}_{jl} - x_l^k)^2}{\sum_{k=1}^{K} \hat{P}(s^* = j|x^k)}. \tag{A.4}$$

Appendix B

Boston housing data

Table B.1 describes the features in the Boston housing data set.

Table B.1. The variables and their abbreviations.

1	crime rate	crim
2	percent land zoned for lots	zn
3	percent nonretail business	indus
4	1 if on Charles river, 0 otherwise	chas
5	nitrogen oxide concentration, pphm	nox
6	average number of rooms	rm
7	percent built before 1940	age
8	weighted distance to employment center	dis
9	accessibility to radial highways	rad
10	tax rate	tax
11	pupil/teacher ratio	p/t
12	percent black	b
13	percent lower-status population	lstat
14	median value of homes in thousands of dollars	mv

Notes

1. The Gaussian basis function weights κ_i were not used by Moody and Darken.
2. Gaussian mixtures are introduced in Sections 3 and 4.
3. Note that the first level of hierarchy has the flavor of a disjunction of the form: IF $s = i$ THEN $s^* = j_1$, or $s^* = j_2, \ldots$ (with the appropriate probabilities). Since the second level implements a conjunction we obtain disjunctions of conjunctions.
4. We use $E()$ to indicate the expected value.
5. Often a subset of the components of x is considered to describe input variables and the remaining components are output variables. This result indicates that "inverse" models in which one of the input variable is estimated from knowledge about the states of output variables and other input variables can be calculated as easily as forward models.

6. The likelihood of a sample of K observations is the joint probability density function of the observations given the model and model parameters. The maximum likelihood parameter estimator is the set of parameters which maximize the likelihood. Since the logarithm is a monotonic function we can alternatively maximize the log-likelihood which is in general computationally simpler. Note, that the hat ($\hat{}$) indicates an estimated quantity.
7. Bayesian approaches to model selection are used in the AutoClass system by Cheeseman *et al.* (1988).
8. Under certain restrictions, fuzzy inference systems can be mapped onto a network of normalized basis functions as described by Wang and Mendel (1992) and Hollatz (1993).

References

Ahmad, S., & Tresp, V. (1993). Some solutions to the missing feature problem in vision. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Neural Information Processing Systems 5,* San Mateo, CA: Morgan Kaufmann.

Bernardo, J. M., & Smith, A. F. M. (1993). *Bayesian Theory.* New York: J. Wiley & Sons.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford: Clarendon Press.

Buntine, W. L., & Weigend, A. S. (1991). Bayesian back-propagation. *Complex Systems, 5,* 605-643.

Buntine, W. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research, 2,* 159-225.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). AutoClass: A Bayesian classification system. *Proceedings of the Fifth International Workshop on Machine Learning (pp. 54–64).* Ann Arbor, MI: Morgan Kaufmann.

Cover, T. M., & Thomas, J. A. (1991). *Elements of Information Theory.* New York: J. Wiley & Sons.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data vie the EM algorithm. *J. Royal Statistical Society Series B, 39,* 1-38.

Duda, R. O., & Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* New York: J. Wiley and Sons.

Fu, L. M. (1989). Integration of neural heuristics into knowledge-based inference. *Connection Science, 1,* 325-340.

Fu, L. M. (1991). Rule learning by searching on adapted nets. *Proceedings of the National Conference on Artificial Intelligence (pp. 590-595).* Anaheim, CA: Morgan Kaufmann.

Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM, 31,* 152-169.

Ghahramani, Z., & Jordan, M. I. (1993). *Function approximation via density estimation using an EM approach* (Technical Report 9304). Cambridge, MA: MIT Computational Cognitive Sciences.

Giles, C. L., & Omlin, C. W. (1992). Inserting rules into recurrent neural networks. In S. Kung, F. Fallside, J. A. Sorenson, & C. Kamm (Eds.), *Neural Networks for Signal Processing 2,* Piscataway: IEEE Press.

Hampshire, J., & Waibel, A. (1989). *The meta-pi network: building distributed knowledge representations for robust pattern recognition* (Technical Report CMU-CS-89-166). Pittsburgh, PA: Carnegie Mellon University.

Harrison, D., & Rubinfeld, D. L. (1978). Hedonic prices and the demand for clean air. *J. Envir. Econ. and Management, 5,* 81-102.

Heckerman, D. (1995). *A tutorial on learning Bayesian networks* (Technical Report MSR-TR-95-06). Redmond, WA: Microsoft Research.

Heckerman, D., Geiger, D., & Chickering, D. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning, 20,* 197-243.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation.* Redwood City, CA: Addison-Wesley.

Hofmann, R., & Tresp, V. (1996). Discovering structure in continuous variables using Bayesian networks. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Neural Information Processing Systems 8,* Cambridge, MA: MIT Press.

Hollatz, J. (1993). *Integration von regelbasiertem Wissen in neuronale Netze.* Doctoral dissertation, Technische Universität, München.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, J. E. (1991). Adaptive mixtures of local experts. *Neural Computation, 3,* 79-87.

Jordan, M., & Jacobs, R. (1993). *Hierarchical mixtures of experts and the EM algorithm* (Technical Report TR 9302). Cambridge, MA: MIT Computational Cognitive Sciences.

MacKay, J. C. (1992). Bayesian model comparison and backprop nets. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Neural Information Processing Systems 4,* San Mateo, CA: Morgan Kaufmann.

Mendenhall, W., & Sincich, T. (1992). *Statistics for Engineering and the Sciences.* San Francisco, CA: Dellen Publishing Company.

Moody, J. E., & Darken, C. (1989). Fast learning in networks of locally-tuned processing units, *Neural Computation, 1,* 281-294.

Nowlan, S. J. (1990). Maximum likelihood competitive learning. In D. S. Touretzky (Ed.), *Neural Information Processing Systems 2,* San Mateo, CA: Morgan Kaufmann.

Nowlan, S. J. (1991). *Soft competitive adaptation: Neural network learning algorithms based on fitting statistical mixtures.* Doctoral dissertation, Pittsburgh, PA: Carnegie Mellon University.

Ormoneit, D., & Tresp, V. (1996) Improved Gaussian mixture density estimates using Bayesian penalty terms and network averaging. In D. S. Touretzky, M. C. Mozer, & M. E. Hasselmo (Eds.), *Neural Information Processing Systems 8,* Cambridge, MA: MIT Press.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* San Mateo, CA: Morgan Kaufmann.

Poggio, T., & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE, 78,* 1481-1497.

Röscheisen, M., Hofmann, R., & Tresp, V. (1992). Neural control for rolling mills: Incorporating domain theories to overcome data deficiency. In J. E. Moody, J. E. Hanson, & R. P. Lippmann (Eds.), *Neural Information Processing Systems 4,* San Mateo, CA: Morgan Kaufmann.

Rumelhart, D. E., & McClelland, J. L. (1986). *Parallel Distributed Processing: Exploration in the Microstructures of Cognition* (Vol. 1), Cambridge, MA: MIT Press.

Sen, A., & Srivastava, M. (1990). *Regression Analysis.* New York: Springer Verlag.

Shavlik, J. W., & Towell, G. G. (1989). An approach to combining explanation-based and neural learning algorithms. *Connection Science, 1,* 233-255.

Smyth, P. (1994). Probability density estimation and local basis function neural networks. In Hanson, S., Petsche, T, Kearns, M., & Rivest, R. (Eds.), *Computational Learning Theory and Natural Learning Systems,* Cambridge, MA: MIT Press.

Specht, D. F. (1990). Probabilistic neural networks. *Neural Networks, 3,* 109-117.

Specht, D. F. (1991). A general regression neural network. *IEEE Transactions on Neural Networks, 2,* 568-576.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning, 8,* 257 - 278.

Thrun, S. (1995). Extracting rules from artificial neural networks with distributed representations. In G. Tesauro, D. S. Touretzky, & T. K. Leen (Eds.), *Neural Information Processing Systems 7,* MIT Press, Cambridge, MA.

Towell, G. G., & Shavlik, J. W. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning, 13*, 71-101.

Towell, G. G., & Shavlik, J. W. (1994). Knowledge-based neural networks. *Artificial Intelligence, 70,* 119-165.

Tresp, V., Hollatz J., & Ahmad, S. (1993). Network structuring and training using rule-based knowledge. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Neural Information Processing Systems 5,* San Mateo, CA: Morgan Kaufmann.

Tresp, V., Ahmad, S., & Neuneier, R. (1994). Training neural networks with deficient data. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Neural Information Processing Systems 6,* San Mateo, CA: Morgan Kaufmann.

Wang, L.-X., & Mendel, J. M. (1992). Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE Transactions on Neural Networks, 3,* 807-814.

Wettscherek, D., & Dietterich, T. (1992). Improving the performance of radial basis function networks by learning center locations. In J. E. Moody, S. J. Hanson, & R. P. Lippmann (Eds.), *Neural Information Processing Systems 4,* San Mateo, CA: Morgan Kaufmann.

Received December 11, 1993
Accepted December 6, 1994
Final Manuscript November 14, 1996