# A new algorithm for training RBF networks

Aldebaro Klautau

*Abstract*— **The radial basis function (RBF) network is the main practical alternative to the multi-layer perceptron for non-linear modeling. This work presents a new discriminative algorithm for the first training-stage of classifiers consisting of RBF networks with diagonal covariance Gaussians as basis functions. The experimental results show that the algorithm leads to improved performance when compared to the conventional expectation-maximization algorithm.**

*Keywords*— **Pattern recognition, radial basis function, discriminative training.**

## I. Introduction

The RBF network is the main practical alternative to the multi-layer perceptron (MLP) for non-linear modeling [1]. One attraction of RBF networks is that there is a two-stage training procedure that is considerably faster than the methods used to train MLPs.

Typically, the basis functions are given in terms of a *radial distance* $||\mathbf{x}-\mathbf{x}'||$. The most popular choice is the Gaussian function, which is assumed throughout this work. The output of the network is then given by

$$z_r(\mathbf{x}) = \sum_{g=1}^{G} w_{rg}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) + w_{r0},$$

where $w_{rg}$ are the output layer weights and $\mathcal{N}(\cdot|\boldsymbol{\mu},\boldsymbol{\Sigma})$ represents a Gaussian.

In the first stage, the parameters of the basis functions are set so that they model the *evidence* $P(\mathbf{x})$. The second stage of training determines the weights in the output layer, and this is a quadratic optimization problem, which can be solved efficiently using methods from linear algebra [1].

In the first stage of training, one needs to determine the set of parameters $\Theta$ associated to the Gaussians (means and variances). The conventional way of estimating $\Theta$ is through maximum likelihood estimation (MLE). More specifically, RBFs are typically trained using the expectation-maximization (EM) algorithm [2] in the first stage of learning. Therefore, the first stage of training an RBF network is almost identical to learning a Gaussian mixture model (GMM) classifier. We note that alternatives to the EM-based initialization have been proposed in the literature, e.g., [3].

In this paper, we present a new algorithm for training RBF networks, which increases the accuracy at the expense of a higher computational cost. The algorithm assumes a conventional classification scenario and that the Gaussians have diagonal covariance matrices.

The paper is organized as follows. In Section II we discuss two learning techniques and establish the notation. In Section III we present the discriminative EM algorithm. Section IV presents the simulations results and is followed by our conclusions.

The author is with the ECE Department, UCSD, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: a.klautau@ieee.org.

A. Klautau was supported by CAPES, Brazil.

## II. Generative and discriminative learning

In this section we make a connection between the first stage of training an RBF network and training Bayes classifiers. For more details, see, e.g., [1].

We assume a conventional classification problem, where one is given a *training set* $\{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_N,y_N)\}$ containing $N$ *examples*, which are independently and identically distributed (iid) samples from an unknown but fixed distribution $P(\mathbf{x},y)$. Each example $(\mathbf{x},y)$ consists of an instance $\mathbf{x}\in\mathcal{X}$ and a label $y\in\{1,\ldots,Y\}$. The input $\mathbf{x}$ is a vector of dimension $L$. A *classifier* is a mapping $F:\mathcal{X}\rightarrow\{1,\ldots,Y\}$.

Throughout this work, we adopt the nomenclature used in [4], where[1] $P(y|\mathbf{x})$, $P(\mathbf{x}|y)$, $P(y)$ and $P(\mathbf{x})$ are called *posterior*, *likelihood*, *prior* and *evidence*, respectively, and are related through Bayes' rule

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \quad (1)$$

Bayes classifiers implement

$$F = \arg\max_{y=1,\ldots,Y} \hat{P}(\mathbf{x}|y)\hat{P}(y).$$

Training a Bayes classifier consists in estimating the parameters $\Theta$ of all its likelihood functions $\hat{P}(\mathbf{x}|y)$.

The conventional way of estimating $\Theta$ for both RBF and Bayes classifiers is through maximum likelihood estimation (MLE). MLE-based classifiers seek $\Theta^g = \arg\max_\Theta R^g(\Theta)$, where

$$R^g(\Theta) = \prod_{n=1}^{N} \hat{P}(\mathbf{x}_n|y_n).$$

Such classifiers are called *generative* [5] or *informative* [6].

The term generative is used because if the estimated $\hat{P}(\mathbf{x},y)$ is "close" to the true distribution $P(\mathbf{x},y)$, we could use $\hat{P}(\mathbf{x},y)$ to generate samples with statistics similar to the ones of our original training set. However, for the sake of classification, we do not need to keep $\Theta$. In such cases, the term informative seems more appropriate.

By contrast, discriminative Bayes classifiers (and RBF networks trained with the method proposed in this work) seek $\Theta^d = \arg\max_\Theta R^d(\Theta)$, where

$$R^d(\Theta) = \hat{P}(y|\mathbf{x}).$$

[1] We use $P$ to denote both probability mass functions and densities.

Note that

$$R^d(\Theta) = \prod_{n=1}^{N} \frac{\hat{P}(\mathbf{x}_n|y_n)\hat{P}(y_n)}{\hat{P}(\mathbf{x}_n)} = \prod_{n=1}^{N} \left(1 + \frac{\sum_{j\neq y_n} \hat{P}(\mathbf{x}_n|j)\hat{P}(j)}{\hat{P}(\mathbf{x}_n|y_n)\hat{P}(y_n)}\right)^{-1}.$$

It follows that discriminative procedures try not only to maximize the likelihood of examples $(\mathbf{x}, y)$, but, at the same time, minimize the likelihood of competing classes $j \neq y$.

Discriminative is often harder than generative training. There are no closed-form solutions and iterative optimization algorithms are needed. Roughly speaking, if the modeling assumptions are correct, adopting a generative classifier is more appropriate [7], [6], [5].

The key idea of this paper is that, similarly to Bayes classifiers, we can improve the performance of RBF networks using a discriminative leaning technique in the first stage of training. The next section presents such technique, which we call discriminative expectation-maximization (DEM) algorithm.

## III. The discriminative EM algorithm

The next subsections briefly describe DEM training. More information can be found at [8].

### A. EM applied to mixture of Gaussians with diagonal covariances

We now review how EM is used for learning, for each class, a mixture of Gaussians with diagonal covariance matrices as the likelihood model. Hence, we want to learn the parameters $\Theta = \{\mu_{jgl}, \sigma_{jgl}, w_{jg}\}$, where $|\Theta| = 2M(L+1)$. The indices $(j, g, l)$ specify the class, the mixture component (Gaussian) and the dimension, respectively. We model each class without taking in account the examples from other classes. Hence, to simplify the notation, we drop the index $j$ along this initial discussion.

To understand the application of EM,[2] we first identify that the "missing information" in this problem is the index of the Gaussian that should be associated to a given example $\mathbf{x}_n$. Let us assume that some *oracle* had given us such information as a "hard" assignment of examples to one among the $G$ Gaussians of a mixture. This assignment can be represented through an indicator function $I(\mathcal{G}_n = g)$, $g = 1, \ldots, G$, which is one for the Gaussian with the "correct" index $\mathcal{G}_n$ provided by the oracle, and 0 for other Gaussians. For example, when learning a mixture with $G = 3$ Gaussians, the indicator function would return $(0, 1, 0)$ when an example $\mathbf{x}_n$ should be assigned to the Gaussian with index $g = 2$. Having this hard assignment for all training examples, the maximization step (M-step) of EM obtains a new Gaussian simply taking the sample mean and sample covariance of all examples associated to that Gaussian by the oracle.

[2]Here we are mainly interested on defining the notation that is needed to describe the discriminative EM. For a very readable discussion of EM as applied to GMM see, e.g., [9]. For a more rigorous and complete treatment of EM see, e.g., [10].

The expectation step (E-step) of EM plays the role of the oracle but, instead of the hard assignment, it provides a probabilistic "soft" assignment $P(g|\mathbf{x}_n)$ of examples to all $G$ states (i.e., a discrete distribution over states for each $\mathbf{x}_n$). For example, assuming $G = 3$, we could have $P(g|\mathbf{x}_n) = (0.1, 0.7, 0.2)$. This would indicate to the M-step that $\mathbf{x}_n$ should be taken in account in the reestimation of all three Gaussians, but influencing more strongly the second one. This contrasts with the previous example of a hard assignment, where each example is associated to only one Gaussian. More specifically, for each example, the E-step calculates

$$P(g|\mathbf{x}_n) = \frac{\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_g, \boldsymbol{\Sigma}_g)w_g}{\sum_{g'=1}^{G} \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_{g'}, \boldsymbol{\Sigma}_{g'})w_{g'}}.$$

The distribution $P(g|\mathbf{x}_n)$ is all that we need to apply the reestimation equations associated to the maximization step (M-step) of EM. The new values for the mixture weights, means and variances can be written as

$$\tilde{w}_g = \frac{\mathcal{C}(1)_g}{\sum_{g'=1}^{G} \mathcal{C}(1)_{g'}},$$

$$\tilde{\mu}_{gl} = \frac{\mathcal{C}(x)_{gl}}{\mathcal{C}(1)_g}$$

and

$$\tilde{\sigma}_{gl} = \frac{\mathcal{C}(x^2)_{gl}}{\mathcal{C}(1)_g} - \tilde{\mu}_{gl}^2,$$

where the *occupation counts* $\mathcal{C}(x)_{gl}$, $\mathcal{C}(x^2)_{gl}$ and $\mathcal{C}(1)_g$, which are statistics for calculating the new set of parameters, are given by

$$\mathcal{C}(x)_{gl} = \sum_{n'} P(g|\mathbf{x}_n)\mathbf{x}_n(l),$$

$$\mathcal{C}(x^2)_{gl} = \sum_{n'} P(g|\mathbf{x}_n)\mathbf{x}_n^2(l)$$

and

$$\mathcal{C}(1)_g = \sum_{n'} P(g|\mathbf{x}_n).$$

In these equations we used $\mathbf{x}(l)$ to indicate the $l$-th component of vector $\mathbf{x}$, and the summation over $n'$ includes only the examples of the specific class in consideration. The algorithm can then be efficiently implemented, as shown in Figure 1.

Note that the number of computations in each epoch is $\mathcal{O}(N)$, i.e., increases linearly with the number of training examples, and the memory requirements is very small: we only need space for the set $\Lambda = \{\mathcal{C}(x)_{jgl}, \mathcal{C}(x^2)_{jgl}, \mathcal{C}(1)_{jg}\}$ of counts, where $|\Lambda| = |\Theta|$.

### B. DEM applied to mixture of Gaussians with diagonal covariances

The main purpose of the previous section was to establish the notation for the DEM algorithm, which is shown in Figure 2. The E-step of DEM is similar to the one in

1) Initialization
• Pick the initial parameters $\overline{\Theta}$ and set the iteration counter $i = 1$
2) Do:
• Set the total log-likelihood $\log R^g(i) = 0$, reset all counts
• E-step: for $n = 1, \ldots, N$ (one epoch)
– For all Gaussians of mixture $j = y_n$, calculate $\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_{jg}, \boldsymbol{\Sigma}_{jg})$
– Calculate
$P(g|\mathbf{x}_n, y_n = j) = \frac{\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_{jg}, \boldsymbol{\Sigma}_{jg})w_{jg}}{\sum_{g'=1}^{G_j}\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_{jg'}, \boldsymbol{\Sigma}_{jg'})w_{jg'}}$, $\forall g$ of class $j$
– Update $\log R^g(i) \leftarrow \log R^g(i) + \log P_{\overline{\Theta}}(\mathbf{x}_n|y_n)$
– Accumulate counts:
$\mathcal{C}(x)_{jgl} \leftarrow \mathcal{C}(x)_{jgl} + P(g|\mathbf{x}_n, y_n = j)\mathbf{x}(l)$
$\mathcal{C}(x^2)_{jgl} \leftarrow \mathcal{C}(x^2)_{jgl} + P(g|\mathbf{x}_n, y_n = j)\mathbf{x}^2(l)$
$\mathcal{C}(1)_{jg} \leftarrow \mathcal{C}(1)_{jg} + P(g|\mathbf{x}_n, y_n = j)$
• M-step: calculate a new set of parameters
– Mixture weight $\tilde{w}_{jg} = \frac{\mathcal{C}(1)_{jg}}{\sum_{g'=1}^{G}\mathcal{C}(1)_{jg'}}$
– Mean element $\tilde{\mu}_{jgl} = \frac{\mathcal{C}(x)_{jgl}}{\mathcal{C}(1)_{jg}}$
– Covariance element $\tilde{\sigma}^2_{jgl} = \frac{\mathcal{C}(x^2)_{jgl}}{\mathcal{C}(1)_{jg}} - \tilde{\mu}^2_{jgl}$
• Update for next iteration $\overline{\Theta} \leftarrow \tilde{\Theta}$ and increment $i$
Until convergence based on $\log R^g(i)$ and / or maximum number of iterations
3) Calculate (optional) the total log-likelihood $\log R^g(i+1) = \sum_{n=1}^{N}\log P_{\overline{\Theta}}(\mathbf{x}_n|y_n)$ of the final model

Fig. 1. EM algorithm as applied to learning a mixture of Gaussians for each class.

the EM algorithm, and the M-step uses different equations based in the procedure described in [11] (see [8]).

When implementing the DEM algorithm, it is convenient to create for each iteration, a model that would output the evidence $P(\mathbf{x})$ of Equation (1) (the denominator of $R^d$). In other words, we deal with the marginal distribution

$$P(\mathbf{x}) = \sum_{y=1}^{Y} P(\mathbf{x}|y)P(y)$$

as one mixture with all $G = \sum_{y=1}^{Y} G_y$ Gaussians, namely

$$P(\mathbf{x}) = \sum_{y=1}^{Y}\sum_{g=1}^{G_y}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{yg}, \boldsymbol{\Sigma}_{yg})w'_{yg},$$

where $w'_{yg} = P(y)w_{yg}$ is the original weight scaled by the associated prior. We conceptually treat this mixture as modeling a fictitious class associated to the denominator of $R^d$, and call it *denominator* model. This fiction is convenient when writing the equations and also when implementing the code, because it allows using methods created for dealing with mixtures.

The DEM algorithm uses two sets of occupation counts:

$$\Lambda^{\mathtt{num}} = \{\mathcal{C}(x)^{\mathtt{num}}_{jgl}, \mathcal{C}(x^2)^{\mathtt{num}}_{jgl}, \mathcal{C}(1)^{\mathtt{num}}_{jg}\}$$

1) Initialization
• Pick the initial parameters $\overline{\Theta}$ and set the iteration counter $i = 1$. Calculate the influence of the prior probabilities $p = \sum_{n=1}^{N}\log P(y_n)$
2) Do:
• Set the total log-posterior $\log R^d(i) = p$, reset all counts
• E-step: for $n = 1, \ldots, N$ (one epoch)
– For all Gaussians $j = 1, \ldots, G$, calculate $\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_{jg}, \boldsymbol{\Sigma}_{jg})$
– Calculate $P(g|\mathbf{x}_n, y_n)$, $\forall g$ of class $y_n$
– Calculate $P(g|\mathbf{x}_n)$, $\forall g$ (i.e., for the Gaussians of the denominator model)
– Update
$\log R^d(i) \leftarrow \log R^d(i) + \log P_{\overline{\Theta}}(\mathbf{x}_n|y_n) - \log P_{\overline{\Theta}}(\mathbf{x}_n)$
– Accumulate counts for the correct class $j = y_n$ (numerator): $\mathcal{C}(x)^{\mathtt{num}}_{jgl}$, $\mathcal{C}(x^2)^{\mathtt{num}}_{jgl}$, $\mathcal{C}(1)^{\mathtt{num}}_{jg}$, and for the denominator $j = 1, \ldots, G$: $\mathcal{C}(x)^{\mathtt{den}}_{jgl}$, $\mathcal{C}(x^2)^{\mathtt{den}}_{jgl}$, $\mathcal{C}(1)^{\mathtt{den}}_{jg}$
• M-step: calculate a new set of parameters

$$\tilde{w}_{jg} = \frac{\mathcal{C}(1)^{\mathtt{num}}_{jg} - \mathcal{C}(1)^{\mathtt{den}}_{jg} + D\overline{w}_{jg}}{\sum_{g'=1}^{G_j}\left(\mathcal{C}(1)^{\mathtt{num}}_{jg'} - \mathcal{C}(1)^{\mathtt{den}}_{jg'}\right) + D}$$

$$\tilde{\mu}_{jgl} = \frac{\mathcal{C}(x)^{\mathtt{num}}_{jgl} - \mathcal{C}(x)^{\mathtt{den}}_{jgl} + D\overline{\mu}_{jgl}}{\mathcal{C}(1)^{\mathtt{num}}_{jg} - \mathcal{C}(1)^{\mathtt{den}}_{jg} + D}$$

$$\tilde{\sigma}^2_{jgl} = \frac{\mathcal{C}(x^2)^{\mathtt{num}}_{jgl} - \mathcal{C}(x^2)^{\mathtt{den}}_{jgl} + D(\overline{\sigma}^2_{jgl} + \overline{\mu}^2_{jgl})}{\mathcal{C}(1)^{\mathtt{num}}_{jg} - \mathcal{C}(1)^{\mathtt{den}}_{jg} + D} - \tilde{\mu}^2_{jgl}$$

• Update for next iteration $\overline{\Theta} \leftarrow \tilde{\Theta}$ and increment $i$
Until convergence based on $\log R^d(i)$ and / or maximum number of iterations
3) Calculate (optional) the total log-posterior $\log R^d(i+1) = \sum_{n=1}^{N}\log P_{\overline{\Theta}}(\mathbf{x}_n|y_n) + p - \sum_{n=1}^{N}\log P_{\overline{\Theta}}(\mathbf{x}_n)$ of the final model

Fig. 2. DEM algorithm as applied to learning a mixture of Gaussians for each class.

and

$$\Lambda^{\mathtt{den}} = \{\mathcal{C}(x)^{\mathtt{den}}_{jgl}, \mathcal{C}(x^2)^{\mathtt{den}}_{jgl}, \mathcal{C}(1)^{\mathtt{den}}_{jg}\},$$

associated to the numerator and denominator of $R^d$, respectively.

Similarly to the EM algorithm, in the E-step of DEM, each example is used to update the counts in $\Lambda^{\mathtt{num}}$ that correspond to the correct class $y_n$. However, DEM also updates the counts in $\Lambda^{\mathtt{den}}$, independent of the label $y_n$ (one can think of it as if an instance $\mathbf{x}$ belongs to both "denominator class" and $y_n$). In other words, we compute $P(g|\mathbf{x}_n)$ for the denominator model using all $G$ Gaussians and update all counts in $\Lambda^{\mathtt{den}}$, while $P(g|\mathbf{x}_n, y_n = j)$ (associated to the numerator) is computed using only the Gaussians of the correct class $y_n$. Hence, in the end of each epoch, the counts in $\Lambda^{\mathtt{num}}$ are exactly the same as the ones that would be obtained with EM.

An example can clarify the role of $\Lambda^{\mathtt{den}}$ in the E-step of

DEM. Let us say that there are $Y = 2$ classes and each class is associated to a mixture of $G_1 = G_2 = 3$ Gaussians. We assume that, when presented to an instance $\mathbf{x}_n$ belonging to class $y_n = 1$, the iteration of the E-step calculated the distribution $(0.1, 0.7, 0.2)$ for the Gaussians of this class. This distribution is then used to update the counts in $\Lambda^{\texttt{num}}$ associated to class 1. Assume that the first three Gaussians of the mixture associated to the denominator model correspond to class 1, and the other three to class 2. If the distribution associated to the denominator is $(0.05, 0.35, 0.1, 0.5, 0, 0)$, we can infer that the first Gaussian of class 2 outputs a relatively high value when presented to $\mathbf{x}_n$. This is the kind of problem that the discriminative training procedure will try to correct.

In the maximization step, the discrete distributions associated to the mixture weights are reestimated according to

$$\tilde{w}_{jg} = \frac{\mathcal{C}(1)^{\texttt{num}}_{jg} - \mathcal{C}(1)^{\texttt{den}}_{jg} + D\overline{w}_{jg}}{\sum_{g'=1}^{G_j}\left(\mathcal{C}(1)^{\texttt{num}}_{jg'} - \mathcal{C}(1)^{\texttt{den}}_{jg'}\right) + D},$$

where $D$ is an empirical constant related to the learning rate [8].

The means and variances are reestimated according to the equations derived in [12], respectively:

$$\tilde{\mu}_{jgl} = \frac{\mathcal{C}(x)^{\texttt{num}}_{jgl} - \mathcal{C}(x)^{\texttt{den}}_{jgl} + D\overline{\mu}_{jgl}}{\mathcal{C}(1)^{\texttt{num}}_{jg} - \mathcal{C}(1)^{\texttt{den}}_{jg} + D}$$

and

$$\tilde{\sigma}^2_{jgl} = \frac{\mathcal{C}(x^2)^{\texttt{num}}_{jgl} - \mathcal{C}(x^2)^{\texttt{den}}_{jgl} + D(\overline{\sigma}^2_{jgl} + \overline{\mu}^2_{jgl})}{\mathcal{C}(1)^{\texttt{num}}_{jg} - \mathcal{C}(1)^{\texttt{den}}_{jg} + D} - \tilde{\mu}^2_{jgl}.$$

In our implementation we use a different $D_{jg}$ for each Gaussian, which depend on a learning rate $\eta$ that varies over time. If the last step resulted in an improvement of $R^d$, $\eta$ is increased. If $R^d$ decreases (a negative improvement), $\eta$ is decreased. The value of $D_{jg}$ also depends on the fact that we want the variances to be positive.

We note that DEM spends most of the CPU cycles computing the values of the $G$ Gaussians for each instance $\mathbf{x}_n$. In order to alleviate this effort, we propose using a K-d tree.

### C. Reducing the computational cost of DGMM with a K-d tree

In [13], Bentley proposed a data structure to speed up database search and called it K-d tree. The K-d tree has been used, for example, in vector quantization (VQ) to reduce the computational cost of encoding (see, e.g., [14]).

We now briefly describe the K-d tree, which is based on hyperplanes that are orthogonal to one of the axes. Consider the input space being split into two half spaces by means of a hyperplane orthogonal to one of the $L$ coordinate axes (recall that $L$ is the number of features). This hyperplane requires storing only two scalar quantities: the index $i$ of the coordinate axis and the location $h_i$ of the plane on this axis. It is possible to locate any vector point $\mathbf{x}$ with respect to this hyperplane by a single scalar comparison $x_i > h_i$ of the $i$-th element of $\mathbf{x}$. The root of the K-d tree is associated to the whole input space, and successive divisions of the subregions by hyperplanes orthogonal to the coordinate axis result into terminal regions, known as *leafs* or *buckets*. In VQ, only the codewords associated to the leaf that $\mathbf{x}$ falls in are considered for representing (quantizing) $\mathbf{x}$.

In [15], the K-d tree data structure was used to speed up the calculations related to the Gaussians in ASR systems. Instead of splitting based on Voronoi regions as in VQ, Fritsch and Rogina established a multidimensional box around each Gaussian mean, and this box plays a role similar to the Voronoi region in VQ. Assuming the covariance matrices are diagonal, the lower and upper limits, for each dimension $k$ of such *Gaussian box* can be easily calculated as follows. Assume we want to find the box associated to some value $V$, that is

$$V = \frac{1}{\sqrt{(2\pi)^L|\mathbf{\Sigma}|}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})'\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right).$$

Taking the log and rearranging we have (here, $x_k$ represents the $k$-th element of $\mathbf{x}$)

$$\sum_{k=1}^{L}\frac{(x_k-\mu_k)^2}{\sigma_k^2} = c, \tag{2}$$

where

$$c = -\ln\left[V^2(2\pi)^L\prod_{k'=1}^{L}\sigma^2_{k'}\right].$$

For each dimension $k$, the lower and upper limits $t_k$ of $\mathbf{x}$ that obey Equation (2) are obtained when $x_{k'} = \mu_{k'}, \forall k' \neq k$. From Equation (2), these limits are

$$t_k = \mu_k \pm \sigma_k\sqrt{c}.$$

An example for which $L = 2$ is shown in Figure 3 (from [16]). Having the box associated to each Gaussian, we can apply, for example, one of the algorithms proposed in [14] to build a K-d tree.

In this work, we adopted the "relative threshold" discussed in [15] to calculate the box of each Gaussian, and the GOC algorithm proposed in [14] to build the K-d tree. Before each E-step of the DEM algorithm, we build a K-d tree using all $G$ Gaussians. Then, for each instance $\mathbf{x}_n$, we assume $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_g, \mathbf{\Sigma}_g) = 0$ if Gaussian $g$ does not belong to the leaf that $\mathbf{x}_n$ falls in.

## IV. Simulation results

In this section we present experimental results comparing RBF networks trained with EM and DEM. When MLPs are applied to classification tasks, it is common practice to use logistic or softmax output activation functions and the corresponding cross-entropy error function so as to ensure that the outputs sum to one and all lie in the interval [0, 1]. This does not add significantly to the time taken to train

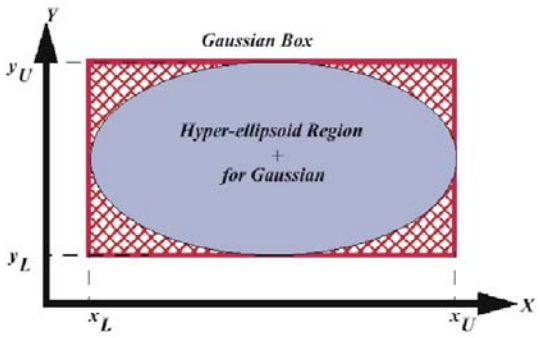


Fig. 3. Example of Gaussian box (from Srivastava's thesis), which plays the role of a Voronoi region.

TABLE I
DESCRIPTION OF THE DATASETS.

| Name | train | test | classes ($Y$) | attributes ($K$) |
|---|---|---|---|---|
| synth | 250 | 1000 | 2 | 2 |
| waveform | 400 | 4600 | 3 | 40 |
| pima | 200 | 332 | 2 | 7 |
| pbvowel | 599 | 600 | 10 | 2 |
| 0-6 (mnist) | 500 | 1938 | 2 | 256 |
| 7-9 (mnist) | 500 | 2037 | 2 | 256 |
| d-t (timit) | 500 | 300 | 2 | 118 |
| iy-ih (timit) | 500 | 446 | 2 | 118 |

an MLP since even with linear outputs, general purpose optimization routines must be used [1].

However, an RBF network with logistic or softmax outputs no longer has a quadratic error surface for the output layer. If general purpose optimization algorithms are used, much of the speed advantage over MLPs is lost [1]. For this reason, we used only linear output units in our simulations.

We used eight standard datasets listed in Table I. The datasets *pima* and *synth* were made available by B. Ripley[3]. The waveform dataset is described in [17]. The *pbvowel* dataset corresponds to a version of the Peterson and Barney's vowel data described in [18]. Two binary problems (digits 0 vs. 6 and 7 vs. 9) were extracted[4] from MNIST, which is a dataset of handwritten digits available from Y. LeCun. Two other binary problems (phones d vs. t and iy vs. ih) were were extracted from the TIMIT speech dataset[5]. We converted each occurrence of these phones into fixed-length vectors ($K = 118$) using a linear warping procedure. MNIST and TIMIT are relatively large datasets. Hence, we used only 500 examples of each of their class. The datasets synth and waveform are toy examples, for which we know the Bayes errors are 8% and 14%, respectively.

We standardized each attribute to have zero mean and unit variance. We selected the number of Gaussians using ten-fold cross-validation on the training set. These preliminary results show that DEM outperforms EM in terms of accuracy.

[3]http://www.stats.ox.ac.uk/pub/PRNN.

[4]We subsampled MNIST to obtain images with $16 \times 16$ pixels using Matlab's function imresize.

[5]http://www.ldc.upenn.edu/.

TABLE II
ERROR RATE FOR RBF NETWORKS TRAINED USING EM AND DEM. GAUSSIANS.

| Dataset | RBF / EM | RBF / DEM |
|---|---|---|
| synth | 8.1 | 8.1 |
| waveform | 16.2 | 14.2 |
| pima | 18.3 | 17.8 |
| 0-6 | 2.5 | 2.1 |
| 7-9 | 8.3 | 8.1 |
| d-t | 16.2 | 15.1 |
| iy-ih | 15.2 | 13.1 |
| pbvowel | 19.8 | 18.9 |

We note that, as other non-linear optimization approaches, running DEM is always susceptible to convergence problems. Figure 4 shows an example of the convergence of the algorithm, illustrating the behavior of the simple mechanism we used to adaptively adjust the learning rate. This is a case where the adaptation heuristic was too aggressive, and the value of $R^d$ abruptly decreased on the third iteration, after the learning rate was increased to around 0.7.

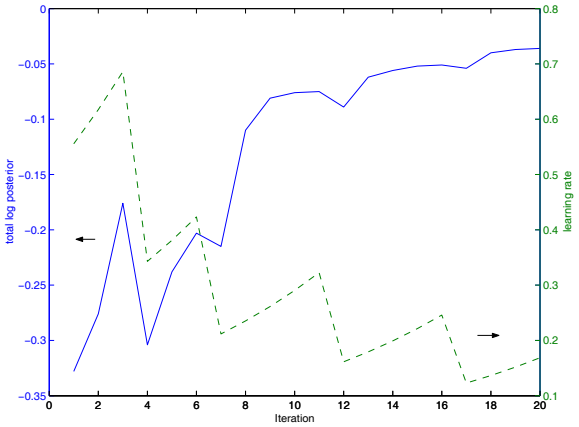


Fig. 4. Convergence of the DEM algorithm for the dataset sd-t (note that 0 is the upper-bound on $\log R^d$). The plot also shows the evolution of the learning rate.

Figure 5 shows the same simulation as in Figure 4, but we added three plots corresponding to a fixed learning rate of $\eta$=0.1, 0.01 and 0.001. It can be seen that 0.1 does not lead to convergence, while 0.001 leads to a steady but very slow convergence. When using $\eta$=0.01, the algorithm converges to almost the same value of $R^d$ as the one obtained with the adaptive heuristic. The problem is that this "reasonable" fixed value of $\eta$ is problem-dependent, i.e., requires tuning for each problem.

## V. CONCLUSIONS

We described a new algorithm for training RBF networks, which is based on the extended Baum-Welch algorithm. The experimental results indicate that the accuracy is improved when compared to an RBF trained with EM. Future works include comparing RBF trained with DEM

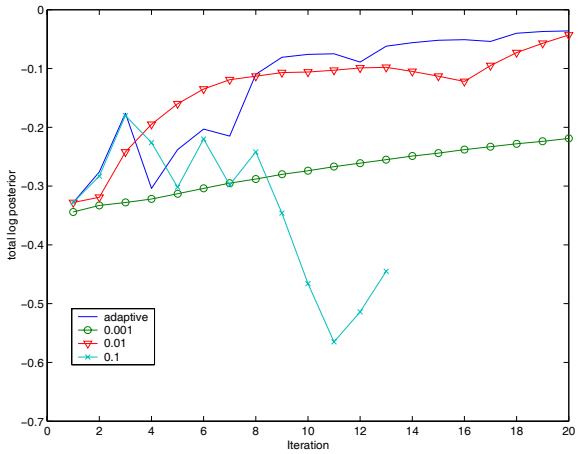


Fig. 5. Comparison of convergence with fixed and adaptive learning rates.

to MLP networks, and eliminating the restriction of using diagonal covariance matrices.

## References


[1] I. Nabney. *Netlab algorithms for pattern recognition*. Springer, 2002.

[2] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society (B)*, 39:pp. 1–22, 1977.

[3] T. Kaylani and S. Dasgupta. A new method for initializing radial basis function classifiers. In *IEEE International Conference on Systems, Man, andCybernetics. Human, Information and Technology*, volume 3, pages 2584–7, 1994.

[4] R. Duda, P. Hart, and D. Stork. *Pattern classification*. Wiley, 2001.

[5] A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2002.

[6] Y. Rubinstein and T. Hastie. Discriminative vs informative learning. In *Knowledge Discovery and Data Mining*, pages 49–53, 1997.

[7] A. Nádas, D. Nahamoo, and M. Picheny. On a model-robust training method for speech recognition. *IEEE Trans. on ASSP*, 36:1432–6, 1988.

[8] A. Klautau. *Speech Recognition Using Discriminative Classifiers*. PhD thesis, UCSD, 2003.

[9] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[10] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*. Wiley, 1997.

[11] L. Baum and J. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bulletin of the AMS*, 73:360–363, 1967.

[12] Y. Normandin. *Hidden Markov Models, Maximum Mutual Information Estimation and the Speech Recognition Problem*. PhD thesis, McGill University, 1991.

[13] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Transactions on Software Engineering*, 5(4):333–340, 1979.

[14] V. Ramasubramanian and K. Paliwal. Fast K-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding. *IEEE Trans. on Signal Processing*, 40(3), 1992.

[15] J. Fritsch and I. Rogina. The bucket box intersection (BBI) algorithm for fast approximative evaluation of diagonal mixture gaussians. In *ICASSP*, pages 837–840, 1996.

[16] S. Srivastava. Fast Gaussian evaluations in large vocabulary continuous speech recognition. Master's thesis, Mississippi State University, 2002.

[17] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[18] A. Klautau. Classification of Peterson and Barney's vowels using Weka. Technical report, UFPA, *http://www.deec.ufpa.br/tr*, 2002.