Letters

Local Linear Perceptrons for Classification

Ethem Alpaydın and Michael I. Jordan

Abstract—**A structure composed of local linear perceptrons for approximating global class discriminants is investigated. Such local linear models may be combined in a cooperative or competitive way. In the cooperative model, a weighted sum of the outputs of the local perceptrons is computed where the weight is a function of the distance between the input and the position of the local perceptron. In the competitive model, the cost function dictates a mixture model where only one of the local perceptrons give output. Learning of the local models' positions and the linear mappings they implement are coupled and both supervised. We show that this is preferrable to the uncoupled case where the positions are trained in an unsupervised manner before the separate, supervised training of mappings. We use goodness criteria based on the cross-entropy and give learning equations for both the cooperative and competitive cases. The coupled and uncoupled versions of cooperative and competitive approaches are compared among themselves and with multi-layer perceptrons of sigmoidal hidden units and radial basis functions (RBF's) of Gaussian units on the application of recognition of handwritten digits. The criteria of comparison are the generalization accuracy, learning time, and the number of free parameters. We conclude that even on such a high-dimensional problem, such local models are promising. They generalize much better than RBF's and use much less memory. When compared with multilayer perceptrons, we note that local models learn much faster and generalize as well and sometimes better with comparable number of parameters.**

I. Introduction

The relative advantages of local and distributed representations have been frequently discussed in neural-network literature. Local methods, like kernel estimators and radial basis functions (RBF's), learn fast as only few hidden units respond to a given input, thus only a small percentage of weights need be updated at each iteration. But more hidden units are used with local coding and larger training samples are needed for good generalization. Distributed methods like the multilayer perceptrons (MLP's) with sigmoidal hidden units learn slowly but find compact representations with few parameters and do not require large samples.

In a usual multilayer network, local or distributed, the response of each hidden unit is scaled by a constant value which is the weight from the hidden unit to the output unit

$$y = \sum_h u_h g_h(\boldsymbol{x}) + u_0. \tag{1}$$

u_h are the weights, u_0 is the "bias" weight, and $g_h(\boldsymbol{x})$ are the hidden unit values for the d dimensional input $\boldsymbol{x}$. Alternatively, y is a superposition of the u_h values where $g_h(\boldsymbol{x})$ are the weights that determine how much of each u_h will be taken into account for input $\boldsymbol{x}$. Frequently $g_h(\cdot)$ values are normalized to sum to one. This implies that for any input, at least one of $g_h(\cdot)$ is nonzero.

Manuscript received April 12, 1995; revised September 25, 1995. This work was supported by a scholarship from TÜBİTAK, Turkish Scientific and Technical Research Council, and TÜBİTAK Grant EEEAG-143.

E. Alpaydın is with the Department of Computer Engineering, Boğaziçi University, TR-80815, Istanbul, Turkey.

M. I. Jordan is with the Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge MA 02139 USA.

Publisher Item Identifier S 1045-9227(96)03192-X.

In the case of purely local representation where only one of the $g_h(\cdot)$ is one and all others are zero, y is a piecewise constant function. If as in Parzen windows and RBF's, $g_h(\cdot)$ are taken as Gaussians, we get a smoother function. The more the Gaussians overlap, the smoother is the final approximation. It is evident that just like any Boolean function can be represented as a disjunction of conjunctions, any continuous function can be approximated to a desired precision as a juxtaposition of piecewise constant functions. If the function is varying considerably around a point, however, a piecewise constant approximation may require many units. Taking into account one more term in the Taylor expansion, we may also look at the linear function of the input, denoted as $w_h(\boldsymbol{x})$

$$\begin{aligned} y &= \sum_h w_h(\boldsymbol{x}) g_h(\boldsymbol{x}) + u_h g_h(\boldsymbol{x}) \\ &= \sum_h (\boldsymbol{W}_h^T \boldsymbol{x}) g_h(\boldsymbol{x}). \end{aligned} \tag{2}$$

We absorbed u_h as constant "bias" into $\boldsymbol{W}_h$ and added a constant dimension of one to $\boldsymbol{x}$. A higher-order function than linear is also possible but we want to minimize variance and not have too many free parameters. It is a good idea to have $g_h(\cdot)$ local like the Gaussian as only few of the $\boldsymbol{W}_h$ are then updated at each iteration. Each of the $\boldsymbol{W}_h^T \boldsymbol{x}$ computation then is a linear "expert" which has a position encoded by $\boldsymbol{V}_h$ and $g_h(\cdot)$ measures the (normalized) distance of the input $\boldsymbol{x}$ from this position. This requires an application-dependent distance measure $\mathcal{D}(\boldsymbol{x}, \boldsymbol{V}_h)$

$$g_h(\boldsymbol{x}) = \frac{\exp[\mathcal{D}(\boldsymbol{x}, \boldsymbol{V}_h)]}{\sum_i \exp[\mathcal{D}(\boldsymbol{x}, \boldsymbol{V}_i)]}. \tag{3}$$

Possibilities for $\mathcal{D}(\cdot, \cdot)$ include (negative) Euclidean distance, i.e., $\mathcal{D}(\boldsymbol{x}, \boldsymbol{V}_h) = -\|\boldsymbol{x} - \boldsymbol{V}_h\|$ (possibly modulated by a spread parameter) and Mahalanobis distance, i.e., $\mathcal{D}(\boldsymbol{x}, \boldsymbol{V}_h) = -(\boldsymbol{x} - \boldsymbol{V}_h)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{V}_h)$ where $\boldsymbol{\Sigma}$ is the covariance matrix. $\boldsymbol{V}_h$ in this case corresponds to a center, the mean, where $g_h(\cdot)$ takes its maximum. Another possibility that is simpler to compute is the dot product, i.e., $\mathcal{D}(\boldsymbol{x}, \boldsymbol{V}_h) = \boldsymbol{V}_h^T \boldsymbol{x}$ where $\boldsymbol{V}_h$ defines a hyperplane with a certain orientation and distance from the origin. As long as the norms of $\boldsymbol{V}_h$ are comparable, orderings given by the dot product and Euclidean distance are equal.

Learning in such a framework is estimating the positions $\boldsymbol{V}$ and linear mappings $\boldsymbol{W}$ from a given labeled training sample. We make distinction between two cases: In the case where learning is coupled, there is a single cost function for supervised training of both sets of parameters. In the uncoupled case, first proposed by Moody and Darken [1], training the positions $\boldsymbol{V}$ is an unsupervised process which precedes the separate, supervised training of the linear mappings $\boldsymbol{W}$.

A second type of distinction is by the degree of overlap of the linear experts. In a competitive scheme, the architecture is so designed that the final output is equal to the output of one of the linear mappings, i.e., one of $g_h(\cdot)$ is one and all others are zero. This is done either explicitly by choosing one, e.g., the closest and updating its parameters only, or implicitly by a cost measure that favors competition. In a cooperative scheme, there is no such requirement and the final output is a blend of the outputs of separate linear mappings.

In the next section, we review existing literature on the idea. Then we formalize the architecture and discuss two ways of combining the

TABLE I
COMPARISON OF PREVIOUS APPROACHES ACCORDING TO THE LEARNING STRATEGY AND THE TYPE OF COMBINING THE LOCAL MAPPINGS

Method	Learning	Combining
Hampshire and Waibel (1990)	uncoupled	cooperative
Stokbro et al. (1990)	uncoupled	cooperative
Jacobs et al. (1991)	coupled	competitive
Bottou and Vapnik (1992)	uncoupled	competitive
Martinetz et al. (1993)	uncoupled	competitive
Murray-Smith (1994)	uncoupled	cooperative

local perceptrons. We then apply it to the problem of recognition of handwritten digits and compare them empirically among themselves and with a single perceptron, multilayer network of sigmoidal hidden units, and RBF network of Gaussian hidden units. The three criteria on which we base our comparisons are learning time, number of free parameters, and generalization accuracy. In the final section, we summarize our findings and conclude.

II. LITERATURE SURVEY

Constructs like used in (2) where there are multiplicative connections in which the output values of two units are multiplied, thus allowing one unit to gate another, is known as a Sigma-Pi unit [2] or a product unit [3]. Previously many authors proposed structures composed of localized linear perceptrons both for classification and function approximation. In Table I, we compare them according to the two axes of coupled/uncoupled learning and competitive/cooperative combination.

- The Meta-Pi Network proposed by Hampshire and Waibel [4] for speech recognition contains a number of stimulus-specific networks each separately trained to recognize the speech of one individual. There is also an additional network that is trained to integrate the outputs of the subnetworks to maximize the phoneme recognition rate of the overall structure. The recognition modules or the gating network are not restricted to be linear perceptrons.
- Stokbro *et al.* [5] use a similar approach for predicting chaotic time series. Instead of a "flat" layer of units, they first compute a k-d tree using the input samples to compute the positions of the Gaussians. The linear mappings are then learned using least squares where the means and spreads of the Gaussians are fixed. Their method performs, as expected, better than the RBF's.
- Jacobs *et al.* [6] propose the "adaptive mixtures of local experts" where each local expert is a linear perceptron and there is a competitive gating mechanism that localizes the experts. The gating network which uses the dot product can also be seen as a set of perceptrons. This approach has later been generalized to learn "hierarchies of experts" [7].
- Bottou and Vapnik [8] propose to use "local learning algorithms" where simple systems are trained with a small subset of data around the given input as opposed to training one big, complex system with all of the data. They achieved significant improvement for an optical character recognition problem with this approach when for each test input, a fixed number of neighboring samples are used to train a separate linear perceptron.
- The "neural-gas" architecture of Martinetz *et al.* [9] is similar in that there is a competitive scheme where the input space is quantized and then local linear mappings are fit to data. These two steps are uncoupled, however. A soft competition takes place for placing the clusters that maximize the log-likelihood and a separate least squares minimization is done afterwards for fitting the linear mappings. For prediction of time series, Martinetz *et al.* find that the "neural-gas" performs better than backpropagation and RBF networks. Previous work of the same group [10] used Kohonen's self-organizing map that is a hard competitive scheme for input quantization and was for the control of a robot arm.
- Murray-Smith [11] similarly extends RBF networks where each "local model net" is a linear function of the input. One can have hierarchies of them and a constructive method is given where new models are incrementally added when needed. It is applied to control problems with significant success.

III. FORMALISM

For each class C_j, we are given a set of data pairs $\{\boldsymbol{x}^t, y^t\}_t$ where $\boldsymbol{x}$ is a d-dimensional input vector and y is a binary value that is one if $\boldsymbol{x} \in C_j$ and zero otherwise. Using this sample, we need to compute the posterior probabilities of classes $P(C_j|\boldsymbol{x}, \theta)$ for a novel input $\boldsymbol{x}$, where θ is the set of modifiable parameters of our model. If we assume that all errors are equally costly, to minimize risk we assign the input to the most probable class

$$c = \arg\max_j [P(C_j|\boldsymbol{x}, \theta)]. \tag{4}$$

For classification, we can write a "softmax" model for the posterior class probabilities [12], [13]

$$P(C_j|\boldsymbol{x}) = \frac{\exp[A_j(\boldsymbol{x})]}{\sum_k \exp[A_k(\boldsymbol{x})]}. \tag{5}$$

A_j denotes the total output for class j. One possibility is to take it as a weighted sum of the responses of the n local experts

$$A_j(\boldsymbol{x}) = \sum_{r=1}^{n} \mu_{jr}(\boldsymbol{x}) g_r(\boldsymbol{x}). \tag{6}$$

$\mu_{jr}(\boldsymbol{x})$ is the output of expert r for class j and $g_r(\boldsymbol{x})$ is the "weight" of expert r. Each local expert is a simple perceptron whose output is a linear function of the input (j ranges over classes and r over experts)

$$\mu_{jr}(\boldsymbol{x}) = \boldsymbol{W}_{jr}^T \boldsymbol{x}. \tag{7}$$

The mappings implemented by the experts are defined by the parameter vectors $\boldsymbol{W}_{jr}$. Each expert is responsible for inputs in a certain region only. In the simplest case, this region is delimited by a hyperplane. The softmax function is used again to make sure that the gating values sum up to one

$$g_r(\boldsymbol{x}) = \frac{\exp[\boldsymbol{V}_r^T \boldsymbol{x}]}{\sum_i \exp[\boldsymbol{V}_i^T \boldsymbol{x}]}. \tag{8}$$

The positions of the experts are coded by the parameter vectors $\boldsymbol{V}_r$. One can think of the gating network as another classifier where a given input is assigned to one of the experts and in this regard g_r values can be seen as probabilities, $P(\omega_r|\boldsymbol{x})$, the posterior probability that $\boldsymbol{x}$ is taken care of by expert r.

We use the cross-entropy measure that is more suited for classification tasks than least squares for optimization of the parameters $\theta = \{\boldsymbol{W}_{jr}, \boldsymbol{V}_r\}_{j,r}$. We favor coupled learning and use the same goodness measure for optimizing both the expert positions and the mappings. Experts can be combined in a competitive or cooperative manner as discussed below.

A. Cooperating Learners

In the cooperative scheme, each expert decides on the output by itself and then a weighted sum of the expert outputs is computed as in (6) which is then converted to probabilities using softmax

$$O_j = \frac{\exp[A_j]}{\sum_k \exp[A_k]} \tag{9}$$

and we maximize the cross-entropy to decrease the Kullback–Leibler distance between this value, O_j, and the desired output, y_j for all classes

$$E = \sum_j y_j \log O_j. \tag{10}$$

By gradient-ascent, taking the derivative of (10) with respect to the parameters and with η denoting the learning factor, we get the following update rules for the expert mappings:

$$\Delta W_{jr} = \eta(y_j - O_j) g_r \boldsymbol{x} \tag{11}$$

and their positions

$$\Delta V_r = \eta(y_j - O_j) g_r (\mu_{jr} - A_j) \boldsymbol{x}. \tag{12}$$

As seen in (12), when determining the expert positions, supervised error is also taken into account and not only the input as done by unsupervised procedures like k-means. Through coupled training of expert mappings and positions, experts are placed in the input space in such a way so as to minimize error. We discuss this in more detail in Section III-C.

B. Competing Learners

First proposed by Jacobs *et al.* [6], a measure that forces competition is to view the architecture as a mixture model [14]. The gating values are the mixture proportions and the expert perceptron outputs are the means. If we take Gaussian components with equal variances, the likelihood of the sample point $\boldsymbol{x}$ is given as

$$E = \log \sum_r g_r \exp \left[\sum_j y_j \log O_{jr} \right]. \tag{13}$$

Equation (13) forces experts to compete because the likelihood is higher when they overlap less. Thus generally for a given input, only one g_r is close to one and others are close to zero and it is this expert that is responsible from giving the whole correct output. That is we want to minimize the distance between the required output and the output of expert r whose g_r is close to one, after it is converted to probabilities

$$O_{jr} = \frac{\exp[\mu_{jr}]}{\sum_k \exp[\mu_{kr}]}.$$

The update equations for the expert mappings and positions with gradient-ascent are as follows:

$$\begin{aligned} \Delta W_{jr} &= \eta h_r (y_j - O_{jr}) \boldsymbol{x} \\ \Delta V_r &= \eta (h_r - g_r) \boldsymbol{x} \end{aligned} \tag{14}$$

with

$$h_r = \frac{g_r \exp[\sum_j y_j \log O_{jr}]}{\sum_i g_i \exp[\sum_j y_j \log O_{ji}]}.$$

C. Coupled and Uncoupled Learners

To make the mathematical connection between the coupled and the uncoupled methods clearer, let us consider a competitive model where we reparameterize the gating network to describe the density directly in the input space (rather than parameterizing the discriminant surfaces g_r using softmax) where the distance measure is Euclidean, thus getting a scenario similar to one used in unsupervised, competitive learning for uncoupled learning. That is, let

$$\begin{aligned} g_r &= P(\omega_r|\boldsymbol{x}) \\ &= \frac{P(\omega_r) p(\boldsymbol{x}|\omega_r)}{\sum_i P(\omega_i) p(\boldsymbol{x}|\omega_i)} \end{aligned} \tag{15}$$

where $P(\omega_r|\boldsymbol{x})$ is the posterior probability that input $\boldsymbol{x}$ is handled by expert r and $P(\omega_r)$ is the prior probability. We take the likelihoods, $p(\boldsymbol{x}|\omega_r)$, as Gaussians and we represent them directly by storing their means and possibly covariances.

We want our learning algorithm to move the means of these Gaussians, either in an uncoupled way, i.e., a clustering procedure, or in a coupled way by making use of the experts' outputs and the supervised target y.

We then have the following measure:

$$E = \log \sum_r P(\omega_r) p(\boldsymbol{x}|\omega_r) \exp \left[\sum_j y_j \log O_{jr} \right] \tag{16}$$

which differs from (13) in not having a softmax at the gating network. We assume that the priors, $P(\omega_r)$ are equal; one can obviously also adjust them as well as a part of the learning algorithm. Considering the simplest case of hyperspheric Gaussians of unit variance, i.e., $p(\boldsymbol{x}|\omega_r) \sim \mathcal{N}(\boldsymbol{V}_r, 1)$, with gradient-ascent, one gets the following update equations:

$$\begin{aligned} \Delta W_{jr} &= \eta h_r (y_j - O_{jr}) \boldsymbol{x} \\ \Delta V_r &= \eta h_r (\boldsymbol{x} - \boldsymbol{V}_r) \end{aligned} \tag{17}$$

with

$$h_r = \frac{g_r \exp[\sum_j y_j \log O_{jr}]}{\sum_i g_i \exp[\sum_j y_j \log O_{ji}]}$$

which can also be written as

$$P(\omega_r|y, x) = \frac{P(\omega_r|x) p(y|x, \omega_r)}{\sum_i P(\omega_i|x) p(y|x, \omega_i)}. \tag{18}$$

An uncoupled learning algorithm sets up a separate uncoupled mixture model for learning the likelihoods $p(\boldsymbol{x}|\omega_i)$. That is, we use a log likelihood that is local to the gating network, i.e., it does not depend on the supervised targets y

$$l_g = \log \sum_r P(\omega_r) p(\boldsymbol{x}|\omega_r). \tag{19}$$

Taking the derivative of l_g with respect to $\boldsymbol{V}_r$, we obtain the uncoupled learning rule

$$\Delta \boldsymbol{V}_r = \eta g_r (\boldsymbol{x} - \boldsymbol{V}_r). \tag{20}$$

This is a soft competitive rule. A method like k-means is a hard-competitive rule where g_r is one if expert r is the closest to input and zero otherwise.

The difference between is that (20) uses g_r, $P(\omega_r|\boldsymbol{x})$, whereas (17) uses h_r, $P(\omega_r|y, \boldsymbol{x})$. From (18), we see that ignoring the likelihood terms $P(y|\boldsymbol{x}, \omega_r)$ reduces h_r to g_r, and therefore reduces (17) to (20). Thus the uncoupled learning algorithm is a special case of the coupled learning algorithm where we are ignoring the likelihood terms $P(y|\boldsymbol{x}, \omega_r)$. That is, in uncoupled learning, we are ignoring the supervised errors at the output of the network in deciding where to place the cluster centers for the hidden units. Clearly, in general one does not want to ignore these likelihoods, i.e., the coupled approach is generally to be preferred.

IV. Simulation Results

We tested the approaches outlined above on a large handwritten digit database taken from the CD-ROM made available by NIST [15]. We used 10 000 patterns for training, 5000 for cross-validation

to determine the point to stop training, and another 5000 for testing. After low-pass filtering and undersampling, 32 by 32 bitmaps are reduced to 8 by 8 matrices where each element is an integer value in the range $0 \cdots 16$. This regularization step improves generalization. Thus input dimensionality is 64 and there are 10 classes.

We implemented the cooperative and competitive approaches where learning of expert positions and mappings are coupled, as discussed in Section III.

For comparison purposes, we also implemented their uncoupled versions. We have taken the model used in Section III-C and updated expert positions using (20). This implements a soft competition between experts. We also used a parameter for the spread of the Gaussians that is computed as a function of the inter-expert distance [1]. With the cooperative model, expert mappings are trained using (11). With the competitive model, only the closest expert gives output and is updated. That is, there is a hard competition with a winner-take all

$$g_r = \begin{cases} 1, & \text{if } \|\boldsymbol{V}_r - \boldsymbol{x}\| = \min_i \|\boldsymbol{V}_i - \boldsymbol{x}\| \\ 0, & \text{otherwise} \end{cases}$$

and update the mapping of only the winner expert.

We also implemented one single perceptron, MLP with one layer of hidden units and RBF's where we also used gradient ascent on the cross-entropy measure after softmaxing the output units. In the RBF network, the Gaussian centers are trained using the unsupervised procedure k-means.

Each model is trained 10 times independently and we report averages and standard deviations of the success on test set and the learning epochs made. The learning factor, η, is adaptive for improved convergence, that is, it is decreased by multiplying with a factor less than one when training error does not decrease and training stops when the learning factor becomes less than 0.001. A momentum factor of 0.7 is used. We test the network after each epoch till training stops and we choose the one that performs best on the cross-validation set and report its performance on the test set. This is better than stopping when error on cross-validation increases.

In terms of the number of parameters, a single perceptron where d is the dimensionality of the input and c is the number of classes has $(d+1)*c$ parameters. With h hidden units, a one-hidden-layer MLP has $(d+1)*h+(h+1)*c$ parameters. RBF does not have a bias unit in the first layer thus has $d*h+(h+1)*c$ parameters. One can also initialize the Gaussian centers to random patterns chosen from the training set thereby decreasing the number of free parameters to $(h+1)*c$; success then is around 1% less on the test set for this application. An adaptive mixture model, for each expert, has $c*(d+1)$ parameters for the classifiers and $d+1$ for the gating.

All methods are compared based on the three criteria of generalization accuracy, memory requirement and learning time. Results are reported in Table II.

TABLE II
COMPARISON OF VARIOUS APPROACHES FOR THE HANDWRITTEN DIGIT RECOGNITION PROBLEM. THE THREE CRITERIA OF COMPARISON ARE THE MEMORY REQUIREMENT MEASURED AS THE NUMBER OF FREE PARAMETERS, LEARNING SPEED MEASURED AS THE NUMBER LEARNING EPOCHS, AND GENERALIZATION ACCURACY MEASURED AS SUCCESS PERCENTAGE ON A TEST SET UNSEEN DURING TRAINING OR CROSS-VALIDATION. VALUES ARE AVERAGES AND STANDARD DEVIATIONS OF 10 INDEPENDENT RUNS

Method		No of Parameters	Learning Epochs	Test Success
Single Layer		650	2.80, 0.42	91.03, 0.21
MLP	10 units	760	6.90, 2.60	92.80, 0.46
	20 units	1,510	6.80, 2.30	94.33, 0.29
	30 units	2,260	7.60, 2.27	94.83, 0.16
	40 units	3,010	5.90, 1.37	94.89, 0.32
	50 units	3,760	6.70, 2.36	94.92, 0.31
	60 units	4,510	7.00, 1.33	95.00, 0.18
RBF	50 units	3,710	2.00, 0.00	87.35, 0.81
	100 units	7,410	1.70, 0.48	89.91, 0.47
	250 units	18,510	1.60, 0.52	92.10, 0.45
	500 units	37,010	2.00, 0.47	93.30, 0.41
Cooperative (uncoupled)	2 experts	1,430	4.70, 0.48	91.54, 0.25
	4 experts	2,860	4.90, 0.32	91.51, 0.34
	6 experts	4,290	5.10, 0.32	92.20, 0.36
Competitive (uncoupled)	2 experts	1,430	4.00, 0.47	91.14, 0.09
	4 experts	2,860	3.40, 0.97	91.31, 0.21
	6 experts	4,290	3.70, 0.67	91.29, 0.12
Cooperative (coupled)	2 experts	1,430	4.50, 0.97	93.43, 0.30
	4 experts	2,860	4.30, 0.82	94.61, 0.26
	6 experts	4,290	5.10, 0.99	95.08, 0.28
Competitive (coupled)	2 experts	1,430	3.60, 0.70	93.25, 0.60
	4 experts	2,860	3.50, 0.85	93.75, 0.39
	6 experts	4,290	3.20, 0.63	94.03, 0.35

V. CONCLUSIONS

It can be noticed that local linear perceptrons learn faster than MLP's with sigmoid hidden units and generalize as well and sometimes better. RBF's learn faster but do not generalize well. This is due to the fact that in RBF's only a one-layer perceptron is trained in a supervised manner. In a local linear network, we train two one-layer perceptrons and in a multilayer perceptron, we train a two-layer network.

We note that a coupled approach where both the mappings and their positions are trained in a supervised manner generalizes better than the uncoupled case where the positions are trained in an unsupervised manner. It seems like that in both the coupled and uncoupled cases, the competitive model learns faster due to the localization of experts and the cooperative model generalizes better due to the smoothness introduced by averaging several experts.

We have also tested the approach using Euclidean distance as the metric where separate networks have different spreads. The success is somewhat lower in that case as the dimensionality is high and spheric Gaussians assume equal variance on all dimensions. We have not tested the method using the full covariance matrix as, in 64 dimensions the full covariance matrix has on the order of a thousand parameters. This is both costly to compute and requires a much larger training sample.

Here we used a gradient-based method for computing the parameters. When as in normal mixtures, there is a probabilistic setting where the goodness function is maximum likelihood, the expectation-maximization algorithm can also be used to train the networks [7].

To conclude, we believe that a structure composed of local linear perceptrons, by allowing a good compromise between local and distributed approaches, is a good alternative for difficult classification tasks as it learns fast and generalizes quite well even when the input dimensionality is high.

ACKNOWLEDGMENT

The authors thank Z. Ghahramani and R. Murray-Smith for stimulating discussions. The preprocessing routines for the NIST database were made available by F. Masulli of the University of Genoa, Italy. The authors also thank two anonymous reviewers for constructive comments.

REFERENCES

[1] J. Moody and C. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computa.,* vol. 1, pp. 281–294, 1989.

[2] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland, "A framework for PDP," in *Parallel Distributed Processing—Explorations in the Microstructure of Cognition,* vol. 1, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 45–76.

[3] R. Durbin and D. E. Rumelhart, "Product units: A computationally powerful and biologically plausible extension to backpropagation networks," *Neural Computa.,* vol. 1, pp. 133–142, 1989.

[4] J. B. Hampshire, II and A. Waibel, "Connectionist architectures for multispeaker phoneme recognition," in *Advances in Neural Information Processing Systems,* vol. 2, D. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 203–210.

[5] K. Stokbro, D. K. Umberger, and J. A. Hertz, "Exploiting neurons with localized receptive fields to learn chaos," *Complex Syst.,* vol. 4, pp. 603–622, 1990.

[6] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Computa.,* vol. 3, pp. 79–87, 1991.

[7] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Computa.,* vol. 6, pp. 181–214, 1994.

[8] L. Bottou and V. Vapnik, "Local learning algorithms," *Neural Computa.,* vol. 4, pp. 888–900, 1992.

[9] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, " 'Neural-gas' network for vector quantization and its application to time-series prediction," *IEEE Trans. Neural Networks,* vol. 4, 558–569, 1993.

[10] H. Ritter, T. Martinetz, and K. Schulten, "Topology-conserving maps for learning visuomotor coordination," *Neural Networks,* vol. 2, pp. 159–168, 1988.

[11] R. Murray-Smith, "A local model network approach to nonlinear modelling," Ph.D. dissertation, Dep. Compute. Sci., Univ. Strathclyde, 1994.

[12] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing,* F. Fogelman-Soulié and J. Hérault, Eds. Berlin: Springer-Verlag, 1990, pp. 227–236.

[13] B. D. Ripley, "Neural networks and related methods for classification," *J. Roy. Statist. Soc. B,* vol. 56, pp. 409–456, 1994.

[14] S. J. Nowlan, "Soft competitive adaptation: Neural network learning algorithms based on fitting statistical mixtures," Ph.D. dissertation, School Comput. Sci., Carnegie Mellon Univ., 1990.

[15] M. D. Garris, J. L. Blue, G. T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C. L. Wilson, *NIST Form-Based Handprint Recognition System,* NISTIR 5469, 1994.