Published in: *Neural Computation*, *6*, 181-214, 1994.

Hierarchical mixtures of experts and the EM algorithm*

Michael I. Jordan
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology

Robert A. Jacobs
Department of Psychology
University of Rochester

Abstract

We present a tree-structured architecture for supervised learning. The statistical model underlying the architecture is a hierarchical mixture model in which both the mixture coefficients and the mixture components are generalized linear models (GLIM's). Learning is treated as a maximum likelihood problem; in particular, we present an Expectation-Maximization (EM) algorithm for adjusting the parameters of the architecture. We also develop an on-line learning algorithm in which the parameters are updated incrementally. Comparative simulation results are presented in the robot dynamics domain.

*We want to thank Geoffrey Hinton, Tony Robinson, Mitsuo Kawato and Daniel Wolpert for helpful comments on the manuscript.

This project was supported in part by a grant from the McDonnell-Pew Foundation, by a grant from ATR Human Information Processing Research Laboratories, by a grant from Siemens Corporation, by by grant IRI-9013991 from the National Science Foundation, and by grant N00014-90-J-1942 from the Office of Naval Research. The project was also supported by NSF grant ASC-9217041 in support of the Center for Biological and Computational Learning at MIT, including funds provided by DARPA under the HPCC program, and NSF grant ECS-9216531 to support an Initiative in Intelligent Control at MIT. Michael I. Jordan is a NSF Presidential Young Investigator.

Introduction

The principle of divide-and-conquer is a principle with wide applicability throughout applied mathematics. Divide-and-conquer algorithms attack a complex problem by dividing it into simpler problems whose solutions can be combined to yield a solution to the complex problem. This approach can often lead to simple, elegant and efficient algorithms. In this paper we explore a particular application of the divide-and-conquer principle to the problem of learning from examples. We describe a network architecture and a learning algorithm for the architecture, both of which are inspired by the philosophy of divide-and-conquer.

In the statistical literature and in the machine learning literature, divide-and-conquer approaches have become increasingly popular. The CART algorithm of Breiman, Friedman, Olshen, and Stone (1984), the MARS algorithm of Friedman (1991), and the ID3 algorithm of Quinlan (1986) are well-known examples. These algorithms fit surfaces to data by explicitly dividing the input space into a nested sequence of regions, and by fitting simple surfaces (e.g., constant functions) within these regions. They have convergence times that are often orders of magnitude faster than gradient-based neural network algorithms.

Although divide-and-conquer algorithms have much to recommend them, one should be concerned about the statistical consequences of dividing the input space. Dividing the data can have favorable consequences for the bias of an estimator, but it generally increases the variance. Consider linear regression, for example, in which the variance of the estimates of the slope and intercept depend quadratically on the spread of data on the x-axis. The points that are the most peripheral in the input space are those that have the maximal effect in decreasing the variance of the parameter estimates.

The foregoing considerations suggest that divide-and-conquer algorithms generally tend to be variance-increasing algorithms. This is indeed the case and is particularly problematic in high-dimensional spaces where data become exceedingly sparse (Scott, 1992). One response to this dilemma—that adopted by CART, MARS, and ID3, and also adopted here—is to utilize piecewise constant or piecewise linear functions. These functions minimize variance at a cost of increased bias. We also make use of a second variance-decreasing device; a device familiar in the neural network literature. We make use of "soft" splits of data (Bridle, 1989; Nowlan, 1991; Wahba, Gu, Wang, & Chappell, 1993), allowing data to lie simultaneously in multiple regions. This approach allows the parameters in one region to be influenced by data in neighboring regions. CART, MARS, and ID3 rely on "hard" splits, which, as we remarked above, have particularly severe effects on variance. By allowing soft splits the severe effects of lopping off distant data can be ameliorated. We also attempt to minimize the bias that is incurred by using piecewise linear functions, by allowing the splits to be formed along hyperplanes at arbitrary orientations in the input space. This lessens the bias due to high-order interactions among the inputs and allows the algorithm to be insensitive to the particular choice of coordinates used to encode the data (an improvement over methods such as MARS and ID3, which are coordinate-dependent).

The work that we describe here makes contact with a number of branches of statistical theory. First, as in our earlier work (Jacobs, Jordan, Nowlan, & Hinton, 1991), we formulate the learning problem as a mixture estimation problem (cf. Cheeseman, et al, 1988; Duda & Hart, 1973; Nowlan, 1991; Redner & Walker, 1984; Titterington, Smith, & Makov, 1985). We show that the algorithm that is generally employed for the *unsupervised* learning of mixture parameters—the Expectation-Maximization (EM) algorithm of Dempster, Laird and Rubin (1977)—can also be exploited for *supervised* learning. Second, we utilize generalized linear model (GLIM) theory (McCullagh & Nelder, 1983) to provide the basic statistical structure for the components of the architecture. In particular, the "soft splits" referred to above are modeled as *multinomial logit* models—a specific form of GLIM. We also show that the algorithm developed for fitting GLIM's—the iteratively reweighted least squares (IRLS) algorithm—can be usefully employed in our model, in particular as the M step of the EM algorithm. Finally, we show that these ideas can be developed in a recursive manner, yielding a tree-structured approach to estimation that is reminiscent of CART, MARS, and ID3.

The remainder of the paper proceeds as follows. We first introduce the hierarchical mixture-of-experts architecture and present the likelihood function for the architecture. After describing a gradient descent algorithm, we develop a more powerful learning algorithm for the architecture that is a special case of the general Expectation-Maximization (EM) framework of Dempster, Laird and Rubin (1977). We also describe a least-squares version of this algorithm that leads to a particularly efficient implementation. Both of the latter algorithms are batch learning algorithms. In the final section, we present an on-line version of the least-squares algorithm that in practice appears to be the most efficient of the algorithms that we have studied.

Hierarchical mixtures of experts

The algorithms that we discuss in this paper are supervised learning algorithms. We explicitly address the case of regression, in which the input vectors are elements of $\Re^m$ and the output vectors are elements of $\Re^n$. We also consider classification models and counting models in which the outputs are integer-valued. The data are assumed to form a countable set of paired observations $\mathcal{X} = \{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\}$. In the case of the *batch* algorithms discussed below, this set is assumed to be finite; in the case of the *on-line* algorithms, the set may be infinite.

We propose to solve nonlinear supervised learning problems by dividing the input space into a nested set of regions and fitting simple surfaces to the data that fall in these regions. The regions have "soft" boundaries, meaning that data points may

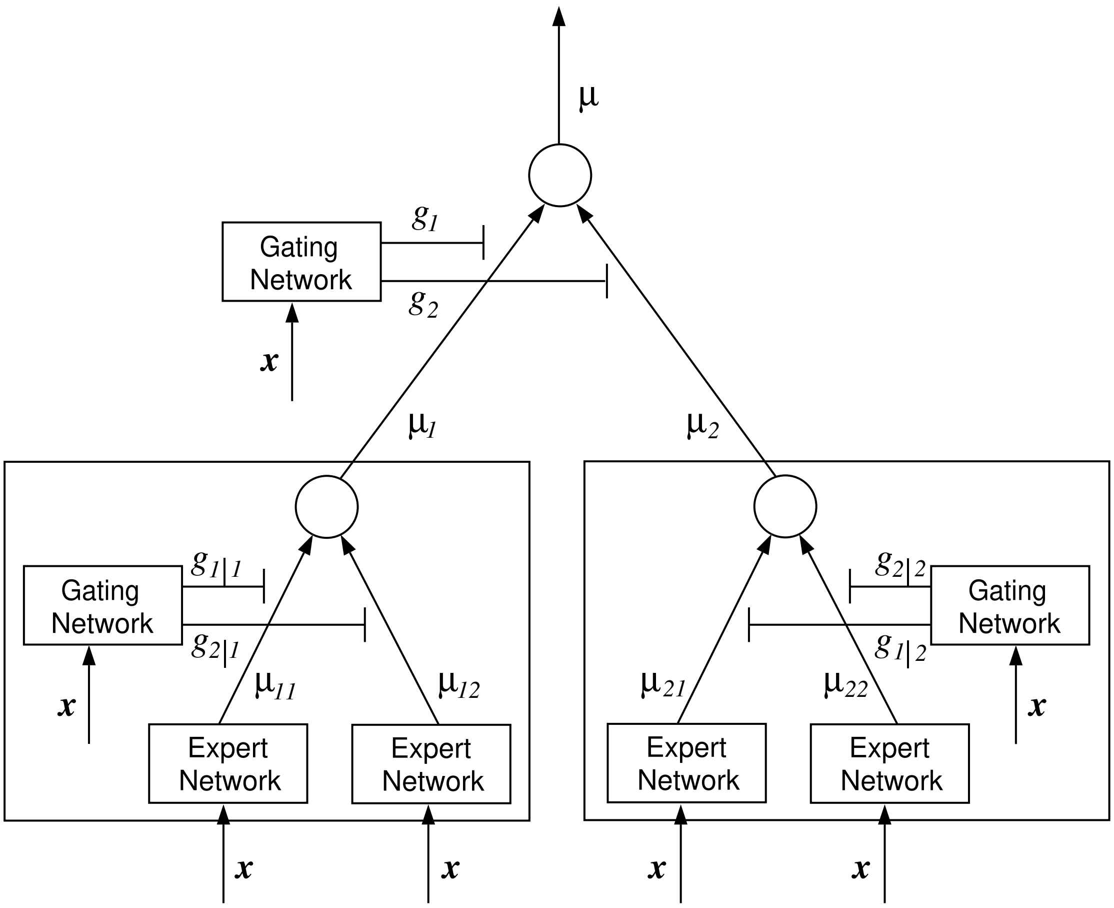

Figure 1: A two-level hierarchical mixture of experts. To form a deeper tree, each expert is expanded recursively into a gating network and a set of sub-experts.

lie simultaneously in multiple regions. The boundaries between regions are themselves simple parameterized surfaces that are adjusted by the learning algorithm.

The hierarchical mixture-of-experts (HME) architecture is shown in Figure 1.[1] The architecture is a tree in which the *gating networks* sit at the nonterminals of the tree. These networks receive the vector $\mathbf{x}$ as input and produce scalar outputs that are a partition of unity at each point in the input space. The *expert networks* sit at the leaves of the tree. Each expert produces an output vector $\boldsymbol{\mu}_{ij}$ for each input vector. These output vectors proceed up the tree, being blended by the gating network outputs.

[1]To simplify the presentation, we restrict ourselves to a two-level hierarchy throughout the paper. All of the algorithms that we describe, however, generalize readily to hierarchies of arbitrary depth. See Jordan and Xu (1993) for a recursive formalism that handles arbitrary hierarchies.

All of the expert networks in the tree are linear with a single output nonlinearity. We will refer to such a network as "generalized linear," borrowing the terminology from statistics (McCullagh & Nelder, 1983). Expert network (i,j) produces its output $\boldsymbol{\mu}_{ij}$ as a generalized linear function of the input $\mathbf{x}$:

$$\boldsymbol{\mu}_{ij} = f(U_{ij}\mathbf{x}), \tag{1}$$

where U_{ij} is a weight matrix and f is a fixed continuous nonlinearity. The vector $\mathbf{x}$ is assumed to include a fixed component of one to allow for an intercept term.

For regression problems, $f(\cdot)$ is generally chosen to be the identity function (i.e., the experts are linear). For binary classification problems, $f(\cdot)$ is generally taken to be the logistic function, in which case the expert outputs are interpreted as the log odds of "success" under a Bernoulli probability model (see below). Other models (e.g., multiway classification, counting, rate estimation and survival estimation) are handled by making other choices for $f(\cdot)$. These models are smoothed piecewise analogs of the corresponding GLIM models (cf. McCullagh & Nelder, 1983).

The gating networks are also generalized linear. Define intermediate variables ξ_i as follows:

$$\xi_i = \mathbf{v}_i^T \mathbf{x}, \tag{2}$$

where $\mathbf{v}_i$ is a weight vector. Then the i^{th} output of the top-level gating network is the "softmax" function of the ξ_i (Bridle, 1989; McCullagh & Nelder, 1983):

$$g_i = \frac{e^{\xi_i}}{\sum_k e^{\xi_k}}. \tag{3}$$

Note that the g_i are positive and sum to one for each $\mathbf{x}$. They can be interpreted as providing a "soft" partitioning of the input space.

Similarly, the gating networks at lower levels are also generalized linear systems. Define ξ_{ij} as follows:

$$\xi_{ij} = \mathbf{v}_{ij}^T \mathbf{x}. \tag{4}$$

Then

$$g_{j|i} = \frac{e^{\xi_{ij}}}{\sum_k e^{\xi_{ik}}} \tag{5}$$

is the output of the j^{th} unit in the i^{th} gating network at the second level of the architecture. Once again, the $g_{j|i}$ are positive and sum to one for each $\mathbf{x}$. They can be interpreted as providing a nested "soft" partitioning of the input space within the partitioning providing by the higher-level gating network.

The output vector at each nonterminal of the tree is the weighted output of the experts below that nonterminal. That is, the output at the i^{th} nonterminal in the second layer of the two-level tree is:

$$\boldsymbol{\mu}_i = \sum_j g_{j|i} \boldsymbol{\mu}_{ij}$$

and the output at the top level of the tree is:

$$\boldsymbol{\mu} = \sum_i g_i \boldsymbol{\mu}_i.$$

Note that both the g's and the $\boldsymbol{\mu}$'s depend on the input $\mathbf{x}$, thus the total output is a nonlinear function of the input.

Regression surface

Given the definitions of the expert networks and the gating networks, the regression surface defined by the hierarchy is a piecewise blend of the regression surfaces defined by the experts. The gating networks provide a nested, "soft" partitioning of the input space and the expert networks provide local regression surfaces within the partition. There is overlap between neighboring regions. To understand the nature of the overlap, consider a one-level hierarchy with two expert networks. In this case, the gating network has two outputs, g_1 and g_2. The gating output g_1 is given by:

$$\begin{aligned} g_1 &= \frac{e^{\xi_1}}{e^{\xi_1} + e^{\xi_2}} && (6) \\ &= \frac{1}{1 + e^{-(\mathbf{v}_1 - \mathbf{v}_2)^T \mathbf{x}}}, && (7) \end{aligned}$$

which is a logistic ridge function whose orientation is determined by the direction of the vector $\mathbf{v}_1 - \mathbf{v}_2$. The gating output g_2 is equal to $1 - g_1$. For a given $\mathbf{x}$, the total output $\boldsymbol{\mu}$ is the convex combination $g_1\boldsymbol{\mu}_1 + g_2\boldsymbol{\mu}_2$. This is a weighted average of the experts, where the weights are determined by the values of the ridge function. Along the ridge, $g_1 = g_2 = \frac{1}{2}$, and both experts contribute equally. Away from the ridge, one expert or the other dominates. The amount of smoothing across the ridge is determined by the magnitude of the vector $\mathbf{v}_2 - \mathbf{v}_1$. If $\mathbf{v}_2 - \mathbf{v}_1$ is large, then the ridge function becomes a sharp split and the weighted output of the experts becomes piecewise (generalized) linear. If $\mathbf{v}_2 - \mathbf{v}_1$ is small, then each expert contributes to a significant degree on each side of the ridge, thereby smoothing the piecewise map. In the limit of a zero difference vector, $g_1 = g_2 = \frac{1}{2}$ for all $\mathbf{x}$, and the total output is the same fixed average of the experts on both sides of the fictitious "split."

In general, a given gating network induces a smoothed planar partitioning of the input space. Lower-level gating networks induce a partition within the partition induced by higher-level gating networks. The weights in a given gating network determine the amount of smoothing across the partition at that particular level of resolution: large weight vectors imply sharp changes in the regression surface across a ridge and small weights imply a smoother surface. In the limit of zero weights in all gating networks, the entire hierarchy reduces to a fixed average (a linear system in the case of regression).

A probability model

The hierarchy can be given a probabilistic interpretation. We suppose that the mechanism by which data are generated by the environment involves a nested sequence of decisions that terminates in a regressive process that maps $\mathbf{x}$ to $\mathbf{y}$. The decisions are modeled as multinomial random variables. That is, for each $\mathbf{x}$, we interpret the values $g_i(\mathbf{x}, \mathbf{v}_i^0)$ as the multinomial probabilities associated with the first decision and the $g_{j|i}(\mathbf{x}, \mathbf{v}_{ij}^0)$ as the (conditional) multinomial probabilities associated with the second decision, where the superscript "0" refers to the "true" values of the parameters. The decisions form a decision tree. We use a statistical model to model this decision tree; in particular, our choice of parameterization (cf. Equations 2, 4, 3 and 5) corresponds to a *multinomial logit* probability model at each nonterminal of the tree (see Appendix 2). A multinomial logit model is a special case of a GLIM that is commonly used for "soft" multiway classification (McCullagh & Nelder, 1983). Under the multinomial logit model, we interpret the gating networks as modeling the input-dependent, multinomial probabilities associated with decisions at particular levels of resolution in a tree-structured model of the data.

Once a particular sequence of decisions has been made, resulting in a choice of regressive process (i, j), output $\mathbf{y}$ is assumed to be generated according to the following statistical model. First, a linear predictor $\boldsymbol{\eta}_{ij}$ is formed:

$$\boldsymbol{\eta}_{ij}^0 = U_{ij}^0 \mathbf{x}.$$

The expected value of $\mathbf{y}$ is obtained by passing the linear predictor through the *link function* f:[2]

$$\boldsymbol{\mu}_{ij}^0 = f(\boldsymbol{\eta}_{ij}^0).$$

The output $\mathbf{y}$ is then chosen from a probability density P, with mean $\boldsymbol{\mu}_{ij}^0$ and "dispersion" parameter $\boldsymbol{\phi}_{ij}^0$. We denote the density of $\mathbf{y}$ as:

$$P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{ij}^0),$$

where the parameter vector $\boldsymbol{\theta}_{ij}^0$ includes the weights U_{ij}^0 and the dispersion parameter $\boldsymbol{\phi}_{ij}^0$:

$$\boldsymbol{\theta}_{ij}^0 = \begin{bmatrix} U_{ij}^0 \\ \boldsymbol{\phi}_{ij}^0 \end{bmatrix}.$$

We assume the density P to be a member of the exponential family of densities (McCullagh & Nelder, 1983). The interpretation of the dispersion parameter depends on the particular choice of density. For example, in the case of the n-dimensional Gaussian, the dispersion parameter is the covariance matrix Σ_{ij}^0.[3]

[2] We utilize the neural network convention in defining links. In GLIM theory, the convention is that the link function relates η to μ; thus, $\eta = h(\mu)$, where h is equivalent to our f^{-1}.

[3] Not all exponential family densities have a dispersion parameter; in particular, the Bernoulli density discussed below has no dispersion parameter.

Given these assumptions, the total probability of generating $\mathbf{y}$ from $\mathbf{x}$ is the mixture of the probabilities of generating $\mathbf{y}$ from each of the component densities, where the mixing proportions are multinomial probabilities:

$$P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}^0) = \sum_i g_i(\mathbf{x}, \mathbf{v}_i^0) \sum_j g_{j|i}(\mathbf{x}, \mathbf{v}_{ij}^0) P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}_{ij}^0), \tag{8}$$

Note that $\boldsymbol{\theta}^0$ includes the expert network parameters $\boldsymbol{\theta}_{ij}^0$ as well as the gating network parameters $\mathbf{v}_i^0$ and $\mathbf{v}_{ij}^0$. Note also that we have explicitly indicated the dependence of the probabilities g_i and $g_{j|i}$ on the input $\mathbf{x}$ and on the parameters. In the remainder of the paper we drop the explicit reference to the input and the parameters to simplify the notation:

$$P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}^0) = \sum_i g_i^0 \sum_j g_{j|i}^0 P_{ij}^0(\mathbf{y}), \tag{9}$$

We also utilize Equation 9 without the superscripts to refer to the probability model defined by a particular HME architecture, irrespective of any reference to a "true" model.

Example (regression)

In the case of regression the probabilistic component of the model is generally assumed to be Gaussian. Assuming identical covariance matrices of the form $\sigma^2 I$ for each of the experts yields the following hierarchical probability model:

$$P(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{(2\pi)^{n/2}\sigma^n} \sum_i g_i \sum_j g_{j|i} e^{-\frac{1}{2\sigma^2}(\mathbf{y}-\boldsymbol{\mu}_{ij})^T(\mathbf{y}-\boldsymbol{\mu}_{ij})}.$$

Example (binary classification)

In binary classification problems the output y is a discrete random variable having possible outcomes of "failure" and "success." The probabilistic component of the model is generally assumed to be the Bernoulli distribution (Cox, 1970). In this case, the mean μ_{ij} is the conditional probability of classifying the input as "success." The resulting hierarchical probability model is a mixture of Bernoulli densities:

$$P(y|\mathbf{x}, \boldsymbol{\theta}) = \sum_i g_i \sum_j g_{j|i} \mu_{ij}^y (1 - \mu_{ij})^{1-y}.$$

Posterior probabilities

In developing the learning algorithms to be presented in the remainder of the paper, it will prove useful to define posterior probabilities associated with the nodes of the tree. The terms "posterior" and "prior" have meaning in this context during the training of the system. We refer to the probabilities g_i and $g_{j|i}$ as *prior* probabilities, because they are computed based only on the input $\mathbf{x}$, without knowledge of the corresponding target output $\mathbf{y}$. A *posterior* probability is defined once both the input and the target output are known. Using Bayes' rule, we define the posterior probabilities at the nodes of the tree as follows:

$$h_i = \frac{g_i \sum_j g_{j|i} P_{ij}(\mathbf{y})}{\sum_i g_i \sum_j g_{j|i} P_{ij}(\mathbf{y})} \tag{10}$$

and

$$h_{j|i} = \frac{g_{j|i} P_{ij}(\mathbf{y})}{\sum_j g_{j|i} P_{ij}(\mathbf{y})}. \tag{11}$$

We will also find it useful to define the joint posterior probability h_{ij}, the product of h_i and $h_{j|i}$:

$$h_{ij} = \frac{g_i g_{j|i} P_{ij}(\mathbf{y})}{\sum_i g_i \sum_j g_{j|i} P_{ij}(\mathbf{y})} \tag{12}$$

This quantity is the probability that expert network (i, j) can be considered to have generated the data, based on knowledge of both the input and the output. Once again, we emphasize that all of these quantities are conditional on the input $\mathbf{x}$.

In deeper trees, the posterior probability associated with an expert network is simply the product of the conditional posterior probabilities along the path from the root of the tree to that expert.

The likelihood and a gradient descent learning algorithm

Jordan and Jacobs (1992) presented a gradient descent learning algorithm for the hierarchical architecture. The algorithm was based on earlier work by Jacobs, Jordan, Nowlan, and Hinton (1991), who treated the problem of learning in mixture-of-experts architectures as a maximum likelihood estimation problem. The log likelihood of a data set $\mathcal{X} = \{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\}_1^N$ is obtained by taking the log of the product of N densities of the form of Equation 9, which yields the following log likelihood:

$$l(\boldsymbol{\theta}; \mathcal{X}) = \sum_t \ln \sum_i g_i^{(t)} \sum_j g_{j|i}^{(t)} P_{ij}(\mathbf{y}^{(t)}). \tag{13}$$

Let us assume that the probability density P is Gaussian with an identity covariance matrix and that the link function is the identity. In this case, by differentiating $l(\boldsymbol{\theta}; \mathcal{X})$ with respect to the parameters, we obtain the following gradient descent learning rule for the weight matrix U_{ij}:

$$\Delta U_{ij} = \rho \sum_t h_i^{(t)} h_{j|i}^{(t)} (\mathbf{y}^{(t)} - \boldsymbol{\mu}^{(t)}) \mathbf{x}^{(t)T}, \tag{14}$$

where ρ is a learning rate. The gradient descent learning rule for the i^{th} weight vector in the top-level gating network is given by:

$$\Delta \mathbf{v}_i = \rho \sum_t (h_i^{(t)} - g_i^{(t)})\mathbf{x}^{(t)}, \tag{15}$$

and the gradient descent rule for the j^{th} weight vector in the i^{th} lower-level gating network is given by:

$$\Delta \mathbf{v}_{ij} = \rho \sum_t h_i^{(t)}(h_{j|i}^{(t)} - g_{j|i}^{(t)})\mathbf{x}^{(t)}, \tag{16}$$

Updates can also be obtained for covariance matrices (Jordan & Jacobs, 1992).

The algorithm given by Equations 14, 15, and 16 is a batch learning algorithm. The corresponding on-line algorithm is obtained by simply dropping the summation sign and updating the parameters after each stimulus presentation. Thus, for example,

$$U_{ij}^{(t+1)} = U_{ij}^{(t)} + \rho h_i^{(t)} h_{j|i}^{(t)}(\mathbf{y}^{(t)} - \boldsymbol{\mu}^{(t)})\mathbf{x}^{(t)T} \tag{17}$$

is the stochastic update rule for the weights in the $(i,j)^{th}$ expert network based on the t^{th} stimulus pattern.

The EM algorithm

In the following sections we develop a learning algorithm for the HME architecture based on the Expectation-Maximization (EM) framework of Dempster, Laird, and Rubin (1977). We derive an EM algorithm for the architecture that consists of the iterative solution of a coupled set of iteratively-reweighted least-squares problems.

The EM algorithm is a general technique for maximum likelihood estimation. In practice EM has been applied almost exclusively to unsupervised learning problems. This is true of the neural network literature and machine learning literature, in which EM has appeared in the context of clustering (Cheeseman, et al. 1988; Nowlan, 1990) and density estimation (Specht, 1991), as well as the statistics literature, in which applications include missing data problems (Little & Rubin, 1987), mixture density estimation (Redner & Walker, 1984), and factor analysis (Dempster, Laird, & Rubin, 1977). Another unsupervised learning application is the learning problem for Hidden Markov Models, for which the Baum-Welch reestimation formulas are a special case of EM. There is nothing in the EM framework that precludes its application to regression or classification problems; however, such applications have been few.[4]

EM is an iterative approach to maximum likelihood estimation. Each iteration of an EM algorithm is composed of two steps: an Estimation (E) step and a Maximization (M) step. The M step involves the maximization of a likelihood

[4]An exception is the "switching regression" model of Quandt and Ramsey (1972). For further discussion of switching regression, see Jordan and Xu (1993).

function that is redefined in each iteration by the E step. If the algorithm simply increases the function during the M step, rather than maximizing the function, then the algorithm is referred to as a Generalized EM (GEM) algorithm. The Boltzmann learning algorithm (Hinton & Sejnowski, 1986) is a neural network example of a GEM algorithm. GEM algorithms are often significantly slower to converge than EM algorithms.

An application of EM generally begins with the observation that the optimization of the likelihood function $l(\boldsymbol{\theta}; \mathcal{X})$ would be simplified if only a set of additional variables, called "missing" or "hidden" variables, were known. In this context, we refer to the observable data $\mathcal{X}$ as the "incomplete data" and posit a "complete data" set $\mathcal{Y}$ that includes the missing variables $\mathcal{Z}$. We specify a probability model that links the fictive missing variables to the actual data: $P(\mathbf{y}, \mathbf{z} | \mathbf{x}, \boldsymbol{\theta})$. The logarithm of the density P defines the "complete-data likelihood," $l_c(\boldsymbol{\theta}; \mathcal{Y})$. The original likelihood, $l(\boldsymbol{\theta}; \mathcal{X})$, is referred to in this context as the "incomplete-data likelihood." It is the relationship between these two likelihood functions that motivates the EM algorithm. Note that the complete-data likelihood is a random variable, because the missing variables $\mathcal{Z}$ are in fact unknown. An EM algorithm first finds the expected value of the complete-data likelihood, given the observed data and the current model. This is the E step:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(p)}) = E[l_c(\boldsymbol{\theta}; \mathcal{Y}) | \mathcal{X}],$$

where $\boldsymbol{\theta}^{(p)}$ is the value of the parameters at the p^{th} iteration and the expectation is taken with respect to $\boldsymbol{\theta}^{(p)}$. This step yields a deterministic function Q. The M step maximizes this function with respect to $\boldsymbol{\theta}$ to find the new parameter estimates $\boldsymbol{\theta}^{(p+1)}$:

$$\boldsymbol{\theta}^{(p+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(p)}).$$

The E step is then repeated to yield an improved estimate of the complete likelihood and the process iterates.

An iterative step of EM chooses a parameter value that increases the value of Q, the expectation of the complete likelihood. What is the effect of such a step on the incomplete likelihood? Dempster, et al. proved that an increase in Q implies an increase in the incomplete likelihood:

$$l(\boldsymbol{\theta}^{(p+1)}; \mathcal{X}) \geq l(\boldsymbol{\theta}^{(p)}; \mathcal{X}).$$

Equality obtains only at the stationary points of l (Wu, 1983). Thus the likelihood l increases monotonically along the sequence of parameter estimates generated by an EM algorithm. In practice this implies convergence to a local maximum.

Applying EM to the HME architecture

To develop an EM algorithm for the HME architecture, we must define appropriate "missing data" so as to simplify the likelihood function. We define indicator variables

z_i and $z_{j|i}$, such that one and only one of the z_i is equal to one, and one and only one of the $z_{j|i}$ is equal to one. These indicator variables have an interpretation as the labels that correspond to the decisions in the probability model. We also define the indicator variable z_{ij}, which is the product of z_i and $z_{j|i}$. This variable has an interpretation as the label that specifies the expert (the regressive process) in the probability model. If the labels z_i, $z_{j|i}$ and z_{ij} were known, then the maximum likelihood problem would decouple into a separate set of regression problems for each expert network and a separate set of multiway classification problems for the gating networks. These problems would be solved independently of each other, yielding a rapid one-pass learning algorithm. Of course, the missing variables are not known, but we can specify a probability model that links them to the observable data. This probability model can be written in terms of the z_{ij} as follows:

$$\begin{aligned} P(\mathbf{y}^{(t)}, z_{ij}^{(t)}|\mathbf{x}^{(t)}, \boldsymbol{\theta}) &= g_i^{(t)} g_{j|i}^{(t)} P_{ij}(\mathbf{y}^{(t)}) && (18) \\ &= \prod_i \prod_j \{g_i^{(t)} g_{j|i}^{(t)} P_{ij}(\mathbf{y}^{(t)})\}^{z_{ij}^{(t)}}, && (19) \end{aligned}$$

using the fact that $z_{ij}^{(t)}$ is an indicator variable. Taking the logarithm of this probability model yields the following complete-data likelihood:

$$\begin{aligned} l_c(\boldsymbol{\theta}; \mathcal{Y}) &= \sum_t \sum_i \sum_j z_{ij}^{(t)} \ln\{g_i^{(t)} g_{j|i}^{(t)} P_{ij}(\mathbf{y}^{(t)})\} && (20) \\ &= \sum_t \sum_i \sum_j z_{ij}^{(t)} \{\ln g_i^{(t)} + \ln g_{j|i}^{(t)} + \ln P_{ij}(\mathbf{y}^{(t)})\}. && (21) \end{aligned}$$

Note the relationship of the complete-data likelihood in Equation 21 to the incomplete-data likelihood in Equation 13. The use of the indicator variables z_{ij} has allowed the logarithm to be brought inside the summation signs, substantially simplifying the maximization problem. We now define the E step of the EM algorithm by taking the expectation of the complete-data likelihood:

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(p)}) = \sum_t \sum_i \sum_j h_{ij}^{(t)} \{\ln g_i^{(t)} + \ln g_{j|i}^{(t)} + \ln P_{ij}(\mathbf{y}^{(t)})\}, \qquad (22)$$

where we have used the fact that:

$$\begin{aligned} E[z_{ij}^{(t)}|\mathcal{X}] &= P(z_{ij}^{(t)} = 1|\mathbf{y}^{(t)}, \mathbf{x}^{(t)}, \boldsymbol{\theta}^{(p)}) && (23) \\ &= \frac{P(\mathbf{y}^{(t)}|z_{ij}^{(t)} = 1, \mathbf{x}^{(t)}, \boldsymbol{\theta}^{(p)}) P(z_{ij}^{(t)} = 1|\mathbf{x}^{(t)}, \boldsymbol{\theta}^{(p)})}{P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \boldsymbol{\theta}^{(p)})} && (24) \\ &= \frac{P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \boldsymbol{\theta}_{ij}{}^{(p)}) g_i^{(t)} g_{j|i}^{(t)}}{\sum_i g_i^{(t)} \sum_j g_{j|i}^{(t)} P(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}, \boldsymbol{\theta}_{ij}{}^{(p)})} && (25) \\ &= h_{ij}^{(t)}. && (26) \end{aligned}$$

(Note also that $E[z_i^{(t)}|\mathcal{X}] = h_i^{(t)}$ and $E[z_{j|i}^{(t)}|\mathcal{X}] = h_{j|i}^{(t)}$.)

The M step requires maximizing $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(p)})$ with respect to the expert network parameters and the gating network parameters. Examining Equation 22, we see that the expert network parameters influence the Q function only through the terms $h_{ij}^{(t)} \ln P_{ij}(\mathbf{y}^{(t)})$, and the gating network parameters influence the Q function only through the terms $h_{ij}^{(t)} \ln g_i^{(t)}$ and $h_{ij}^{(t)} \ln g_{j|i}^{(t)}$. Thus the M step reduces to the following separate maximization problems:

$$\boldsymbol{\theta}_{ij}^{(p+1)} = \arg\max_{\boldsymbol{\theta}_{ij}} \sum_t h_{ij}^{(t)} \ln P_{ij}(\mathbf{y}^{(t)}), \tag{27}$$

$$\mathbf{v}_i^{(p+1)} = \arg\max_{\mathbf{v}_i} \sum_t \sum_k h_k^{(t)} \ln g_k^{(t)}, \tag{28}$$

and

$$\mathbf{v}_{ij}^{(p+1)} = \arg\max_{\mathbf{v}_{ij}} \sum_t \sum_k h_k^{(t)} \sum_l h_{l|k}^{(t)} \ln g_{l|k}^{(t)}. \tag{29}$$

Each of these maximization problems are themselves maximum likelihood problems. Equation 27 is simply the general form of a weighted maximum likelihood problem in the probability density P_{ij}. Given our parameterization of P_{ij}, the log likelihood in Equation 27 is a weighted log likelihood for a GLIM. An efficient algorithm known as iteratively reweighted least-squares (IRLS) is available to solve the maximum likelihood problem for such models (McCullagh & Nelder, 1983). We discuss IRLS in Appendix A.

Equation 28 involves maximizing the cross-entropy between the posterior probabilities $h_k^{(t)}$ and the prior probabilities $g_k^{(t)}$. This cross-entropy is the log likelihood associated with a multinomial logit probability model in which the $h_k^{(t)}$ act as the output observations (see Appendix B). Thus the maximization in Equation 28 is also a maximum likelihood problem for a GLIM and can be solved using IRLS. The same is true of Equation 29, which is a weighted maximum likelihood problem with output observations $h_{l|k}^{(t)}$ and observation weights $h_k^{(t)}$.

In summary, the EM algorithm that we have obtained involves a calculation of posterior probabilities in the outer loop (the E step), and the solution of a set of IRLS problems in the inner loop (the M step). We summarize the algorithm as follows:

Algorithm 1

1. For each data pair $(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})$, compute the posterior probabilities $h_i^{(t)}$ and $h_{j|i}^{(t)}$ using the current values of the parameters.
2. For each expert (i, j), solve an IRLS problem with observations $\{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\}_1^N$ and observation weights $\{h_{ij}^{(t)}\}_1^N$.
3. For each top-level gating network, solve an IRLS problem with observations $\{(\mathbf{x}^{(t)}, h_k^{(t)})\}_1^N$.
4. For each lower-level gating network, solve a weighted IRLS problem with observations $\{(\mathbf{x}^{(t)}, h_{l|k}^{(t)})\}_1^N$ and observation weights $\{h_k^{(t)}\}_1^N$.
5. Iterate using the updated parameter values.

A least-squares algorithm

In the case of regression, in which a Gaussian probability model and an identity link function are used, the IRLS loop for the expert networks reduces to weighted least squares, which can be solved (in one pass) by any of the standard least-squares algorithms (Golub & van Loan, 1989). The gating networks still require iterative processing. Suppose, however, that we fit the parameters of the gating networks using least squares rather than maximum likelihood. In this case, we might hope to obtain an algorithm in which the gating network parameters are fit by a one-pass algorithm. To motivate this approach, note that we can express the IRLS problem for the gating networks as follows. Differentiating the cross-entropy (Equation 28) with respect to the parameters $\mathbf{v}_i$ (using the fact that $\partial g_i / \partial \xi_j = g_i(\delta_{ij} - g_j)$, where δ_{ij} is the Kronecker delta) and setting the derivatives to zero yields the following equations:

$$\sum_t (h_i^{(t)} - g_i(\mathbf{x}^{(t)}, \mathbf{v}_i))\mathbf{x}^{(t)} = 0, \tag{30}$$

which are a coupled set of equations that must be solved for each i. Similarly, for each gating network at the second level of the tree, we obtain the following equations:

$$\sum_t h_i^{(t)} (h_{j|i}^{(t)} - g_{j|i}(\mathbf{x}^{(t)}, \mathbf{v}_{ij}))\mathbf{x}^{(t)} = 0, \tag{31}$$

which must be solved for each i and j. There is one aspect of these equations that renders them unusual. Recall that if the labels $z_i^{(t)}$ and $z_{j|i}^{(t)}$ were known, then the gating networks would be essentially solving a set of multiway classification problems. The supervised errors $(z_i^{(t)} - g_i^{(t)})$ and $(z_{j|i}^{(t)} - g_{j|i}^{(t)})$ would appear in the algorithm for solving these problems. Note that these errors are differences between indicator variables and probabilities. In Equations 30 and 31, on the other hand, the errors that drive the algorithm are the differences $(h_i^{(t)} - g_i^{(t)})$ and $(h_{j|i}^{(t)} - g_{j|i}^{(t)})$, which are differences between probabilities. The EM algorithm effectively "fills in" the missing labels with estimated probabilities h_i and $h_{j|i}$. These estimated probabilities can be thought of as targets for the g_i and the $g_{j|i}$. This suggests that we can compute "virtual targets" for the underlying linear predictors ξ_i and $\xi_{j|i}$, by inverting the softmax function. (Note that this option would not be available for the z_i and $z_{j|i}$, even if they were known, because zero and one are not in the range of the softmax function.) Thus the targets for the ξ_i are the values:

$$\ln h_i^{(t)} - \ln C,$$

where $C = \sum_k e^{\xi_k}$ is the normalization constant in the softmax function. Note, however, that constants that are common to all of the ξ_i can be omitted, because such constants disappear when ξ_i are converted to g_i. Thus the values $\ln h_i^{(t)}$ can be used as targets for the ξ_i. A similar argument shows that the values $\ln h_{l|k}^{(t)}$ can be used as targets for the ξ_{ij}, with observation weights $h_k^{(t)}$.

The utility of this approach is that once targets are available for the linear predictors ξ_i and ξ_{ij}, the problem of finding the parameters $\mathbf{v}_i$ and $\mathbf{v}_{ij}$ reduces to a coupled set of weighted least-squares problems. Thus we obtain an algorithm in which all of the parameters in the hierarchy, both in the expert networks and the gating networks, can be obtained by solving least-squares problems. This yields the following learning algorithm:

Algorithm 2

1. For each data pair $(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})$, compute the posterior probabilities $h_i^{(t)}$ and $h_{j|i}^{(t)}$ using the current values of the parameters.

2. For each expert (i, j), solve a weighted least-squares problem with observations $\{(\mathbf{x}^{(t)}, \mathbf{y}^{(t)})\}_1^N$ and observation weights $\{h_{ij}^{(t)}\}_1^N$.

3. For each top-level gating network, solve a least-squares problem with observations $\{(\mathbf{x}^{(t)}, \ln h_k^{(t)})\}_1^N$.

4. For each lower-level gating network, solve a weighted least-squares problem with observations $\{(\mathbf{x}^{(t)}, \ln h_{l|k}^{(t)})\}_1^N$ and observation weights $\{h_k^{(t)}\}_1^N$.

5. Iterate using the updated parameter values.

It is important to note that this algorithm does not yield the same parameter estimates as Algorithm 1; the gating network residuals $(h_i^{(t)} - g_i^{(t)})$ are being fit by least squares rather than maximum likelihood. The algorithm can be thought of as an approximation to Algorithm 1, an approximation based on the assumption that the differences between $h_i^{(t)}$ and $g_i^{(t)}$ are small. This assumption is equivalent to the assumption that the architecture can fit the underlying regression surface (a consistency condition) and the assumption that the noise is small. In practice we have found that the least squares algorithm works reasonably well, even in the early stages of fitting when the residuals can be large. The ability to use least squares is certainly appealing from a computational point of view. One possible hybrid algorithm involves using the least squares algorithm to converge quickly to the neighborhood of a solution and then using IRLS to refine the solution.

Simulation results

We tested Algorithm 1 and Algorithm 2 on a nonlinear system identification problem. The data were obtained from a simulation of a four-joint robot arm moving in three-dimensional space (Fun & Jordan, 1993). The network must learn the *forward dynamics* of the arm; a state-dependent mapping from joint torques to joint accelerations. The state of the arm is encoded by eight real-valued variables: four positions (rad) and four angular velocities (rad/sec). The torque was encoded as four real-valued variables ($\text{N} \cdot \text{m}$). Thus there were twelve inputs to the learning system. Given these twelve input variables, the network must predict the four accelerations at the joints (rad/sec^2). This mapping is highly nonlinear due to the rotating coordinate systems and the interaction torques between the links of the arm.

We generated 15,000 data points for training and 5,000 points for testing. For each epoch (i.e., each pass through the training set), we computed the relative error on the test set. Relative error is computed as a ratio between the mean squared error and the mean squared error that would be obtained if the learner were to output the mean value of the accelerations for all data points.

We compared the performance of a binary hierarchy to that of a backpropagation network. The hierarchy was a four-level hierarchy with 16 expert networks and 15 gating networks. Each expert network had 4 output units and each gating network had 1 output unit. The backpropagation network had 60 hidden units, which yields approximately the same number of parameters in the network as in the hierarchy.

The HME architecture was trained by Algorithms 1 and 2, utilizing Cholesky decomposition to solve the weighted least-squares problems (Golub & van Loan, 1989). Note that the HME algorithms have no free parameters. The free parameters for the backpropagation network (the learning rate and the momentum term) were chosen based on a coarse search of the parameter space. (Values of 0.00001 and 0.15

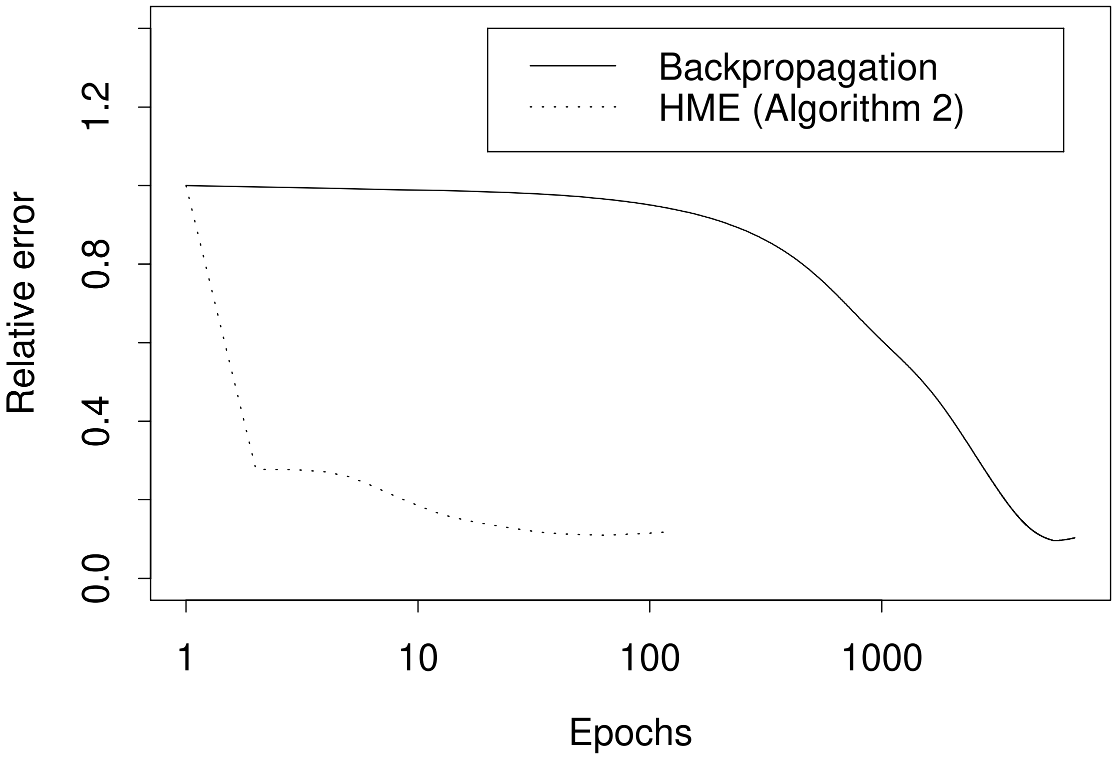

Figure 2: Relative error on the test set for a backpropagation network and a four-level HME architecture trained with batch algorithms. The standard errors at the minima of the curves are 0.013 for backprop and 0.002 for HME.

were chosen for these parameters.) There were difficulties with local minima (or plateaus) using the backpropagation algorithm: Five of ten runs failed to converge to "reasonable" error values. (As we report in the next section, no such difficulties were encountered in the case of *on-line* backpropagation). We report average convergence times and average relative errors only for those runs that converged to "reasonable" error values. All ten runs for both of the HME algorithms converged to "reasonable" error values.

Figure 2 shows the performance of the hierarchy and the backpropagation network. The horizontal axis of the graph gives the training time in epochs. The vertical axis gives generalization performance as measured by the average relative error on the test set.

Table 1 reports the average relative errors for both architectures measured at the minima of the relative error curves. (Minima were defined by a sequence of three successive increases in the relative error.) We also report values of relative error for the best linear approximation, the CART algorithm, and the MARS algorithm. Both CART and MARS were run four times, once for each of the output variables.

Architecture	Relative Error	# Epochs
linear	.31	1
backprop	.09	5,500
HME (Algorithm 1)	.10	35
HME (Algorithm 2)	.12	39
CART	.17	NA
CART (linear)	.13	NA
MARS	.16	NA

Table 1: Average values of relative error and number of epochs required for convergence for the batch algorithms.

We combined the results from these four computations to compute the total relative error. Two versions of CART were run; one in which the splits were restricted to be parallel to the axes and one in which linear combinations of the input variables were allowed.

The MARS algorithm requires choices to be made for the values of two structural parameters: the maximum number of basis functions and the maximum number of interaction terms. Each basis function in MARS yields a linear surface defined over a rectangular region of the input space, corresponding roughly to the function implemented by a single expert in the HME architecture. Therefore we chose a maximum of 16 basis functions to correspond to the 16 experts in the four-level hierarchy. To choose the maximum number of interactions (mi), we compared the performance of MARS for $mi = 1$, 2, 3, 6, and 12, and chose the value that yielded the best performance ($mi = 3$).

For the iterative algorithms, we also report the number of epochs required for convergence. Because the learning curves for these algorithms generally have lengthy tails, we defined convergence as the first epoch at which the relative error drops within five percent of the minimum.

All of the architectures that we studied performed significantly better than the best linear approximation. As expected, the CART architecture with linear combinations performed better than CART with axis-parallel splits.[5] The HME architecture yielded a modest improvement over MARS and CART. Backpropagation produced the lowest relative error of the algorithms tested (ignoring the difficulties with convergence).

These differences in relative error should be treated with some caution. The need to set free parameters for some of the architectures (e.g., backpropagation) and

[5]It should be noted that CART is at an advantage relative to the other algorithms in this comparison, because no structural parameters were fixed for CART. That is, CART is allowed to find the best tree of any size to fit the data.

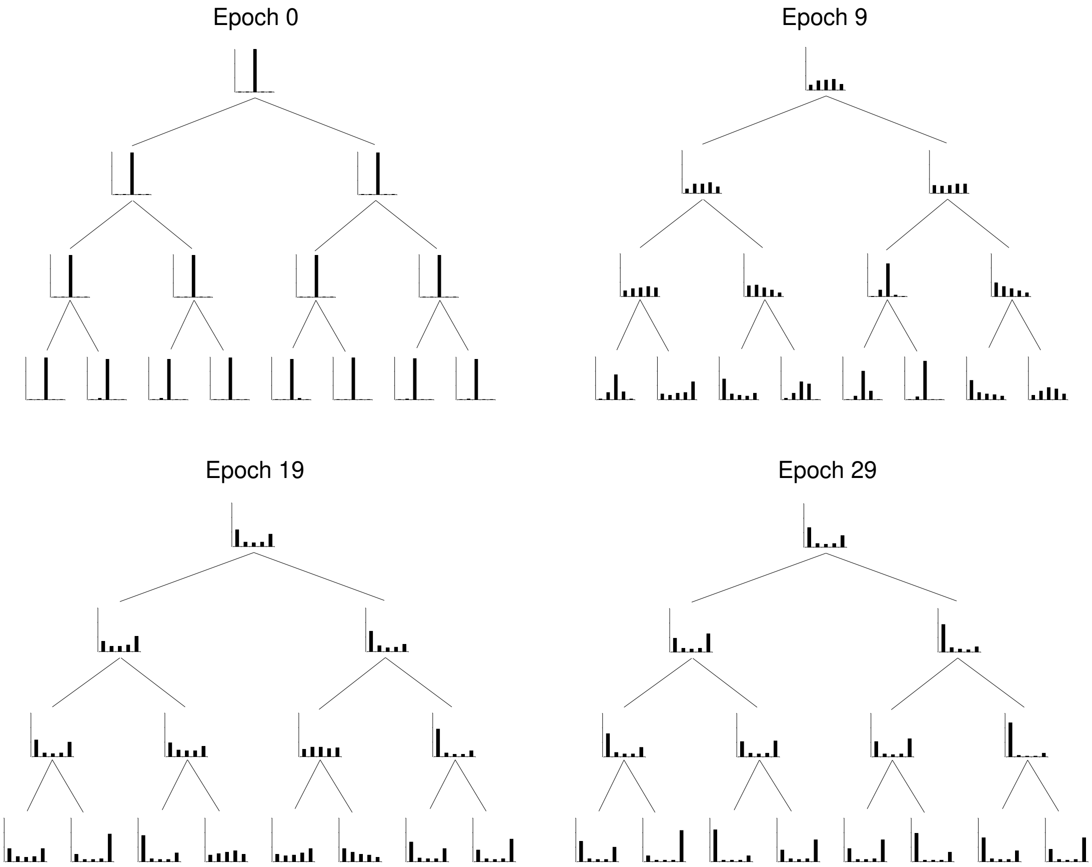

Figure 3: A sequence of histogram trees for the HME architecture. Each histogram displays the distribution of posterior probabilities across the training set at each node in the tree.

the need to make structural choices (e.g., number of hidden units, number of basis functions, number of experts) makes it difficult to match architectures. The HME architecture, for example, involves parameter dependencies that are not present in a backpropagation network. A gating network at a high level in the tree can "pinch off" a branch of the tree, rendering useless the parameters in that branch of the tree. Raw parameter count is therefore only a very rough guide to architecture capacity; more precise measures are needed (e.g., VC dimension) before definitive quantitative comparisons can be made.

The differences between backpropagation and HME in terms of convergence time are more definitive. Both HME algorithms reliably converge more than two orders of magnitude faster than backpropagation.

As shown in Figure 3, the HME architecture lends itself well to graphical investigation. This figure displays the time sequence of the distributions of posterior probabilities across the training set at each node of the tree. At Epoch 0, before any learning has taken place, most of the posterior probabilities at each node are approximately 0.5 across the training set. As the training proceeds, the histograms flatten out, eventually approaching bimodal distributions in which the posterior probabilities are either one or zero for most of the training patterns. This evolution is indicative of increasingly sharp splits being fit by the gating networks. Note that there is a tendency for the splits to be formed more rapidly at higher levels in the tree than at lower levels.

Figure 4 shows another graphical device that can be useful for understanding the way in which a HME architecture fits a data set. This figure, which we refer to as a "deviance tree," shows the deviance (mean squared error) that would be obtained at each level of the tree if the tree were clipped at that level. We construct a clipped tree at a given level by replacing each nonterminal at that level with a matrix that is a weighted average of the experts below that nonterminal. The weights are the total prior probabilities associated with each expert across the training set. The error for each output unit is then calculated by passing the test set through the clipped tree. As can be seen in the figure, the deviance is substantially smaller for deeper trees (note that the ordinate of the plots is on a log scale). The deviance in the right branch of the tree is larger than in the left branch of the tree. Information such as this can be useful for purposes of exploratory data analysis and for model selection.

An on-line algorithm

The batch least-squares algorithm that we have described (Algorithm 2) can be converted into an on-line algorithm by noting that linear least squares and weighted linear least squares problems can be solved by recursive procedures that update the parameter estimates with each successive data point (Ljung & Söderström, 1986). Our application of these recursive algorithms is straightforward; however, care must be taken to handle the observation weights (the posterior probabilities) correctly. These weights change as a function of the changing parameter values. This implies that the recursive least squares algorithm must include a decay parameter that allows the system to "forget" older values of the posterior probabilities.

In this section we present the equations for the on-line algorithm. These equations involve an update not only of the parameters in each of the networks,[6] but also the storage and updating of an inverse covariance matrix for each network. Each matrix has dimensionality mxm, where m is the dimensionality of the input vector. (Note that the size of these matrices depends on the square of the number of *input* variables, not the square of the number of *parameters*. Note also that the update equation for the inverse covariance matrix updates the inverse matrix directly; there is never a need to invert matrices.)

[6]Note that in this section we use the term "parameters" for the variables that are traditionally called "weights" in the neural network literature. We reserve the term "weights" for the observation weights.

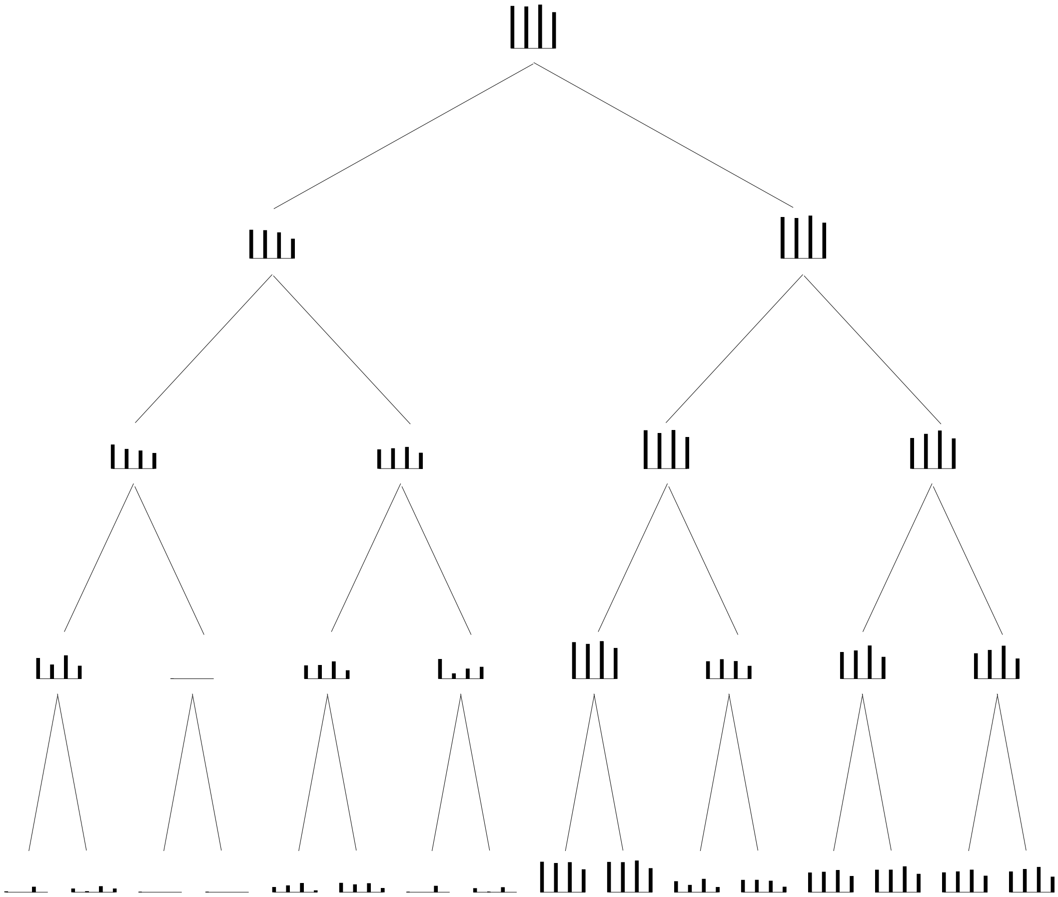

Figure 4: A deviance tree for the HME architecture. Each plot displays the mean squared error (MSE) for the four output units of the clipped tree. The plots are on a log scale covering approximately three orders of magnitude.

The on-line update rule for the parameters of the expert networks is given by the following recursive equation:

$$U_{ij}^{(t+1)} = U_{ij}^{(t)} + h_i^{(t)} h_{j|i}^{(t)} (\mathbf{y}^{(t)} - \boldsymbol{\mu}_{ij}^{(t)}) \mathbf{x}^{(t)T} R_{ij}^{(t)}, \tag{32}$$

where R_{ij} is the inverse covariance matrix for expert network (i, j). This matrix is updated via the equation:

$$R_{ij}^{(t)} = \lambda^{-1} R_{ij}^{(t-1)} - \lambda^{-1} \frac{R_{ij}^{(t-1)} \mathbf{x}^{(t)} \mathbf{x}^{(t)T} R_{ij}^{(t-1)}}{\lambda [h_{ij}^{(t)}]^{-1} + \mathbf{x}^{(t)T} R_{ij}^{(t-1)} \mathbf{x}^{(t)}}, \tag{33}$$

where λ is the decay parameter.

It is interesting to note the similarity between the parameter update rule in Equation 32 and the gradient rule presented earlier (cf. Equation 14). These updates are essentially the same, except that the scalar ρ is replaced by the matrix $R_{ij}^{(t)}$. It can be shown, however, that $R_{ij}^{(t)}$ is an estimate of the inverse Hessian of the least-squares cost function (Ljung & Söderström, 1986), thus Equation 32 is in fact a stochastic approximation to a Newton-Raphson method rather than a gradient method.[7]

Similar equations apply for the updates of the gating networks. The update rule for the parameters of the top-level gating network is given by the following equation (for the i^{th} output of the gating network):

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + S_i^{(t)}(\ln h_i^{(t)} - \xi_i^{(t)})\mathbf{x}^{(t)}, \tag{34}$$

where the inverse covariance matrix S_i is updated by:

$$S_i^{(t)} = \lambda^{-1} S_i^{(t-1)} - \lambda^{-1} \frac{S_i^{(t-1)}\mathbf{x}^{(t)}\mathbf{x}^{(t)T} S_i^{(t-1)}}{\lambda + \mathbf{x}^{(t)T} S_i^{(t-1)} \mathbf{x}^{(t)}}. \tag{35}$$

Finally, the update rule for the parameters of the lower-level gating network are as follows:

$$\mathbf{v}_{ij}^{(t+1)} = \mathbf{v}_{ij}^{(t)} + S_{ij}^{(t)} h_i^{(t)}(\ln h_{j|i}^{(t)} - \xi_{ij}^{(t)})\mathbf{x}^{(t)}, \tag{36}$$

where the inverse covariance matrix S_i is updated by:

$$S_{ij}^{(t)} = \lambda^{-1} S_{ij}^{(t-1)} - \lambda^{-1} \frac{S_{ij}^{(t-1)}\mathbf{x}^{(t)}\mathbf{x}^{(t)T} S_{ij}^{(t-1)}}{\lambda [h_i^{(t)}]^{-1} + \mathbf{x}^{(t)T} S_{ij}^{(t-1)} \mathbf{x}^{(t)}}. \tag{37}$$

Simulation results

The on-line algorithm was tested on the robot dynamics problem described in the previous section. Preliminary simulations convinced us of the necessity of the decay parameter (λ). We also found that this parameter should be slowly increased as training proceeds—on the early trials the posterior probabilities are changing rapidly so that the covariances should be decayed rapidly, whereas on later trials the posterior probabilities have stabilized and the covariances should be decayed less rapidly. We used a simple fixed schedule: λ was initialized to 0.99 and increased a fixed fraction (0.6) of the remaining distance to 1.0 every 1000 time steps.

The performance of the on-line algorithm was compared to an on-line backpropagation network. Parameter settings for the backpropagation network were

[7] This is true for fixed values of the posterior probabilities. These posterior probabilities are also changing over time, however, as required by the EM algorithm. The overall convergence rate of the algorithm is determined by the convergence rate of EM, not the convergence rate of Newton-Raphson.

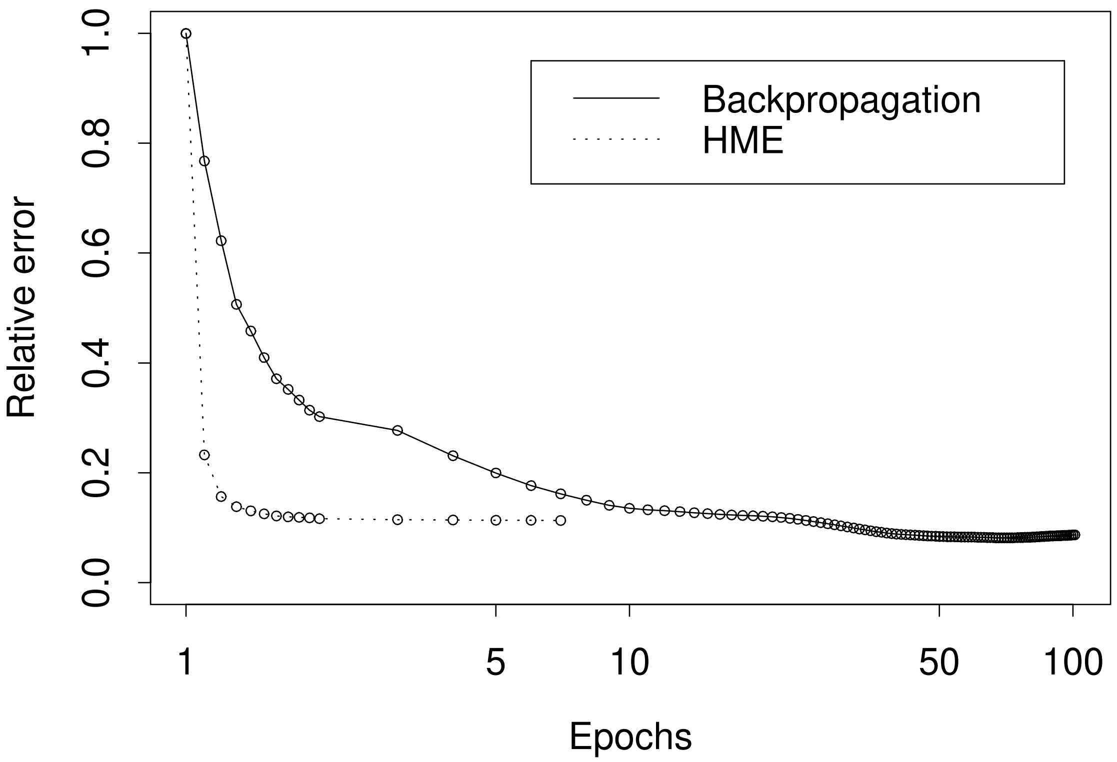

Figure 5: Relative error on the test set for a backpropagation network and a four-level hierarchy trained with on-line algorithms. The standard errors at the minima of the curves are 0.008 for backprop and 0.009 for HME.

obtained by a coarse search through the parameter space, yielding a value of 0.15 for the learning rate and 0.20 for the momentum. The results for both architectures are shown in Figure 5. As can be seen, the on-line algorithm for backpropagation is significantly faster than the corresponding batch algorithm (cf. Figure 2). This is also true of the on-line HME algorithm, which has nearly converged within the first epoch.

The minimum values of relative error and the convergence times for both architectures are provided in Table 2. We also provide the corresponding values for a simulation of the on-line gradient algorithm for the HME architecture (Equation 17).

We also performed a set of simulations which tested a variety of different HME architectures. We compared a one-level hierarchy with 32 experts to hierarchies with five levels (32 experts), and six levels (64 experts). We also simulated two three-level hierarchies, one with branching factors of 4, 4, and 2 (proceeding from the top of the tree to the bottom), and one with branching factors of 2, 4, and 4. (Each three-level hierarchy contained 32 experts.) The results are shown in Figure 6.

Architecture	Relative Error	# Epochs
linear	.32	1
backprop (on-line)	.08	63
HME (on-line)	.12	2
HME (gradient)	.15	104

Table 2: Average values of relative error and number of epochs required for convergence for the on-line algorithms.

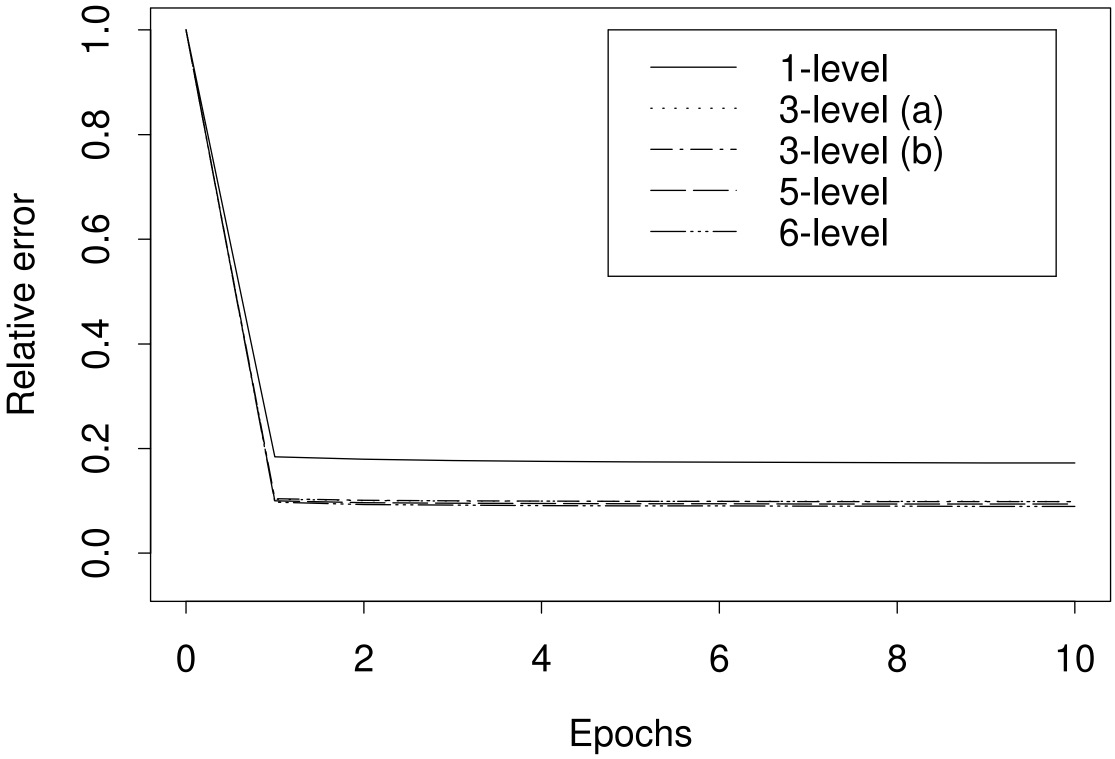

Figure 6: Relative error on the test set for HME hierarchies with different structures. "3-level (a)" refers to a 3-level hierarchy with branching factors of 4, 4, and 2, and "3-level (b)" refers to a 3-level hierarchy with branching factors of 2, 4, and 4. The standard errors for all curves at their respective minima were approximately 0.009.

As can be seen, there was a significant difference between the one-level hierarchy and the other architectures. There were smaller differences among the multi-level hierarchies. No significant difference was observed between the two different 3-level architectures.

Model selection

Utilizing the HME approach requires that choices be made regarding the structural parameters of the model, in particular the number of levels and the branching factor of the tree. As with other flexible estimation techniques, it is desirable to allow these structural parameters to be chosen based at least partly on the data. This model selection problem can be addressed in a variety of ways. In this paper we have utilized a test set approach to model selection, stopping the training when the error on the test set reaches a minimum. As is the case with other neural network algorithms, this procedure can be justified as a complexity control measure. As we have noted, when the parameters in the gating networks of an HME architecture are small, the entire system reduces to a single "averaged" GLIM at the root of the tree. As the training proceeds, the parameters in the gating networks begin to grow in magnitude and splits are formed. When a split is formed the parameters in the branches of the tree on either side of the split are decoupled and the effective number of degrees of freedom in the system increases. This increase in complexity takes place gradually as the values of the parameters increase and the splits sharpen. By stopping the training of the system based on the performance on a test set, we obtain control over the effective number of degrees of freedom in the architecture.

Other approaches to model selection can also be considered. One natural approach is to use ridge regression in each of the expert networks and the gating networks. This approach extends naturally to the on-line setting in the form of a "weight decay." It is also worth considering Bayesian techniques of the kind considered in the decision tree literature by Buntine (1991), as well as the MDL methods of Quinlan and Rivest (1989).

Related work

There are a variety of ties that can be made between the HME architecture and related work in statistics, machine learning, and neural networks. In this section we briefly mention some of these ties and make some comparative remarks.

Our architecture is not the only nonlinear approximator to make substantial use of GLIM's and the IRLS algorithm. IRLS also figures prominently in a branch of nonparametric statistics known as generalized additive models (GAM's; Hastie & Tibshirani, 1990). It is interesting to note the complementary roles of IRLS in these two architectures. In the GAM model, the IRLS algorithm appears in the outer loop, providing an adjusted dependent variable that is fit by a backfitting procedure in the inner loop. In the HME approach, on the other hand, the outer loop is the E step of EM and IRLS is in the inner loop. This complementarity suggests that it might be of interest to consider hybrid models in which a HME is nested inside a GAM or vice versa.

We have already mentioned the close ties between the HME approach and other tree-structured estimators such as CART and MARS. Our approach differs from MARS and related architectures—such as the basis-function trees of Sanger (1990)—by allowing splits that are oblique with respect to the axes. We also differ from these architectures by using a statistical model—the multinomial logit model—for the splits. We believe that both of these features can play a role in increasing predictive ability—the use of oblique splits should tend to decrease bias, and the use of smooth multinomial logit splits should generally decrease variance. Oblique splits also render the HME architecture insensitive to the particular choice of coordinates used to encode the data. Finally, it is worth emphasizing the difference in philosophy behind these architectures. Whereas CART and MARS are entirely nonparametric, the HME approach has a strong flavor of parametric statistics, via its use of generalized linear models, mixture models and maximum likelihood.

Similar comments can be made with respect to the decision tree methodology in the machine learning literature. Algorithms such as ID3 build trees that have axis-parallel splits and use heuristic splitting algorithms (Quinlan, 1986). More recent research has studied decision trees with oblique splits (Murthy, Kasif & Salzberg, 1993; Utgoff & Brodley, 1990). None of these papers, however, have treated the problem of splitting data as a statistical problem, nor have they provided a global goodness-of-fit measure for their trees.

There are a variety of neural network architectures that are related to the HME architecture. The multi-resolution aspect of HME is reminiscent of Moody's (1989) multi-resolution CMAC hierarchy, differing in that Moody's levels of resolution are handled explicitly by separate networks. The "neural tree" algorithm (Strömberg, Zrida, & Isaksson, 1991) is a decision tree with multi-layer perceptions (MLP's) at the non-terminals. This architecture can form oblique (or curvilinear) splits, however the MLP's are trained by a heuristic that has no clear relationship to overall classification performance. Finally, Hinton and Nowlan (see Nowlan, 1991) have independently proposed extending the Jacobs et al. (1991) modular architecture to a tree-structured system. They did not develop a likelihood approach to the problem, however, proposing instead a heuristic splitting scheme.

Conclusions

We have presented a tree-structured architecture for supervised learning. We have developed the learning algorithm for this architecture within the framework of maximum likelihood estimation, utilizing ideas from mixture model estimation and generalized linear model theory. The maximum likelihood framework allows standard tools from statistical theory to be brought to bear in developing inference procedures and measures of uncertainty for the architecture (Cox & Hinkley, 1974). It also opens the door to the Bayesian approaches that have been found to be useful in the context of unsupervised mixture model estimation (Cheeseman, et al., 1988).

Although we have not emphasized theoretical issues in this paper, there are a number of points that are worth mentioning. First, the set of exponentially-smoothed piecewise linear functions that we have utilized are clearly dense in the set of piecewise linear functions on compact sets in $\Re^m$, thus it is straightforward to show that the hierarchical architecture is dense in the set of continuous functions on compact sets in $\Re^m$. That is, the architecture is "universal" in the sense of Hornik, Stinchcombe, and White (1989). From this result it would seem straightforward to develop consistency results for the architecture (cf. Geman, Bienenstock, & Doursat, 1992; Stone, 1977). We are currently developing this line of argument and are studying the asymptotic distributional properties of fixed hierarchies. Second, convergence results are available for the architecture. We have shown that the convergence rate of the algorithm is linear in the condition number of a matrix that is the product of an inverse covariance matrix and the Hessian of the log likelihood for the architecture (Jordan & Xu, 1993).

Finally, it is worth noting a number of possible extensions of the work reported here. Our earlier work on hierarchical mixtures of experts utilized the multilayer perceptron as the primitive function for the expert networks and gating networks (Jordan & Jacobs, 1992). That option is still available, although we lose the EM proof of convergence (cf. Jordan & Xu, 1993) and we lose the ability to fit the sub-networks efficiently with IRLS. One interesting example of such an application is the case where the experts are auto-associators (Bourlard & Kamp, 1988), in which case the architecture fits hierarchically-nested local principal component decompositions. Another area in unsupervised learning worth exploring is the non-associative version of the hierarchical architecture. Such a model would be a recursive version of classical mixture-likelihood clustering and may have interesting ties to hierarchical clustering models. Finally, it is also of interest to note that the recursive least squares algorithm that we utilized in obtaining an on-line variant of Algorithm 2 is not the only possible on-line approach. Any of the fast filter algorithms (Haykin, 1991) could also be utilized, giving rise to a family of on-line algorithms. Also, it is worth studying the application of the recursive algorithms to PRESS-like cross-validation calculations to efficiently compute the changes in likelihood that arise from adding or deleting parameters or data points.

References

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics, 59*, 291-294.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees.* Belmont, CA: Wadsworth International Group.

Bridle, J. (1989). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Fogelman–Soulie & J. Hérault (Eds.), *Neuro-computing: Algorithms, Architectures, and Applications.* New York: Springer-Verlag.

Buntine, W. (1991). *Learning classification trees.* NASA Ames Technical Report FIA-90-12-19-01, Moffett Field, CA.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). Autoclass: A Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning,* Ann Arbor, MI.

Cox, D. R. (1970). *The Analysis of Binary Data.* London: Chapman-Hall.

Cox, D. R., & Hinkley, D. V. (1974). *Theoretical Statistics.* London: Chapman-Hall.

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

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

Finney, D. J. (1973). *Statistical Methods in Biological Assay.* New York: Hafner.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics, 19*, 1-141.

Fun, W. & Jordan, M. I. (1993). *The moving basin: Effective action search in forward models.* MIT Computational Cognitive Science Tech Report 9205, Cambridge, MA.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1-52.

Golub, G. H., & Van Loan, G. F. (1989). *Matrix Computations.* Baltimore, MD: The Johns Hopkins University Press.

Hastie, T. J., & Tibshirani, R. J. (1990). *Generalized Additive Models.* London: Chapman and Hall.

Haykin, S. (1991). *Adaptive Filter Theory.* Englewood Cliffs, NJ: Prentice-Hall.

Hinton, G. E. & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing: Volume 1*, 282-317. Cambridge, MA: MIT Press.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2*, 359-366.

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

Jordan, M. I., & Jacobs, R. A. (1992). Hierarchies of adaptive experts. In J. Moody, S. Hanson, & R. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4*. San Mateo, CA: Morgan Kaufmann. pp. 985-993.

Jordan, M. I., & Xu, L. (1993). *Convergence properties of the EM approach to learning in mixture-of-experts architectures.* Computational Cognitive Science Tech. Rep. 9301, MIT, Cambridge, MA.

Little, R. J. A., & Rubin, D. B. (1987). *Statistical Analysis with Missing Data.* New York: John Wiley.

Ljung, L. & Söderström, T. (1986). *Theory and practice of recursive identification.* Cambridge: MIT Press.

McCullagh, P. & Nelder, J.A. (1983). *Generalized Linear Models.* London: Chapman and Hall.

Moody, J. (1989). Fast learning in multi-resolution hierarchies. In D.S. Touretzky (Ed.), *Advances in Neural Information Processing Systems.* San Mateo, CA: Morgan Kaufmann Publishers.

Murthy, S. K., Kasif, S., & Salzberg, S. (1993). *OC1: A randomized algorithm for building oblique decision trees.* Technical Report, Department of Computer Science, Johns Hopkins University.

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

Nowlan, S.J. (1991). *Soft competitive adaptation: Neural network learning algorithms based on fitting statistical mixtures.* Tech. Rep. CMU-CS-91-126, CMU, Pittsburgh, PA.

Quandt, R.E., & Ramsey, J.B. (1972). A new approach to estimating switching regressions. *Journal of the American Statistical Society, 67*, 306-310.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81-106.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the Minimum Description Length Principle. *Information and Computation, 80*, 227-248.

Redner, R. A., & Walker, H. F. (1984). Mixture densities, maximum likelihood and the EM algorithm. *SIAM Review, 26*, 195-239.

Sanger, T. D. (1991). A tree-structured adaptive network for function approximation in high dimensional spaces. *IEEE Transactions on Neural Networks, 2*, 285-293.

Scott, D. W. (1992). *Multivariate Density Estimation.* New York: John Wiley.

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

Stone, C. J. (1977). Consistent nonparametric regression. *The Annals of Statistics,* *5*, 595-645.

Strömberg, J. E., Zrida, J., & Isaksson, A. (1991). Neural trees—using neural nets in a tree classifier structure. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 137-140.

Titterington, D. M., Smith, A. F. M., & Makov, U. E. (1985). *Statistical Analysis of Finite Mixture Distributions.* New York: John Wiley.

Utgoff, P. E., & Brodley, C. E. (1990). An incremental method for finding multivariate splits for decision trees. In *Proceedings of the Seventh International Conference on Machine Learning*, Los Altos, CA.

Wahba, G., Gu, C., Wang, Y., & Chappell, R. (1993). *Soft classification, a.k.a. risk estimation, via penalized log likelihood and smoothing spline analysis of variance.* Tech. Rep. 899, Department of Statistics, University of Wisconsin, Madison.

Wu, C. F. J. (1983). On the convergence properties of the EM algorithm. *The Annals of Statistics,* *11*, 95-103.

Appendix A – Iteratively reweighted least squares

The iteratively reweighted least squares (IRLS) algorithm is the inner loop of the algorithm that we have proposed for the HME architecture. In this section, we describe the IRLS algorithm, deriving it as a special case of the Fisher scoring method for generalized linear models. Our presentation derives from McCullagh and Nelder (1983).

IRLS is an iterative algorithm for computing the maximum likelihood estimates of the parameters of a generalized linear model. It is a special case of a general algorithm for maximum likelihood estimation known as the Fisher scoring method (Finney, 1973). Let $l(\boldsymbol{\beta}; \mathcal{X})$ be a log likelihood function—a function of the parameter vector $\boldsymbol{\beta}$—and let $(\partial l / \partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T)$ denote the Hessian of the log likelihood. The Fisher scoring method updates the parameter estimates $\boldsymbol{\beta}$ as follows:

$$\boldsymbol{\beta}_{r+1} = \boldsymbol{\beta}_r - \{E[\frac{\partial l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T}]\}^{-1} \frac{\partial l}{\partial \boldsymbol{\beta}}, \tag{38}$$

where $\boldsymbol{\beta}_r$ denotes the parameter estimate at the r^{th} iteration and $\partial l / \partial \boldsymbol{\beta}$ is the gradient vector. Note that the Fisher scoring method is essentially the same as the

Newton-Raphson algorithm, except that the expected value of the Hessian replaces the Hessian. There are statistical reasons for preferring the expected value of the Hessian—and the expected value of the Hessian is often easier to compute—but Newton-Raphson can also be used in many cases.

The likelihood in generalized linear model theory is a product of densities from the exponential family of distributions. This family is an important class in statistics and includes many useful densities, such as the normal, the Poisson, the binomial and the gamma. The general form of a density in the exponential family is the following:

$$P(y, \eta, \phi) = \exp\{(\eta y - b(\eta))/\phi + c(y, \phi)\}, \tag{39}$$

where η is known as the "natural parameter" and ϕ is the dispersion parameter.[8]

Example (Bernoulli density)

The Bernoulli density with mean π has the following form:

$$\begin{aligned} P(y, \pi) &= \pi^y (1-\pi)^{1-y} \\ &= \exp\{\ln(\frac{\pi}{1-\pi})y + \ln(1-\pi)\} \\ &= \exp\{\eta y - \ln(1+e^\eta)\}, \end{aligned} \tag{40}$$

where $\eta = \ln(\pi/1-\pi)$ is the natural parameter of the Bernoulli density. This parameter has the interpretation as the log odds of "success" in a random Bernoulli experiment.

In a generalized linear model, the parameter η is modeled as a linear function of the input $\mathbf{x}$:

$$\eta = \boldsymbol{\beta}^T \mathbf{x},$$

where $\boldsymbol{\beta}$ is a parameter vector. Substituting this expression into Equation 39 and taking the product of N such densities yields the following log likelihood for a data set $\mathcal{X} = \{(\mathbf{x}^{(t)}, y^{(t)})\}_1^N$:

$$l(\boldsymbol{\beta}, \mathcal{X}) = \sum_t \{(\boldsymbol{\beta}^T \mathbf{x}^{(t)} y^{(t)} - b(\boldsymbol{\beta}^T \mathbf{x}^{(t)}))/\phi + c(y^{(t)}, \phi)\}.$$

The observations $y^{(t)}$ are assumed to be sampled independently from densities $P(y, \eta^{(t)}, \phi)$, where $\eta^{(t)} = \boldsymbol{\beta}^T \mathbf{x}^{(t)}$.

We now compute the gradient of the log likelihood:

$$\frac{\partial l}{\partial \boldsymbol{\beta}} = \sum_t (y^{(t)} - b'(\boldsymbol{\beta}^T \mathbf{x}^{(t)}))\mathbf{x}^{(t)}/\phi \tag{41}$$

[8] We restrict ourselves to scalar-valued random variables to simplify the presentation, and describe the (straightforward) extension to vector-valued random variables at the end of the section.

and the Hessian of the log likelihood:

$$\frac{\partial l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = -\sum_t b''(\boldsymbol{\beta}^T \mathbf{x}^{(t)}) \mathbf{x}^{(t)} \mathbf{x}^{(t)T} / \phi. \tag{42}$$

These quantities could be substituted directly into Equation 38, however there is additional mathematical structure that can be exploited. First note the following identity, which is true of any log likelihood:

$$E[\frac{\partial l}{\partial \boldsymbol{\beta}}] = 0.$$

(This fact can be proved by differentiating both sides of the identity $\int P(y, \boldsymbol{\beta}, \phi) dy = 1$ with respect to $\boldsymbol{\beta}$). Because this identity is true for any set of observed data, including all subsets of $\mathcal{X}$, we have the following:

$$E[y^{(t)}] = b'(\boldsymbol{\beta}^T \mathbf{x}^{(t)}),$$

for all t. This equation implies that the mean of $y^{(t)}$, which we denote as $\mu^{(t)}$, is a function of $\eta^{(t)}$. We therefore include in the generalized linear model the *link function*, which models μ as a function of η:

$$\mu^{(t)} = f(\eta^{(t)}).$$

Example (Bernoulli density)

Equation 40 shows that $b(\eta) = \ln(1 + e^\eta)$ for the Bernoulli density. Thus

$$\mu = b'(\eta) = \frac{e^\eta}{1 + e^\eta},$$

which is the logistic function. Inverting the logistic function yields $\eta = \ln(\mu / 1 - \mu)$; thus, μ equals π, as it must.

The link function $f(\eta) = b'(\eta)$ is known in generalized linear model theory as the *canonical link*. By parameterizing the exponential family density in terms of η (cf. Equation 39), we have forced the choice of the canonical link. It is also possible to use other links, in which case η no longer has the interpretation as the natural parameter of the density. There are statistical reasons, however, to prefer the canonical link (McCullagh & Nelder, 1983). Moreover, by choosing the canonical link, the Hessian of the likelihood turns out to be constant (cf. Equation 42), and the Fisher scoring method therefore reduces to Newton-Raphson.[9]

[9]Whether or not the canonical link is used, the results presented in the remainder of this section are correct for the Fisher scoring method. If noncanonical links are used, then Newton-Raphson will include additional terms (terms that vanish under the expectation operator).

To continue the development, we need an additional fact about log likelihoods. By differentiating the identity $\int P(y, \boldsymbol{\beta})dy = 1$ twice with respect to $\boldsymbol{\beta}$, the following identity can be established:

$$E[\frac{\partial l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T}] = -E[\frac{\partial l}{\partial \boldsymbol{\beta}}][\frac{\partial l}{\partial \boldsymbol{\beta}}]^T.$$

This identity can be used to obtain a relationship between the variance of η and the function $b(\eta)$ in the exponential family density. Beginning with Equation 42, we have:

$$\begin{aligned}
-E[\sum_t b''(\boldsymbol{\beta}^T \mathbf{x}^{(t)})\mathbf{x}^t \mathbf{x}^{(t)T}/\phi] &= E[\frac{\partial l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T}] \\
&= -E[\frac{\partial l}{\partial \boldsymbol{\beta}}][\frac{\partial l}{\partial \boldsymbol{\beta}}]^T \\
&= -\frac{1}{\phi^2} E[\sum_t (y^{(t)} - b'(\boldsymbol{\beta}^T \mathbf{x}^{(t)}))\mathbf{x}^{(t)} \sum_s (y^{(s)} - b'(\boldsymbol{\beta}^T \mathbf{x}^{(s)}))\mathbf{x}^{(s)T}] \\
&= -\frac{1}{\phi^2} E[\sum_t (y^{(t)} - b'(\boldsymbol{\beta}^T \mathbf{x}^{(t)}))^2 \mathbf{x}^{(t)} \mathbf{x}^{(t)T}] \\
&= -\frac{1}{\phi^2} \sum_t \mathrm{Var}[y^{(t)}]\mathbf{x}^{(t)} \mathbf{x}^{(t)T},
\end{aligned}$$

where we have used the independence assumption in the fourth step. Comparing Equation 42 with the last equation, we obtain the following relationship:

$$\mathrm{Var}[y^{(t)}] = \phi b''(\boldsymbol{\beta}^T \mathbf{x}^{(t)}).$$

Moreover, because $f(\eta) = b'(\eta)$, we have

$$\mathrm{Var}[y^{(t)}] = \phi f'(\boldsymbol{\beta}^T \mathbf{x}^{(t)}). \tag{43}$$

We now assemble the various pieces. First note that Equation 43 can be utilized to express the Hessian (Equation 42) in the following form:

$$\frac{\partial l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = -\sum_t \mathbf{x}^{(t)} \mathbf{x}^{(t)T} w^{(t)},$$

where the weight $w^{(t)}$ is defined as follows:

$$w^{(t)} = \frac{f'(\eta^{(t)})^2}{\mathrm{Var}[y^{(t)}]}.$$

In matrix notation we have:

$$\frac{\partial l}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^T} = -X^T W X, \tag{44}$$

where X is the matrix whose rows are the input vectors $\mathbf{x}^{(t)}$ and W is a diagonal matrix whose diagonal elements are $w^{(t)}$. Note also that the Hessian is a constant, thus the expected value of the Hessian is also X^TWX.

Similarly, Equation 43 can be used to remove the dependence of the gradient (Equation 41) on ϕ:

$$\frac{\partial l}{\partial \boldsymbol{\beta}} = \sum_t (y^{(t)} - \mu^{(t)})\mathbf{x}^{(t)} w^{(t)} / f'(\eta^{(t)}).$$

This equation can be written in matrix notation as follows:

$$\frac{\partial l}{\partial \boldsymbol{\beta}} = X^T W \mathbf{e}, \tag{45}$$

where $\mathbf{e}$ is the vector whose components are:

$$e^{(t)} = (y^{(t)} - \mu^{(t)})/f'(\eta^{(t)}).$$

Finally, substitute Equation 44 and Equation 45 into Equation 38 to obtain:

$$\begin{aligned} \boldsymbol{\beta}_{r+1} &= \boldsymbol{\beta}_r + (X^TWX)^{-1}X^TW\mathbf{e} && (46) \\ &= (X^TWX)^{-1}X^TW\mathbf{z}, && (47) \end{aligned}$$

where $\mathbf{z} = X\boldsymbol{\beta}_r + \mathbf{e}$.[10] These equations are the normal equations for a weighted least squares problem with observations $\{(\mathbf{x}^{(t)}, z^{(t)})\}_1^N$ and observation weights $w^{(t)}$. The weights change from iteration to iteration, because they are a function of the parameters $\boldsymbol{\beta}_r$. The resulting iterative algorithm is known as iteratively reweighted least squares (IRLS).

It is easy to generalize the derivation to allow additional fixed observation weights to be associated with the data pairs. Such weights simply multiply the iteratively varying weights $w^{(t)}$, leading to an iteratively reweighted *weighted* least squares algorithm. Such a generalization is in fact necessary in our application of IRLS: The EM algorithm defines observation weights in the outer loop that IRLS must treat as fixed during the inner loop.

Finally, it is also straightforward to generalize the derivation in this section to the case of vector outputs. In the case of vector outputs, each row of the weight matrix (e.g., U for the expert networks) is a separate parameter vector corresponding to the vector $\boldsymbol{\beta}$ of this section. These row vectors are updated independently and in parallel.

[10]As McCullagh and Nelder (1983) note, $\mathbf{z}$ has the interpretation as the linearization of the link function around the current value of the mean.

Appendix B – Multinomial logit models

The multinomial logit model is a special case of the generalized linear model in which the probabilistic component is the multinomial density or the Poisson density. It is of particular interest to us because the gating networks in the HME architecture are multinomial logit models.

Consider a multiway classification problem on n variables $y_1, y_2, \ldots, y_n$. A natural probability model for multiway classification is the multinomial density:

$$P(y_1, y_2, \ldots, y_n) = \frac{m!}{(y_1!)(y_2!)\ldots(y_n!)} p_1^{y_1} p_2^{y_2} \ldots p_n^{y_n},$$

where the p_i are the multinomial probabilities associated with the different classes and $m = \sum_{i=1}^{n} y_i$ is generally taken to equal one for classification problems. The multinomial density is an member of the exponential family and can be written in the following form:

$$P(y_1, y_2, \ldots, y_n) = \exp\{\ln \frac{m!}{(y_1!)(y_2!)\ldots(y_n!)} + \sum_{i=1}^{n} y_i \ln p_i\}. \tag{48}$$

Taking the logarithm of both sides, and dropping the terms that do not depend on the parameters p_i, we see that the log likelihood for the multinomial logit model is the cross-entropy between the observations y_i and the parameters p_i.

Implicit in Equation 48 is the constraint that the p_i sum to one. This constraint can be made explicit by defining p_n as follows: $p_n = 1 - \sum_i^{n-1} p_i$, and rewriting Equation 48:

$$P(y_1, y_2, \ldots, y_n) = \exp\{\ln \frac{m!}{(y_1!)(y_2!)\ldots(y_n!)} + \sum_{i=1}^{n-1} y_i \ln \frac{p_i}{p_n} + n \ln p_n\}. \tag{49}$$

The natural parameters in an exponential family density are those quantities that appear linearly in the y_i (cf. Equation 39), thus we define:

$$\eta_i = \ln \frac{p_i}{p_n}. \tag{50}$$

Using $p_n = 1 - \sum_i^{n-1} p_i$ implies:

$$p_n = \frac{1}{1 + \sum_{i=1}^{n-1} e^{\eta_i}}$$

and therefore Equation 50 can be inverted to yield:

$$\begin{aligned} p_i &= \frac{e^{\eta_i}}{1 + \sum_{j=1}^{n-1} e^{\eta_j}} \\ &= \frac{e^{\eta_i}}{\sum_{j=1}^{n} e^{\eta_j}} \end{aligned} \tag{51}$$

using $\eta_n = 0$ from Equation 50. This latter expression is the "softmax" function (Bridle, 1989).

Finally, note that Equation 49 implies that $b(\boldsymbol{\eta})$ must be defined as follows (cf. Equation 39):

$$b(\boldsymbol{\eta}) = n \ln(\sum_{i=1}^{n} e^{\eta_i}),$$

which implies:

$$\mu_i = \frac{\partial b(\boldsymbol{\eta})}{\partial \eta_i} = \frac{n e^{\eta_i}}{\sum_{j=1}^{n} e^{\eta_j}} = n p_i. \tag{52}$$

The fitting of a multinomial logit model proceeds by IRLS as described in Appendix A, using Equation 51 and Equation 52 for the link function and the mean, respectively.