Learning in High Dimensions: Modular Mixture Models

Hagai Attias
hagaia@microsoft.com
Microsoft Research, USA

Abstract

We present a new approach to learning probabilistic models for high dimensional data. This approach divides the data dimensions into low dimensional subspaces, and learns a separate mixture model for each subspace. The models combine in a principled manner to form a flexible modular network that produces a total density estimate. We derive and demonstrate an iterative learning algorithm that uses only local information.

1 Introduction

High dimensional data pose a challenge that machine learning researchers are facing increasingly often. Many domains which have come to the attention of this community, such as audio and video processing, text modeling, computational biology, and finance, are characterized by high data dimensionality. This raises problems which are much less significant in lower dimensions. Among these problems is the often exponential increase in the number of parameters required for constructing models, which leads to overfitting and to an exponential increase in the computational resources required to learn these models. Another problem is the fast increase in the number of local minima of the error surfaces, which makes training considerably more difficult.

Several approaches to efficient learning in high dimensions have been proposed, among them are multiresolution kd-trees [1], mixture of tree distributions [2], and acceleration methods for the EM algorithm for sparse data [3]. Here we describe a new approach termed *modular mixture models* (MMM). This approach is based on fitting separate low-dimensional models to subsets of the data dimensions, and combining these models in a flexible modular structure to get a total density estimator.

2 Modular Mixture Models

The problem of learning in high dimension is addressed in this paper within the framework of probabilistic models (see, e.g., [6]). In this framework, one specifies a model, and infers the model parameters from a given dataset via Bayes' rule. For sufficiently large datasets, this inference is usually performed using maximum likelihood. The structure of the model may also be inferred from data. Once a model is obtained, it can be applied to solve classification and regression problems.

To describe real world data well, models are usually required to be sufficiently rich. The class of mixture models offers a convenient solution. For continuous data, it is known that a mixture of Gaussian distributions can model them to any desired accuracy, provided that the number of mixture components is sufficiently large. The parameters of a mixture model may be learned efficiently using the EM algorithm.

However, for high dimensional data the number of parameters in a mixture of Gaussians model may become prohibitively large. Using m components in l dimensions would require the order of ml^2 parameters. For e.g. handwritten digit recognition problems, the number of parameters could easily reach several millions. Such a high number would create severe problems of overfitting and bad local maxima. Whereas feature extraction methods may decrease this number, finding an effective general method for identifying the relevant features in a given dataset is a difficult open problem.

The idea of modular mixture models is to divide the data space into low-dimensional subspaces, and fit a separate mixture model to each subspace. Such a mixture model is termed a *module*. Thus we have a set of modules, where each module estimates the density of its inputs. Fitting a mixture model to low dimensional data is relatively easy to do efficiently.

The problem is, how should the different modules combine to give a total density estimator. If the subspaces are non-overlapping, one could simply take a product over the module densities to be the total density. However, the resulting model ignores correlations between subspaces, originating, e.g., from multiscale characteristics of the data, and the estimator it produces may be highly suboptimal.

MMM combine these modules using additional, hidden modules. We refer to the modules describing the data as visible modules. Each visible module is assigned a single parent module, which is hidden. A hidden module has more than one child, and is a tool to model correlations between the children. The hidden modules may be combined in the same fashion using yet additional hidden modules. Hence, MMM creates a tree of modules with a flexible structure, producing a single density estimate that takes into account correlations on multiple scales.

In the following, the structure of a module and the parent-child dependence are specified, and an EM algorithm for estimating the MMM parameters is presented. An interesting feature of this algorithm is that its update rules use only local (parent-children) information.

3 The Basic Module

The basic module is a mixture model with parent-dependent mixing proportions. The modules are indexed by $i = 1, 2, \ldots$. Module i has m_i states (or components), labeled by $s_i = 1, ..., m_i$. This module describes an l_i-dimensional 'data' vector denoted by x_i, with coordinates $x_{ij},\ j = 1, ..., l_i$. In visible modules (the leaves to the MMM tree), x_i actually corresponds to observed data variables, but in hidden modules x_i are hidden variables. We refer to x_i in either case as *input variables*. The states s_i, of course, are hidden for both hidden and visible modules.

In the present paper we focus on Gaussian components, but the discussion can be extended to include the exponential family. Given state label $s_i = s$, the input x_i is Gaussian with mean μ_{is} and precision (inverse covariance) matrix Λ_{is},

$$p(x_i = x \mid s_i = s) = g_{is}(x) = \frac{1}{z_{is}} e^{-(x-\mu_{is})^T \Lambda_s (x-\mu_{is})/2} ,$$

where $z_{is} = \det(\Lambda_{is}/2\pi)^{1/2}$ is the normalization constant.

Module i depends only on its parent module $pa(i)$. The child-parent relationship is further restricted to a dependence of the child state s_i on the parent input $x_{pa(i)}$. Hence,

$$p(x_i, s_i \mid \text{all modules}) = p(x_i \mid s_i) p(s_i \mid x_{pa(i)}) . \quad (1)$$

The child dependence on its parent is parametrized by the softmax form,

$$p(s_i = s \mid x_{pa(i)} = x) = w_{is}(x) = \frac{1}{z_{ix}} e^{a_{is}^T x} . \quad (2)$$

It is understood that a_{is} has a 0th component, such that $a_{is}^T x = a_{is,0} + \sum_j a_{is,j} x_j$. The normalization constant is $z_{ix} = \sum_s e^{a_{is}^T x}$.

4 Combining Modules

A flexible tree of modules is constructed as described above. The data dimensions are divided into subspaces, and a module is constructed for each subspace. For simplicity, we will be focusing on non-overlapping subspaces. Hidden modules are added and each visible module is assigned a single hidden module as a parent, while no hidden module remains childless. The process of constructing the tree by adding new modules as parents of existing ones continues until reaching a single module.

The joint density of our MMM is obtained by taking a product over all module densities,

$$p(\{x_i, s_i\}) = \prod_i p(x_i, s_i \mid x_{pa(i)}) . \quad (3)$$

We will use y to refer collectively to the observed (data) variables. Denote the set of visible modules (i.e. modules with no children) by V, and the set comprised of the rest by H. Then $y = \{x_i,\ i \in V\}$. The joint MMM distribution (3) defines a probabilistic latent variable model for the data via marginalization, $p(y) = \sum p(\{x_i, s_i\})$, where the sum runs over s_i for all i, and implies integration over x_i for all $i \in H$.

5 A Local Learning Algorithm

In this section we present an algorithm for training MMM. Exact maximum likelihood is clearly intractable, since it requires integrating the softmax form (2) over the hidden input variables $x_{i\in H}$, which cannot be done analytically. In [5], a variational approach has been developed for a related model. That approach approximates the conditional distribution over the continuous hidden variables given the data, $p(x_{i\in H} \mid y)$, by a Gaussian, whose parameters are inferred from the data using an iterative algorithm. Similar ideas may be applied in the present context.

However, here we take a simpler approach and use a MAP estimate for the hidden inputs. The reason is the same reason for constructing MMM in the first place. The hidden modules were introduced to combine the visible modules and model correlations between the latter. A MAP estimate of the hidden x_i's suffices to accomplish this aim. In addition, whereas exact or approximate marginalization would probably produce a more powerful model for $p(y)$, the same effect would be obtained by increasing the number of modules and keeping the MAP estimate. Finally, a MAP estimate leads to a simple and elegant algorithm with the attractive feature of local learning rules.

The locality results from the fact that using MAP decouples the states of the different modules. Consequently, each module separately estimates its parameters from its own and its parent's inputs. Given the parameter values, the inputs are updated, where the input to each module uses information only from that module and its children. This procedure is iterated to convergence.

To derive the learning algorithm, observe that the joint conditional distribution over all hidden variables reduces to the conditional over the module states times a δ-function, $p(\{x_{i\in H}\}, \{s_i\} \mid y) = p(\{s_i\} \mid y) \prod_{i\in H} \delta(x_i - \hat{x}_i(y))$, where $\hat{x}_i$ are the data dependent MAP estimates.

The joint state conditional further factorizes over the modules, since $p(\{s_i\} \mid y) \propto \prod_i p(x_i, s_i \mid x_{pa(i)})$. Hence we have

$$p(\{s_i\} \mid y) = \prod_i p(s_i \mid y) \ , \tag{4}$$

where $p(s_i = s \mid y)$ is the responsibility of state s in module i for the data vector y. This factorization is key to the local features of the learning algorithm for MMM.

Let y^n, $n = 1, ..., N$ denote the nth observed data vector. There are three types of quantities that should be obtained from the dataset:
(i) The parameters $\theta_i = \{\mu_{is},\ \Lambda_{is},\ a_{is}\}$ for all modules i.
(ii) The input variables x_i^n to all hidden modules $i \in H$ for data vector n.
(iii) The responsibilities $\gamma_{is}^n = p(s_i^n = s \mid y^n)$ of state s in module i for data vector n.

Estimating the parameters (i) is usually termed learning, whereas obtaining the inputs and responsibilities (ii,iii) is termed inference. Both tasks are accomplished by maximizing the objective function

$$\mathcal{F} = \sum_{n,i,s} \gamma_{is}^n \left[\log p(x_i^n, s_i^n = s \mid x_{pa(i)}^n, \theta_i) - \log \gamma_{is}^n \right] \tag{5}$$

w.r.t. θ_i, x_i^n, and γ_{is}^n. The first term inside the sum is the complete data likelihood, with the module states averaged over using their responsibilities, as usual in EM. The second term ensure that $\gamma_{is}^n = p(s_i^n = s \mid y^n)$ as required.

Here are the update rules for MMM.

For module i's means and precisions,

$$\begin{aligned} \mu_{is} &= \frac{\sum_n \gamma_{is}^n x_i^n}{\sum_n \gamma_{is}^n} \ , \\ \Lambda_{is}^{-1} &= \frac{\sum_n \gamma_{is}^n (x_i^n - \mu_{is})(x_i^n - \mu_{is})^T}{\sum_n \gamma_{is}^n} \ . \end{aligned} \tag{6}$$

For module i's dependence on its parent, we have the incremental rule

$$\delta a_{is} = \eta \sum_n \gamma_{is}^n \left[1 - w_{is}(x_{pa(i)}^n) \right] x_{pa(i)\cdot}^n \ , \tag{7}$$

where η is a suitably chosen learning rate.

For module i's inputs at the nth data instance, we also have an incremental rule,

$$\begin{aligned} \delta x_i^n &= -\epsilon \sum_s \gamma_{is}^n \Lambda_s (x_i^n - \mu_{is}) \\ &+ \ \epsilon \sum_{j, pa(i)=j} \sum_s \left[\gamma_{js}^n - w_{js}(x_i^n) \right] a_{js} \ , \end{aligned} \tag{8}$$

where the sum in the second term runs over the children of module i, and ϵ sets the learning rate.

Finally, for the responsibilities of module i's states,

$$\gamma_{is}^n = \frac{1}{Z_i} g_{is}(x_i^n) w_{is}(x_{pa(i)}^n) \ , \tag{9}$$

where Z_i is set to ensure $\sum_s \gamma_{is}^n = 1$.

After initialization (see below), the above iterations may be performed at any order. The scheme corresponding to ordinary EM directs us to update γ_{is}^n and x_i^n in the E-step, where the incremental rule for x_i^n is iterated to convergence. Then turn to the M-step and update μ_{is}, Λ_{is} and a_{is}, where again the incremental rule for a_{is} is iterated to convergence. Any alternative scheme, in particular early stopping in the incremental rules, corresponds to generalized EM and is guaranteed convergence.

6 Top-Down Interpretation

Viewed as a generative model for the data, MMM has a natural interpretation. Suppose the data vector y_n is an image of a handwritten digit, e.g. 2. Assume a simple MMM with four visible modules which subdivide the pixel array into equal non-overlapping parts. Add one hidden module as the top module, and connect the visible modules as his children.

Several writing styles may be used for 2, and they correspond to the different states of the top module. Moving on to the visible modules, each of the four parts of the image would display one of several possible templates. These templates are generated by the states of the module which covers that part of the image. The variance associated with each template is described by the Gaussian distribution associated with the corresponding state.

Finally, which combination of template is selected generally depends on the writing style. For one style, visible modules i, j are likely to be in states $s_i = 1,\ s_j = 2$, whereas for another style these modules are likely to be in states $s_i = 3,\ s_j = 2$. The style controls the visible modules via the dependence of their states on the top module's input, $p(s_i \mid x_{pa(i)})$.

This interpretation holds for arbitrary MMM with any configuration of hidden modules. Modules higher in the hierarchy control the coarser structure of the generated image, and lower modules add finer details.

7 Bottom-Up Interpretation

Viewed as a bottom-up network that processes the observed data, MMM perform a sequence of local clustering and local dimensionality reduction operations. Clustering, because each visible module i estimates the density of its inputs x_i, by fitting a mixture model to the data in its image patch.

Dimensionality reduction, because the inputs $\{x_j\}$ to the parents $j = pa(i)$ form a lower dimensional representation of the data $\{x_i\}$. Notice that the mapping $\{x_i\} \to \{x_j, j = pa(i)\}$ is defined implicitly by the MMM joint distribution, and is implemented by the update rules (8,9). This mapping is nonlinear, and is deterministic due to our use of the MAP estimate. It is not taken in isolation, but is coupled to the clustering performed by module i and to the rest of the network.

The sequence we have in mind is therefore (i) local clustering of inputs x_i, (ii) dimensionality reduction $x_i \to x_{pa(i)}$, (iii) repeat at the next level. Intuitively, one can view the dimensionality reduction as a linear transformation applied to the log-responsibilities $\log \gamma_{is}$ to give the inputs $x_{pa(i)}$, but strictly speaking this is only an approximation. We revisit this point in the section on initialization.

8 Considerations for Model Selection

For simplicity, we assume in this section a symmetric tree structure. Each module has m states and k children, and its input dimensionality is l. What are the optimal m, k, and l? This is a model selection question which, in principle, could be addressed using Bayesian methods. Here we focus on selecting the dimension l, and employ two different (but related) heuristic considerations.

The first consideration aims to minimize the number of parameters. Observe that we could have constructed the tree without using any hidden inputs $x_{i\in H}$, only hidden states s_i. The states in a given module i would then have depended directly on the states $s_{pa(i)}$ of its parent, rather than on the inputs to its parent. A simple calculation shows that the resulting model would have had a larger number of parameters than ours if $l < mk/(k+2)$ (precise conditions are omitted), and thus more prone to overfitting. Thus a small l is encouraged.

The second consideration aims at dimensionality reduction. One may think of the inputs x_j to module j as arising from the *outputs* of its children i, $pa(i) = j$. These outputs may be, e.g., the indicator variables associated with the states of each child, or equivalently the responsibilities γ_{is}. The dimensionality of these outputs is $(m-1)k$. Now, if the above inequality is satisfied, it implies the weaker inequality $l < (m-1)k$. Hence, minimizing the number of parameters leads to a large dimensionality reduction.

Assuming the model parameters are fixed, we choose l as follows. Perform SVD on the $\log \gamma_{is}$. Set a threshold t, and count the number n_t of eigenvalues above the threshold. Then set $l = \min(n_t, mk/(k+2))$.

9 Initialization

MMM is a parametric model with many hidden variables. Training thus has complicated dynamics which are sensitive to initial conditions. In this section we provide a careful initialization procedure. This procedure is based on the interpretation discussed above.

Let us focus on the visible modules $i \in V$ and assume their parameters have already been initialized. How should the inputs x_j to their parent modules $j = pa(i)$ be initialized? Recall the responsibilities $\gamma_{is} = g_{is}(x_i)w_{is}(x_j)/Z_i(x_i, x_j)$ (9). Taking the logarithm on both sides, using (2), and rearranging terms, we have

$$a_{is}^T x_j = \log \gamma_{is} + b_{is} \ , \tag{10}$$

where $b_{is} = -\log g_{is}(x_i) + \log Z_i(x_i, x_j) + \log z_i(x_j)$ depends on x_j via the normalization constants. Hence, if we approximate Z_i and z_i as flat w.r.t. x_j, b_{is} are constants and the inputs x_j can be initialized as linear combinations of the log responsibilities $\log \gamma_{is}$ of the children modules. Selecting the combination coefficients may be viewed as a problem of linear dimensionality reduction, where SVD could be applied.

These considerations lead the following proposal.
(i) Set the inputs x_j to the parent modules to zero, and train the means and precisions of the visible modules on the data x_i.
(ii) For each j, perform SVD on $\log \gamma_{is}$ with $pa(i) = j$, and initialize x_j to the first l_j principal components.
(iii) Proceed to the parents of modules j and apply the same procedure.

10 Handwritten Digit Classification

To use MMM for Bayesian classification, we train a separate MMM for each class c. Given a new data point y, we compute its likelihood $\log p(y \mid c) = \mathcal{F}$ from (5) for each model. Notice that, whereas the model parameters are now fixed, we still have to do inference using the update rules (8,9). The data point is assigned to class $\hat{c} = \arg\max_c p(c \mid y)$. The class posterior is obtained using Bayes' rule, $p(c \mid y) \propto p(y \mid c)p(c)$, where the class prior is measured from the training data.

The algorithm was applied to the Buffalo post office dataset, which contains 1100 examples for each digit $0 - 9$. Each digit is a gray-level 8×8 pixel array. We used 1000-digit batches for training, and a separate batch of 100 for testing. For each digit, we used MMM with 4 visible modules and one hidden module. The visible modules had $l_i = 16$, $m_i = 10$, and the hidden one $l_j = 8$, $m_j = 6$. Results gave misclassification rate of 1.9%. A single 64-dimensional, 30 component mixture of Gaussians model with full covariance matrices gave a slightly higher rate of 2.1%. More extensive tests using the MNIST dataset are currently underway.

11 Discussion

One important restriction on the structure of MMM imposed in the present paper is that each module has only one parent. In order to prevent edge effects, it would be advantageous for parents to share children. An easy way to achieve this would be allowing the states s_i of module i to depend on the inputs x_j of more than one module j, while taking care to avoid loops. This would leave the form of the update rules essentially unmodified.

Another potential cause for edge effects is the requirement of non-overlapping data variables, i.e. of x_i and x_j for $i, j \in V$ and $i \neq j$. This restriction is harder to overcome, since allowing the x_i to depend not only on s_i but also on states of other modules would change the model parametrization. One proposal is to allow the inputs of the visible modules to be a linear transformation of the data, i.e. $x_i = A_i y$, where the additional parameter matrices A_i would be estimated by maximum likelihood. The A_i are determined essentially by second order correlations between the data variables, and would tend to vanish for pixels that are far apart.

Learning the optimal structure of a MMM from data is an important issue which has been touched upon here only lightly. Whereas Bayesian methods for model selection can in principle provide a solution, in practice there are two major problems. First, exact Bayesian inference is intractable for mixture models. Second, even if it were tractable, the space of all possible structures is prohibitively large. The first problem may be addressed using variational techniques [4], and the second by local search methods [6].

Many other topics in MMM remain, among them extensions to discrete data, to non-Gaussian mixture models, and to dynamical systems.

References

[1] Moore, A. (1999). Very fast EM-based mixture model clustering using multiresolution kd-trees. In *Proc. NIPS-98*.

[2] Meila, M. (1998). Learning with mixtures of trees. *PhD thesis*, MIT.

[3] Chickering, D.M. & Heckerman, D. (1999). Fast learning from sparse data. In *Proc. UAI-99*.

[4] Attias, H. (2000). A variational Bayesian framework for graphical models. In *Proc. NIPS-99*.

[5] Attias, H. (1999). Learning a hierarchical belief network of independent factor analyzers. In *Proc. NIPS-98*.

[6] Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In *Learning in Graphical Models* (Ed. Jordan, M.I.).