Hierarchical Expectation Refinement for Learning Generative Perception Models

Thomas Dean
Department of Computer Science
Brown University, Providence, RI 02912

August 24, 2005

Abstract

We present a class of generative models well suited to modeling perceptual processes and an algorithm for learning their parameters that promises to scale to learning very large models. The models are hierarchical, composed of multiple levels, and allow input only at the lowest level, the base of the hierarchy. Connections within a level are generally local and may or may not be directed. Connections between levels are directed and generally do not span multiple levels. The learning algorithm falls within the general family of expectation maximization algorithms. Parameter estimation proceeds level-by-level starting with components in the lowest level and moving up the hierarchy. Having learned the parameters for the components in a given level, those parameters are permanently fixed and are never revisited for the purposes of learning. These parameters do, however, play an important role in learning the parameters for higher-level components by helping to generate the samples used in subsequent parameter estimation. Within levels, learning is decomposed into many local subproblems suggesting a straightforward parallel implementation. The inference required for learning is carried out by local message passing and the arrangement of connections within the underlying networks is designed to facilitate this method of inference. Learning is unsupervised but can be easily adapted to accommodate labeled data. A prototype implementation is discussed along with some preliminary experimental results on a standard benchmark machine-learning data set.

1 Introduction

Generative statistical models have been applied with some success in modeling the diverse patterns and structural regularities found in naturally occurring phenomena [11, 12]. They have provided the representational foundation for computational theories used to explain such inferential processes as biological perception [3, 25] and natural language generation [2, 20]. In the case of modeling perception, the generative models typically correspond to large, multi-layer networks consisting of many components intended to model features of the perceptual data. These networks tend to be *hierarchical*, involving multiple layers of increasingly abstract and contextually invariant features, and *compositional*, the features in one layer of the hierarchy representing relationships among features appearing in layers lower in the hierarchy. The hierarchies are often relatively shallow when compared to the total number of components — no more than a few tens of layers. The underlying graphs are sparsely connected and the shortest path between two components in a given layer is often a path that first ascends and then descends the hierarchy rather than connecting directly through the containing layer.

However attractive from a representational perspective, training such networks presents significant challenges for existing machine learning algorithms. The challenges arise in acquiring sufficient training data and in coping with the combinatorics inherent in performing inference in large networks. Labeled data for supervised learning is generally difficult to obtain and unsupervised learning in models with many hidden parameters often requires prohibitively large amounts of data and computation. Inference in large networks is often plagued with problems involving poor accuracy and slow or unpredictable convergence.

In this paper, we present a class of generative models well suited to modeling perceptual processes and an algorithm for learning their parameters that promises to scale to learning very large models. The models are hierarchical, composed of multiple levels, and allow input only at the lowest level, the base of the hierarchy. Connections within a level are generally local and may or may not be directed. Connections between levels are directed and generally do not span multiple levels.

The learning algorithm falls within the general family of *expectation maximization* algorithms [5]. Parameter estimation proceeds level-by-level starting with components in the lowest level and moving up the hierarchy. Having learned the parameters for the components in a given level, those parameters are permanently fixed and are never revisited for purposes of learning.[1] These parameters do, however, play an important role in learning the parameters for higher-level components by helping to generate the samples used in subsequent parameter estimation. Within a level, learning is decomposed into many local subproblems suggesting a straightforward parallel implementation. The inference required for learning is carried out by local message passing [26] and the arrangement of connections within the underlying networks is designed to facilitate this method of inference [22]. Learning is unsupervised but can be easily adapted to accommodate labeled data.

There are any number of approaches that have some subset of the characteristics mentioned above, e.g., [1, 6, 7, 9, 29, 32, 31]. They differ in that the basic components consist variously of attractor networks, Boltzmann machines, logistic (sigmoid) functions, vector quantizers, support-vector machines or self-organizing maps. The network structures are differentiated with respect to different flavors of artificial neural network or graphical statistical model. The work presented here draws inspiration from [10, 14, 15, 33, 34, 28].

While neither the general architecture of the proposed models nor the general approach to learning their parameters are particularly novel, the combination of representational and algorithmic techniques and the way in which they leverage existing algorithms is novel. Our primary motivation (described in [4]) is to develop an algorithm for simulating large-scale models of the neocortex based on the work of Lee and Mumford [19]. However, the general class of models and the algorithm for learning them is applicable to a variety of pattern recognition and perceptual inference problems.

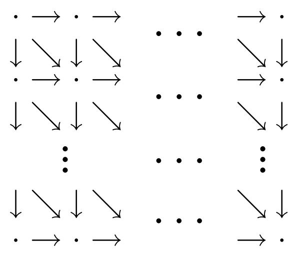
2 Generalized Pyramid Graphs

In the following discussion, we assume some knowledge of graphical models and Bayesian networks in particular [27, 24, 17], but the basic ideas do not require a deep understanding of the mathematics underlying inference in such models. Depending on the context, the term *node* will be used to refer to a random variable in the underlying statistical model or a vertex in the graph associated with the model. We begin by presenting a class of graphs called *generalized pyramid graphs*.

[1] We are exploring models in which learning continues at all levels simultaneously and indefinitely, with the rate of learning increasing and the size of the requisite training sample decreasing as you ascend the hierarchy.

$$\begin{bmatrix} n & n+1 & \cdots & n+m-1 \\ n+m & n+m+1 & \cdots & n+2m-1 \\ \vdots & \vdots & \vdots & \vdots \\ n+m^2-m & \cdots & \cdots & n+m^2-1 \end{bmatrix}$$

(a)



(b)

Figure 1: One level in a pyramid graph illustrating (a) the pyramid-graph node-numbering scheme and (b) the pattern of intra-level connections (if present) showing how this pattern together with the node-numbering scheme ensure that the graph is acyclic; the width of a level is $m = w + (p-1)(w-o)$ where w is the width of the receptive fields, p is the width of the parent level and o is the overlap between adjacent receptive fields.

A *generalized pyramid graph* is a directed acyclic graph $G = (V, E)$ whose $N = |V|$ nodes are partitioned into K levels, with level one designating the bottommost or *input* level and level K designating the topmost or *root* level. Each directed edge in E connects a node in level k to a node in level $k-1$ (*inter-level edges*) or to another node in level k (*intra-level edges*). Each *child* level k has a unique *parent* level $k+1$ for all k such that $0 < k < K$. Let i be a node in level k and $f_i = \{j : (i,j) \in E\}$ be the set of children of i in level $k-1$; f_i is said to be the *receptive field* of i. For any two adjacent nodes i and j in level k, their respective receptive fields f_i and f_j may or may not overlap.

The nodes in each level are arranged in a square grid as shown in Figure 1.a. Two nodes in a particular level are *grid adjacent* if they are next to one another in the grid either along a diagonal or along one of the four directions aligned with the grid axes. Two nodes that are grid adjacent may or may not be adjacent in G. The nodes comprising a given receptive field are arranged in a square sub grid of the grid for the level containing the receptive field.

The receptive fields for nodes that are grid adjacent to one another in a given parent level can overlap or share nodes in the child level. It is also possible for the receptive fields for two nodes that are not grid adjacent in a given level to overlap in the child level, for example, in the case in which the overlap for a level is greater than half the width of the receptive fields for that level. The overlap for a given level is specified as an integer used to determine how two receptive fields share nodes depending on the arrangement of the nodes associated with the receptive fields in the parent level.

The number of nodes in a given level — except for the top or highest numbered level — is determined by the number of nodes in the parent level and the width and overlap of the receptive fields associated with nodes in the parent level. Since the top level has no parent, the number of nodes in the top level is specified by providing the width of the top level grid. In specifying a pyramid graph, we provide the width of the top level and the width and overlap for every other level.

If there are intra-level connections among the nodes in a given level, then they always follow the same fixed pattern of connections shown in Figure 1.b. The indices 1 through N with the pattern of connections and the numbering scheme illustrated in Figure 1.a constitute a *topological sort* of the nodes in G. Such graphs are said to be *generalized pyramid graphs* since they are roughly pyramidal in shape but appear truncated if their are multiple nodes in the root level. The term *pyramid graph* is borrowed from [22] in which it is used to characterize a class of Bayesian networks for which *loopy belief propagation* works particularly well. Figure 2 provides representative examples of generalized pyramid graphs.

3 Pyramidal Bayesian Networks

A *graphical model* is a compact representation of a joint probability distribution over a set of random variables whose conditional dependencies are completely characterized by a graph in which each random variable is associated with a vertex [27, 24, 17]. Bayesian networks (BNs) and Markov random fields (MRFs) are examples of graphical models. In a given graphical model, some of the nodes are *observed* and are assigned values subsequent to inference and the remaining variables are *hidden*. A graphical model is *parameterized* by assigning each node in the graph a conditional

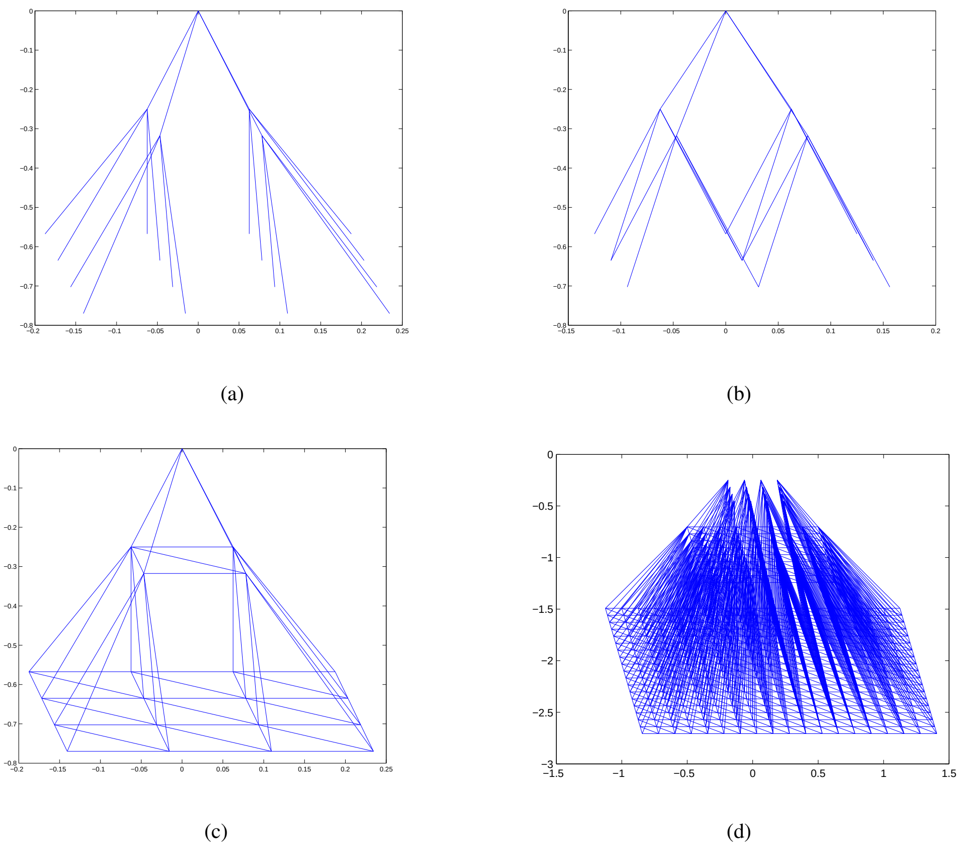


Figure 2: (a) A three-level pyramid graph with no overlap between adjacent receptive fields, (b) a three-level pyramid graph with an overlap of one, (c) a three-level pyramid graph with no overlap and intra-level edges and (d) a generalized pyramid graph with three levels, multiple roots, an overlap of one and intra-level edges in levels one and two.

probability density (CPD) (in the case of directed models such as Bayesian networks) or a potential function (in the case of undirected Markov random fields).

We use generalized pyramid graphs to represent the dependency structure of a class of graphical models called *pyramidal Bayesian networks* (PBNs). In this paper we focus on directed graphical models, but much of the discussion applies to a class of hybrid models in which the intra-level edges are undirected (and hence each level constitutes a Markov random field) and the inter-level edges are directed (and hence the entire graph with each level representing a composite (*super*) node constitutes a particularly simple Bayesian network corresponding to a Markov chain). Level one corresponds to the input level — the level in which evidence is added to the Bayesian network — and consists of all the terminal nodes in the Bayesian network.

A given level is either *homogeneous* meaning all the nodes in the level have the same CPDs or *inhomogeneous* meaning their parameters differ and are established separately. In the case of homogeneous levels, the parameters for all nodes in a given level are *tied* to expedite inference. For the models of interest here, the size of the joint space for a given level (the product of the state spaces for the set of all random variables in a given level) decreases as you ascend the hierarchy. We limit the families of allowable densities to just Gaussians and tabular CPDs though the methods described here can be extended to a broader class of continuous densities using non-parametric belief propagation methods [30, 16]. A level can have a mix of discrete (tabular) and continuous (Gaussian) nodes. The input level (level one) need not be the only level containing variables with continuous state spaces, though this is the case in the applications we have explored so far.

The basic structure of the graphical model is supplied in the form of a generalized pyramid graph and so learning corresponds to estimating the parameters of the CPDs. The sizes of the respective state spaces are either specified in advance (in the case of terminal nodes) or computed as a function of the size of their children's state spaces (in the case of non-terminal nodes). Initially every node is quantified by a uniform distribution so that every node (random variable) is independent of every other node.

In constructing the Bayesian network, we assign the size of the state space for each hidden node to be proportional to the log base two of the product of the state spaces for its children. We do so for a couple of reasons, the first being conceptual and representational and the second being practical and computational. Conceptually, a hidden node constitutes an abstraction that explains, reduces, simplifies and accounts for the variation in product space of its children. Practically, the logarithmic reduction avoids an explosion in the size of the state spaces of the nodes in higher levels. The reduction also prevents higher levels from having more information capacity than lower levels and using that capacity to over fit the data.

Suppose that each node in level L has the same number of children in the next lower level L'. And suppose that each node in level L' has the same size state space and the same number of parents in level L. In this case the size of the joint space for all the variables in level L' is proportional to the size of the joint space for all the variables in level L — no information is lost. If we construct the graph so the average number of parents is smaller than the average number of children, higher levels will have less information capacity than lower levels. This representational discipline makes sense given that the higher levels are meant to account for more abstract concepts that generalize information at lower levels.

4 Hierarchical Expectation Refinement

In this section, we describe an algorithm called *hierarchical expectation refinement* that operates by decomposing the problem of learning a large pyramidal Bayesian network (PBN) into a set of more tractable component learning problems. The problem of estimating the parameters of a PBN is decomposed into K subproblems, one for each level. Each of these subproblems is further subdivided into smaller component learning problems corresponding to graph fragments that are used to create small Bayesian networks that can be solved in isolation. Learning proceeds level-by-level from the bottom up starting from the input level. Learning within a level can proceed simultaneously, in parallel on all the component learning problems associated with that level. For simplicity of presentation, we assume that all the nodes in the BN are discrete and their corresponding CPDs are represented as conditional probability tables (CPTs). We begin with the example shown in Figure 3.

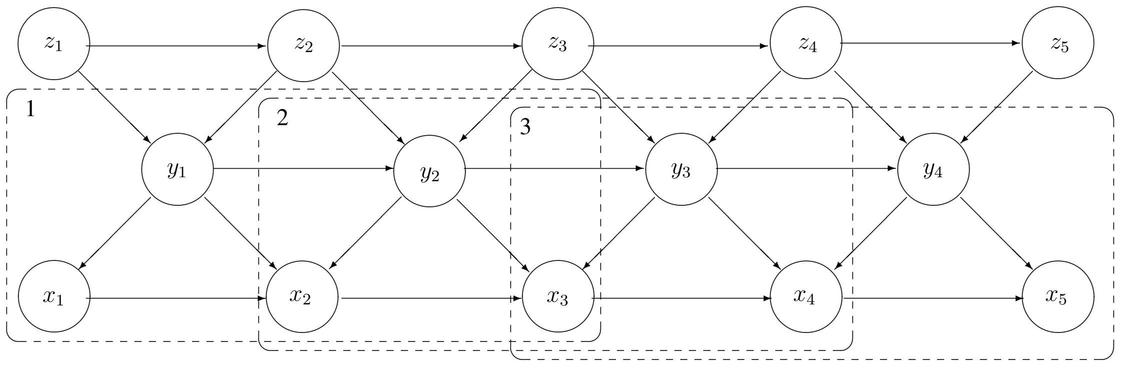
Figure 3.a depicts a fragment of a three-level PBN showing the decomposition of the first layer into three separate parameter estimation problems. The three dashed boxes enclose the variables involved in each of the component learning problems associated with the first layer. All edges that extend outside of a dashed box are eliminated for purposes of parameter estimation thereby isolating a relatively small and thus tractable subgraph characterized by the following joint distribution for the learning component labeled 1 in Figure 3.a:

$$\mathtt{Pr}(x_1, x_2, x_3, y_1, y_2) = \mathtt{Pr}(x_1|y_1)\mathtt{Pr}(x_2|x_1, y_1, y_2)\mathtt{Pr}(x_3|y_2)\mathtt{Pr}(y_1)\mathtt{Pr}(y_2|y_1)$$

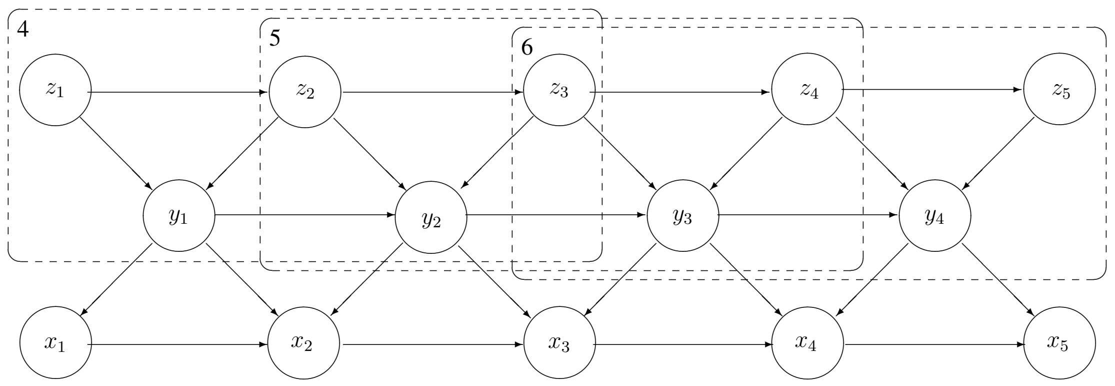
We estimate the parameters for this subgraph using expectation maximization with a training set of assignments to the variables $\{x_1, x_2, x_3\}$ — the set of variables $\{y_1, y_2\}$ are hidden. We use the estimated parameters to quantify $\mathtt{Pr}(x_1|y_1)$ and $\mathtt{Pr}(x_2|x_1, y_1, y_2)$ in the PBN and throw away the rest.

Now we generalize the above example. Assume we want to learn the parameters for a node in level k having already learned the parameters for all nodes in levels $k-1, k-2, \ldots, 1$. Recall that for any node in level k, its parents are either in level k or $k+1$. We define the *extended family* of a node x in level k as the parents of x and the children of those parents of x that are in level k. To learn the parameters associated with a node x in level k (the parameters for $\mathtt{Pr}(x|\mathtt{Parents}(x))$), we construct a component BN with a set nodes corresponding to the extended family of x. In the component BN, the nodes in level k are observed and those in level $k+1$ are hidden. We initialize the CPTs of variables in the component BN to random distributions with Dirichlet priors, collect a sufficiently large set of samples of the hidden nodes in the component, and use this set of samples as a training set for learning the component parameters. We assign the parameters of the CPT for x in PBN to be the same as those of the CPT for x in the component BN.

In the most straightforward version of the learning algorithm, we learn all the parameters for variables in level k before proceeding to level $k+1$ (see Figure 3.b for the learning components associated with level-two variables). In learning the parameters for variables in level one, the training set of assignments to variables in level one correspond to inputs. In the case of the assignments required for estimating the parameters of variables at level $k > 1$, the training data is acquired by sampling from the posterior *belief* function $\mathtt{Bel}(\vec{x}) = \mathtt{Pr}(\vec{x}|\xi)$ where $\vec{x}$ is the vector of (observed)



(a)



(b)

Figure 3: A fragment of a three-level PBN showing (a) the decomposition of the first layer into three separate parameter estimation problems and (b) an analogous decomposition of the second layer.

variables in the component learning problem at level k and ξ is an assignment to the (observed) variables in the input level of the PBN.

That's really all there is to the algorithm. It's simple to explain and relatively simple to implement given a suitable support library such as Kevin Murphy's Bayes Net Toolbox [21] (`http://www.cs.ubc.ca/~murphyk/Software/`) or Jeff Bilmes and Geoff Zweig's Graphical Model Tool Kit (`http://ssli.ee.washington.edu/~bilmes/gmtk/`). There are a number of obvious optimizations. Instead of creating a separate component problem for each node in a level, we compute the maximal sets [35] of the collection of sets corresponding to the extended families for each node. We then use these maximal sets to construct the set of component BNs. When we estimate the parameters of a component BN, we use the learned parameters to quantify all those variables whose extended family is subset of the maximal set associated with the component BN thereby reducing the total number of component problems considerably. For an homogeneous level, we construct one component BN which effectively ties the parameters for all the nodes in that level.

For relatively small networks consisting of a thousand or so discrete and Gaussian nodes, some variant of Lauritzen and Spiegelhalter's junction-tree algorithm [18] for exact inference works fine to update the belief function: $\texttt{Bel}(x) = \texttt{Pr}(x|\xi)$. For larger networks, the class of generalized pyramid graphs is one for which loopy belief propagation appears to work rather well [22] and non-parametric methods such as [30, 16] offer the prospect of using a richer set of CPD families.

5 Implementation and Preliminary Experiments

The algorithm described in the previous section was designed to be implemented on a compute cluster using MPI (Message Passing Interface) code to handle communication between processes responsible for the component learning

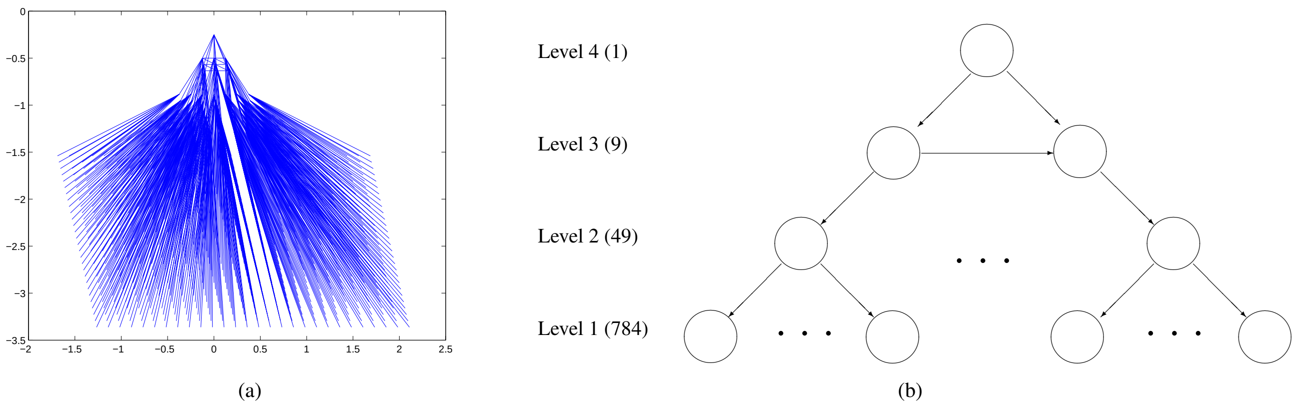


Figure 4: The four-level PBN used in the experiments with MNIST data set.

problems. While we haven't yet realized a parallel implementation, we have developed a non-parallel prototype in Matlab using Kevin Murphy's Bayes Net Toolbox [21] (BNT). The prototype is available for download at `http://www.cs.brown.edu/~tld/projects/cortex/prototype.tar.gz` and includes all of the code required to replicate the experiments discussed in this section.

In addition to an implementation of the algorithm described in the previous section, the supplied code includes a demonstration of the algorithm running on Yann LeCun's MNIST data set.[2] Our objective here was not to compete with the performance of the best machine learning algorithms on this data set (some of them are capable of recognition rates surpassing humans), but rather simply to provide a first-pass proof of concept.

We constructed the four-level PBN shown in Figure 4 with a 28×28 input level corresponding to the size of images in the MNIST data set. Nodes in levels four and three have 3×3 receptive fields with an overlap of 1 (the overlapping fields are in levels three and two). Nodes in level two have 4×4 receptive fields that do not overlap (the receptive fields are in level one). Only level three has intra-level edges. The nodes in level one are Gaussians to accommodate the eight-bit depth (255 possible values) of the pixels in the images. All other nodes are discrete valued with level-two nodes having sixteen possible values and the others having ten possible values. Level two was made homogeneous on the assumption that the level-two nodes would capture local image features such as oriented strokes and boundaries.

The choice of sixteen values for level three nodes was — apart from the number being a pleasing power of two — arbitrary. The nodes in level two implement mixture-of-Gaussians classifiers. Since level two is homogeneous, we need only learn one such classifier trained on a random sample of 4×4 image subregions. Figure 5 provides some insight into what the classifier actually does learn. For each of the sixteen learned classes, we generated 100 samples. Figure 5 is partitioned into sixteen pairs of subplots — one for each class — arranged in an eight-by-two matrix. The left subplot in each pair shows the 100 samples for the corresponding class plotted as sequences of sixteen values, one for each pixel in a 4×4 image patch. The right subplot in each pair shows a 4×4 contour plot of intensities corresponding to the averages of the corresponding pixel values in the 100 samples.

We're going to ignore the single component corresponding to the only node in level four for reasons that will soon become apparent. Given this exclusion, the PBN in Figure 4 is decomposed into one component learning problem for level three, four component problems in level two, and one component, associated with the mixture-of-Gaussians classifier, accounting for all the nodes in level one. Conceptually, think of there being a three-level pyramid graph that takes as input the output of a filter that produces 7×7 feature maps with four-bit depth from the original 28×28 images with eight-bit depth. The filter applies the mixture-of-Gaussians classifier to each of the 49 non-overlapping 4×4 regions tiling the 28×28 image.

We initialize the nodes in the PBN to a default (uniform) CPT: $\mathtt{Pr}(x|\mathtt{Parents}(x)) = (1/|\Omega_x|, ..., 1/|\Omega_x|)$ where Ω_x denotes the state space for the random variable x.[3] This default CPT results in the same belief functions as the

[2]The original database of handwritten digits is available from the National Institute of Science and Technology (NIST). LeCun has taken a subset of images from this database and size-normalized each digit and centered it in a fixed-sized image to produce a data set consisting of 30,000 training images of digits produced by a set of 250 writers and 10,000 test images produced by an additional set of 250 writers disjoint from the first set. LeCun's data set is available at `http://yann.lecun.com/exdb/mnist/`.

[3]This is particularly easy to arrange in BNT using the form: `bnet.CPD{i} = tabular_CPD(bnet, i, 'CPT', 'unif')` where `bnet` is a BNT Bayesian network structure and `i` is the index of a node.

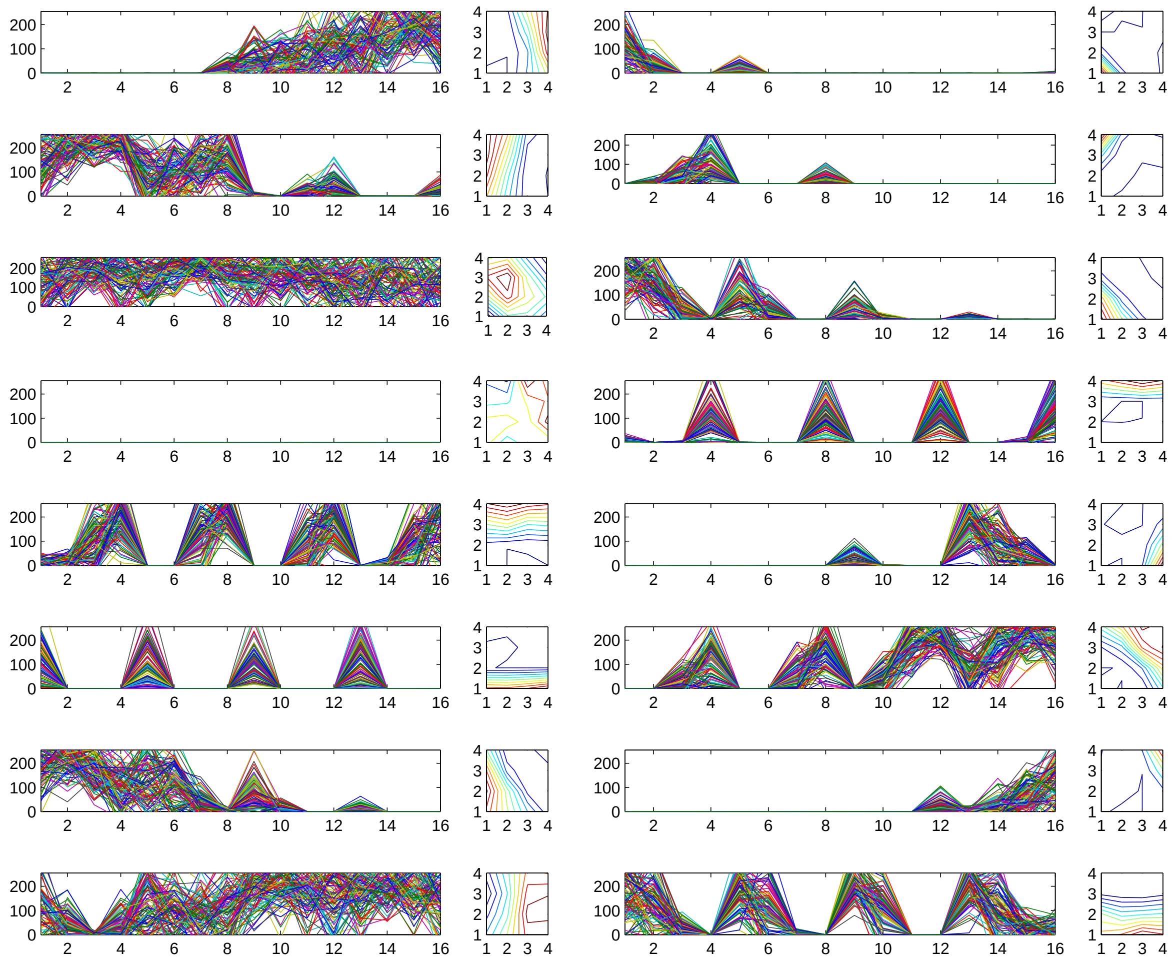


Figure 5: Two visualizations of the mixture-of-Gaussians classifier learned from the MNIST data set.

case in which we start from a forest of small trees and build the PBN from the bottom up adding nodes as we learn their associated parameters. We initialize the nodes in the component learning problems to random distributions with Dirichlet priors.[4]

For the experiments involving the MNIST data set, we learned the mixture-of-Gaussians classifier using a sample of between 10,000 and 40,000 4×4 image patches depending on the experiment. Assuming the PBN shown in Figure 5 there are four component BNs covering the 49 nodes in level two. We learn these four components in an unsupervised manner using between 1,000 and 30,000 training images and the BNT implementation of EM with a maximum of ten iterations and a threshold of 10^{-3}. There is only one learning component covering the nine nodes in level three and we learn this in a supervised manner using the image labels and the same images we used in learning the level-two components. In this case, however, we compute the posterior belief function to sample the level-three variables (the observed nodes in the solitary level-three component) assigning the level-one variables from the training images as evidence. We use the level-three samples plus the corresponding image labels to construct complete assignments to the variables in the level-three component BN. Since we have complete assignments, we simply calculate frequencies and assign the resulting parameters to the nodes in both level three and four (the single root node).

During evaluation, we assign the level-one nodes to the pixel values in an image, compute the MAP assignment for the root node, and compare it with the known image label. Since our hope is that hierarchical expectation refine-

[4]Again, BNT makes this initialization particularly easy to arrange using the form: `bnet.CPD{i} = tabular_CPD(bnet, i, 'prior_type', 'dirichlet', 'dirichlet_type', 'BDeu', 'dirichlet_weight', equivalent_sample_size)` where `bnet` is a BNT Bayesian network structure, `i` is the index of a node and `equivalent_sample_size` is a positive integer or *pseudo count* used to specify the conjugate (Dirichlet) prior.

ment offers an efficient alternative to applying expectation maximization, we compare the performance of expectation refinement (ER) to applying the expectation maximization (EM) directly. To give EM a little better chance, we set the maximum number of iterations to 20. The experiments were run on a 2.3 GHz PowerPC G5 with 1GB of memory. Here are the results of our preliminary experiments.

ALGORITHM	# PATCHES	# TRAINING	# TESTING	% TRAINING	% TESTING	CPU HOURS
ER	10,000	1,000	1,000	99.5	52.70	2.89
ER	20,000	10,000	5,000	99.6	77.86	20.16
EM	10,000	1,000	1,000	100.0	50.03	2.18
EM	20,000	10,000	5,000	99.4	81.34	19.85

As expected this doesn't compare with the best classifiers. It is, however, quite respectable performance given that we use only a small portion of the training set (less than 10% of the full set of 60,000 training images), most of the model parameters were learned without supervision, and no attempt was made to tailor the network to the problem aside from having the size of the input level match the resolution of the images. The expectation refinement algorithm spends most of its time running expectation maximization on the component BNs. If we had learned the four level-two component BNs in parallel using separate processors, the time on the large training set would have been reduced from a little over 20 hours to less than 10. These results are clearly preliminary but just as clearly they suggest that the basic idea merits further study. In the off chance that the algorithm proves useful and has not yet appeared in the literature, I would like to christen it HERA for Hierarchical Expectation Refinement Algorithm.[5]

6 Related Work

Hinton, Osindero and Teh [15] present a hybrid model combining Boltzmann machines and directed acyclic Bayesian networks and an improvement and generalization on Hinton's method of contrastive divergence [13]. The authors present a greedy, level-by-level approach to learning a generative model not, as they point out, unlike the model presented in Lee and Mumford [19]. They claim "we can learn the weights one layer at a time with a guarantee[6] that we will never decrease the log probability of the data under the full generative model." They later qualify this claim as it applies to the particular approximation proposed in the paper. The authors provide an analogy to boosting [8] which applies equally well to the algorithm described in this paper: "the greedy algorithm resembles boosting in its repeated use of the same 'weak' learner, but instead of re-weighting each data-vector to ensure that the next step learns something new, it re-represents it."

Hinton, Osindero and Bao [14] describe generative models in which each layer in a hierarchy of layers corresponds to a Markov random field with causal arcs from parent layer to child layer. This seems a natural instantiation of the Lee and Mumford cortical model. Fei-Fei and Perona [6] and Fei-Fei, Fergus and Perona [7] present algorithms for learning to classify images in new categories by relying on a small number of examples, having learned to correctly classify images from other categories (which presumably rely on features shared by the new categories).

We are essentially implementing expectation maximization in a distributed but highly structured model (hierarchically arranged in levels with topologies that embed in the plane — or nearly so). We expect that for many perceptual learning problems the particular way in which we decompose the parameter-estimation problem (which is related to Hinton's weight-sharing and parameter-tying techniques) will avoid potential problems due to early (and permanently) assigning parameters to the lower layers. If problems do arise, it may be possible to use some variant of Hinton, Osindero and Teh's "up-down" back-fitting algorithm. This perspective on EM is in keeping with Neal and Hinton's "A view of the EM algorithm that justifies incremental, sparse, and other variants" [23].

We were originally attracted to the idea of level-by-level parameter estimation by reading George and Hawkins' "A hierarchical Bayesian model of invariant pattern recognition in the Visual Cortex" [10]. In the algorithm described in [10], each node in a Bayesian network collects instantiations of its children that it uses to derive the state space for its associated random variable. The states that comprise the state space for a given node correspond to perturbed versions (exemplars) of the most frequently appearing vectors of instantiations of its children. Their algorithm uses a distance

[5]Hera is the queen of the Olympian deities. My one reservation in invoking her name is that she tried to prevent the birth of Artemis in a fit of rage over the infidelity of her husband Zeus. Artemis is also the name of an outreach program for young girls I am quite attached to and have helped run at Brown University for the last six years (`http://www.cs.brown.edu/orgs/artemis/`).

[6]The guarantee is on the *expected* change in the log probability.

metric (Hamming distance) to map vectors of instantiations of its children to the nearest exemplar. Generalizing on their published algorithm, each node employs vector quantization to learn a set of code vectors (the states of the state space) and a vector transformation or *codec* to map instantiations of its children into its state space. Once a node has learned its state space and codec, it is straightforward to collect samples and quantify the conditional probability tables for each of its children.

Learning proceeds level-by-level starting from the input level. The samples for the first-level nodes come from observations. Samples for higher levels can be acquired recursively by either performing inference on that portion of the Bayesian network whose nodes are already quantified or by using the codecs to propagate observations up through the network. This basic algorithm is simple to implement (see Appendix A) by utilizing existing libraries for vector quantization and inference in Bayesian networks.

References

[1] J.A. Anderson. The BSB model: A simple nonlinear autoassociative neural network. In M. Hassoun, editor, *Associative Neural Memories*. Oxford University Press, New York, NY, 1993.

[2] Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts, 1993.

[3] P. Dayan and L. F. Abbott. *Theoretical Neuroscience*. MIT Press, Cambridge, MA, 2001.

[4] Thomas Dean. A computational model of the cerebral cortex. In *Proceedings of AAAI-05*, pages 938–943, Cambridge, Massachusetts, 2005. MIT Press.

[5] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39 (Series B):1–38, 1977.

[6] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. In *Proceedings of CVPR-05 Workshop on Generative-Model Based Vision*, 2004.

[7] Li Fei-Fei and Pietro Perona. A Bayesian hierarchical model for learning natural scene categories. In *CVPR*, 2005.

[8] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.

[9] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernnetics*, 36(4):93–202, 1980.

[10] Dileep George and Jeff Hawkins. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2005.

[11] Ulf Grenander. *General Pattern Theory: A Mathematical Study of Regular Structures*. Oxford University Press, New York, NY, 1993.

[12] Ulf Grenander. *Elements of Pattern Theory*. Johns Hopkins University Press, Baltimore, Maryland, 1996.

[13] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[14] Geoffrey E. Hinton, Simon Osindero, and Kejie Bao. Learning causally linked Markov random fields. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics, January 6–8, 2005, Savannah Hotel, Barbados*, pages 128–135. Society for Artificial Intelligence and Statistics, 2005.

[15] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Submitted to *Neural Computation*, 2005.

[16] M. Isard. PAMPAS: real-valued graphical models for computer vision. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition: Volume I*, pages 613–620, 2003.

[17] M.I. Jordan, editor. *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1998.

[18] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–194, 1988.

[19] Tai Sing Lee and David Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America*, 2(7):1434–1448, July 2003.

[20] Chris Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

[21] Kevin Murphy. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33, 2001.

[22] Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy-belief propagation for approximate inference: An empirical study. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufmann, 2000.

[23] R.M. Neal and G.E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M.I. Jordan, editor, *Learning in Graphical Models*. MIT Press, Cambridge, MA, 1998.

[24] Richard E. Neapolitan. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. John Wiley and Sons, New York, 1990.

[25] Randall O'Reilly and Yuko Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MIT Press, Cambridge, Massachusetts, 2000.

[26] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[27] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, California, 1988.

[28] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, November 1999.

[29] D. Ross and R. Zemel. Multiple cause vector quantization. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, Cambridge, MA, 2003. MIT Press.

[30] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition: Volume I*, pages 605–612, 2003.

[31] Heiko Wersing, Julian Eggert, and Edgar Körner. Sparse coding with invariance constraints. In *International Conference on Artificial Neural Networks*, pages 385–392, 2003.

[32] Heiko Wersing and Edgar Körner. Learning optimized features for hierarchical models of invariant object recognition. *Neural Computation*, 15:1559–1588, 2003.

[33] Laurenz Wiskott. How does our visual system achieve shift and size invariance? In J. L. van Hemmen and T. J. Sejnowski, editors, *Problems in Systems Neuroscience*. Oxford University Press, 2003.

[34] Laurenz Wiskott and Terrence Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[35] Daniel Yellin. Algorithms for subset testing and finding maximal sets. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 386–392, 1992.

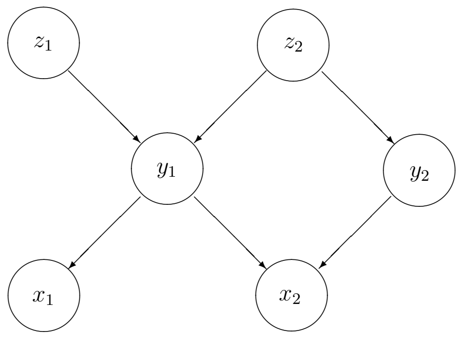


Figure 6: A fragment of a pyramid graph used to illustrate the process of learning state spaces and conditional probability tables.

Appendix A: Hierarchical Learning Using Vector Quantization

In the following, we present a generalization of the algorithm described by George and Hawkins in [10]. To understand the process of learning state spaces and conditional probability tables, consider the fragment of a pyramid graph shown in Figure 6. Suppose that we already know the state space for the variables x_1 and x_2, but we don't know the state space for any of the other variables. Let's work through the process of learning the state spaces for y_1 and y_2 and constructing the CPTs for x_1 and x_2. Note that this process will not involve the parents of y_1 and y_2; z_1 and z_2 are shown primarily to make the point that the parents of a node play no role in the determination of the size of its state space.

To construct the CPT for $\texttt{Pr}(x_1|y_1)$, we need to map each entry in a sample drawn from Ω_{x_1} to an appropriate element in Ω_{y_1}. We use vector quantization to obtain Ω_{y_1} (a set of code vectors) and a mapping f_{y_1} (codec) by analyzing a sample of vectors from the cross product $\Omega_{x_1} \times \Omega_{x_2}$. Why? Think of the random variable y_1 as an abstract feature representing a relationship involving a set of more concrete (less abstract) features $\{x_1, x_2\}$. The CPTs conditioned on y_1 ($\texttt{Pr}(x_1|y_1)$ and $\texttt{Pr}(x_2|y_1, y_2)$) have to account for the variation observed in $\Omega_{x_1} \times \Omega_{x_2}$. Similarly, we use scalar quantization to get Ω_{y_2} and f_{y_2} by analyzing a sample of scalars drawn from Ω_{x_1}.

Once we have Ω_{y_1} and Ω_{y_2} we can can construct the CPTs for $\texttt{Pr}(x_1|y_1)$ and $\texttt{Pr}(x_2|y_1, y_2)$. In the case of $\texttt{Pr}(x_1|y_1)$, we collect a sample $S = \{(x, y)|x \in \Omega_{x_1}, y = f_{y_1}(x))\}$. If x_1 is an observed variable (terminal node), the sample is drawn from observation. If x_1 is a hidden variable (non-terminal), then the sample is constructed (recursively) from the children of x_1 using previously learned codecs. Given a sufficiently large S we approximate $\texttt{Pr}(x_1 = x|y_1 = y)$ as the number of times (x, y) appears in S divided by the number of times $(., y)$ appears in S. The case of $\texttt{Pr}(x_2|y_1, y_2)$ is similar except that the sample looks like $\{(x, y, y')|x \in \Omega_{x_2}, y = f_{y_1}(x), y' = f_{y_2}(x)\}$. In describing the general algorithm, the following definitions will come in handy:

Definition: the *family* of a node i is the set consisting of i and the parents of i.

Definition: a *family assignment* for a node i is an assignment of values to the nodes (random variables) in i's family.

In the following description, we assume that each node is associated with an instance of a data structure with the following fields:

1. the index i of the node,
2. an array of indices corresponding to the parents of i,
3. an array of indices corresponding to the children of i,
4. the codec used to map vectors of samples of the children of i to the state space (code vectors) for i,
5. the CPT for i,

6. the assignment to i in the current sample (see Step 1 below),
7. a set of assignments to the children of i to be used in learning the state space and codec for i, and
8. a set of family assignments for i to be used in learning the CPT for i.

We also assume that all of the children of nodes in level k reside in level $k-1$ and all the parents of nodes in level k reside in level $k+1$. This assumption does not hold in the case of there being intra-level edges in the pyramid graph and represents a limitation of this algorithm.

For each node i in level k, we use vectors constructed from concatenations of assignments to the children of i in level $k-1$ to define the state space and codec for i. Once you have defined the state space and codec for a node in level k, you can learn the CPTs for its children in level $k-1$; for each node i in level $k-1$, collect samples of its parent nodes in level k to use in quantifying the CPT for i. To collect a sample from nodes in level k, you first need a sample for the nodes in level $k-1$; given a sample of the nodes in level $k-1$, use the codecs for the nodes in level k applied to the sample from $k-1$ to obtain the sample for level k.

The algorithm consists of performing the following three steps on each iteration until the nodes in all levels are learned. Note that on each iteration a new sample is acquired for the subset of nodes that have codecs, and each node whose parents have codecs adds a new family assignment to the set that will be used to compute its CPT.

1. Generate the sample for this iteration — recurse starting from the leaves (the observed nodes):
   (a) If a node has already been sampled this iteration, then don't bother to recurse any further.
   (b) If a node doesn't have a codec as yet, then don't bother to recurse further.
   (c) If a node has a codec and has not been sampled, then use the codec to sample the node, mark it as sampled and recurse on its parents.
2. Add to the local node samples for learning CPTs — recurse starting from the leaves:
   (a) If all the nodes in the family of a node have been sampled, then add the family sample to the set of samples used to learn the CPT for the node and recurse on the parents.
   (b) If a node hasn't been marked as sampled (see the previous step), then add the sample of children to the set of samples used to learn the state space and codec for a node.
   (c) If a node has been sampled but at least one of its parents has not, then recurse on its parents.
3. Check sample sizes and when appropriate initiate parameter learning — recurse starting from the leaves:
   (a) If the node doesn't have a codec and the size of its sample of instantiations of its children is sufficiently large, then use this sample to learn the state space and codec for the node.
   (b) If the node has a codec but not a CPT and the size of its sample of instantiations of its family is sufficiently large, the use this sample to learn the parameters of the CPT for this node.
   (c) If the node has a codec and a CPT, then recurse on its parents.