INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A Hierarchical Community of Experts

by

Brian Sallans

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

A Hierarchical Community of Experts

Brian Sallans

Master of Science

Graduate Department of Computer Science

University of Toronto

1998

The unsupervised extraction of high-level representations from data is a difficult problem. Many hierarchical graphical models have been proposed to solve this problem, but all suffer from representational difficulties or problems in generalizing the architecture to multiple layers.

A multilayer graphical model is proposed that avoids the difficulties of earlier models. The model combines binary and linear stochastic units in a novel way: The binary units gate the outputs of the linear units. Inference is intractable, so a Markov chain Monte Carlo approximation scheme called Gibbs sampling is used. Normal Gibbs sampling requires that the Markov chain reach equilibrium before samples are used for learning. It is shown that "brief" Gibbs sampling can be used for learning, where the samples are used even though the Markov chain is far from equilibrium. Simulations demonstrate that the network can extract high-level representations from some small but interesting data sets.

Acknowledgements

I am grateful to the many people who have contributed to this thesis, and to making my time in the Computer Science graduate program a great learning experience. First I would like to thank my thesis advisor Geoffrey Hinton for his patience, guidance and seemingly boundless enthusiasm. I would also like to thank Radford Neal for his honest and accurate input. I owe a great deal to Alberto Mendelzon and Vassos Hadzilacos, who have encouraged and supported me during my time at the University of Toronto.

Current and former members of the Neural Networks Research Group have greatly contributed to this thesis, and to making the group a fun place to work: Zoubin Ghahramani, Andrew Brown, Brendan Frey, Alberto Paccanaro, Mike Revow and Yee Whye Teh. Additional thanks to Zoubin Ghahramani for factor analysis, independent component analysis and rectified Gaussian belief net Matlab code. This research has also benefited from the constant flow of visitors to the Neural Networks lab. Particularly I would like to thank Josh Tenenbaum, Chris Williams, Ryohei Nakano and Naonori Ueda for useful discussions.

My love and thanks to my parents and brother for their constant encouragement and support.

I was financially supported by a Natural Sciences and Engineering Research Council post-graduate scholarship. My work has been financially supported by grants from the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

Contents

List of Tables

List of Figures

Chapter 1

Background

1.1 Introduction

Connectionist models are often used to perform classification tasks. In a classification task there is a set of observations and a set of classes. The task is to learn the mapping from observations to classes so that new observations can be assigned to the correct class, or so that a probability distribution can be computed over possible class assignments. In practice, classification tasks are learned by networks trained with supervised learning algorithms. In supervised learning all of the training cases are labeled, and the error between the predicted and actual label is minimized over the entire training set.

Supervised learning is efficient because it makes use of information about the desired targets as well as information about the input data. Unfortunately, supervised learning requires labeled data that can be expensive or impossible to obtain. Also, there is something unsatisfying about supervised learning: We humans only have to be told the class label of a few examples of something before we can recognize objects of a similar kind.

A model could learn a classification task with a minimum of supervision if first an unsupervised learning algorithm was used to cluster the data, and then a supervised

algorithm was used to learn labels for each of the clusters. This way we need only a few labeled training cases for each class. The model first extracts a high-level representation of the data in an unsupervised fashion by making use of underlying statistical structure in the data. Once this structure is learned it is easier to learn the classification task. Of course, this assumes that the underlying statistical structure in the data corresponds to the classes to be learned. As long as class membership is assigned based on some kind of clusters in the data, this is a reasonable assumption.

With unsupervised learning there are no target values, so we must minimize something other than residual error between predicted and actual targets. Density estimation is one alternative. If we assume that the observations have been drawn from some probability distribution, then by learning this distribution we can gain some insight about the data. First we must make assumptions about the form of the distribution. Typically we assume that it belongs to some class of distributions for which we have a parameterized model. Then we can optimize the parameters of the model so as to maximize the probability that the distribution would generate the given observations. Because we assume that the model generates the data, it is called a *generative model.* We call a set of parameters which maximize the probability that the model would generate the given data a *maximum-likelihood* (ML) set of parameters. The generative model can be used for classification if there are unobserved random variables which can be interpreted as class labels. The posterior distribution over these hidden variables yields a soft classification of a previously unseen datum. The process of calculating the posterior over the hidden units is called *inference*, and the optimization of the model parameters is called *learning.*

The purpose of this thesis is to investigate a generative model for unsupervised learning of statistical structure in real-valued data. This model, called a hierarchical community of experts, combines real-valued and binary stochastic units in a novel way. The goal is to find a model that can extract efficient high-level representations from data, and use these representations to learn minimally-supervised classification tasks.

1.2 Organization of this thesis

This chapter begins with a brief explanation of the Expectation-Maximization (E-M) algorithm, one of the most common methods of learning ML parameters in generative models. Because inference plays a central role in the E-M algorithm, we then discuss methods of inference including exact, variational and Monte Carlo methods. We then review several previous models: factor analysis (section 1.5.1), mixture models (section 1.5.2), logistic belief networks (section 1.5.3), independent component analysis (section 1.5.4) and rectified Gaussian belief networks (section 1.5.5). In chapter 2 we motivate the new model, and describe how inference and learning are performed. In chapter 3 we show the results of learning on some small tasks. In particular, we show that a hierarchical community of experts can extract interesting representations from data and use them to learn a minimally-supervised classification task for a simple binary classification problem (section 3.2.1). In chapter 4 we discuss some attributes of the model, including some shortcomings and possible solutions. Finally we consider possible modifications and extensions to the model.

1.3 The Expectation-Maximization algorithm

The Expectation-Maximization (E-M) algorithm is an iterative method for learning ML parameters of a generative model where some of the random variables are observed, and some are hidden [Dempster et al., 1977]. The hidden random variables might represent quantities that we think are the underlying causes of the observables. For example, a model designed to explain data consisting of shoe size and reading ability might use age as a hidden variable. The hidden variables could be continuous as in the above example, or discrete as in the case of class labels.

Let $\mathbf{x}$ be the values of the visible variables, $\mathbf{y}$ be the values of hidden variables, and let θ be the parameters of the model. As the name implies there are two steps to the algorithm:

Expectation (E) step: Calculate the distribution $P(\mathbf{y}|\mathbf{x};\theta)$ over the hidden variables, given the visible variables and the current value of the parameters.

Maximization (M) step: Compute the values of the parameters θ^* that maximize the expected log-likelihood under the distribution found in the E-step:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} \left\{ \mathrm{E}\left[\sum_{\mathbf{x}} \log P(\mathbf{x},\mathbf{y}|\theta) \right]_{P(\mathbf{y}|\mathbf{x};\theta)} \right\} \\ &= \arg\max_{\theta} \left\{ \sum_{\mathbf{y}} \sum_{\mathbf{x}} P(\mathbf{y}|\mathbf{x};\theta) \log P(\mathbf{x},\mathbf{y}|\theta) \right\} \end{aligned} \tag{1.1}$$

and set $\theta \longleftarrow \theta^*$.

So the E-step involves inferring the distribution over hidden units, and the M-step involves learning new parameters. It can be shown that if these two steps are repeated the true log-likelihood will increase, or stay the same if a maximum has already been reached.

Notice that the M-step might require solving a difficult non-linear optimization prob-

lem. It is sometimes natural to implement a partial M-step instead, where we just find a set of parameters that improve the expected log-likelihood instead of fully maximizing it. For example, gradient ascent-based partial M-steps are quite common. Algorithms that use a partial M-step are called generalized E-M algorithms (GEM), and they are also guaranteed to improve the true likelihood.

The E-step can also be very difficult, depending on the form of the posterior. Sometimes we must resort to approximate inference, where instead of finding the true posterior, we find an approximation to the true posterior. The approximation is then used to compute the expectation required in the M-step. As we can see, inference is central to the problem of learning ML parameters with the E-M algorithm. Performing exact or approximate inference is a very important problem that must be solved both to learn the parameters of a generative model, and to use the model once ML parameters have been found.

1.4 Inference

1.4.1 Exact inference

In some cases there is no need for approximate inference because we can efficiently compute the correct posterior distribution. Examples include the case of linear-Gaussian models; and singly-connected graphical models, where some variant of probability propagation can be used to compute the posterior correctly[Pearl, 1988]. Factor analysis is an example of the former (see section 1.5.1), and the hidden Markov model [Baum and Petrie, 1966] is an example of the latter. Kalman filters can be both singly-connected and Gaussian [Kalman, 1960].

The advantages of exact inference are obvious: speed and precision. These properties account for the wide adoption of models in which exact inference is possible. Unfortu-

nately exact inference is only possible for a very restricted range of architectures[1], and these do not seem to be expressive enough to capture complicated phenomena such as vision.

1.4.2 Variational methods

Consider a parameterized distribution Q over hidden variables. Given a metric that measures the difference between this distribution and the true posterior P we can optimize the parameters of Q to approximate P. The approximation Q is called a variational approximation, and the parameters are variational parameters. There is a suitable distance metric called the Kullback-Leibler divergence between Q and P:

$$\mathrm{KL}(Q\|P) = \int Q(\mathbf{y}|\mathbf{x}) \log \frac{Q(\mathbf{y}|\mathbf{x})}{P(\mathbf{y}|\mathbf{x})} d\mathbf{y} \tag{1.2}$$

where $\mathbf{y}$ is a vector of hidden unit activities, and $\mathbf{x}$ is an observation.

To evaluate (1.2) directly we would need to evaluate $P(\mathbf{y}|\mathbf{x})$ which is what we are trying to approximate in the first place. We can get around this difficulty by evaluating the *free energy* which is a function of the joint probability of the hidden and visible units $P(\mathbf{x}, \mathbf{y})$. The free energy is comparable to "variational free energy" from statistical physics. The joint probability is readily available in directed acyclic graphical models (called belief networks).

The free energy is actually an upper bound on the negative log-probability of the data. It is the negative log-probability of the data plus the KL divergence between Q and P[2]:

$$F = -\log P(\mathbf{x}) + \mathrm{KL}(Q\|P) \tag{1.3}$$

Instead of optimizing the parameters of Q by minimizing (1.2), we can minimize (1.3),

[1] This assumes that $\mathcal{P} \neq \mathcal{NP}$ [Cooper, 1990].

[2] The reader should note that the free energy is sometimes defined with opposite sign: $F = \log P(\mathbf{x}) - \mathrm{KL}(Q\|P)$. However, we will use the sign which is consistent with the free energy from statistical physics

which will have the same effect (since $\log P(\mathbf{x})$ is independent of Q). It turns out that we will also train the model by minimizing (1.3) with respect to the model parameters. See section 2.4 for a further discussion of free energy.

Variational methods have several advantages. They allow us to calculate an upper and lower bound on the log-probability of the data under the current generative model [Jaakkola, 1997] and they are fast. They are also deterministic which can be an advantage in situations where gradients are shallow and sampling noise might hinder learning. The down side is that it is not always easy to come up with a good approximating distribution. Typically a good variational approximation is very architecture-specific. If care is not taken, the approximating distribution can be too simplistic to capture important features of the posterior, or it can be too complicated to be used in subsequent calculations. Finding good approximating distributions is as much art as science.

Two commonly-used variational approximations are the *mean field* approximation, and the *maximum a posteriori* (MAP) approximation. Under a mean field approximation we assume that the hidden units are independent. Instead of having a distribution whose representation requires space exponential in the number of hidden units we have one that only needs linear space. Under a MAP approximation we assume that Q is a single spike of infinite density at a point $\mathbf{y}_m$. Like replacing a tent by a tent pole, the entire mass of the distribution is replaced with this single spike. In this case, the KL distance between Q and P is infinite, so we optimize the parameters of Q with respect to the expected energy under Q, given by:

$$\begin{aligned} E_Q[energy] &= E_Q[-\log P(\mathbf{y},\mathbf{x})] \\ &= -\int Q(\mathbf{y}|\mathbf{x}) \log P(\mathbf{y},\mathbf{x}) d\mathbf{y} \\ &= -\log P(\mathbf{y},\mathbf{x})\delta(\mathbf{y}-\mathbf{y}_m) \\ &= -\log P(\mathbf{y}_m,\mathbf{x}) \end{aligned} \tag{1.4}$$

This quantity is proportional to the posterior, so during the optimization of Q we move the spike to a point of maximum density under the posterior (the tallest part of the tent), thus the name *maximum a posteriori*. This can be a reasonable approximation if the true posterior is unimodal and sharply-peaked. Both of these approximations are common because they are simple, and they seem to work for a large class of problems. However if the posterior is multimodal or broad they may be insufficient.

It may appear that by adopting such simple approximations, we loose a lot of the expressiveness of the true posterior. It is true that the posterior is usually much more expressive than the approximation. However, there is a mitigating factor that allows simple approximations to do a reasonable job: Because we update the model parameters by performing gradient descent in F, the model parameters are adjusted so as to reduce $\mathrm{KL}(Q||P)$ as well as to increase $\log P(\mathbf{x})$. The model parameters change so that the true posterior is brought closer to the approximation.

1.4.3 Monte Carlo methods

Monte Carlo approximations can be used to estimate expectations when we cannot evaluate the posterior explicitly, but we can sample from it. Given N samples $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$ from a distribution $P(\mathbf{x})$, we can approximate the expectation of a function $f(\mathbf{x})$ under P as follows:

$$\begin{aligned} E[f(\mathbf{x})] &= \int f(\mathbf{x})P(\mathbf{x})d\mathbf{x} \\ &\approx \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_i) \end{aligned} \tag{1.5}$$

There are a number of different schemes for generating the set of samples which depend on the form of the distribution P (see [Neal, 1993] for a survey of Monte Carlo sampling techniques). In general, Monte Carlo methods are not as dependent on architecture as are variational methods. A single Monte Carlo method can be applied to a

broad range of architectures. The disadvantage of Monte Carlo methods is speed: It can take a long time to approximate an expectation to a high degree of accuracy. Also, because the approximations involve taking a random sample from P, we get a noisy estimate of any quantities of interest. We discuss Monte Carlo methods in more depth in section 1.5.3 and section 2.4.

1.5 Models

1.5.1 Factor Analysis

Factor analysis can be viewed as a two-tier generative model. The value of a D-dimensional real-valued visible vector is given by:

$$\mathbf{x} = \mathbf{W}\mathbf{y} + \eta \tag{1.6}$$

where $\mathbf{y}$ is a real-valued vector, drawn from a multivariate Gaussian distribution with zero mean and unit-diagonal covariance. The vector $\mathbf{y}$ can be viewed as the underlying, or *hidden* cause of the data vector $\mathbf{x}$. The real-valued vector η, which plays the role of independent sensor noise, is drawn from a multivariate Gaussian with zero mean and a $D \times D$ diagonal covariance matrix $\mathbf{\Psi}$, where $\psi_1^2, ..., \psi_D^2$ are the elements along the diagonal. The weights $\mathbf{W}$, which specify the relationship between the hidden and visible units, are called the *factor loadings*. Since $\mathbf{y}$ is drawn from a diagonal-covariance Gaussian, we can consider each unit y_j separately in the generative model. The prior over the j^{th} hidden unit is given by:

$$P(y_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left\{-\frac{(y_j - \widehat{y}_j)^2}{2\sigma_j^2}\right\} \tag{1.7}$$

where in the standard factor analysis model $\widehat{y}_j = 0$ and $\sigma_j^2 = 1$. We will look at models later in which $\widehat{y}_j$ is non-zero and σ_j^2 is not unity. Viewed as a Bayesian network, a factor analyzer consists of Gaussian stochastic units with linear transfer functions (see figure

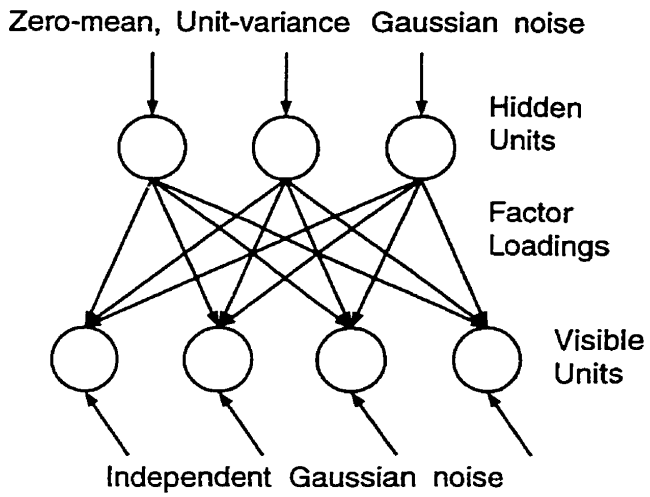

Figure 1.1: Factor analysis, viewed as a Bayesian network. The circles represent linear units, and the arrows between layers represent the factor loadings. When viewed as a generative model, the hidden units are driven with zero mean, unit-variance Gaussian noise. Independent Gaussian noise is also added to the visible units

1.1). We refer to this kind of unit as a *linear-Gaussian* unit.

Because the prior is Gaussian and the hidden units are linear, a factor analyzer in generative mode defines a Gaussian distribution over observation space. Therefore, factor analysis can only capture second-order correlations in the data, and is insensitive to higher-order correlations. The Gaussian defined by a factor analyzer has zero mean if $\hat{\mathbf{y}} = 0$, and it can be shown to have covariance matrix $\mathbf{W}\mathbf{W}^\top + \mathbf{\Psi}$. Typically, factor analysis is used as a means of dimensionality reduction, when it is conjectured that a small number of underlying hidden factors are responsible for the distribution of the visible data. In this case the number of hidden factors used is less than D [Everitt, 1984], and the factor analyzer can only model a restricted subset of possible covariance matrices.

Given a data set, the ML factor loadings ($\mathbf{W}$) and diagonal covariance matrix ($\mathbf{\Psi}$) can be found with the *Expectation-Maximization* (E-M) algorithm [Rubin and Thayer, 1982]. In the E-step the distribution of the hidden units given the visible units is calculated. In the M-step the parameters are updated so as to maximize the probability of generating the given data vector given the posterior found in the E-step. The posterior distribution

can be found using Bayes' rule:

$$p(\mathbf{y}|\mathbf{x};\mathbf{W},\mathbf{\Psi}) = \frac{1}{Z_p} p(\mathbf{x}|\mathbf{y};\mathbf{W},\mathbf{\Psi})p(\mathbf{y}) \tag{1.8}$$

where Z_p is a normalizing constant.

The posterior is Gaussian and can be computed exactly. Factor analysis is a widely-used model in part because exact inference is tractable. The tractability of the posterior results in a straight-forward inference algorithm in which *explaining away* is handled correctly [Pearl, 1988].

Factor analysis learns representations that are fully distributed across the hidden units. The hidden layer tends to use all of the hidden units to represent each data point. Unfortunately, this means that all of the hidden factors, and so all of the basis vectors in $\mathbf{W}$, must concern themselves with each data point. A factor analyzer will not dedicate subsets of its factor loadings to model localized features of a data set. This ability would be advantageous when trying to model images, for example, which are typically compositions of small objects.

Factor analysis models have been extended to use non-Gaussian priors over the hidden factors [Bell and Sejnowski, 1995, Olshausen, 1996, Hinton and Ghahramani, 1997, Éric Moulines et al., 1997, Lewicki and Olshausen, 1998]. In most cases the posterior cannot be computed exactly, and approximations must be used. These approximations include learning an explicit recognition model [Hinton et al., 1995], estimating the posterior with a variational method [Saul et al., 1996] or with MAP inference [Olshausen, 1996], or sampling from the posterior using Monte Carlo sampling techniques [Neal, 1992]. As an alternative to these approximations the model can be simplified by removing the independent noise on the visible units [Bell and Sejnowski, 1995]. This causes the posterior to shrink to a single point which can be found by multiplying the observation by the inverse of the generative weight matrix. This simplification is analogous to the one made

for principal component analysis (PCA) which can be viewed as a factor analysis model with no independent noise on the visible units. Note that if the generative weight matrix is not square then the resulting model does not define a proper probability distribution over the observation space. A well-defined probabilistic version of PCA has been investigated [Roweis, 1997, Tipping and Bishop, 1997].

1.5.2 Mixture Models

A mixture model can be viewed as a probabilistic generative model composed of several distinct density models (or components). To generate an observation, exactly one of the components is chosen, and the new point is drawn from the distribution defined by this component. If the probability of component i generating data point $\mathbf{x}$ is given by $p_i(\mathbf{x}|\theta)$ then the probability of the mixture generating $\mathbf{x}$ is given by:

$$p(\mathbf{x}|\theta) = \sum_{i=1}^{K} p_i(\mathbf{x}|\theta) P(i|\theta) \tag{1.9}$$

where θ represents the parameters of the mixture model, $P(i|\theta)$ is the prior probability of choosing component i, and K is the number of density models in the mixture. To generate a new data point, first component i is chosen with probability $P(i|\theta)$, and then the new data point $\mathbf{x}$ is drawn from the distribution $p_i(\mathbf{x}|\theta)$.

A set of ML model parameters can be found with the E-M algorithm. In the E-step, the responsibility of each component i is calculated for each data point $\mathbf{x}$:

$$R_i(\mathbf{x}) = \frac{p_i(\mathbf{x}|\theta) P(i|\theta)}{\sum_{j=1}^{K} p_j(\mathbf{x}|\theta) P(j|\theta)} \tag{1.10}$$

This is the posterior probability of selecting component i given the data point $\mathbf{x}$, and (1.10) is simply a restatement of Bayes' rule. Notice that $p_i(\mathbf{x}|\theta)$ must be evaluated to find the responsibilities. This may not be easy, depending on the form of the model.

The M-step consists of updating the model parameters θ so as to maximize the expected log-likelihood; typically, the expected negative log-likelihood is minimized instead. Given the responsibilities, the expected log-likelihood is:

$$\mathcal{L}(\theta) = \sum_{\mathbf{x}} \sum_{i=1}^{K} R_i(\mathbf{x}) \log p_i(\mathbf{x}|\theta) \tag{1.11}$$

The parameters can be updated by performing gradient ascent on (1.11). Many different models have been used including Gaussians [Day, 1969], principal component analyzers [Jacobs et al., 1991] and factor analyzers [Hinton et al., 1997a, Ghahramani and Hinton, 1996].

One benefit of a mixture model is that if there are several distinct clusters in the data the mixture model can assign a separate mixture component to each cluster. For this reason mixture models are well suited to solving classification problems where each cluster defines a class of objects. In this case the latent variable i corresponds to the class labels. Not only can the mixture model adapt to each class separately, but the responsibilities computed for a previously unseen data point yield a soft classification of the new observation.

Notice that in generative mode only one mixture component can be active at a time; this is a considerable limitation. Consider the case of images, where each image is composed of a collection of objects. Since each mixture component must try to explain the entire image the model cannot simply adapt one component to each object. Instead, it must adapt one component to each *combination* of objects. If the data contain combinations drawn from D possible objects then 2^D mixture components will be required to model the data. Clearly if there are exponentially many mixture components the calculation of responsibilities (1.10) becomes intractable.

Mixture models learn a localized representation of each data point: Only one unit in the hidden layer is active for each observation. The representational power of mixture models is limited because they do not learn a distributed representation. Unlike with a

factor analyzer, the representation of an input vector is not shared across several hidden units.

1.5.3 Logistic Belief Networks

Logistic belief networks, first investigated by Neal [Neal, 1992], are directed acyclic networks of stochastic binary units. The binary units discussed here output either 0 or 1. A unit i outputs a 1 with probability dependent on the outputs of the units above:

$$P(s_i = 1 | s_j, j < i) = \sigma(b_i + \sum_{j<i} w_{ji} s_j) \tag{1.12}$$

where s_i is the output of unit i, $j < i$ denotes the parents of unit i, w_{ji} is the connection strength from unit j to unit i, b_i is the bias on unit i, and $\sigma(x)$ is the logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.13}$$

We refer to these units as *binary-logistic* units.

In order to learn ML parameters for this model we must compute the posterior distribution over the hidden units given a visible vector. This amounts to finding the probability of each of the possible states of the hidden units given the input vector. If a network has D hidden units there will be 2^D such states making the exact calculation of the full posterior intractable. To approximate the full posterior distribution Neal used a Monte Carlo sampling technique called Gibbs sampling. The posterior has also been approximated with a variational approximation [Saul et al., 1996], and by learning an explicit set of bottom-up recognition connections that, given a training vector, produce a set of hidden unit activities [Hinton et al., 1995].

Gibbs Sampling in Logistic Belief Nets

Gibbs sampling is a Monte Carlo method that can be used to approximate expectations. Gibbs sampling is appropriate when we cannot explicitly calculate these expectations, but we can sample from the distribution of a single unit given the states of all of the other units in the network. During Gibbs sampling hidden units are visited one at a time, and the new state of each unit is drawn from its distribution conditioned on the activities of all of the other units. Let α denote a particular set of hidden unit activities, or *configuration* of the network, and let P^α denote the probability of this configuration, given by:

$$\begin{aligned} P^\alpha &= \Pi_i P(s_i^\alpha | s_j^\alpha, j < i) \\ &= \prod_i \left[\sigma(b_i + \sum_{j<i} w_{ji} s_j^\alpha)^{s_i^\alpha} \left[1 - \sigma(b_i + \sum_{j<i} w_{ji} s_j^\alpha) \right]^{1-s_i^\alpha} \right] \end{aligned} \tag{1.14}$$

where s_i^α denotes the state of unit i in configuration α.

The conditional probability of unit i emitting a 1, given the configuration denoted by α over the other units, is given by:

$$P(s_i = 1 | \alpha) = \frac{P^{\alpha \backslash s_i=1}}{P^{\alpha \backslash s_i=1} + P^{\alpha \backslash s_i=0}} \tag{1.15}$$

where $P^{\alpha \backslash s_i=1}$ and $P^{\alpha \backslash s_i=0}$ are the probabilities of a configuration that is identical to α, with the exception that unit i takes on the values 1 and 0 respectively.

If we define the energy of a configuration as $E^\alpha = -\log P^\alpha$, then (1.15) can be rewritten as:

$$P(s_i = 1 | \alpha) = \sigma(\Delta E_i^\alpha) \tag{1.16}$$

where ΔE_i^α is the difference in energies when unit i takes on the value of either 1 or 0:

$$\Delta E_i^\alpha = E_i^{\alpha \backslash s_i=0} - E_i^{\alpha \backslash s_i=1}$$

$$= \log \hat{s}_i^\alpha - \log(1 - \hat{s}_i^\alpha) + \sum_j [\, s_j^\alpha \log \hat{s}_j^{\alpha \setminus s_i=1} + (1 - s_j^\alpha) \log(1 - \hat{s}_j^{\alpha \setminus s_i=1}) - s_j^\alpha \log \hat{s}_j^{\alpha \setminus s_i=0} - (1 - s_j^\alpha) \log(1 - \hat{s}_j^{\alpha \setminus s_i=0}) \,] \quad (1.17)$$

Here, $\hat{s}_i^\alpha = \sigma(b_i + \sum_{k<i} w_{ki} s_k^\alpha)$ is the probability that $s_i = 1$ given its total input from the other units, plus the bias.

To perform Gibbs sampling each unit is visited in turn, and its state is chosen using (1.16). If sampling continues for long enough, this process is guaranteed to converge to the correct distribution [Neal, 1993]. Unfortunately, "long enough" is hard to determine. Even if we can tell that the Markov chain has reached equilibrium, the amount of time required might make this approximation unattractive. We discuss this problem further in section 2.4.

Given a sample from the posterior distribution of the hidden units, the weights can be updated with the online delta rule:

$$\Delta w_{ji} = \epsilon s_j (s_i - \hat{s}_i) \quad (1.18)$$

where ϵ is the learning rate. This performs steepest ascent in the log-likelihood.

1.5.4 Independent Component Analysis

We have seen in section 1.5.1 and section 1.5.2 that sometimes it is undesirable to produce representations of data that are fully localized or fully distributed. Logistic belief networks finds sparse, distributed representations, but are unable to represent real-valued quantities efficiently. Independent component analysis (ICA) finds sparse, distributed representations of real-valued data.

Factor analysis tries to find factor loadings that model the covariance structure of data, but is insensitive to higher-order structure. ICA tries to capture the higher order statistics as well by finding a weight matrix that makes the hidden factors statistically

independent [Bell and Sejnowski, 1995]. Bell and Sejnowski originally derived their algorithm from within an information-maximization framework, but it can be viewed as the ML optimization of a two-tier generative model with a high-kurtosis non-Gaussian prior over the hidden units [Mackay, 1996, Pearlmutter and Parra, 1997]. An example of such a distribution is shown in figure 1.2. ICA can also be viewed as a generative model

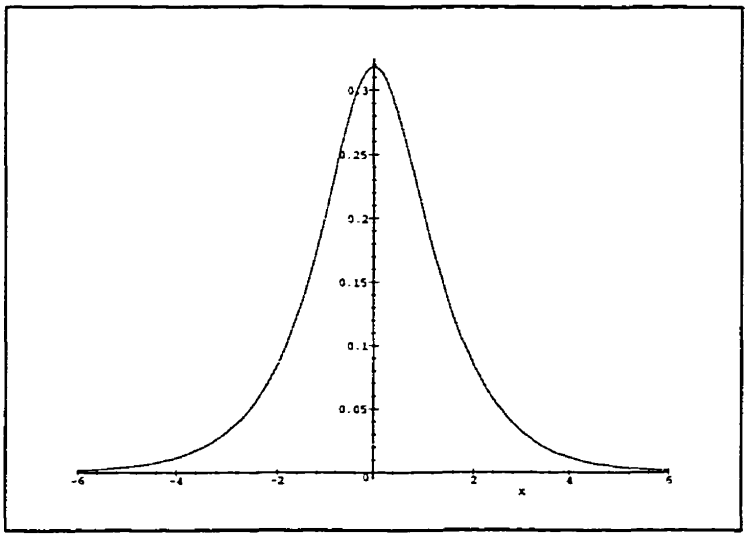

Figure 1.2: A plot of $(\pi \cosh(x))^{-1}$, a high-kurtosis distribution used by the ICA algorithm as a prior distribution over hidden unit activities.

whose hidden units are Gaussian distributed, but have a non-linear transfer function [Mackay, 1996].

Intuitively this prior encourages sparse, distributed representations because it places a large probability mass near zero, and spreads the rest of the probability mass far out in the tails of the distribution. This means that, when operating in generative mode, most hidden units will take on values close to zero while a few others will take on relatively large values. This can be contrasted with a zero-mean Gaussian prior where most hidden units will take on small but non-zero values.

If we assume a Gaussian noise model on the input units the computation of the posterior distribution over the hidden units is intractable. The ICA algorithm is equiv-

alent to assuming that there is no independent sensor noise in the generative model [Mackay, 1996]. In this case, the activities in the hidden layer can be found by multiplying the visible unit activities by the inverse of the generative weights. The result is a simple weight-update rule:

$$\Delta \mathbf{W} \propto [\mathbf{W}^{\mathrm{T}}]^{-1} + \frac{\mathrm{d}\log(p(\mathbf{y}))}{\mathrm{d}\mathbf{y}}\mathbf{x}^{\mathrm{T}} \tag{1.19}$$

where $\mathbf{W}$ is the inverse of the generative weight matrix, $p(\mathbf{y})$ is the density function of the non-Gaussian prior over the hidden units, $\mathbf{y}$ is a vector of the values of the hidden units, and $\mathbf{x}$ is a vector containing the values of the visible units.

The matrix inversion in the first term of (1.19) makes this algorithm computationally unattractive, and biologically implausible since the learning rule is not local. Versions of ICA have been derived which eliminate the matrix inversion, and converge faster than the original algorithm [Amari et al., 1996, Mackay, 1996, Cardoso, 1996].

Unfortunately, the lack of independent sensor noise in the generative model makes the ICA algorithm sensitive to noise in the training data. Also, this algorithm requires that the number of hidden units be equal to the dimensionality of the input data. If the ICA algorithm is used on data that has only a few underlying causes the remainder of the "independent components" will be used to model noise in the data. It is not always easy to distinguish true underlying causes from noise components (see section 3.1 for an example of the performance of ICA on a noisy task). Further, the lack of sensor noise makes it difficult to extend this model to multiple layers.

Olshausen and Field investigated a similar two-layer model [Olshausen and Field, 1996] which can be viewed as a generative model with a high-kurtosis non-Gaussian prior over the hidden units [Olshausen, 1996]. In this model Olshausen and Field retain the independent Gaussian sensor noise, and approximate the posterior distribution over the hidden units with a MAP estimate. The posterior can also been approximated by a best-fit Gaus-

sian which better accounts for the mass under the posterior [Lewicki and Olshausen, 1998]. These are reasonable approximations when the posterior is unimodal and sharply peaked. Given the approximated posterior the weights can be updated by taking a gradient step which increases the expected likelihood.

These latter models avoid some of the limitations of ICA in that the inclusion of sensor noise reduces the model's sensitivity to noise in the training data. Also, there are no limitations on how many hidden units can be employed; the model can be used for the purposes of dimensionality reduction, or to compute an "overcomplete" basis set [Olshausen and Field, 1996]. This greater flexibility is achieved at a price: The approximation of the posterior is more computationally expensive than that required to learn the parameters of an ICA model which, in the case of a covariant algorithm, just requires a few matrix multiplications per data point per weight update.

1.5.5 Rectified Gaussian Belief Nets

A Rectified Gaussian Belief Net (RGBN) is a hierarchical generative model that extracts sparse, distributed representations [Hinton and Ghahramani, 1997]. The prior distribution used in the RGBN is a rectified Gaussian (see figure 1.3). Given the input from its parents y_j, a unit selects an output value $\tilde{y}_i$ as follows:

$$p(y_i|y_j, j<i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{\frac{-(y_i - \sum_{j<i} w_{ji}\, y_j)^2}{2\sigma_i^2}\right\} \tag{1.20}$$

$$\tilde{y}_i = \begin{cases} y_i & \text{if } y_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.21}$$

The intuition behind why the RGBN extracts sparse, distributed representations is similar to that for ICA. The rectified Gaussian distribution allows hidden units to place a large probability mass exactly on zero, and some probability mass far from zero. The rectified Gaussian also provides a good example of viewing a non-Gaussian prior as a

Gaussian prior passed through a non-linearity; in this case the non-linearity is a simple rectification.

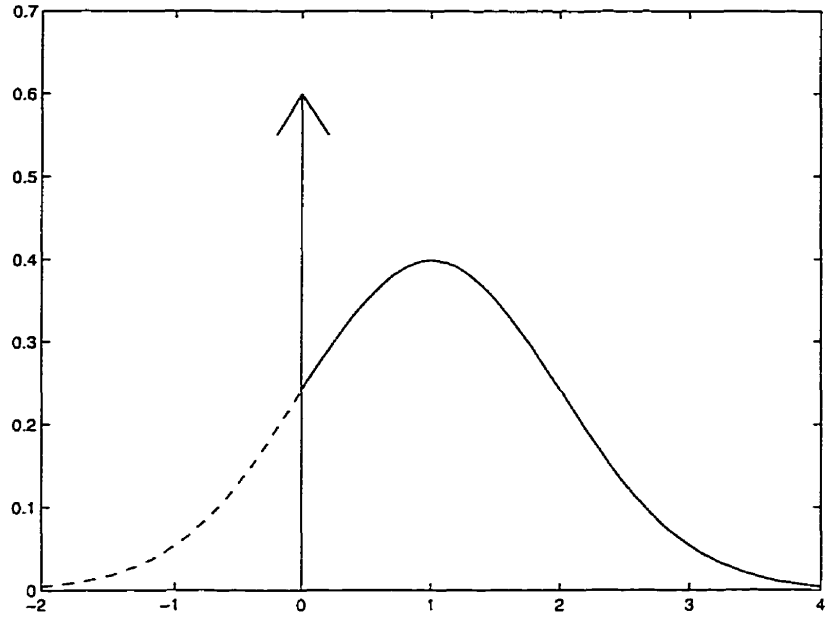

Figure 1.3: The rectified Gaussian concentrates all of the probability mass from $x < 0$ in a spike at zero.

Like ICA, an RGBN uses real-valued units, so real valued random variables can be represented efficiently. Again, because of the non-linear "rectification" function that produces the output value $\tilde{y}_i$, and because independent Gaussian noise is added to visible units, the computation of the posterior is intractable. Instead of using a MAP estimate, Hinton and Ghahramani used Gibbs sampling to approximate the posterior. The rectified Gaussian prior was specifically chosen to make Gibbs sampling feasible while still allowing units to place a large probability mass exactly on zero. By using Gibbs sampling to approximate the posterior Hinton and Ghahramani avoided the potential problem of finding only one mode of a multi-modal posterior as would be the case if using a MAP or Gaussian estimate. It is unclear how much of a problem this is in practice, and MAP inference has been used successfully in single-layer models with rectified Gaussian units [Rao and Ballard, 1997, Socci et al., 1998].

One possible problem with this approach, as discussed in section 1.5.3, is that Gibbs

sampling may take a long time to converge. Hinton and Ghahramani found that the RGBN was able to learn with only 10 to 20 Gibbs samples per hidden unit. We discuss "brief" Gibbs sampling further in section 2.4.

Given samples from the posterior of the unrectified values y_i, the weights of the RGBN can be updated with a simple online delta rule:

$$\Delta w_{ji} = \epsilon \tilde{y}_j (y_i - \widehat{y}_i) \tag{1.22}$$

and the variance of the local Gaussian noise can be learned with:

$$\Delta \sigma_j^2 = \epsilon \left[(y_j - \widehat{y}_j)^2 - \sigma_j^2 \right] \tag{1.23}$$

Again, this update rule simply performs gradient ascent in log-likelihood.

The RGBN was able to find localized features in lower layers, and to find correlations among these features in higher layers, for some small but interesting problems.

Chapter 2

Hierarchical Community of Experts

2.1 Motivation

Sometimes it makes sense to model data using both binary and real-valued quantities. For example, whether or not a particular object appears in a scene is a binary decision, but its position, orientation, scale and colour are real-valued quantities. The only model we have seen so far that makes this distinction is a mixture model; however we have also seen that the mixture model is exponentially inefficient.

All of the models examined in chapter 1 that use exclusively real-valued stochastic units implicitly assume that the presence of a feature is related to its magnitude. In these models, a feature being absent is indistinguishable from a feature being present with a very small magnitude. Even worse, it is impossible to represent large real values and small real values without giving significant probability to everything in between.

This representation does not always make sense. For example, if we wanted to represent the mass of adult elephants, then either we want the feature to be absent (when there is no elephant), or present with a large numerical value. We want the model to give small real values very low probability. Of course, a logistic belief network can represent this kind of information, but it would be inefficient to represent real-valued, locally linear

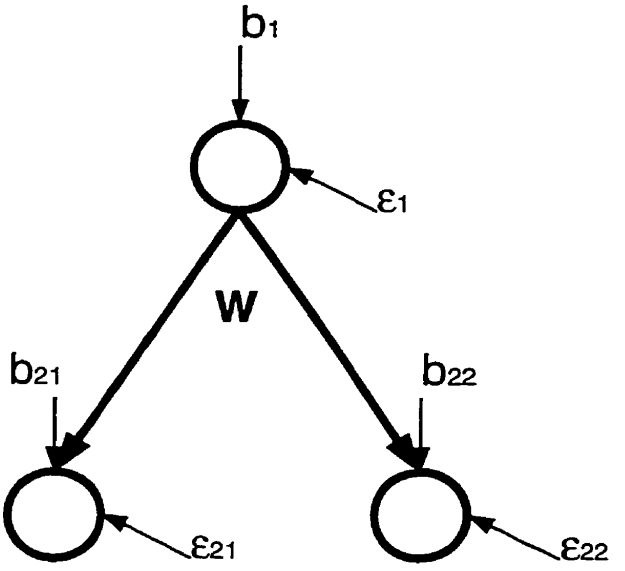

Figure 2.1: Architecture of the RGBN used to model a mixture of two Gaussian distributions. Injected noise ϵ_1 has variance σ_1^2; and noise values ϵ_{21} and ϵ_{22} both have variance σ_2^2.

variables with highly nonlinear binary units.

To demonstrate the problems inherent in the assumption that a feature's existence is related to its magnitude, we trained an RGBN model on a simple data set consisting of 1000 2-dimensional points generated from a mixture of two Gaussians (see figure 2.2(a)). Each observation was generated by the first Gaussian with probability $\pi_1 = 0.6$, and from the second with probability $\pi_2 = 0.4$. (see table 2.1 for a summary of the Gaussian mixture). The RGBN had one top-level unit, and two visible units (see figure 2.1). Ideally, the RGBN should place high density on the two clusters, and little in between. This RGBN can represent two distinct Gaussians: one when the top-level unit is off, and the other when it is on. We will assume that the first Gaussian is being modeled when the top-level unit is off. The mixing proportion for the first Gaussian is determined by the probability mass of the Gaussian prior on the top-level unit that is below zero:

$$\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}\sigma_1} \exp\left\{\frac{-1}{2\sigma_1^2}(x - b_1)^2\right\} dx = \pi_1 \tag{2.1}$$

where b_1 and σ_1^2 are the bias and variance of the top-level unit; and π_1 is the mixing proportion for the first Gaussian (0.6 in this example).

Further, the mean of the Gaussian prior on the top-level unit also influences the means of the two Gaussians in the mixture:

$$\mathbf{b}_2 = \mu_1 \tag{2.2}$$

$$\mathbf{W}\, b_1 + \mathbf{b}_2 = \mu_2 \tag{2.3}$$

where μ_1 and μ_2 are the means of the first and second Gaussians; $\mathbf{W}$ is the pair of weights from the top-level unit to the two visible units; and $\mathbf{b}_2 = [b_{21} b_{22}]^\top$ is the bias on the visible units.

The parameter b_1 must simultaneously determine the mixing proportions of the two Gaussians (see (2.1)), which control which Gaussian is used to generate an observation, and the mean of the second Gaussian (see (2.3)), which control the magnitude of the generated data. Clearly the mixing proportions are independent of the means of the Gaussians, so it is unfortunate that the RGBN must try to model them both with the same parameter.

After training, the top-level unit of the RGBN was on 89% of the time. When this occurred, the RGBN produced data from a full-covariance Gaussian which did not correspond to either of the distributions in the original mixture (see table 2.1).

An equal number of data points were generated from the trained RGBN and plotted (see figure 2.2). Notice that the RGBN is unable to place significant mass on the two clusters without placing mass on everything in between. It manages to model the first Gaussian, but in trying to capture the right mixing proportions, it must use a poor choice of bias on the top-level unit, misplacing the mass of the second Gaussian.

	μ	Ψ	π
Original Gaussian 1	(-2.0,-1.5)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$	0.4
Original Gaussian 2	(1.0, 2.0)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$	0.6
Learned Gaussian 1	(-2.1,-1.9)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$	0.11
Learned Gaussian 2	(-0.1, 0.4)	$\begin{bmatrix} 3.7 & 3.5 \\ 3.5 & 4.5 \end{bmatrix}$	0.89

Table 2.1: Parameters of a mixture of two Gaussians, and parameters estimated by a trained RGBN.

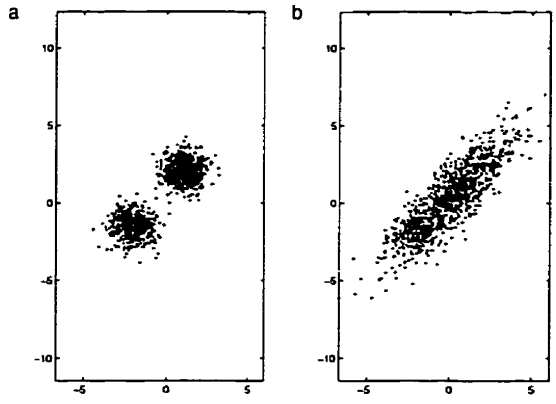

Figure 2.2: a) The training data, generated from a mixture of two Gaussians. b) An equal number of data points generated from the trained RGBN.

2.2 Combining Binary and Linear Units

We can model both kinds of information by gating the output of each linear-Gaussian unit with the output of a binary-logistic unit. The binary unit in a pair will be used to code the presence or absence of a feature while the linear unit can model real values that are approximately locally linear (see figure 2.3). We will use y to denote the values of linear-Gaussian units and s to denote the values of binary-logistic units, and paired units will share the same subscript.

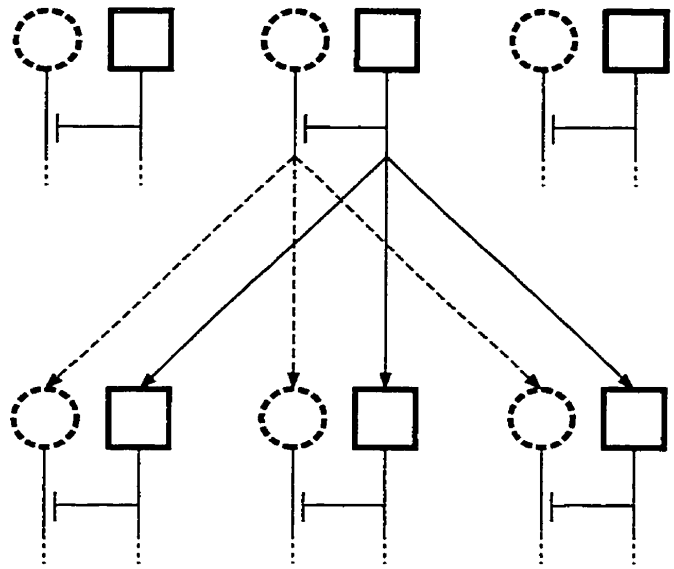

Figure 2.3: Units in a community of experts, a network of paired binary and linear units. Binary units (solid squares) gate the outputs of corresponding linear units (dashed circles) and also send generative connections to the binary units in the layer below. Linear units send generative connections to linear units in the layer below (dashed arrows).

Recall from the definition of a linear-Gaussian unit (see (1.7)):

$$P(y_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left\{-\frac{(y_j - \widehat{y}_j)^2}{2\sigma_j^2}\right\} \tag{2.4}$$

where y_j is the output and σ_j^2 is the variance of the linear-Gaussian unit. Here, the mean of the Gaussian distribution is the weighted sum of the *gated* outputs of the units above plus the bias:

$$\widehat{y}_j = b_j + \sum_{k<j} w_{kj} y_k s_k \tag{2.5}$$

There are weighted connections from linear units to linear units. We also include weighted connections from binary units to binary units, and use (1.12) to determine the output of the binary units in the generative model. We could also include connections from binary units to linear units, although in the simulations reported in chapter 3 these connections are not used. If these connections were included, then the prior on a linear unit $\widehat{y}_j$ would just be the weighted sum of the linear and binary outputs of the units above. To make

inference and learning feasible (see section 2.3) connections from linear units to binary units are prohibited.

The result is a network of binary units that synthesizes a linear network from a large set of available linear units. The binary units indicate the existence of features while the linear units represent real-valued variables that are locally linear. Given the activities of the binary units, the network is linear, and can be viewed as a multilayer version of factor analysis. The model uses a non-linear mechanism (the binary units) to select a linear model from a large set of possible linear models. In this respect it is similar to a hierarchical mixture of experts (HME) [Jacobs et al., 1991, Jordan and Jacobs, 1994], with the binary units playing the role of gating networks. However, unlike an HME, each linear unit acts as an expert, and any combination of experts can cooperate to explain an observation. We call this network a hierarchical community of experts (HCE) [Hinton et al., 1997b].

2.3 Learning and Inference

In order to find an ML set of model parameters we want to compute the posterior distribution over the hidden units. As with the LBN and RGBN, we cannot compute the full posterior, but can approximate it with Gibbs sampling. The obvious way of performing Gibbs sampling is to visit the units one at a time, and choose the state of the current unit while holding all others constant. This obvious approach has two disadvantages: First, the distribution for a binary unit is affected by the value of its corresponding linear unit. Instead of using the current (sampled) value of the linear unit to determine the distribution of the binary unit, we can integrate over all possible values of the linear unit, weighted by their probabilities. This should lead to faster convergence to the posterior.

Second, the distribution of a linear unit is affected by the pre-gated values of the

linear units below it. If a unit below is gated out (i.e. disabled), then its distribution is completely determined by the values of the units in the layer above. There is no connection to the data, so the values of the gated out units below do not provide any data-driven information about what values should be taken by units in the layer above. Allowing the values of gated out units in the layer below to influence the sampling will only increase sampling noise. This is unfortunate since we hope that the representation found by the binary units will be sparse, and so most of the linear units will be gated out at any particular time. In this case we would like to integrate out linear units below that are gated out. The correct way to integrate out linear-Gaussian units that are gated out is simply to ignore their energy contribution.

Given a set of binary activities, the network is linear. We could therefore choose the state of a binary unit while holding the other binary units constant, and while integrating out the entire linear network. However, integrating out a layer of k linear units requires the inversion of a $k \times k$ matrix (see appendix A). It is not clear that the reduction in sampling noise is worth this additional computational effort, especially early on when most of the units are not doing anything meaningful. We have chosen the intermediate solution of integrating out just the corresponding linear unit when sampling from a binary unit, and integrating out the gated-out linear units in the layer below when sampling from units in a given layer.

Given samples from the posterior distribution of the binary units, the binary-to-binary connections are updated using (1.18). The linear-to-linear connections are updated with a similar rule:

$$\Delta w_{ji} = \epsilon y_j s_j (y_i - \widehat{y}_i) s_i / \sigma_i^2 \tag{2.6}$$

and the local noise variance of the linear-Gaussian units can be learned with:

$$\Delta \sigma_j^2 = \epsilon s_j \left[(y_j - \widehat{y}_j)^2 - \sigma_j^2 \right] \tag{2.7}$$

If there were binary-to-linear connections they would also be updated with (2.6), but $\widehat{y}_i$ would be a weighted sum of binary and linear unit activities.

We do not include linear-to-binary connections because it would make it difficult to sample from the posterior of the linear unit. The non-Gaussian likelihood term in the posterior caused by the binary-logistic connections would make the posterior of the linear unit non-Gaussian. The posterior would, however, correspond to an unnormalized energy function that we can calculate explicitly. This raises the possibility of using a sampling technique such as the Metropolis algorithm to sample from the linear units in the case of linear-to-binary connections [Metropolis et al., 1953]. Alternatively, because the distribution over a linear unit with connections to linear-Gaussian and binary-logistic units would be log-concave, we could use adaptive rejection sampling for Gibbs sampling [Gilks, 1992]. These possibilities have not been explored.

2.4 Brief Gibbs Sampling

It might seem at first glance that Gibbs sampling is a bad choice for an efficient on-line learning algorithm. If considered as a method of computing expectations with respect to the posterior distribution over hidden units, Gibbs sampling faces several difficulties:

1. We should let the Markov chain reach equilibrium before we use the samples for parameter learning, but it is hard to tell when the equilibrium distribution has been reached.

2. Even if we can determine this, it may take the Gibbs sampler far too long to reach equilibrium to make Gibbs sampling viable. To make matters worse, as the weights change, this equilibrium distribution changes as well. The Gibbs sampler must chase a moving target.

3. To avoid having to start the Gibbs sampling from a state that is far from equilibrium we must provide a separate realization of the Markov chain for each training example. In practice we need to store the last set of hidden unit activities that the sampler provided for each observation. For large training sets, or for online training schemes, this requirement is infeasible.

We can try to avoid the convergence problems by allowing for a burn-in period, and by using a small learning rate. Even if the burn-in period is not long enough for the Markov chain to reach equilibrium, the first part of the run (when the Gibbs sampler was not at equilibrium) will be overshadowed by later samples. With a small learning rate, the equilibrium distribution after a weight update should be close to the equilibrium distribution before the update, so the Markov chain will remain close to its equilibrium distribution. However, this approach may still be too slow and will require a separate Markov chain for each observation.

In chapter 3 we show that an HCE network can learn some small but interesting tasks when each Gibbs sampling pass consists of only a few iterations of Gibbs sampling for each unit. The model can learn even if the distribution being sampled is far from the true equilibrium distribution. As a byproduct, we do not have to store previous network states for each training example. This "brief" Gibbs sampling has been used previously to train RGBN's [Hinton and Ghahramani, 1997].

To understand why brief Gibbs sampling works, we can consider a cost function other than the negative log-likelihood. If Q is the distribution over the hidden units produced by the Gibbs sampler given the energy E, then the *free energy* of the network is defined as the expected energy under Q minus the entropy of Q. For a network of binary and real-valued units, this is given by:

$$F = \sum_{\alpha} \int_{\mathbf{x}} Q(\mathbf{x}, \alpha) E(\mathbf{x}, \alpha, \mathbf{z}) d\mathbf{x} - \left(- \sum_{\alpha} \int_{\mathbf{x}} Q(\mathbf{x}, \alpha) \log Q(\mathbf{x}, \alpha) d\mathbf{x} \right) \tag{2.8}$$

where $\mathbf{x}$ is a real-valued vector of linear-Gaussian hidden unit activities; α denotes a configuration of the binary-logistic hidden units; $Q(\mathbf{x}, \alpha)$ is the joint probability of the real-valued units having value $\mathbf{x}$ and the binary hidden units being in configuration α; and $\mathbf{z}$ is a (fixed) vector of visible unit activities. The E-M algorithm can be viewed as coordinate descent in this new objective function [Neal and Hinton, 1997].

For simplicity, we will drop the dependence on $\mathbf{x}$ and α in the following discussion. Let P be the actual posterior distribution given E. If $Q = P$, then (2.8) is just the negative log probability of the visible units. Otherwise, $\mathrm{K}L(Q\|P) > 0$, so (2.8) is the negative log probability of the hidden units plus the Kullback-Leibler divergence between Q and P:

$$F = -\log P(\mathbf{z}) + \int Q \log \frac{Q}{P} \tag{2.9}$$

In our case we approximate the actual posterior with brief Gibbs sampling. Since the sampled distribution has not had time to reach equilibrium we actually sample from the approximate posterior Q. Assume that we have an infinite ensemble of networks, so that the approximate distribution Q and the gradient of F with respect to the parameters can be computed exactly. We will denote the distribution reached at the end of partial E-step t by Q^t and the energy function used during partial E-step t by E^t. Partial M-step t updates the energy function from E^t to E^{t+1}. By cautiously following the gradient of the log-likelihood during the partial M-step we reduce the energy:

$$\int Q^t E^{t+1}(\mathbf{z}) \leq \int Q^t E^t(\mathbf{z}) \tag{2.10}$$

Gibbs sampling during the partial E-step moves the approximate posterior Q closer to the true posterior even if the Markov chain has not reached equilibrium [Goutsias, 1991] (see appendix C):

$$\int Q^{t+1} E^{t+1}(\mathbf{z}) + \int Q^{t+1} \log Q^{t+1} \leq \int Q^t E^{t+1}(\mathbf{z}) + \int Q^t \log Q^t \tag{2.11}$$

Therefore $F^{t+1} \leq F^t$ for an infinite ensemble of networks.

The above argument assumes that Gibbs sampling in partial E-step $t+1$ starts from the final hidden states visited in step t. In this case, we can approximate an infinite ensemble by using a small learning rate in a single network, so that the energy functions of successive E-steps will be similar. Then (2.10) and (2.11) tell us that, even if the Markov chain has not reached equilibrium[1], we will be adjusting the weights so as to minimize an upper bound on the negative log-probability of the data.

We would like to avoid storing the result of previous Gibbs sampling sweeps. One alternative would be to learn an explicit bottom-up recognition model. Before Gibbs sampling begins we could do a bottom up pass to initialize the activities of the hidden units, and let Gibbs sampling proceed from this point. The recognition model could be learned using the difference between the next sample generated by the Gibbs sampler and this bottom-up initialization value. In this way, we could hope to start the Gibbs sampler close to what would have been the previous sample without having the extra overhead of storing the previous sample for each training case.

Instead we take the simpler approach of always starting the Gibbs sampler from the same initial state, and sampling for only a few sweeps, generating samples from some approximate posterior distribution Q. The free energy F is an upper bound on the negative log-probability of the data; the bound exceeds the negative log-probability by the Kullback-Leibler divergence between the true posterior P and the approximation Q. If the equilibrium distribution of the Markov chain is far from the initial value, this divergence term will be large, and the partial M-step will tend to improve F by decreasing the divergence. The model will be regularized towards models that can converge to the posterior from the initial state in only a few Gibbs sweeps. This is similar to the way that a model that uses a mean-field approximation tends to learn weights such that the

[1]Of course, the learning rate must be small enough that successive energy functions will be similar, but not so small that the weights remain unchanged, which would eventually let the Markov chain reach equilibrium

posterior comes closer to a factorial distribution.

In fact, brief Gibbs sampling is similar to a variational approximation. With a variational approximation, the parameters of Q are optimized so as to minimize the K-L divergence between the approximation and the true posterior. The approximation is explicitly chosen so that after the optimization step the expectations required can be calculated exactly. In the case of brief Gibbs sampling, we do not know the form of the approximation Q, but we can sample from it. We calculate expectations with respect to Q with Monte Carlo approximations. There are advantages to this approach: First, the approximating distribution can potentially be as complex as the true posterior. Second, we do not need to explicitly state what our approximating distribution is. We do not need to come up with one that is a good approximation to the posterior and can be used to analytically compute expectations. The disadvantage is that we are introducing two levels of approximation: The first is the approximate posterior Q which is caused by terminating Gibbs sampling prematurely. The second is the Monte Carlo approximation of Q. In other words, even the expectations found with respect to the approximation Q are only approximate due to sampling noise.

2.5 Comparison to Mixture of Gaussians, Factor Analysis and RGBN's

One advantage of the HCE model is that not only can it extract sparse, distributed representations from data, but it can also produce representations at either extreme. To demonstrate this, we generated two very simple 2-dimensional data sets of 1000 points each; one from a mixture of two circular Gaussians; and the second from a non-circular 2-dimensional Gaussian. The former can be well modeled by a mixture of Gaussians model, and the latter can be modeled by a factor analysis model with only one hidden factor. We trained the same simple two-layer HCE 100 times on each data set. In all

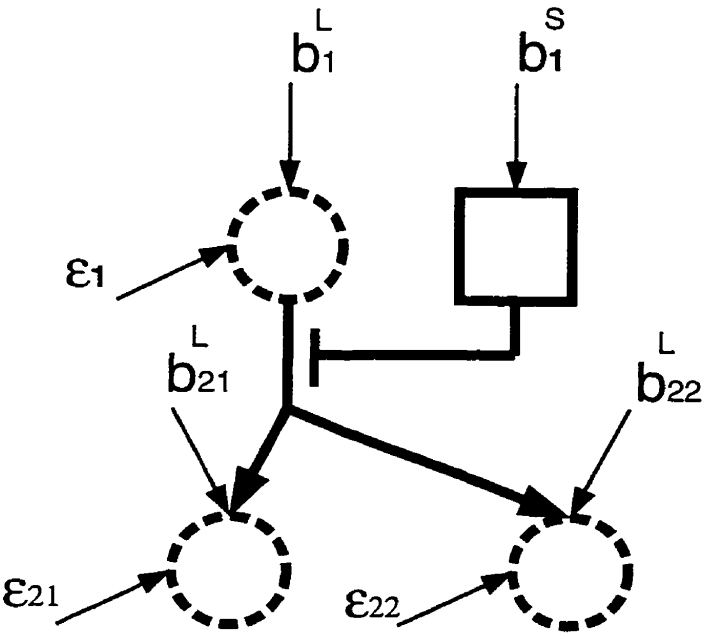

Figure 2.4: Architecture of a simple HCE to be used to model simple 2-D data sets. The top-level binary-logistic unit (solid square) gates the output of the top-level linear-Gaussian unit (dashed circle). The injected noise values ϵ_1, and ϵ_{21} and ϵ_{22} have variances σ_1^2 and σ_2^2 respectively.

cases, the HCE had one pair of units in the top layer, and two linear units in the visible layer (see figure 2.4).

2.5.1 Mixture of Gaussians

Each 2-D mixture of Gaussians datum was generated in the following way: one of the two possible Gaussians was chosen, and the data point was drawn from this Gaussian. The two Gaussian distributions used and their mixing proportions were the same as those used in section 2.1. See table 2.2 for a summary of the distribution used to produce the training data, and the distributions produced by the HCE's after training. A plot of the training data and a sample of 1000 points generated from the one of the trained models are shown in figure 2.5.

An HCE with the above architecture can model two Gaussians: one when the top-level binary unit is on, and one when it is off. Notice that, in this case, the HCE can model the

	μ	Ψ	π
Original Gaussian 1	(1.0, 2.0)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$	0.6
Original Gaussian 2	(-2.0,-1.5)	$\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$	0.4
Learned Gaussian 1	(0.98±0.13, 2.00±0.15)	$\begin{bmatrix} 0.51 \pm 0.02 & 0.0 \\ 0.0 & 0.51 \pm 0.02 \end{bmatrix}$	0.58
Learned Gaussian 2	(-1.92±0.56,-1.46±0.51)	$\begin{bmatrix} 0.76 \pm 0.11 & 0.30 \pm 0.08 \\ 0.30 \pm 0.08 & 0.87 \pm 0.12 \end{bmatrix}$	0.42

Table 2.2: Parameters of mixture of two Gaussians, and average parameters learned by the 100 trained HCE's (± two standard deviations). Learned Gaussian 1 is the Gaussian generated when the top-level binary unit is off, and Learned Gaussian 2 is generated when the top-level binary unit is on.

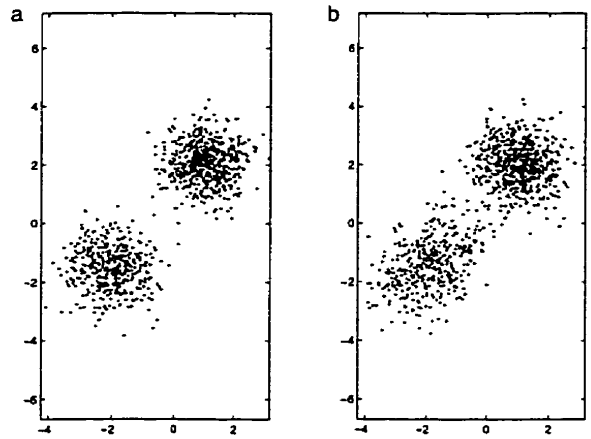

Figure 2.5: a) Training data generated from a mixture of circular Gaussians. b) Data generated from one of the trained HCE's.

one-on behaviour of the Gaussian mixture because there are only two Gaussians in the mixture, requiring one binary unit at the top level. If there were n Gaussian distributions in the mixture, the HCE would need $\log n$ layers to capture the one-on behaviour in a binary tree. Alternatively, lateral inhibitory connections between top-layer binary units could be learned.

2.5.2 Factor Analysis

Data was generated from a 2-D Gaussian that can be properly modeled by a factor analysis model with one hidden unit; see table 2.6 for a summary of the distribution used to produce the training data, and the distributions produced by the HCE's after training. A plot of the training data and a sample of 1000 points generated from one of the trained models are shown in figure 2.7.

	μ	Ψ	π
Original Gaussian	(3.5,-7.0)	$\begin{bmatrix} 0.8 & -1.6 \\ -1.6 & 5.0 \end{bmatrix}$	1.0
Learned Gaussian 1	(2.76±0.77,-4.80±2.27)	$\begin{bmatrix} 0.31 \pm 0.05 & 0.0 \\ 0.0 & 0.31 \pm 0.05 \end{bmatrix}$	0.02
Learned Gaussian 2	(3.56±0.24,-7.17±0.47)	$\begin{bmatrix} 0.84 \pm 0.22 & -1.59 \pm 0.44 \\ -1.59 \pm 0.44 & 5.00 \pm 1.10 \end{bmatrix}$	0.98

Figure 2.6: Parameters of a full covariance Gaussian, and average parameters learned by the 100 trained HCE's (± two standard deviations). Learned Gaussian 1 is the Gaussian generated when the top-level binary unit is off, and Learned Gaussian 2 is generated when the top-level binary unit is on.

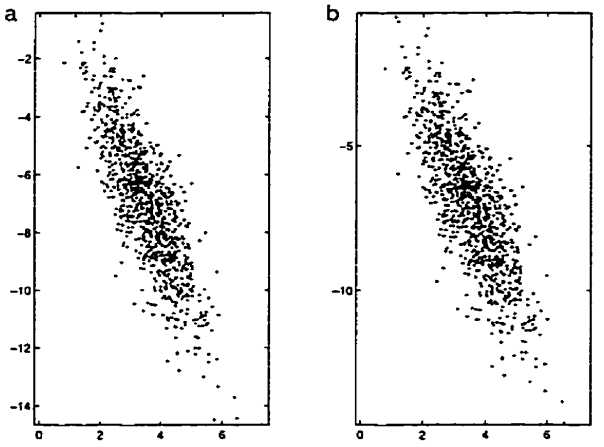

Figure 2.7: a) Training data generated from a full-covariance Gaussian. b) Data generated from one of the trained HCE's.

In this case, the HCE model faces similar restrictions as a factor analysis model. If there are more independent causes of the data than there are hidden units, then the HCE will have trouble modeling the data. However, unlike a factor analyzer, if the HCE has more hidden units than are required to model the data, it can simply bias the binary

units of the unnecessary pairs so that they are always gated out. Turning off unused units will not increase the expressiveness of the model, but it does make the model easier to interpret.

2.5.3 Rectified Gaussian Belief Nets

In the previous two sections we saw that an HCE can represent any distribution that can be represented by a factor analyser or by a mixture of Gaussians. It can duplicate the factor analyzer by biasing binary units to be always on or always off. It can potentially represent any distribution that can be modeled by a mixture of Gaussians (given enough units, and if the HCE has lateral connections or a sufficient number of hidden layers) by modeling each Gaussian in the mixture with a linear-Gaussian unit in the first hidden layer, and learning a one-on behaviour for the corresponding binary units.

We saw in section 2.1 that an HCE can also represent distributions that cannot be effectively modeled with an RGBN. A reasonable question is whether or not the converse is true: Can an RGBN model densities that an HCE cannot? Clearly, since an HCE can duplicate the behaviour of a mixture of Gaussians, it can potentially model arbitrary distributions. In fact, the set of densities modeled by an RGBN is a subset of those that can be modeled by a simple variant of the HCE with only one pair of units for every rectified Gaussian unit [Hinton and Ghahramani, 1997].

To show this we must first describe another type of binary unit: a *binary-probit* unit. Assume that the binary-probit unit i has inputs s_j, $j < i$. Define $\widehat{y}_i$ as:

$$\widehat{y}_i = \sum_{j<i} s_j + b_i \tag{2.12}$$

where b_i is the bias on unit i. Then the binary-probit unit emits a 1 with probability given by:

$$P(s_i = 1 | s_j, j < i) = \frac{1}{\sqrt{2\pi}} \int_{x=0}^{\infty} \exp\{-\frac{1}{2}(x - \widehat{y}_i)^2\} dx \tag{2.13}$$

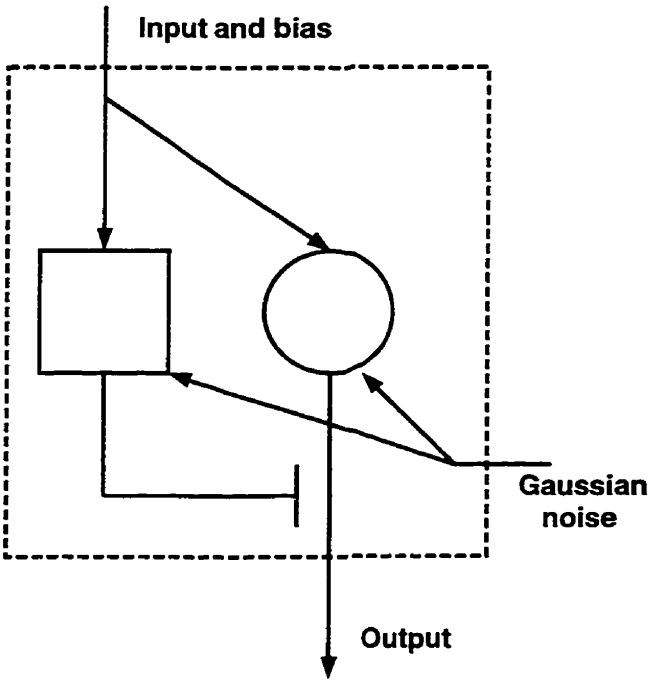

Figure 2.8: Converting an HCE to an RGBN. The solid square represents a binary-probit unit and the solid circle represents a linear-Gaussian unit. When they share the same input, bias and Gaussian noise, the result is a rectified-Gaussian unit (dashed square)

A binary-probit unit is like a binary-logistic unit with the logistic function replaced by a cumulative Gaussian.

Now consider the following modification to the HCE architecture (see figure 2.8): First, instead of using binary-logistic units, we will use binary-probit units. Second, we will require that the input, bias and internal Gaussian noise of each unit in a pair of binary and linear units be shared.

The combined linear-Gaussian/binary-probit pair emit a non-zero value when the the internal random variable in the binary-probit unit is greater than zero. However, since the random Gaussian noise and input are shared, this occurs exactly when the linear-Gaussian unit emits a value greater than zero. The combination of the two units is a rectified-Gaussian unit.

Chapter 3

Simulation Results

3.1 Noisy Bars

3.1.1 The Problem

The noisy bars task is a toy problem that demonstrates the need for sparse distributed representations [Hinton et al., 1995, Hinton and Ghahramani, 1997]. There are four stages in generating each $K \times K$ image. First a global orientation is chosen, either horizontal or vertical, with both cases being equally probable. Given this choice, each of the K bars of the appropriate orientation is turned on independently with probability 0.4. Next, each active bar is given an intensity, chosen from a uniform distribution. Finally, independent Gaussian noise is added to each pixel. A sample of images generated in this way is shown in figure 3.1(a).

3.1.2 HCE Results

We trained a 3-layer HCE network on the 6×6 noisy bars problem. The network consisted of one pair of units in the top hidden layer, where each pair consists of a linear-Gaussian unit gated by its corresponding binary logistic unit; 24 pairs of units in the first hidden layer; and 36 linear-Gaussian units in the visible layer. We used update rules with

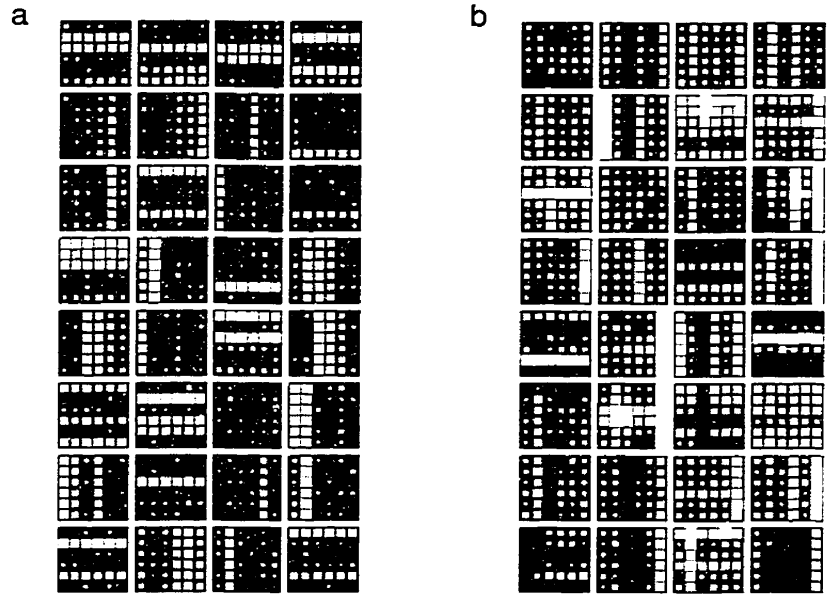

Figure 3.1: a) Training data for the noisy bars problem. b) Images generated by the trained network. The area of each square represents the value of the corresponding pixel in the 6×6 images. White represents positive values and black represents negative values.

weight-decay. The modified update rule for the binary-to-binary connections is given by:

$$\Delta w_{ji} = \epsilon[s_j(s_i - \widehat{s_i}) - \lambda w_{ji}] \tag{3.1}$$

where ϵ is the learning rate and λ is the weight decay parameter. The update rule for the linear-to-linear connections was similarly modified. The network was trained for 12 passes through a data set of of 1000 images, with a learning rate of 0.04 and a weight decay parameter of 0.04. The images were presented in a different, random order for each pass.

For each image presented, 16 Gibbs sampling iterations were performed. Gibbs sampling was performed by visiting each pair of units in a layer in random order, where for each pair the binary unit was visited first, followed by the linear unit. Of the 16 network states visited, the first four were discarded, and the next 12 were used for learning. The weights from the linear units in the first hidden layer to the units in the visible layer were

constrained to be positive. Without this constraint, the trained model learns a similar distribution, but the solution is not so easily interpreted. The result of training is shown in figure 3.2.

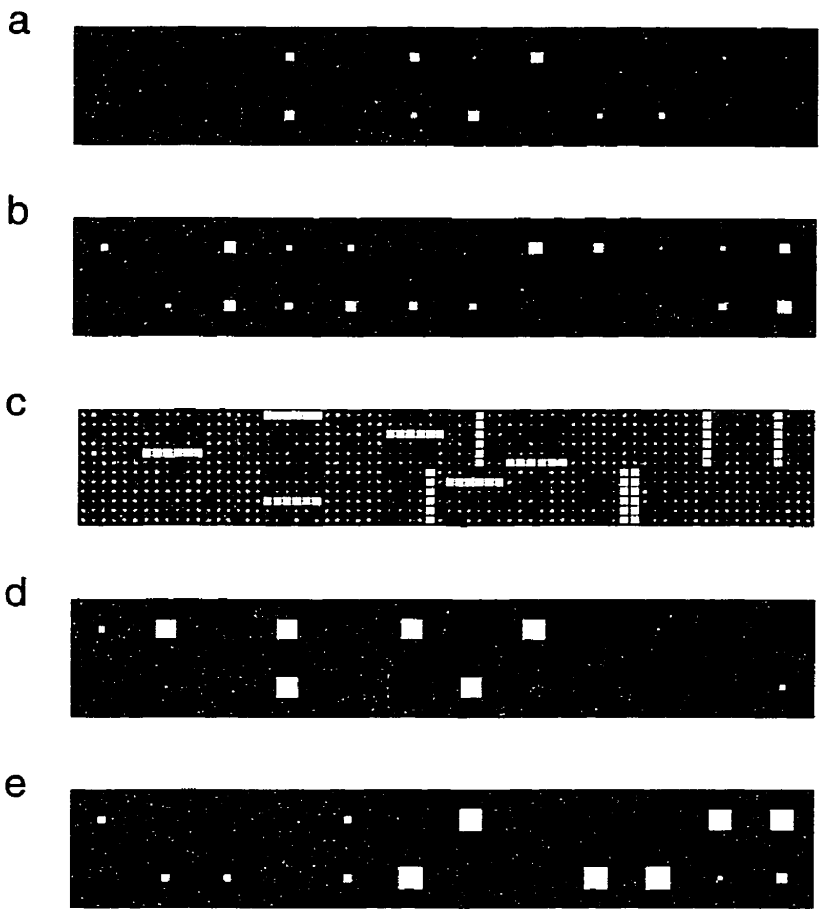

Figure 3.2: Generative weights and biases of a three-layered network after being trained on the noisy bars problem. a) weights from the top layer linear-Gaussian unit to the 24 middle layer linear-Gaussian units. b) Biases of the middle layer linear units. c) weights from the 24 middle layer linear units to the 36 visible units. d) weights from the top layer binary logistic unit to the 24 middle layer binary logistic units. e) Biases of the middle layer binary logistic units.

The trained network is using 12 of the linear-Gaussian units in the first hidden layer to represent each of the 12 possible horizontal and vertical bars. The top level binary unit is selecting the linear units in the first hidden layer that represent horizontal bars by exciting the corresponding binary units; these binary units are biased to be off otherwise. Similarly, the binary units that correspond to vertical bars, which are often active due to

positive biases, are being inhibited by the top binary unit. The top linear unit is simply acting as an additional bias on the linear units in the first hidden layer. Examples of data generated by the trained network are shown in figure 3.1(b). As can be seen in figure 3.1(b), the distribution learned by the model is not perfect. The top binary unit is biased to be on 51.7% of the time, which is quite close to the 50% used by the process that generated the bars. However, when generating vertical images (when the top-level binary unit is off), the trained HCE still allows horizontal bars to be used (with average probability 24.8% for each horizontal bar). Ideally, the negative biases on the binary units corresponding to horizontal bars should be larger (see figure 3.2(e)).

The network was shown novel images, and 10 iterations of Gibbs sampling were performed. After the final iteration, the top level binary unit was found to be off for 90% of vertical images, and on for 84% of horizontal images.

The results shown in figure 3.2 should be contrasted with the features learned by other models on the same problem. Figure 3.3 shows the features learned by a factor analyzer with 24 hidden factors, a mixture of 24 Gaussians, an ICA model (with the required 36 hidden units), and an RGBN with 24 hidden units in the middle-layer, and one hidden unit in the top-layer.

The factor analyzer has not learned the distinction between horizontal and vertical bars. The mixture of Gaussians has learned this, and with enough Gaussians in the mixture, could learn the distribution exactly. Unfortunately, even if the bars were binary, it would require 2^7 Gaussians just to represent all possible combinations of horizontal and vertical bars. For the real-valued bars problem, it would need additional Gaussians to model the uniform intensity distribution for each bar. The ICA and RGBN models learn the distinction between horizontal and vertical bars, and find sparse, distributed representations for the bars. The ICA features are obscured by noise, but the RGBN learns the same features as the HCE with its weights restricted to be non-negative. If the HCE is free to choose its weights with no restrictions, the resultant features are

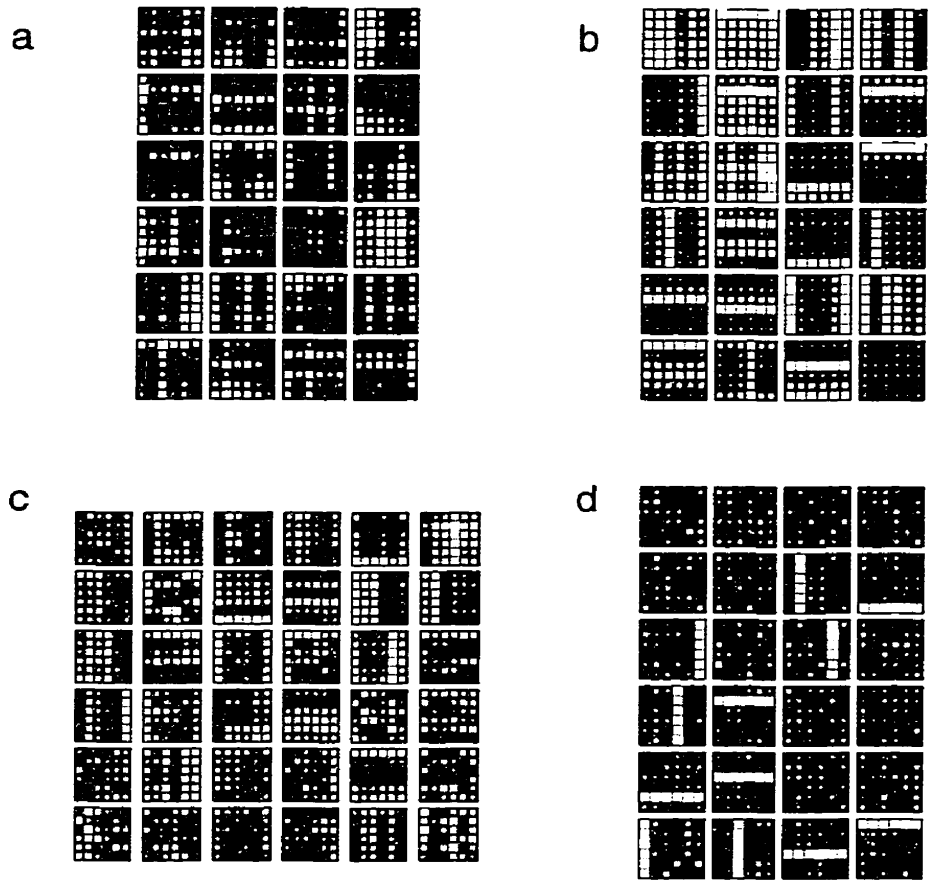

Figure 3.3: Generative weights of other density estimators, trained on the noisy bars data. a) Factor analyzer with 24 hidden factors. b) Mixture of 24 (diagonal covariance) Gaussians. c) ICA (with 36 hidden units). d) 3-layer RGBN with 24 hidden units in the middle-layer, and 1 unit in the top-layer.

qualitatively similar to those learned by the ICA model, but without being so obscured by noise (see figure 3.4). With this restriction removed, the HCE's top-level binary unit still learns to distinguish between horizontal and vertical bars.

3.2 Hand-Written Digits

3.2.1 Classification

We trained a similar three-layer network on handwritten twos and threes from the CEDAR CD ROM 1 database [Hull, 1994]. The digits were scaled to an 8×8 grid,

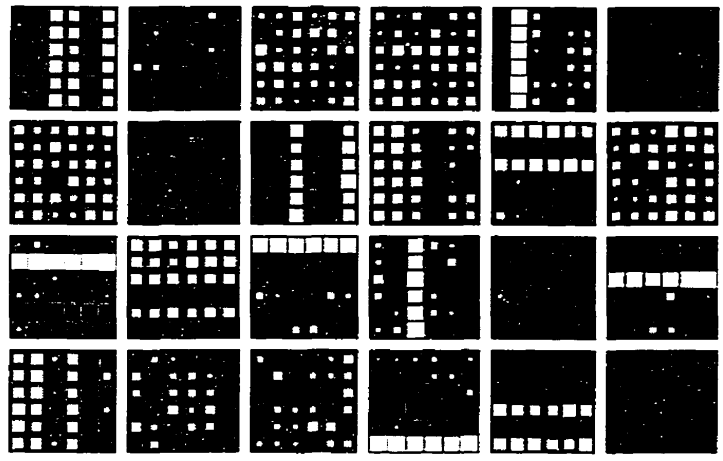

Figure 3.4: Generative weights of an HCE trained on the noisy bars, with no restrictions on the weights. The features are reminiscent of those learned by ICA.

and the 256-gray-scale pixel values were rescaled to lie within $[0, 1]$. The 2000 digits were divided into a training set of 1400 digits, and a test set of 600 digits, with twos and threes being equally represented in both sets. A small subset of the training data is shown in figure 3.5(a).

The network consisted of a single pair of units in the top hidden layer, 24 pairs of units in the first hidden layer, and 64 linear-Gaussian units in the visible layer. During training, the network made 43 passes through the data set, with a learning rate of 0.01 and a weight decay parameter of 0.02. Gibbs sampling was performed as in the bars problem, with 4 discarded Gibbs sampling iterations, followed by 12 iterations used for learning. For this task, there were no constraints placed on the sign of the weights from the linear-Gaussian units in the first hidden layer to the units in the visible layer. The result of training is shown in figure 3.6.

In this case, the network uses all 24 linear units in the first hidden layer to represent digit features. Some of the features span the entire image, and act as templates of digits. Other features are highly localized, and can modify the templates. The top binary unit selects the linear units in the first hidden layer that correspond to features found predominantly in threes, by exciting the corresponding binary units. Features that are

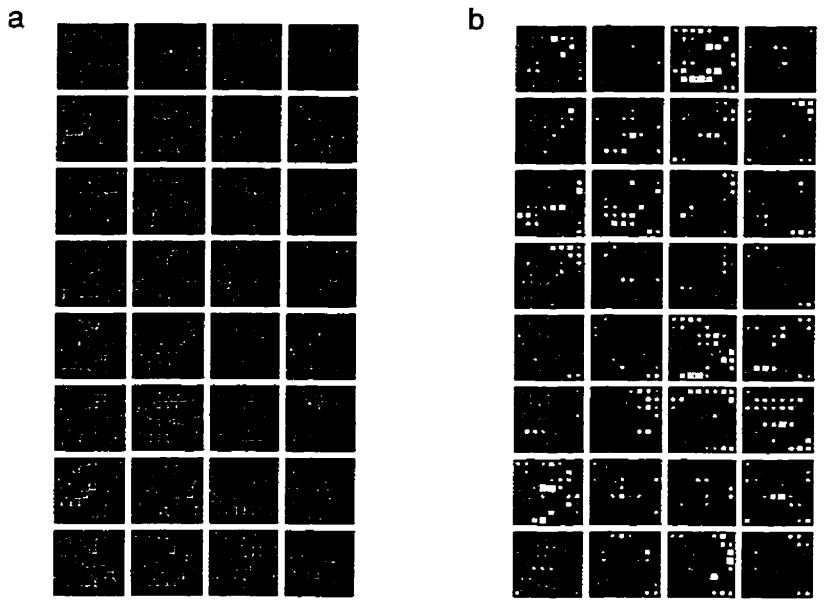

Figure 3.5: a) A subset of the training data. b) Images generated by the trained network. For clarity, black represents positive values in this figure.

exclusively used in twos are being gated out by the top binary unit, while features that can be shared between digits are being only slightly excited or inhibited. When the top binary unit is off, the features found in threes are inhibited by strong negative biases, while features used in twos are gated in by positive biases on the corresponding binary units. Unlike the bars problem where there were no shared features between "horizontal" and "vertical" data points, the two classes of data in this problem can potentially share features. The HCE takes advantage of this by finding common features and biasing them so that they can be active for examples from either class. For example, the feature in the lower right corner of figure 3.6(c) is a template of a two, which is inhibited when the top-level binary unit is on and excited otherwise (figure 3.6 (d) and (e)). The feature in the lower left corner of figure 3.6(c) can be used to shift the position of the upper stroke of a digit. Since such strokes appear in both twos and threes, the corresponding binary unit is only slightly excited by the top-level binary unit. Examples of data generated by the trained network are shown in figure 3.5(b).

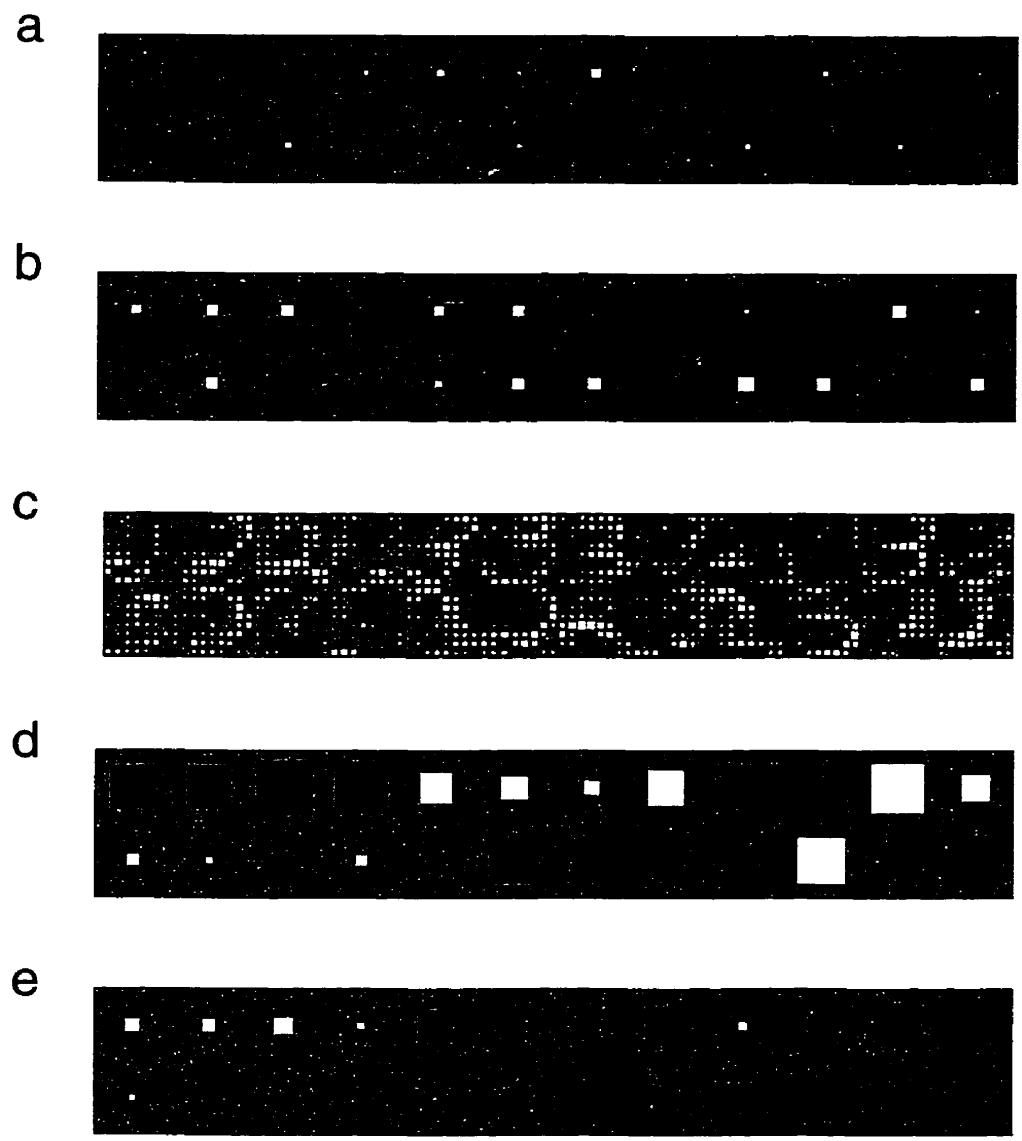

Figure 3.6: Generative weights and biases of a three-layered network after being trained on handwritten twos and threes. a) weights from the top layer linear-Gaussian unit to the 24 middle layer linear-Gaussian units. b) Biases of the middle layer linear-Gaussian units. c) weights from the 24 middle layer linear-Gaussian units to the 36 visible units. d) weights from the top layer binary logistic unit to the 24 middle layer binary logistic units. e) Biases of the middle layer binary logistic units.

The trained network was shown 600 test images, and 10 Gibbs sampling iterations were performed for each image. The top level binary unit was found to be off for 94% of twos, and on for 84% of threes. We then tried to improve classification by using prolonged Gibbs sampling. In this case, the first 300 Gibbs sampling iterations were discarded, and the activity of the top binary unit was averaged over the next 300 iterations. If the average activity of the top binary unit was above a threshold of 0.32, the digit was classified as a three; otherwise, it was classified as a two. The threshold was found by calculating the optimal threshold needed to classify just 10 of the training samples under the same prolonged Gibbs sampling scheme. The reason we used only 10 examples to set the threshold was to demonstrate that good classification can be achieved with very little labeled training data if an unsupervised learning algorithm has already extracted a good representation. With prolonged Gibbs sampling, the average activity of the top binary unit was found to be below threshold for 96.7% of twos, and above threshold for 95.3% of threes, yielding an overall misclassification rate of 4% (with no rejections allowed). Histograms of the average activity of the top level binary unit are shown in figure 3.7.

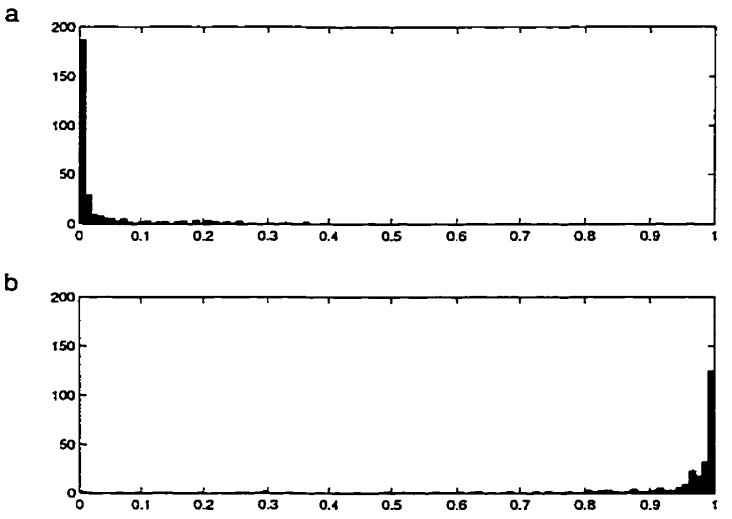

Figure 3.7: Histograms of the average activity of the top level binary unit, after prolonged Gibbs sampling, when shown novel handwritten twos and threes. a) Average activity for twos in the test set. b) Average activity for threes in the test set.

3.2.2 Representation

In the previous example we saw that an HCE can share features between classes. By using a distributed representation of the digits the network can reuse hardware instead of creating redundant features.

Another benefit of this type of representation is that previously unseen classes of data can be accommodated. To demonstrate this, we trained a network on 16×16 handwritten digits from the CEDAR CD-ROM database, scaled to have pixel intensities in the range $[0, 1]$. Digits from classes 0 through 8 were used, and there were 800 examples of each class. No examples of the digit 9 were shown to the network. The network had 256 linear units in the visible layer, and 64 pairs of units in a single hidden layer. The network was trained for 14 passes through the data set with a learning rate of 0.01 and weight decay of 0.005.

After training, the network was shown 300 previously unseen examples of each of the 10 digit classes, including the digit 9. Representations were found for each of these examples in the following way: First 50 iterations of Gibbs sampling were discarded. Then seven additional sweeps of Gibbs sampling were performed. For each of the seven sweeps, a reconstruction was formed by passing the activities in the hidden layer through the generative weights. An average reconstruction was then found by averaging these seven reconstructions. The squared difference between each original digit and its average reconstruction was measured. Figure 3.8 shows the average squared error for each digit class, plus or minus two standard deviations.

Notice that the average reconstruction error for the 9's, a previously unseen class of digits, is no greater than for other similar classes of digits. The network is able to accommodate the new class with the existing features. This is possible because the network finds a sparse, distributed representation, resulting in general, localized features. Unlike in the previous example, no global templates are learned. This reflects the fact that the network had to represent a more diverse data set. Figure 3.9 shows a randomly

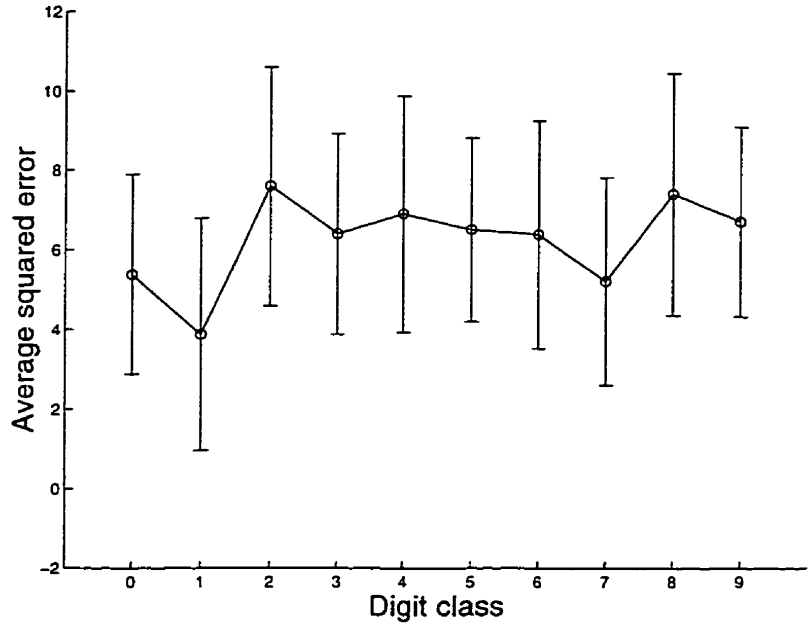

Figure 3.8: Average squared reconstruction error for 10 digit classes (plus or minus two standard deviations).

selected data point from each digit class; the network's reconstruction; and how the network has used the learned set of features in each case.

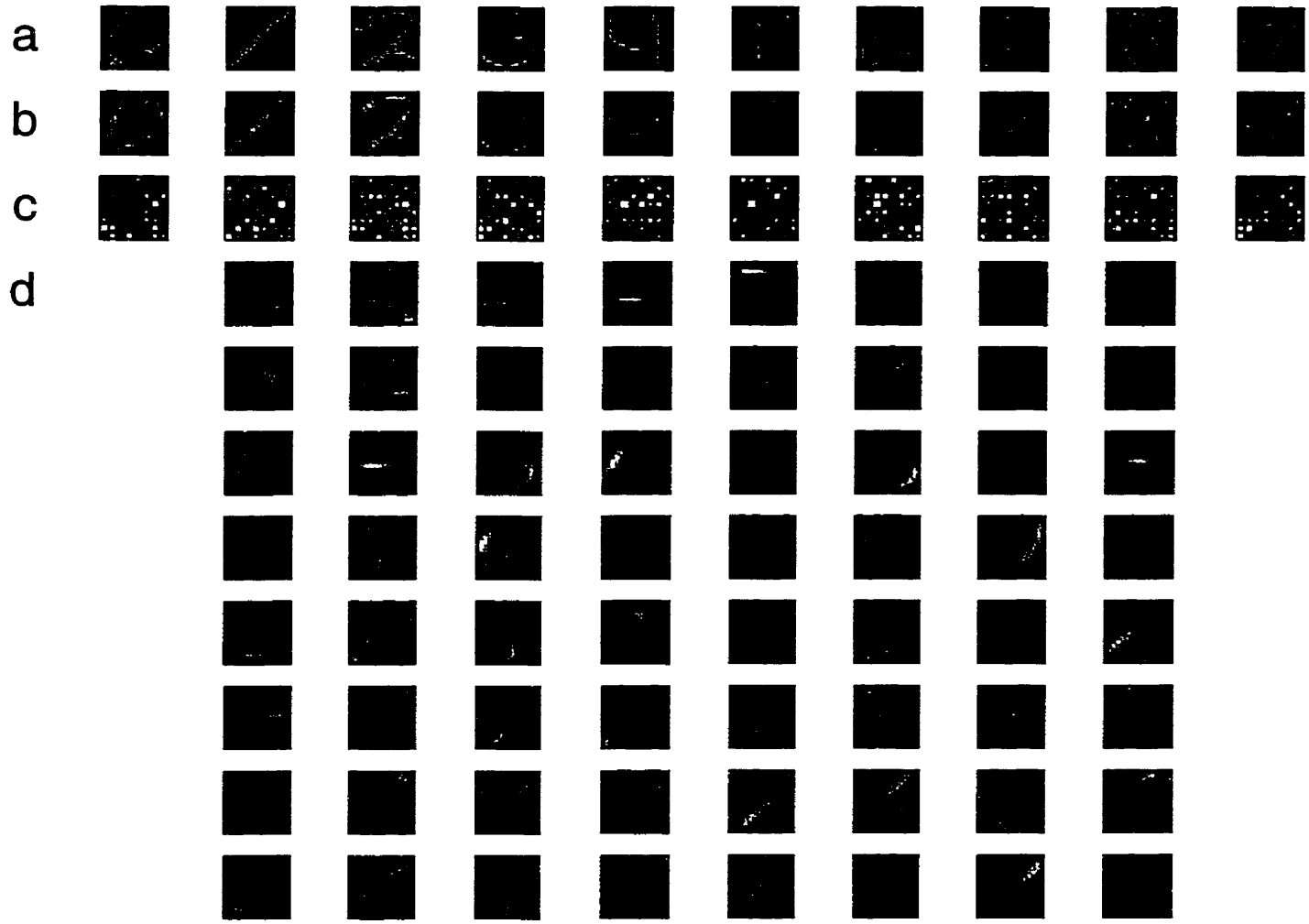

Figure 3.9: a) A randomly selected example from each digit class. b) The network's reconstruction of each example. c) The average gated activity for each feature. White denotes positive and black denotes negative. The area of a square denotes magnitude, where the largest square in the figure has magnitude 0.49. The reconstruction is formed by multiplying the average gated activity by the corresponding feature from (d), and summing over all features. d) The features learned by the network sorted by average activity, over the entire test set, of the associated binary unit. The upper-left-most feature was the least active over the 3000 test examples, and the lower-right-most feature was the most active.

Chapter 4

Conclusion

We have described a hierarchical generative model that can dynamically synthesize a linear network from a large number of available linear-Gaussian units. The synthesis mechanism is a network of binary-logistic units that gate the outputs of the linear units. Maximum-likelihood model parameters can be learned using a local delta rule, where the statistics required can be approximated with Gibbs sampling. We have shown that the network can learn interesting tasks by increasing a lower bound on the log-likelihood, even when the Gibbs sampling is so brief that the Markov chain is far from equilibrium. Because each linear unit specializes in capturing a particular feature, and any combination of linear units can co-operate to explain an observation, we call this model a hierarchical community of experts.

4.1 Discussion

The simulations reported in chapter 3 are encouraging, and show that a Hierarchical Community of Experts has some very desirable features. It makes appropriate use of binary and real-valued random variables to model features of data. It learns low-level localized features in lower layers, and finds relationships among features in higher layers. It uses sparse internal representations that are distributed across many units. The result

is that many hidden units are used to represent each datum, and features are shared across different classes of data. Learned features are general enough to accommodate previously unseen classes of data.

Sparse, distributed representations are encouraged by the prior distribution over the combined linear-binary pairs of units. It is easy for the HCE to place a large probability mass exactly on zero, encouraging it to find representations where many features are gated out for each data point. Each gated unit is like a mixture model, where one density estimator in the mixture is a spike at zero, and the other is a Gaussian. The mixing proportion and the mean of the Gaussian are learned by two distinct parameters, so this model does not suffer from some of the representational difficulties of models with rectified Gaussian priors. There is a clear distinction in the prior of a hidden unit between a feature being absent, and a feature being present with a small magnitude. This distinction is not made by ICA or other models that use a high-kurtosis unimodal prior over hidden units.

Inference in an HCE is intractable, so we use Gibbs sampling. Two of the drawbacks of Gibbs sampling are that it can be slow to converge to the posterior, and it is difficult to tell when convergence has occurred. We avoid these problems by using "brief" Gibbs sampling where we use only 10 to 20 samples per data presentation. Gibbs sampling can be viewed as coordinate descent in free energy, and we have shown that learning can occur even if Gibbs sampling does not reach equilibrium. As a result, we need not start each sweep of Gibbs sampling from the previous state, making online learning feasible. We use a simple constant distribution to initialize the Gibbs sampler before each sweep, but more sophisticated initialization schemes could be used.

Monte Carlo methods are not without disadvantages. If the correlations to be learned by upper layers are subtle, the estimate of the gradient of the weights can be overwhelmed by sampling noise, making learning impossible. By using a constant distribution to initialize the Gibbs sampling, we increase the bias and reduce the variance of the gradient

estimate. Of course, at some point the gradient will be dominated by the bias, again making learning impossible. Since noise in the gradient estimate can cause problems, one possible solution is to use a deterministic approximation of the posterior. It remains to be seen if a version of the HCE which does approximate inference with a variational method can learn more complicated tasks than the Monte Carlo-based HCE.

4.2 Future Work

The current model assumes i.i.d. data, but it should be possible to extend the model to time-series data, by making the prior on the hidden units conditional on the state at previous times. Inference would still be intractable, but if the prior on the linear units were only influenced by the values of other linear units (either at earlier times or in higher layers) then Gibbs sampling should still be feasible.

More important is the problem of improving the representations that are extracted in higher levels of the model. In our simulations, the overwhelming conclusion is that interesting low-level features are consistently extracted by the model, but the model is much less consistent in solving the (much more interesting) problem of finding high-level representations. For some (especially binary) classification tasks an HCE can find the classes in an unsupervised fashion, but when the data becomes more complex the model is not as successful. In fact, this is a problem for many of the generative models being investigated today.

Obviously learning features from clamped data is easier than finding correlations among noisy samples drawn from the posterior of a hidden layer. What, if anything, can be done to make the latter task easier? If we increase the length of Gibbs sampling, presumably the sampling noise will decrease, at the cost of greater computation time. Instead, we could have an ensemble of units, all of which could model similar features and be sampled in parallel. We could encourage units in the hidden layer to learn similar

features by inhibiting explaining away among these units. In the fully-connected model, if two units try to learn the same feature, one unit is typically explained away by the other when the that unit's feature is required to explain the data. The result is that only one of the copies of the feature is in use at any one time, and over time one of the units learns some other feature. Unfortunately, in complex data, one class might be allocated only a few units. Given a noisy sample from the posterior over hidden units, it would be very difficult to detect correlations among these groups of just a few units. If we inhibit explaining away among some groups of units, there is a greater likelihood that multiple units will learn the same feature, and the activities of these units would not be anti-correlated due to explaining away. This might result in samples with greater redundancy, in which it would be easier to detect higher-level structure.

Appendix A

Integrating Out a Hidden Layer

In order to perform Gibbs' sampling for a layer of binary units, we want to integrate out some of the linear units in the same layer. To integrate out all of the linear units in the hidden layer we must evaluate the integral:

$$P(\mathbf{x}, \mathbf{s}) = \int P(\mathbf{x}|\mathbf{s}, \mathbf{y}) P(\mathbf{y}) P(\mathbf{s}) d\mathbf{y} \tag{A.1}$$

where $\mathbf{x}$ are the values of the real-valued visible units; $\mathbf{y}$ are the values of the real-valued hidden units; and $\mathbf{s}$ are the values of the binary hidden units.

If we assume that the linear units are Gaussian and are gated by the binary units, then (A.1) becomes:

$$\begin{aligned} P(\mathbf{x}, \mathbf{s}) \;=\; P(\mathbf{s}) \int C \exp\Big\{ & \frac{-1}{2} (\mathbf{x} - \mathbf{W}(\mathbf{y} \odot \mathbf{s}) - \mathbf{b}_v)^\top \mathbf{\Psi}_v^{-1} (\mathbf{x} - \mathbf{W}(\mathbf{y} \odot \mathbf{s}) - \mathbf{b}_v) \\ & + \frac{-1}{2} (\mathbf{y} - \mathbf{b}_h)^\top \mathbf{\Psi}_h^{-1} (\mathbf{y} - \mathbf{b}_h) \Big\} d\mathbf{y} \end{aligned} \tag{A.2}$$

where $\mathbf{W}$ is the matrix of weights from the hidden to the visible real-valued units; $\mathbf{b}_v$ and $\mathbf{b}_h$ are biases on the visible and hidden real-valued layers respectively; $\mathbf{\Psi}_v$ and $\mathbf{\Psi}_h$ are covariance matrices for the visible and hidden real-valued layers respectively; $\odot$ denotes

element-wise multiplication; and C is the normalizing constant:

$$C = \frac{1}{(2\pi)^{(D+K)/2}|\mathbf{\Psi}_v|^{1/2}|\mathbf{\Psi}_h|^{1/2}} \tag{A.3}$$

where K and D are the dimensionality of the hidden and visible layers respectively.

By expanding (A.2) and comparing to an arbitrary Gaussian distribution we can compute the value of the integral:

$$\begin{aligned} P(\mathbf{x},\mathbf{s}) &= P(\mathbf{s}) \int C \exp\left\{ \frac{-1}{2} \left[\mathbf{x}^\top \mathbf{\Psi}_v^{-1} \mathbf{x} - 2\mathbf{x}^\top \mathbf{\Psi}_v^{-1} (\mathbf{W}(\mathbf{y} \odot \mathbf{s}) + \mathbf{b}_v) \right.\right. \\ &\qquad + (\mathbf{W}(\mathbf{y} \odot \mathbf{s}) + \mathbf{b}_v)^\top \mathbf{\Psi}_v^{-1} (\mathbf{W}(\mathbf{y} \odot \mathbf{s}) + \mathbf{b}_v) \\ &\qquad \left.\left. + \mathbf{y}^\top \mathbf{\Psi}_h^{-1} \mathbf{y} - 2\mathbf{y}^\top \mathbf{\Psi}_h^{-1} \mathbf{b}_h + \mathbf{b}_h^\top \mathbf{\Psi}_h^{-1} \mathbf{b}_h \right] \right\} d\mathbf{y} \qquad \text{(A.4)} \\ &= P(\mathbf{s}) \int C \exp\left\{ \frac{-1}{2} (\mathbf{y} - \mu)^\top \mathbf{\Phi}^{-1} (\mathbf{y} - \mu) - \frac{1}{2} D \right\} d\mathbf{y} \\ &= P(\mathbf{s}) \int C \exp\left\{ \frac{-1}{2} \left[\mathbf{y}^\top \mathbf{\Phi}^{-1} \mathbf{y} - 2\mathbf{y}^\top \mathbf{\Phi}^{-1} \mu + \mu^\top \mathbf{\Phi}^{-1} \mu + D \right] \right\} d\mathbf{y} \qquad \text{(A.5)} \end{aligned}$$

where μ and $\mathbf{\Phi}$ are the mean and covariance matrix of the Gaussian posterior distribution, and D is a constant required to complete the square. Comparing (A.4) to (A.5) yields:

$$\begin{aligned} \mathbf{\Phi}^{-1} &= \mathbf{S}^\top \mathbf{W}^\top \mathbf{\Psi}_v^{-1} \mathbf{W} \mathbf{S} + \mathbf{\Psi}_h^{-1} && \text{(A.6)} \\ \mu &= \mathbf{\Phi} \left[\mathbf{W} \mathbf{S} \mathbf{\Psi}_v^{-1} \mathbf{x} + \mathbf{\Psi}_h^{-1} \mathbf{b}_h \right] && \text{(A.7)} \\ D &= (\mathbf{x} - \mathbf{b}_v)^\top \mathbf{\Psi}_v^{-1} (\mathbf{x} - \mathbf{b}_v) + \mathbf{b}_h^\top \mathbf{\Psi}_h^{-1} \mathbf{b}_h - \mu^\top \mathbf{\Phi}^{-1} \mu && \text{(A.8)} \end{aligned}$$

where $\mathbf{S}$ is a diagonal $K \times K$ matrix with $\mathbf{s}$ placed along the diagonal. Notice that postmultiplying $\mathbf{W}$ by $\mathbf{S}$ simply zeros out the columns of $\mathbf{W}$ that correspond to gated-out units in the hidden layer. If no linear units were gated out, $\mathbf{S}$ would equal the $K \times K$ identity matrix, and (A.6)-(A.8) would be identical to the result for a linear-Gaussian network.

We can now evaluate the integral from (A.1) by combining (A.3) with (A.6)-(A.8):

$$\begin{aligned} P(\mathbf{x},\mathbf{s}) &= P(\mathbf{s})\ C\ \exp\left\{\frac{-1}{2}D\right\}\int \exp\left\{\frac{-1}{2}(\mathbf{y}-\mu)^{\top}\mathbf{\Phi}^{-1}(\mathbf{y}-\mu)\right\}d\mathbf{y} \\ &= P(\mathbf{s})\ C\ \exp\left\{\frac{-1}{2}D\right\}(2\pi)^{K/2}|\mathbf{\Phi}|^{1/2} \\ &= P(\mathbf{s})\frac{|\mathbf{\Phi}|^{1/2}}{(2\pi)^{D/2}|\mathbf{\Psi}_v|^{1/2}|\mathbf{\Psi}_h|^{1/2}}\exp\left\{\frac{-1}{2}D\right\} \end{aligned} \tag{A.9}$$

To perform Gibbs sampling we define $E^{\alpha} = -\log P(\mathbf{x},\mathbf{s}^{\alpha})$, and use (1.16) to sample from the i^{th} binary unit. Notice that in order to evaluate (A.9) we must invert a $K \times K$ matrix. We can avoid this if instead of integrating out the entire linear hidden layer we just integrate out the i^{th} linear unit when sampling from the i^{th} binary unit. The cost is greater sampling noise.

Appendix B

Update Rules

The complete-data likelihood for a two-layer HCE with K pairs of binary and linear units in the top layer and D linear units in the bottom layer is given by:

$$\begin{aligned}\mathcal{L} \;=\; & \left[\frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}_2|^{1/2}}\exp\left\{\frac{-1}{2}(\mathbf{y}-\widehat{\mathbf{y}})^{\top}\mathbf{\Sigma}_2^{-1}(\mathbf{y}-\widehat{\mathbf{y}})\right\}\right] \\ & \times\left[\frac{1}{(2\pi)^{K/2}|\mathbf{\Sigma}_1|^{1/2}}\exp\left\{\frac{-1}{2}(\mathbf{x}-\widehat{\mathbf{x}})^{\top}\mathbf{\Sigma}_1^{-1}(\mathbf{x}-\widehat{\mathbf{x}})\right\}\right] \\ & \times\prod_i \sigma(\widehat{s}_i)^{s_i}(1-\sigma(\widehat{s}_i))^{1-s_i} \qquad \text{(B.1)}\end{aligned}$$

where

$$\begin{aligned}\widehat{y} &= \mathbf{W}(\mathbf{x}\odot\mathbf{s})+\mathbf{b}_2^l \\ \widehat{x} &= \mathbf{b}_1^l \\ \widehat{s}_i &= \mathbf{b}_1^b \qquad \text{(B.2)}\end{aligned}$$

In the above, $\mathbf{W}$ is the matrix of linear-to-linear weights; $\mathbf{\Sigma}_2$ and $\mathbf{\Sigma}_1$ are the diagonal covariance matrices of the visible and hidden layers respectively; $\mathbf{y}$, $\mathbf{x}$ and $\mathbf{s}$ are the activities of the visible linear, hidden linear and hidden binary units respectively; $\mathbf{b}_2^l$, $\mathbf{b}_1^l$ and $\mathbf{b}_1^b$ are the biases on the visible linear, hidden linear and hidden binary units

respectively; and $\odot$ denotes element-wise multiplication.

Taking the negative logarithm of (B.1) yields:

$$\begin{aligned} E \;=\; & \frac{1}{2}\log|\boldsymbol{\Sigma}_2| + \frac{1}{2}(\mathbf{y}-\hat{\mathbf{y}})^\top \boldsymbol{\Sigma}_2^{-1}(\mathbf{y}-\hat{\mathbf{y}}) \\ & + \frac{1}{2}\log|\boldsymbol{\Sigma}_1| + \frac{1}{2}(\mathbf{x}-\hat{\mathbf{x}})^\top \boldsymbol{\Sigma}_1^{-1}(\mathbf{x}-\hat{\mathbf{x}}) \\ & + \sum_i s_i \log\sigma(\hat{s}_i) + (1-s_i)\log(1-\sigma(\hat{s}_i)) + C_n \end{aligned} \tag{B.3}$$

where C_n is a constant.

The update rules for the parameters associated with the binary units are identical to those in [Neal, 1992], and the reader is referred there for a derivation. We can find update rules for the linear-unit parameters by taking derivatives of (B.3) with respect to these parameters:

$$\frac{\partial E}{\partial \mathbf{W}} = -\boldsymbol{\Sigma}^{-1}(\mathbf{y}-\hat{\mathbf{y}})(\mathbf{x}\odot\mathbf{s})^\top \tag{B.4}$$

$$\frac{\partial E}{\partial \mathbf{b}_2^l} = -\boldsymbol{\Sigma}^{-1}(\mathbf{y}-\hat{\mathbf{y}}) \tag{B.5}$$

$$\frac{\partial E}{\partial \boldsymbol{\Sigma}^{-1}} = -\boldsymbol{\Sigma} + (\mathbf{y}-\hat{\mathbf{y}})(\mathbf{y}-\hat{\mathbf{y}})^\top \tag{B.6}$$

These yield the scalar update equations:

$$\Delta w_{ji} = \epsilon(y_i-\hat{y}_i)x_j s_j/\sigma_i^2 \tag{B.7}$$

$$\Delta b_{2i}^l = \epsilon(y_i-\hat{y}_i)/\sigma_i^2 \tag{B.8}$$

$$\Delta\sigma_j = \epsilon\left[(y_i-\hat{y}_i)-\sigma_i^2\right] \tag{B.9}$$

where ϵ is a learning rate. We have enforced the diagonality constraint on $\boldsymbol{\Sigma}_2$ in (B.9) by simply ignoring the off-diagonal terms in (B.6).

Notice that the update rules derived here will use the current (sampled) values of linear units in the lower layer. If these units were gated out by corresponding binary

units, then we could easily (and correctly) integrate over these units by ignoring the associated energy term. The result is that there is no update to the parameters associated with a unit when that unit is gated out. The effect of eliminating the appropriate terms from (B.3) is the same as multiplying the update for parameters associated with a linear unit y_i by the corresponding binary value s_i:

$$\Delta w_{ji} = \epsilon s_i (y_i - \hat{y}_i) x_j s_j / \sigma_i^2 \tag{B.10}$$

$$\Delta b'_{2i} = \epsilon s_i (y_i - \hat{y}_i) / \sigma_i^2 \tag{B.11}$$

$$\Delta \sigma_j = \epsilon s_j \left[(y_j - \hat{y}_j)^2 - \sigma_j^2 \right] \tag{B.12}$$

These are the update rules stated in section 2.3.

Appendix C

Gibbs Sampling Improves K-L Divergence

We need to show that a sweep of Gibbs sampling brings the approximating distribution Q^{t+1} closer in terms of K-L divergence to the true posterior than the previous approximating distribution Q^t. The following theorem and proof are almost exactly those given in [Goutsias, 1991], except that this proof involves $\mathrm{KL}(Q\|P)$ and not $\mathrm{KL}(P\|Q)$.

Theorem: If $\{T_m(x,y)\}$ is a set of transition probabilities such that

$$P(y) = \int P(x)T_m(x,y)dx \tag{C.1}$$

and

$$Q^{t+1}(y) = \int Q^t(x)T_m(x,y)dx \tag{C.2}$$

and if $Q^t(x) > 0$, $P(x) > 0$ and $T_m(x,y) > 0$ for all states x, y then $\mathrm{KL}(Q^{t+1}\|P) \leq \mathrm{KL}(Q^t\|P)$.

Proof: First note that if $Q^t(x) > 0$ and $T_m(x,y) > 0$ for all states x, y then so is $Q^{t+1}(x)$ by (C.2). Define $q_m(y,x)$ and $r_m(y,x)$ as follows:

$$q_m(y,x) = \frac{P(x)T_m(x,y)}{P(y)} \tag{C.3}$$

$$r_m(y,x) = \frac{Q^t(x)T_m(x,y)}{Q^{t+1}(y)} \tag{C.4}$$

Notice that $\int q_m(y,x)dx = 1$ and $\int r_m(y,x)dx = 1$, making $q_m(x,y)$ and $r_m(x,y)$ proper densities over states x. Further, note that:

$$\begin{aligned} \int Q^{t+1}(y)r_m(y,x)dx &= \int Q^{t+1}(y)\frac{Q^t(x)T_m(x,y)}{Q^{t+1}(y)}dx \\ &= Q^{t+1}(y) \end{aligned} \tag{C.5}$$

and

$$\begin{aligned} \int Q^{t+1}(y)r_m(y,x)dy &= \int Q^{t+1}(y)\frac{Q^t(x)T_m(x,y)}{Q^{t+1}(y)}dy \\ &= Q^t(x) \end{aligned} \tag{C.6}$$

Now we have:

$$\mathrm{KL}(Q^t\|P) - \mathrm{KL}(Q^{t+1}\|P) = \int Q^t(x)\log\frac{Q^t(x)}{P(x)}dx - \int Q^{t+1}(y)\log\frac{Q^{t+1}(y)}{P(y)} \tag{C.7}$$

Replacing $Q^t(x)$ with $\int Q^{t+1}(y)r_m(y,x)dy$ and $Q^{t+1}(y)$ with $\int Q^{t+1}(y)r_m(y,x)dx$ (from (C.5) and (C.6)) yields:

$$\begin{aligned} \mathrm{KL}(Q^t\|P) - \mathrm{KL}(Q^{t+1}\|P) &= \int\int Q^{t+1}(y)r_m(y,x)\log\frac{Q^t(x)}{P(x)}dxdy \\ &\quad - \int\int Q^{t+1}(y)r_m(y,x)\log\frac{Q^{t+1}(y)}{P(y)}dxdy \\ &= \int\int Q^{t+1}(y)r_m(y,x)\log\frac{Q^t(x)/Q^{t+1}(y)}{P(x)/P(y)}dxdy \end{aligned}$$

$$
\begin{aligned}
&= \int\int Q^{t+1}(y) r_m(y,x) \log \frac{Q^t(x) T_m(x,y)/Q^{t+1}(y)}{P(x)T_m(x,y)/P(y)} dx dy \\
&= \int \left[\int r_m(y,x) \log \frac{r_m(y,x)}{q_m(y,x)} dx \right] Q^{t+1}(y) dy \\
&= \int [\mathrm{KL}(r_m \| q_m)]\, Q^{t+1}(y) dy \\
&= E_{Q^{t+1}} [\mathrm{KL}(r_m \| q_m)] \\
&\geq 0 \qquad\qquad (C.8)
\end{aligned}
$$

since $\mathrm{KL}(A\|B) \geq 0$ for any density functions A and B.□

The theorem tells us that $\mathrm{KL}(Q^{t+1}\|P) \leq \mathrm{KL}(Q^t\|P)$, so a single sweep of Gibbs sampling brings the approximation Q closer in terms of K-L divergence to the true posterior P.

Bibliography

[Amari et al., 1996] Amari, S., Cichocki, A., and Yang, H. (1996). A new learning algorithm for blind signal separation. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, *Advances in Neural Information Processing Systems*, volume 8, pages 757–763. The MIT Press, Cambridge.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 41.

[Bell and Sejnowski, 1995] Bell, A. J. and Sejnowski, T. J. (1995). An information-maximisation approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1004–1034.

[Cardoso, 1996] Cardoso, J.-F. (1996). Performance and implementation of invariant source separation algorithms. In *Proc. ISCAS'96.*

[Cooper, 1990] Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42:393–405.

[Day, 1969] Day, N. E. (1969). Estimating the components of a mixture of normal distributions. *Biometrika*, 56:463–474.

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

[Éric Moulines et al., 1997] Éric Moulines, Cardoso, J.-F., and Gassiat, E. (1997). Maximum likelihood for blind separation and deconvolution of noisy signals using mixture models. In *Proc. ICASSP'97*, pages 3617–20.

[Everitt, 1984] Everitt, B. S. (1984). *An Introduction to Latent Variable Models.* Chapman and Hall, London.

[Ghahramani and Hinton, 1996] Ghahramani, Z. and Hinton, G. E. (1996). The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1 [`ftp://ftp.cs.toronto.edu/pub/zoubin/tr-96-1.ps.gz`], Department of Computer Science, University of Toronto.

[Gilks, 1992] Gilks, W. R. (1992). Derivative-free adaptive rejection sampling for gibbs sampling. In Bernardo, J. M., Berger, J. O., Dawid, A. P., and Smith, A. F. M., editors, *Bayesian Statistics 4*. Oxford University Press.

[Goutsias, 1991] Goutsias, J. K. (1991). A theoretical analysis of monte carlo algorithms for the simulation of gibbs random field images. *IEEE Transations on Information Theory*, 37:1618–1628.

[Hinton et al., 1995] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The wake-sleep algorithm for unsupervised neural networks. *Science*, 268:1158-1161.

[Hinton et al., 1997a] Hinton, G. E., Dayan, P., and Revow, M. (1997a). Modeling the manifolds of Images of handwritten digits. *IEEE Trans. Neural Networks*, 8(1):65–74.

[Hinton and Ghahramani, 1997] Hinton, G. E. and Ghahramani, Z. (1997). Generative models for discovering sparse distributed representations. *Phil. Trans. Roy. Soc. London B*, 352:1177–1190.

[Hinton et al., 1997b] Hinton, G. E., Sallans, B., and Ghahramani, Z. (1997b). A hierarchical community of experts. In Jordan, M. I., editor, *Learning and Inference in Graphical Models*. Kluwer Academic Publishers.

[Hull, 1994] Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

[Jaakkola, 1997] Jaakkola, T. S. (1997). *Variational Methods for Inference and Estimation in Graphical Models*. Department of Brain and Cognitive Sciences, MIT, Cambridge, MA. Ph.D. thesis.

[Jacobs et al., 1991] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixture of local experts. *Neural Computation*, 3:79–87.

[Jordan and Jacobs, 1994] Jordan, M. I. and Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Trans. ASME, Series D, Journal of Basis Engineering*, (8):35–45.

[Lewicki and Olshausen, 1998] Lewicki, M. S. and Olshausen, B. A. (1998). Inferring sparse, overcomplete image codes using an efficient coding framework. submitted for publication.

[Mackay, 1996] Mackay, D. (1996). Maximum likelihood and covariant algorithms for independent component analysis. Available at http://wol.ra.phy.cam.ac.uk/mackay/ica.ps.gz.

[Metropolis et al., 1953] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092.

[Mozer et al., 1997] Mozer, M. C., Jordan, M. I., and Petsche, T., editors (1997). *Advances in Neural Information Processing Systems*, volume 9. The MIT Press, Cambridge.

[Neal, 1992] Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

[Neal, 1993] Neal, R. M. (1993). Probabilistic inference using Markov chain monte carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.

[Neal and Hinton, 1997] Neal, R. M. and Hinton, G. E. (1997). A new view of the EM algorithm that justifies incremental and other variants. In Jordan, M. I., editor, *Learning and Inference in Graphical Models*. Kluwer Academic Publishers.

[Olshausen, 1996] Olshausen, B. A. (1996). Learning linear, sparse, factorial codes. A.I. Memo No. 1580, MIT Center for Biological and Computational Learning, Cambridge, MA.

[Olshausen and Field, 1996] Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.

[Pearlmutter and Parra, 1997] Pearlmutter, B. A. and Parra, L. C. (1997). Maximum likelihood blind source separation: A context sensitive generalization of ica. In [Mozer et al., 1997], pages 613–619.

[Rao and Ballard, 1997] Rao, R. P. N. and Ballard, D. H. (1997). Efficient encoding of natural time varying images produces oriented space-time receptive fields. Technical report 97.4 [ftp://ftp.cs.rochester.edu/pub/u/rao/papers/space-time.ps.z], National Resource Laboratory for the Study of Brain and Behaviour, Department of Computer Science, University of Rochester.

[Roweis, 1997] Roweis, S. (1997). Em algorithms for pca and spca. In [Mozer et al., 1997].

[Rubin and Thayer, 1982] Rubin, D. and Thayer, D. (1982). EM algorithms for ML factor analysis. *Psychometrika*, 47(1):69–76.

[Saul et al., 1996] Saul, L. K., Jaakkola, T., and Jordan, M. I. (1996). Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76.

[Socci et al., 1998] Socci, N. D., Lee, D. D., and Seung, H. S. (1998). The rectified gaussian distribution. In Jordan, M., Kearns, M., and Solla, S. A., editors, *Advances in Neural Information Processing Systems 10*. MIT Press, Cambridge, MA.

[Tipping and Bishop, 1997] Tipping, M. E. and Bishop, C. M. (1997). Probabilistic principal component analysis. Technical Report NCRG/97/010, Neural Computing Research Group, Aston University.