Product Models for Sequences

by

Andrew Denis Brown

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

Abstract

Product Models for Sequences

Andrew Denis Brown
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2001

This thesis introduces a series of new graphical models for time series which incorporate the ideas of using many small density models combined to attain distributed representation and independence in the hidden state for fast inference. A product of hidden Markov models (PoHMM) is a graphical model which uses many hidden Markov models (HMM's) to model a sequence. A training method for approximate maximum likelihood learning is presented and results show it to be superior to an equivalent sized single HMM for modelling complex structure and long-range structure in sequences. The relative density network (RDN) is a supervised network, which is like the PoHMM in that it is designed to learn the relevant features of a class of data by using many small HMM's. Unlike, other methods of supervised classification of sequences, it does not insist on using a single model for each data class, but uses many models, and lets the network decide how to allocate them among the classes. A set of experimental comparisons with other HMM based supervised learning schemes on a problem in speaker recognition shows it to have superior classification performance. Finally, the spiking Boltzmann machine is a second type of product model which uses a combination of binary and continuous hidden state representation, with smooth temporal dynamics so that it may model slowly varying signals. An example with synthetic data shows that it can learn to model and predict simple dynamics.

Acknowledgements

I would like to thank my supervisor, Geoff Hinton, for his support, and patience, and his inexhaustable capacity to explain ideas over and over until they are properly understood.

I would also like to thank the members of the Gatsby Computational Neurosciece Unit who have provided a stimulating and exciting environment in which to work and learn. In particular I would like to thank Brian Sallans, Sam Roweis, and Zoubin Ghahramani who have all made helpful comments and contributions. I would also like to thank Christopher Williams, and Radford Neal and John Bridle for helpful discussions about this work.

This work has been supported by the Natural Sciences and Engineering Research Council and the Gatsby Foundation.

Contents

Notation

- Lower case text is used for scalar quantities, x, bold face is used for vectors, $\mathbf{x}$, and capital letter typically represent matrices, X.
- Probability distributions are also denoted using capital letters, but the functional notation usually distinguishes them from matrices, *i.e.* $P(x)$. In other cases it should be clear from the context.
- A summation with a vector argument is assumed to be over all the dimensions of the argument *i.e.* $\sum_{\mathbf{x}} = \sum_{x_1} \sum_{x_2} \cdots \sum_{x_d}$
- A set of variables in a sequence is represented using a colon subscript $x_{1:T} = \{x_1, \ldots, x_t, \ldots, x_T\}$
- The 'hat' symbol is used to distinguish between a probability distribution P defined by a model and an observed, empirical distribution, $\widehat{P}$.

“From now on, I’ll describe the cities to you,” the Khan had said, “in your journeys you will see if they exist.”

But the cities visited by Marco Polo were always different from those thought of by the emperor.

“And yet I have constructed in my mind a model city from which all possible cities can be deduced,” Kublai said. “It contains everything corresponding to the norm. Since the cities that exist diverge in varying degree from the norm, I need only forsee the exceptions to the norm and calculate the most probable combinations.”

“I also have thought of a model city from which I deduce all others,” Marco answered. “It is a city made only of exceptions, exclusions, incongruities, contradictions. If such a city is the most improbable, by reducing the number of abnormal elements, we increase the probability that the city actually exists. So I have only to subtract exceptions from my model, and in whatever direction I proceed, I will arrive at one of the cities which, always an exception, exist. But I cannot force my operation beyond a certain limit: I would achieve cities too probable to be real.”

Italo Calvino — *Invisible Cities*

Chapter 1

Introduction

We often try to understand the world around us by constructing mental models of how it works. We then use these models to help us predict what will happen in new and unforeseen circumstances. In the preceding passage from Italo Calvino's *Invisible Cities*, the imagined conversation between Kublai Khan and Marco Polo illustrates some of the problems that we encounter when attempting to build models of a complex world — their behaviour can often run counter to our intuitions, making the act of model building quite a challenge.

In machine learning we also construct models, but rather than qualitative mental models, they are quantitative models of the observed world. Here the goal of model building is to induce structure from our observations so that we can better understand what we have seen, which can then be used to make predictions and make decisions about what we expect to see in the future. The chosen tool for doing this is probability theory, and the work in this thesis focusses on probabilistic models of sequences and time series. Chapters 2 and 3 will introduce the formal theory that we will be using for the models, but in this chapter we would first like to give a motivation for this work by giving a brief introduction to the central issue of representation in probabilistic models.

1.1 The Problem of Representation in Probabilistic Models

The models used in this thesis all make use of the idea of hidden state to represent dependencies among the observations. Hidden state is useful because it makes the models more modular, and it allows more compact representation of the observations. For example, in modelling time series, we assume that the past has some influence on our current and future observations, and we must decide how to represent the information contained in our sequence history. There are two standard ways of doing this. One way is to use the sequence history as its own representation. In this case we directly model the dependency of the current observation on the past observations. Such 'auto-regressive' models have a couple of drawbacks. They are limited in memory to the amount of history which we choose to build into our model. Also, this representation offers no simplification if the underlying dynamics generating the series are quite simple. Another way of representing sequences is to postulate a hidden state and then infer what values of the hidden state could have influenced our observations. Called 'state space' models, they can rectify both of these disadvantages of auto-regressive models, because the inferred state is not limited to simply storing the past observations. The hidden state can store information about the history of the sequence which occurred arbitrarily far back in the past, and it need only store the information about the history which is relevant to predicting future observations, so if the sequence admits a simpler representation than the raw history of observations, then it is possible for a state space model to find it. The question then is what form this hidden state representation should take.

1.1.1 Componential vs. Mixture Models

Some of the probabilistic models which are in common use, such as the naive Bayes classifier, the 'aspect' model in information retrieval (Hofmann 2001), and the hidden Markov models in speech recognition(Rabiner and Juang 1993) are based on mixture models. This is the idea of clustering the data, and then fitting a separate model, or *expert*, to each data cluster. The hidden state representation amounts to having a single variable with a discrete number of states that switches among the categories. However, in many instances observations cannot be so easily assigned to a single category. Some examples might express characteristics of several different categories, and we would like to be able to share information about these characteristics among the experts. It would be much more efficient if we could describe an item using a whole set of features rather than simply labelling it as belonging to one category. The corresponding hidden state representation for such models requires many variables which simultaneously indicate something about the nature of the observations. This is what will be referred to as *componential* structure.

For example, in the task of text classification, there may be a set of documents pertaining to business news, and another related to technology, but there may equally well be articles in a data set which contain both of these topics simultaneously. Should we then create a new category for technology-business news? What about an article describing a technology company which is sponsoring a sports team? We can see that if we want to accurately model such a domain using a simple categorical model, then we face an explosion of possible classes. A much more desirable model would describe each article in our database using a whole set of features. Such an architecture increases the representational power of our model, but it also avoids the exponential explosion in the number of hidden states and the number of parameters which would occur in a categorical model.

Similarly, modern speech recognizers work using hidden Markov models: they start by dividing the speech up into short segments or frames, grouping the frames according to their spectral content and then characterizing the way that the speech tends to move from one spectral cluster to another. This is again based on the idea that a segment of speech can adequately be described as belonging to a single cluster of speech characteristics. However, there are problems with this approach. Speech is generated by a set of loosely couple articulators which move independently to change the speech waveform. The fact that these articulators such as our lips, tongue, and jaw are slow moving mechanisms, constrained by physical laws, means that the pronunciation of a unit of speech (a phoneme) is influenced by the context in which they are spoken. One solution to this problem is to model phonemes in all their likely contexts. Again, a more satisfying approach, which is the focus of some current research interest in articulatory and multiband speech recognition, is to construct a model capable of capturing the underlying componential structure, rather than trying to isolate all the possible variations.

1.1.2 Causal vs. Descriptive Models

This notion of using models with componential structure is not new, and there are different ways of incorporating it into models. Some examples of probabilistic models with multiple explanatory interacting hidden variables include sigmoid belief networks(Neal 1990), the Helmholtz machine(Dayan, Hinton, Neal, and Zemel 1995) and factorial hidden Markov models(Ghahramani and Jordan 1997). These models all have the property that they use multiple interacting factors to represent and explain observations. Because of this they also have the problem that when we try to perform *inference* in the model — computing a distribution over the possible hidden state representations — it is computationally expensive. This is due to the problem of explaining away(Pearl 1988). In such models each of the hidden state variables offers competing explanations for how the data was generated. When we observe the data these explanatory variables interact and we need to consider all their combinations in order to evaluate the posterior probability of any particular configuration of the explanatory variables.

A different way of introducing componential structure to the model is to have experts whose hidden state describes a feature of the data independently of what the other experts' hidden state variables are doing. By forcing this limitation on the model's hidden state representation, we can not expect to uncover the hidden causes of the data in the way that causal models, such as the ones mentioned above, might. The trade off comes in the process of doing inference, because this procedure can be done very efficiently in each of the experts independently. The total model must constitute a good description of the data, and in the process of learning the model parameters this will force the experts to learn different aspects of the data. Chapter 3 will introduce a class of probabilistic models, and a learning algorithm which have these properties.

To summarize, the main goal of this thesis will be to describe work on a set of time-series models which use componential hidden state representations combined with an architecture that allows inference in the hidden states to be done independently. This makes inference in these models very efficient. The learning algorithm must still take into consideration the dependencies between the experts, but we will look at an algorithm which works around this problem.

1.2 Thesis Overview

The remaining sections of this thesis will proceed as follows. Chapters 2 and 3 review background material necessary for understanding the models and learning algorithms described later. Chapter 2 introduces the concept of a graphical model. It defines the probabilistic semantics of both directed and undirected models. It discusses the problem of inference in graphical models with hidden variables and several methods exact, and approximate for doing this. It also talks about methods of learning the parameters of graphical models, mostly in a maximum likelihood setting. Finally the chapter discusses various graphical models for time series, from the basic to more complex models. In particular, section 2.4.1 describes the hidden Markov model (HMM), and gives the details of the forward-backward algorithm for performing inference and the Baum-Welch algorithm for performing maximum likelihood parameter fitting. In addition, a method for reparameterizing the HMM and then computing derivatives for unconstrained gradient optimization is presented. More detailed derivations of this are included in appendix A. Chapter 3 introduces the Products of Experts (PoE), which is a particular kind of mixed directed and undirected graphical model. The chapter discusses what advantages this architecture has and also what it has in common with other graphical models, particularly undirected Markov networks. The chapter also introduces the contrastive divergence algorithm which can be used for fitting general Markov networks, but which is particularly efficient in PoE's. In chapter 4 we look at a the product of hidden Markov models (PoHMM), a product of experts model for time series based on the hidden Markov model. The chapter defines the model and shows how the contrastive divergence algorithm can be applied to it. The chapter also describes a series of examples and experiments using PoHMM's to model discrete valued sequences. Chapter 5 describes a set of models which are inspired by the PoHMM, but are used in an supervised framework. The main architecture is the relative density network (RDN), which like an PoHMM uses many small PoHMM's to model the data. Unlike other supervised models for sequence classification it does not use a single model to fit each data class, but allows the network to decide how to allocate them. The chapter describes the philosophy of the RDN architecture and a set of variations on it. The chapter concludes with a set of experiments in speaker classification. Chapter 6 looks at another product model, the 'spiking' Boltzmann machine (SBM). The SBM is a PoE model for time series which uses a mixed discrete/continuous hidden state representation. The addition of a temporal smoothing function to a basic PoE architecture allows it to model the smooth dynamics of image sequences. An experiment with synthetic images shows this. Finally, chapter 7 summarizes the results and contributions of this thesis as well as directions possible future work.

Chapter 2

Learning and Inference in Graphical Models

2.1 Graphical Models

Graphical models are a convenient way to visualize and to organize computations over probability distributions consisting of many variables. In this section we will give a brief overview of the two main types of graphical models: Bayesian networks, and Markov fields.

2.1.1 Bayesian Networks

Bayesian networks can help solve the problem of representing and computing with joint distributions over potentially large numbers of variables by exploiting the independence structure in the distribution. A Bayesian network represents the variables in probability distribution as a directed, acyclic graph. Each vertex in the graph is a variable, or a group of variables in the probability distribution and the directed edges indicate the conditioning relationships of the variables. The presence of a link between two nodes indicates that the receiving node is conditionally dependent on the value of the source node, and the goal of displaying these dependence relations in a graphical way is intended to make them readily apparent by looking at the connections in the graph.

To give a simple example: by the chain rule of probability a distribution over three variables $\{x_1, x_2, x_3, x_4\}$ can be decomposed as a product of conditional densities using the chain rule:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)P(x_4|x_3, x_2, x_1) \tag{2.1}$$

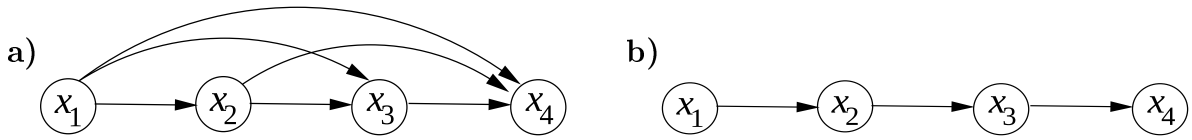

Figure 2.1: Two possible networks for the joint probability $P(x_1, x_2, x_3, x_4)$. a) $P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)P(x_4|x_3, x_2, x_1)$ and b) $P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3)$.

Now this representation is no simpler than the original full joint distribution, however, if we know that there are conditional independencies among the variables then we can drop some of the conditioning variables. For instance, if x_3 is conditionally independent of x_1 given x_2 (denoted $x_3 \perp\!\!\!\perp x_1 | x_2$, and $x_4 \perp\!\!\!\perp x_2, x_1 | x_3$) then the expression of the joint reduces to:

$$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3) \tag{2.2}$$

More importantly, this simplification makes it easier to perform computations. Consider that we would like to find the conditional distribution of $P(x_4|x_1)$. Figure 2.1 shows the Bayesian network diagram for two possible joint distributions over four variables. Each variable is represented by a circle, and conditional dependencies are shown by directed edges. The first shows all the possible dependencies, and the second has the conditional independencies described above. In figure 2.1b we see that there is no edge in the graph between x_1 and x_3, denoting the independence of x_1 and x_3 conditioned on x_2. To condition the graph on a particular value of x_1 then, it seems intuitive that information about x_1 must propagate first through x_2 and then to x_3, since x_2 is the variable which induces the dependence between them, and this is depicted by the graphical structure.

Now imagine that we would like to compute something simple like the marginal distribution of x_4.

$$P(x_4) = \sum_{x_1, x_2, x_3} P(x_1, x_2, x_3, x_4) \tag{2.3}$$

If the variables are discrete valued each with K possible values then this summation would appear to cost K^3 operations. However, if we know that our distribution has the form shown in figure 2.1b then we can simplify the computation,

$$\begin{aligned} P(x_4) &= \sum_{x_1, x_2, x_3} P(x_1)P(x_2|x_1)P(x_3|x_2)P(x_4|x_3) \\ &= \sum_{x_3} P(x_4|x_3) \sum_{x_2} P(x_3|x_2) \sum_{x_1} P(x_2|x_1)P(x_1) \end{aligned} \tag{2.4}$$

Using the distributivity of multiplication over addition we have "pushed through" the

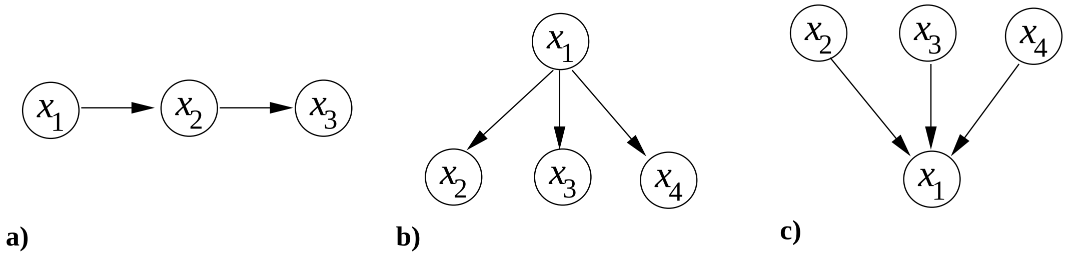

Figure 2.2: Conditional dependency in Bayesian networks

additions to cover only factors which include the relevant terms. We have now reduced a large summation into a series of three intermediate summations each costing K^2 operations:

$$\begin{aligned} P(x_2) &= \sum_{x_1} P(x_2|x_1)P(x_1) \\ P(x_3) &= \sum_{x_2} P(x_3|x_2)P(x_2) \\ P(x_4) &= \sum_{x_3} P(x_4|x_3)P(x_3) \end{aligned} \tag{2.5}$$

If $K >> 3$ then this will save some computation, but if we had a much longer chain, containing N variables then in general we would be able to reduce a K^N operation to NK^2. This principle is the basis of the dynamic programming algorithms described in section 2.2.1 for performing marginalizations of this type in general Bayesian networks.

The general formulation of the joint distribution of a Bayesian network is a product of local conditional probability distributions,

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i|pa(x_i)), \tag{2.6}$$

where $\mathrm{pa}(x_i)$ is the set of parents of the variable x_i. The variables in the network are ordered so that variables are only conditioned on variables which preceded them in the ordering. This *ancestral* ordering ensures that the probability distribution given by (2.6) is consistent.

The graph associated with the Bayes network has directed edges from parent nodes to child nodes, and the graphical representation makes it possible to read off the independence relationships among the variables. Figure 2.2 illustrates the three basic cases. The first example, figure 2.2, shows a pass through connection. In such a connection, x_3 and x_1 can influence each other unless the value of x_2 is known, in which case x_3 is independent of x_1.

In the second example, Figure 2.2b), the diverging connection through x_1 causes x_2, x_3 and x_4 to be dependent, unless we have knowledge about the state of x_1 in which case the other three variables become conditionally independent.

In the last case, x_2, x_3 and x_4 are connected by converging links to x_1 and are independent unless the state of x_1 is known. This is also true if the state of one of x_1's descendants is known. This final effect is known as *explaining away* and can be understood intuitively by thinking of an event which has two possible explanations. Knowing that the event can explained reduces the need for — and our belief in — a second explanation. Thus observing the state of a variable induces correlations among the parents of the variable which do not otherwise exist.

The rule by which we decide that two variables are rendered conditionally independent given the state of a third variable $\mathbf{v}$ is called d-separation, and it can be stated formally as follows (Jensen 1996):

d-separation Two variables in a Bayesian network are d-separated if there is an intervening variable $\mathbf{v}$ such that:

- the connection is serial or diverging and the state of $\mathbf{v}$ is known, or
- the connection is converging and neither the state of $\mathbf{v}$ nor any of its descendants is known

In addition, there should be no other paths in the network between the two variables.

2.1.2 Examples of Bayesian Networks

The Bayes network formalism can represent several common probabilistic models. Examples include: mixtures of Gaussians, factor analysis, and hidden Markov models. The Bayesian network diagrams for mixtures of Gaussians and factor analysis are shown in figure 2.3. In this diagram we have used the convention that squares represent discrete variables and circles represent continuous one. Figure 2.3 a) is a factor analysis model where the latent variables, $\mathbf{y}$, are drawn from a zero-mean, unit covariance Gaussian and the visible variables, $\mathbf{x}$, are linearly related to the latent variables by the "factor loading" matrix, W.

$$\mathbf{x} = W\mathbf{y} + \nu \tag{2.7}$$

ν is drawn from Gaussian with zero-mean and an axis-aligned covariance, R. The second diagram is a mixture of Gaussians, where s is multinomial indicator which selects one

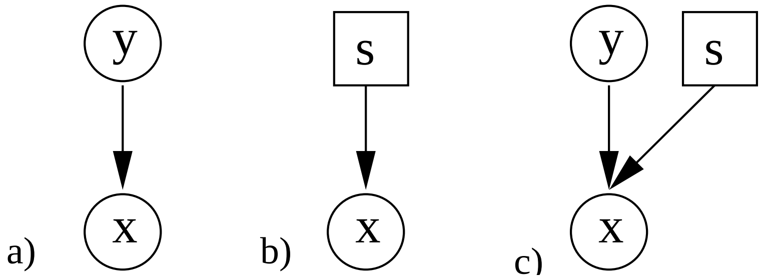

Figure 2.3: Bayes nets for some common probabilistic models (square nodes represent discrete variables, and circles represent continuous variables): a) factor analysis b) mixture of Gaussians and c) mixture of factor analyzers

of the Gaussian components of the mixture. The final model shown in figure 2.3 c) is a combination of the two previous models: a mixture of factor analyzers (Ghahramani 1996). We can generate data from this model by first picking a component from s and generating a Gaussian distributed value for $\mathbf{y}$. Given $\mathbf{y}$ and the the factor loading matrix corresponding to mixture component s we can draw $\mathbf{x}$ from the appropriate Gaussian. Note that s and $\mathbf{y}$ are marginally independent, but when $\mathbf{x}$ is observed then they are dependent: the choice of s restricts the choice of values for $\mathbf{y}$ which could have generated $\mathbf{x}$ and vice-versa.

2.1.3 Markov Networks

A second type of graphical model with different semantics than the Bayes net is called a *Markov* network (Pearl 1988) or a Markov random field (MRF). The graph of a Markov network differs from a Bayes net in that the links between nodes are undirected. This is meant to signify a change in the meaning of the edges in the graph. Whereas the links in a directed network indicate that there is a local conditional density relating the parent node to the child, in a Markov network the link indicates only that there is a local *potential* function which joins the two nodes. Unlike with a Bayesian network the local potential functions are not normalized and their product does not necessarily form a proper joint probability distribution over the variables in the network. For this reason the joint distribution of the variables is a product of their potentials divided by an additional normalizing (called the *partition function* by analogy from physics). The formula for the joint of a Markov network is:

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C) \tag{2.8}$$

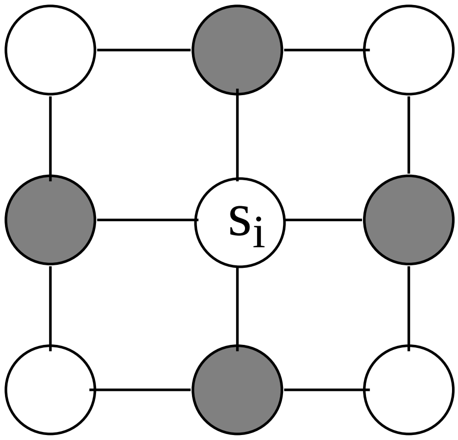

Figure 2.4: Example of a Markov network. The coloured nodes comprise the Markov blanket of s_i

where $\mathcal{C}$ is the set of maximal cliques[1] in the graph, and the partition function is given by:

$$Z = \sum_{\mathbf{x}} \prod_{C \in \mathcal{C}} \psi_C(x_C) \tag{2.9}$$

Figure 2.4, shows an example of a locally connected Markov network. The independence properties of a Markov network are more straightforward than those of a directed Bayesian network. In a Markov network, two variables are dependent if there is a path between them and all nodes along that path are unobserved. This leads to the concept of a Markov blanket: given all of its neighbours, a variable in a Markov network is conditionally independent of all other variables in the network. This property will become more important when we introduce Products of Experts in section 3.1.

2.1.4 Examples of Markov Networks

The Boltzmann machine is one example of a Markov network used in machine learning. In a Boltzmann machine all of the variables are binary and stochastic. The variables are partitioned into two sets: one which corresponds to the observed variables in the problem and a set of latent, or hidden variables which are introduced to capture high order correlations among the visible variables. Figure 2.5 shows a fully connected Boltzmann machine, the most general type of architecture. Following the properties of a Markov graph any unit in the network is independent of the rest of the network given all of its neighbouring nodes. In this particular instance, every node is a neighbour of every other

[1] A clique being a set of variables which are fully connected and a maximal clique is a clique which is not a subset of any other clique

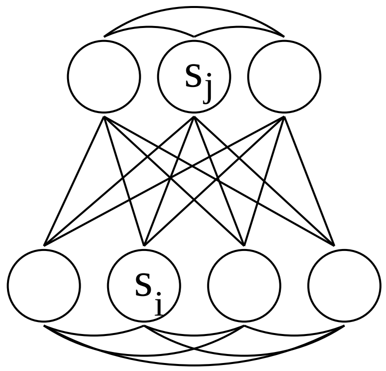

Figure 2.5: Markov network of a Boltzmann machine

node so the Markov blanket for a node is the rest of the network, but later on we will look at restricted forms of this architecture in which the Markov properties come into play to simplify the inference problem. All the units in a Boltzmann machine are binary variables with weighted connections between them. The energy of a Boltzmann machine is defined to be:

$$E(\mathbf{s}) = -\sum_{ij} s_i s_j w_{ij} \tag{2.10}$$

Self connections are not allowed so we assume that all $w_{ii} = 0$. Given the energy function the probability distribution over configurations of the network is the Boltzmann distribution:

$$P(\mathbf{s}) = \frac{1}{Z} e^{-E(\mathbf{s})} \tag{2.11}$$

$$Z = \sum_{\{s_1, \ldots, s_N\}} e^{-E(\mathbf{s})} \tag{2.12}$$

The normalizing constant Z is a sum over all configurations of the network, and, in general, it is intractable to compute exactly. For this reason inference and learning in a Boltzmann machine can be difficult. Later, we will look at a couple of ways of dealing with this problem.

2.2 Inference

When we develop a probabilistic model we would like to be able use it to answer questions such as: given a set of observations on some of the variables in the network, what influence does this have on other nodes in the network? What are the mean values? What are the modes? Given that the structure and parameters of the network have been set and we assume that they capture all the interesting structure in the problem, then inference is the task of computing the effects of observations on unobserved nodes in the network[2]. In a restricted set of graphical models it is possible to perform inference tractably and exactly. But, in general sampling methods or approximations are required which involve limitations on the exactness and completeness of our knowledge of the posterior distribution.

In this section we will give a brief overview of exact methods of computing posterior probabilities in graphical models, as well as Monte Carlo and approximate methods of performing inference.

2.2.1 Exact Methods

Here we look at two of the main algorithms for performing exact inference in graphical models, the "belief propagation" algorithm(Pearl 1988; Lauritzen and Spiegelhalter 1988) and the JLO or Junction Tree algorithm(Jensen, Lauritzen, and Olesen 1990). We will see later in section 2.4.2 that the inference algorithm associated with hidden Markov models, the forward-backward algorithm, is just a special case of belief propagation. Since the forward-backward algorithm is integral to the algorithms introduced in this thesis, we will look at it in detail.

Belief Propagation

Belief propagation is based on two basic ideas: Bayes' theorem for combining new evidence with previous beliefs, and the dynamic programming idea, which we introduced in the last section, for recursively updating beliefs in an efficient way. Information is propagated in the network by passing messages based on locally available information. It is only guaranteed to converge to the correct posterior distributions in networks which are singly-connected, *i.e.* graphs in which there is only a single undirected path between any two nodes. Looking at an example network pictured in figure 2.6, we get an idea of how the message passing scheme works. Each node in a singly-connected network cuts it into two disjoint segments. For example node x_1 in that network separates the set, $\mathcal{U}$, containing all of its ancestors $\mathcal{U} = \{a_1, a_2, a_3, a_4\}$ and the set, containing its descendants and co-parents $\mathcal{D} = \{d_1, d_2, d_3, x_2\}$. Now if some of the variables in $\mathcal{U}$ and $\mathcal{D}$ have been observed we would like to update the distribution over all the unobserved variables in the

[2]Although from the perspective of Bayesian learning, the structure and parameter settings of a network may also be thought of as random variables, throughout this thesis we will be considering only the Maximum Likelihood approach to learning the parameters, and we will assume that only the 'hidden variables' of the network need be inferred.

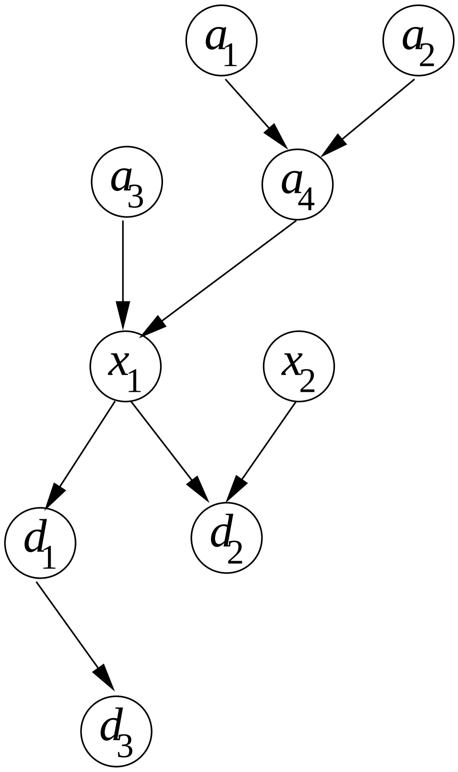

Figure 2.6: An example of message passing in the Belief Propagation algorithm

network. Each node in the network divides the network into its own $\mathcal{U}$ and $\mathcal{D}$ set. If we label the evidence on ancestral nodes $E_{\mathcal{U}}$ and evidence on the descendant and co-parent nodes as $E_{\mathcal{D}}$ then we would like the message arriving at x_i from the parents of x_i to contain the probability $P(x_i, E_{\mathcal{U}})$ and the messages arriving from the children to contain $P(E_{\mathcal{D}}|x_i)$. The top-down message serves as a prior term and the bottom-up message acts like a likelihood term which may then be combined using Bayes' rule to get the posterior probability of the node conditioned on all the evidence in the network ,E:

$$P(x_i|E_{\mathcal{U}\cup\mathcal{D}}) = \frac{P(x_i, E_{\mathcal{U}})P(E_{\mathcal{D}}|x_i)}{\sum_{x_i} P(x_i, E_{\mathcal{U}})P(E_{\mathcal{D}}|x_i)} \tag{2.13}$$

Belief propagation accomplishes this by sending out one message from each node to each of its parents (a λ-message) and one message from each node to each of its children (a π-message). A nodes outgoing *lambda* or *pi* message along a particular link is computed using the messages coming in on all other links, and in this way the algorithm recursively computes the $P(x_i, E_{\mathcal{U}})$ and $P(E_{\mathcal{D}}|x_i)$ for each node in the network.[3] Some care must be taken to arrange the scheduling of messages so that only one of each kind of message

[3]For a more detailed description of the formulas for computing these messages see (Frey 1997)

travels along the link between two nodes, but there is a simple way of doing this. Any singly-connected graph can be thought of as a tree by selecting one node in the graph to be a root. Then messages may be correctly scheduled by passing messages from the leaf node towards the root and then from the root towards the leaves. This ensures that exactly two messages travel over each edge of the graph.

The Junction Tree Algorithm

The belief propagation algorithm is only guaranteed to work correctly if the graph it operates on contains no undirected loops. If it contains cycles of short length then these can be dealt with by clustering variables into larger nodes to eliminate the cycles. The junction tree algorithm (Jensen, Lauritzen, and Olesen 1990; Cowell, Dawid, Lauritzen, and Spiegelhalter 1999) provides a set of transformations which will turn the directed graph into an undirected graph with the same dependencies between variables, and then eliminate loops by the process of *triangulation*. Triangulation breaks loops in the graph by introducing new edges. There is no optimal algorithm for doing this, and the resulting triangulated graph may have many more dependencies than the original graph. Once the triangulation step terminates the junction tree is constructed by grouping together all maximal cliques into single nodes, and this will yield a tree structure.

Once the junction tree has been formed then the second part of the algorithm is the propagation step. The junction tree has its own multiplicative message-passing rule for propagating evidence around the network, although it has similar scheduling properties as the belief propagation algorithm, with a two pass update.

Other Issues

While the junction tree algorithm can perform inference in networks of general structure, in practice if the network is densely connected, then this may yield cliques with a large number variables, and inference is exponential in the size of the largest clique. This is not a problem which is limited to this particular algorithm. Exact inference in general belief networks, and even approximate inference, has been shown to be NP-hard (Cooper 1990; Dagum and Luby 1993).

In describing the belief propagation and the junction tree algorithms I have made no mention of the form of the distribution which describes the local conditional probabilities. If all the variables in the network are discrete with low cardinality then the local conditional probabilities may be implemented as tables and the propagation rules may be implemented in a straightforward fashion. Similarly, if the network consists of continuous

Gaussian distributions with linear relationships between them then all the operations in the update rules result in Gaussians. The property that these two representations have is that they are closed under the operations of the update rules. However, if the network consists of a combination of discrete and continuous variables then the situation is more complex. Inference is exact in cases where continuous nodes have discrete parents, *e.g.* a switch variable in a mixture model, although the operation of summing out the discrete hidden variable does not yield either a Gaussian or a discrete density, but a weighted sum of Gaussians. So there is a limit to how many times this operation may be recursively performed before the representation of the distribution becomes unmanageable.

In the case where a discrete variable has a continuous valued parent, then the distribution over the discrete variable is some continuously parameterized function, such as a softmax. In this case inference cannot be done exactly, because the posterior is a product of a a non-linear likelihood function and a Gaussian, which has no closed form representation.

Finally, while belief propagation is only guaranteed to compute the correct posterior on singly-connected graphs, we make note of recent work on performing inference in networks which are not singly-connected. The common wisdom is that cycles, especially short ones, cause evidence to be "double counted" and the inference algorithm to be overconfident in the computed posterior probabilities. However, it has been empirically demonstrated that belief propagation can work quite well as an approximate inference method on graphs which contain loops, often converging to the correct answer or nearly the correct answer(Frey 1997; Frey and MacKay 1998; Murphy, Weiss, and Jordan 1999).

2.2.2 Sampling

For Bayesian networks which are not amenable to inference using exact methods, Monte Carlo methods are a commonly used method of performing inference. It allows us to forgo computing the entire distribution of a hidden variable in favour of using a set of samples from that distribution. If we have a Bayesian network with visible units, $\mathbf{y}$ and hidden variable $\mathbf{x}$ then the set of points $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ drawn from the distribution $P(\mathbf{x}|\mathbf{y})$ is an approximation to the posterior, and may be used to compute estimates of expectations with respect to the posterior distribution using the formula:

$$E_{P(\mathbf{x})}[g(\mathbf{x})] = \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{x}^{(i)}) \tag{2.14}$$

One of the practical benefits of computing expectations this way, as opposed to some numerical solution to the difficult integrals which emerge, is that sampling methods are not as susceptible to a combinatorial explosion, the way that numerical methods are when the dimensionality of the problem increases (MacKay 1998b). However, drawing samples from the space of $\mathbf{x}$ may be difficult for a couple of reasons. First, the dimensionality of $\mathbf{x}$ may be quite high, and there are not many high-dimensional distributions which may be easily sampled. Also, we may not even be able to directly compute the probability of a point in the distribution easily. Many distributions are defined in terms of some positive function $f(\mathbf{x})$ which may then be converted to a proper probability by normalizing. The normalization constant involves integrating over all possible values of $\mathbf{x}$, and this may be intractable to compute for a high dimensional distribution.

The Metropolis-Hastings algorithm is one method for sampling from high dimensional spaces(Neal 1993). It works by setting up an Markov chain, whose stationary distribution is the one which we would like to sample from. The algorithm works by taking a sample from any initial distribution $P(\mathbf{x}_0)$ and successively sampling from a proposal distribution $T(\mathbf{x}, \mathbf{x}')$ until the chain has reached the stationary, or *equilibrium* distribution. Given the proposal distribution the next sample of the Markov chain is drawn conditioned on the current one. The new sample is then stochastically accepted or rejected. First the quantity a is computed:

$$a = \frac{P(\mathbf{x}^t)T(\mathbf{x}^t, \mathbf{x}')}{P(\mathbf{x}')T(\mathbf{x}', \mathbf{x}^t)} \tag{2.15}$$

If the ratio a is greater than 1, then the proposed sample is accepted, otherwise it is accepted with probability a.

Gibbs sampling is a particular Metropolis-Hastings method which uses a particular proposal distribution in which a variable from the system is drawn conditioned on all the other variables. The variable then updated to be the new value and another variable is selected, and the process continues until all variables have been updated:

$$\begin{aligned} x_1^{t+1} &\sim P(x_1|x_2^t, \ldots, x_n^t), \\ x_2^{t+1} &\sim P(x_2|x_1^{t+1}, x_3^t, \ldots, x_n^{t+1}), \\ x_n^{t+1} &\sim P(x_n|x_1^{t+1}, x_2^{t+1}, \ldots, x_{n-1}^{t+1}) \end{aligned} \tag{2.16}$$

One complete pass through the sequence of proposal distributions, generates a new proposal state for the system which can be shown to have an acceptance probability of 1, thus eliminating the possibility of rejecting the proposed sample under the criteria stated in equation (2.15). Gibbs sampling is useful in very large systems where the distribution of one variable conditioned on all the others is easily computed. Although, we have shown variables being updated in a fixed sequence, asynchronous schemes in which the variables are updated randomly will also work. Gibbs sampling is attractive for two reasons. First, in high dimensional spaces it may be difficult to construct a proposal distribution such that the Metropolis-Hastings will have a reasonable acceptance probability, and the Gibbs sampling proposal distribution assures acceptance. Second, it substitutes sampling from a high dimensional distribution with a sequences of samples from low dimensional distributions which may be computationally more feasible.

In some models, it might be possible to construct the conditional distribution for a block of variables at once,

$$x_1^{t+1}, \dots, x_a^{t+1} \sim P(x_1, \dots, x_a | x_{a+1}^t, \dots, x_n^t) \tag{2.17}$$

thus eliminating sampling steps, by allowing many variables to be sampled at once. This technique of Gibbs sampling the variables in blocks will be important in the discussion of Products of Experts in section 3.1.

Gibbs sampling is a useful method for doing inference in graphical models based on many hidden variables, where it is impossible to compute or sample directly from the posterior distribution. Examples where it has been used include the Boltzmann machine (Hinton and Sejnowski 1986), sigmoid belief networks (Neal 1990), and rectified Gaussian belief networks (Hinton and Gharamani 1997).

Gibbs Sampling in a Boltzmann machine

Here we will look at a particular example of the use of Gibbs sampling in the Boltzmann machine, as later development of the Product of Experts model will depend on it.

The probability that a single unit of the Boltzmann machine is on, conditioned on all the other variables is simple to compute:

$$P(s_i = 1 | s_{j \neq i}) = \frac{e^{\sum_{j \neq i} w_{ij} s_j}}{1 + e^{\sum_{j \neq i} w_{ij} s_j}} = \frac{1}{1 + e^{-\sum_{j \neq i} w_{ij} s_j}} = \sigma(\sum_{j \neq i} w_{ij} s_j) \tag{2.18}$$

where $\sigma(\cdot)$ is the logistic function. This formula says that the probability of a hidden unit being on is just the logistic of the total input coming from the other units in the network. We can get a sample from the posterior distribution over the hidden units by sampling from each unit in sequence and repeating the process until the equilibrium distribution has been reached. One of the problems with this process is that it can take a long time to reach equilibrium and there is no easy way to measure when the Markov chain has converged.

2.2.3 Approximation Methods

While not directly relevant to the following discussions, it is worth mentioning that there has been a great deal of interest in using approximation techniques to perform inference in graphical models. These methods are typically applied in cases where the model has loops and exact methods do not apply. Unfortunately, this is true for the most interesting graphical models and this is where approximate methods become useful.

One class of approximate algorithms, variational methods, are used to obtain a lower bound on a probability distribution by hypothesizing an approximating distribution which has some convenient property such as being factorized in a way which makes taking expectations easier. The simplest of these, the *mean-field* approximation comes from work on Ising models in statistical physics, and was first applied in the field of neural networks to the Boltzmann machine (Peterson and Anderson 1987; Hinton 1989). It assumes that the posterior distribution of the hidden variables conditioned on the observed variables is fully factorized — that is, all the units are independent — an assumption which is clearly untrue for the general Boltzmann machine with fully interconnected hidden units. In the mean field approximation each node in the network is a Bernoulli random variable with its own mean, and the idea then is to optimize the means as free parameters to achieve the best possible fit to the posterior given the strong constraint imposed on the distribution by the assumption of independence. This approximation leads to an iterative method of settling the means of the approximating distribution.

This variational method of approximate inference has been extended by using more sophisticated approximating distributions which allow coupling among substructures of the graph (Saul and Jordan 1996). This effect has been exploited in graphical models of time series which have several Markov chains in the hidden state that are coupled via interactions in the observed variables. For a small number of hidden state chains exact propagation methods might be feasible, but if we want to scale our model to have many hidden state chains then exact inference grows exponentially. Examples of this kind of time series model include factorial hidden Markov models (Ghahramani and Jordan 1997) and switching state-space models (Ghahramani and Hinton 1998). We will revisit these in the section on dynamic graphical models (section 2.4).

Other approximation methods do not compute bounds, but use other principles to approximate the posterior probabilities. For example, if we have a large densely connected network, then, by the central limit theorem, as the the number of edges connected to a node grows, their effect is roughly Gaussian. Thus, the incoming connections to a node may be approximated as a Gaussian with the mean and covariance as parameters of the approximation which are estimated (Saul and Jordan 1999a; Barber and Sollich 2000).

A more recent development is to devise an approximate energy function whose minimum is exact for a class of tractable graphical models, and then apply it to others which are not in that class. In this vein, the belief propagation algorithm has been shown to have fixed points which are minima of the Bethe free energy (Yedidia, Freeman, and Weiss 2001). The Bethe free energy is exact for trees, but it is only an approximation to the free energy of networks with loops. However, it can be corrected with higher order terms to encompass graphs with local loops.

2.3 Learning

The method of maximum likelihood (ML) estimation gives us a way of fitting a model to data by finding the single set of parameters, $\hat{\theta}$ which were most likely to have generated a dataset $\mathcal{D}$. These parameters may be found by maximizing the log-likelihood function:

$$\mathcal{L}(\theta) = \log P(\mathcal{D}|\theta) \tag{2.19}$$

It is sometimes useful to think of the maximum likelihood objective as minimizing the Kullback-Liebler divergence between the log-probability of the model, P, and the observed empirical distribution, $\widehat{P}$:

$$\mathrm{KL}(\widehat{P}(d)||P(d|\theta)) = \langle \log(\widehat{P}(d) \rangle_{\widehat{P}(d)} - \langle \log(P(d|\theta)\rangle_{\widehat{P}(d)} \tag{2.20}$$

the first term is constant with respect to the parameters, and so we need only maximize the the second term. Since the observed data set, $\mathcal{D}$ is a sample from $\widehat{P}$ then equation 2.19 may be thought of as an approximation to 2.20, but in practice both methods are implemented the same way.

A complete Bayesian treatment of the parameter estimation problem would require specifying a prior distribution over the model parameters, and then combining the likelihood and the prior using Bayes rule to obtain a posterior distribution over the parameters:

$$P(\theta|\mathcal{D}) = kP(\mathcal{D}|\theta)P(\theta) \tag{2.21}$$

Predictions could then be made by integrating out our uncertainty about the parameters, or obtaining the maximum a posterior (MAP) parameter estimate. However, the learning algorithms in the following chapters will be exclusively maximum likelihood. The next few sections will discuss how maximum likelihood learning is implemented in various types of graphical models.

2.3.1 Maximum Likelihood Estimation in Graphical Models

The simplest case of maximum likelihood learning occurs when all the variables in the model are observed. As we saw earlier the probability distribution corresponding to a graphical model factorizes into a product of local conditional probabilities, in the case of a Bayesian network, and unnormalized local potentials in the case of a Markov graph. Taking the logarithm of these yields a sum over the logs of the local terms. In the case of Bayesian networks, this sum of local terms causes the parameters to be decoupled into independent optimization problems:

$$\log P(\mathbf{x}|\theta) = \log \prod_{i=1}^{n} P(x_i|pa(x_i), \theta) = \sum_{i=1}^{n} \log P(x_i|pa(x_i), \theta). \tag{2.22}$$

Taking expectations of this function with respect to the empirical data distribution, will cause the sufficient statistics of the data to be substituted into the objective function. For example, if the log-probabilities have a quadratic form, *i.e.* the distributions are Gaussian, then this expression may be maximized directly by setting the first derivatives to zero and solving for the model parameters. The parameters of the model will then be expressed in terms of sufficient statistics of the data. If the form of the distribution is more complex, then the function might be minimized using gradient descent or more powerful second-order techniques. If the distributions are discrete and are represented as conditional probability tables, then they can be solved for directly.

For Markov networks, however, the situation is more complicated, because the normalizing constant also appears in the maximum likelihood objective function. We will consider the problems that this presents in section 2.3.3.

2.3.2 Networks with Hidden Variables: The EM algorithm

Here we will look at the situation of a Bayesian network with hidden variables. The objective of maximizing the likelihood now involves the step of integrating out the hidden variables from the joint distribution of the visible and hidden variables (also called the *complete data* likelihood):

$$\log P(\mathbf{v}|\theta) = \log \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}|\theta) \tag{2.23}$$

where $\mathbf{v}$ and $\mathbf{h}$ are the visible and hidden variables of the model respectively. This optimization is problematic, because the summation is inside the log and the problem will not decouple in the convenient way that it does for a completely visible Bayesian network. However, a strategy of handling this problem is suggested by simply taking the derivatives of the objective with respect to the parameters:

$$\begin{aligned}
\frac{\partial}{\partial\theta} \log P(\mathbf{v}|\theta) &= \frac{\partial}{\partial\theta} \log \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}|\theta) && (2.24)\\
&= \frac{1}{\sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}|\theta)} \sum_{\mathbf{h}} \frac{\partial}{\partial\theta} P(\mathbf{v}, \mathbf{h}|\theta) && (2.25)\\
&= \frac{1}{P(\mathbf{v}|\theta)} \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}|\theta) \frac{\partial}{\partial\theta} \log P(\mathbf{v}, \mathbf{h}|\theta) && (2.26)\\
&= \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}, \theta) \frac{\partial}{\partial\theta} \log P(\mathbf{v}, \mathbf{h}|\theta) && (2.27)\\
&= \left\langle \frac{\partial}{\partial\theta} \log P(\mathbf{v}, \mathbf{h}|\theta) \right\rangle_{P(\mathbf{h}|\mathbf{v},\theta)} && (2.28)
\end{aligned}$$

This shows that derivatives of the log-likelihood are equal to the expected derivatives of the complete data log-likelihood under the posterior distribution of the model. This implies that the critical points of the two expressions are also the same so that maximizing the log-likelihood is equivalent to maximizing the expected complete data likelihood under the posterior distribution of the model. However, we cannot simply follow the gradient of the complete-data log-likelihood because changing the model parameters changes the posterior, so the maximization must be done iteratively. This directly gives rise to the Expectation-Maximization (EM) algorithm of Dempster, Laird and Rubin (Dempster, Laird, and Rubin 1977). The two steps of the EM algorithm may be stated as follows:

E-Step Compute the posterior distribution $P(\mathbf{h}|\mathbf{v})$

M-Step Maximize the expected complete data log-likelihood with respect to the parameters, θ

This process is repeated until convergence. If the complete data likelihood is of a form which cannot be directly minimized then some form of gradient step may be taken. The

EM algorithm with a partial M-step is sometimes referred to as the Generalized EM (GEM) algorithm(McLachlan and Krishnan 1996).

If the posterior of the model can be computed tractably then application of the EM algorithm is straightforward. However, in many models of interest, the posterior distribution is too complicated to be computed, or represented, tractably. A variant of the EM algorithm allows us to deal with these situations by approximating the posterior distribution $P(\mathbf{h}|\mathbf{v},\theta)$ with another, simpler distribution $Q(\mathbf{h}|\xi)$(Neal and Hinton 1998):

$$\begin{aligned}
\log P(\mathbf{v}|\theta) &= \log \sum_{\mathbf{h}} P(\mathbf{v},\mathbf{h}|\theta) \\
&= \log \sum_{\mathbf{h}} Q(\mathbf{h}|\xi) \frac{P(\mathbf{v},\mathbf{h}|\theta)}{Q(\mathbf{h}|\xi)} \\
&\geq \sum_{\mathbf{h}} Q(\mathbf{h}|\xi) \log \frac{P(\mathbf{v},\mathbf{h}|\theta)}{Q(\mathbf{h}|\xi)} \\
&= \langle \log P(\mathbf{v},\mathbf{h}|\theta) \rangle_{Q(\mathbf{h}|\xi)} - \langle \log Q(\mathbf{h}|\xi) \rangle_{Q(\mathbf{h}|\xi)} \qquad (2.29) \\
&= \mathcal{F}(Q,\theta)
\end{aligned}$$

where the third step uses *Jensen's inequality* (Cover and Thomas 1991) to establish that $\mathcal{F}$ is a lower bound on the log-likelihood. This re-formulation of the problem now admits partial E-steps as well as partial M-steps. The variant EM algorithm can be stated as follows:

E-Step Holding θ fixed, maximize $\mathcal{F}$ with respect to the parameters ξ of the variational distribution Q

M-Step Holding Q fixed, maximize $\mathcal{F}$ with respect to the model parameters θ

This re-formulation of the EM algorithm permits learning in models with an intractable posterior distribution, and we are guaranteed to be increasing a lower bound on the log-likelihood, even if we are not guaranteed to be increasing the log-likelihood itself.

To make the relationship between $\mathcal{F}$ and the log-likelihood clearer we can rearrange it as follows:

$$\begin{aligned}
\mathcal{F}(Q,\theta) &= \sum_{\mathbf{h}} Q(\mathbf{h}|\xi) \log \frac{P(\mathbf{v},\mathbf{h}|\theta)}{Q(\mathbf{h}|\xi)} \\
&= \sum_{\mathbf{h}} Q(\mathbf{h}|\xi) \log \frac{P(\mathbf{v}|\theta) P(\mathbf{h}|\mathbf{v},\theta)}{Q(\mathbf{h}|\xi)} \\
&= \sum_{\mathbf{h}} Q(\mathbf{h}|\xi) \log P(\mathbf{v}|\theta) - \sum_{\mathbf{h}} Q(\mathbf{h}|\xi) \log \frac{Q(\mathbf{h}|\xi)}{P(\mathbf{h}|\mathbf{v},\theta)} \\
&= \log P(\mathbf{v}|\theta) - \mathrm{KL}(Q(\mathbf{h}|\xi) \| P(\mathbf{h}|\mathbf{v},\theta)), \qquad (2.30)
\end{aligned}$$

Since the KL-divergence between two distributions is always positive, it is clear that $\mathcal{F}$ is a lower bound on the log-likelihood of the data with equality when Q is exactly equal to the posterior distribution.

2.3.3 Learning in Markov Networks

Now we consider the problem performing maximum likelihood learning in Markov networks. Looking at equation 2.8 we can see that in terms of the potential functions the log-likelihood of a Markov network is:

$$\log P(\mathbf{x}) = \sum_{C \in \mathcal{C}} \log \psi_C(x_C) - \log Z \tag{2.31}$$

As with the Bayesian network the log-likelihood is a sum-of-logs of a set of local functions. The additional term which appears — the log of the partition function — is due to the fact that the Markov network requires the product of potential functions to be globally normalized. In contrast, the Bayesian network is constructed as a product of conditional probabilities, such that the no global normalization is required, so the problem does not arise.

By taking derivatives with respect to the parameters, we can see that optimizing this function poses problems. If each potential function, ψ_C, is parameterized by its own set of local parameters, θ_C, then the derivative with respect to a particular set of parameters θ_c is as follows:

$$\begin{aligned}
\frac{\partial}{\partial \theta_c} \log P(\mathbf{x}|\theta) &= \frac{\partial}{\partial \theta_c} \log \psi_c(x_c|\theta_c) - \frac{\partial}{\partial \theta_c} \log Z \\
&= \frac{\partial}{\partial \theta_c} \log \psi_c(x_c|\theta_c) - \frac{1}{Z} \sum_x \left[\prod_{c' \in \mathcal{C} \setminus c} \psi_{c'}(x_{c'}|\theta_{c'}) \right] \frac{\partial}{\partial \theta_c} \psi_c(x_c|\theta_c) \\
&= \frac{\partial}{\partial \theta_c} \log \psi_c(x_c|\theta_c) - \frac{1}{Z} \sum_x \left[\prod_{c' \in \mathcal{C}} \psi_{c'}(x_{c'}|\theta_{c'}) \right] \frac{\partial}{\partial \theta_c} \log \psi_c(x_c|\theta_c) \\
&= \frac{\partial}{\partial \theta_c} \log \psi_c(x_c|\theta_c) - \sum_x P(\mathbf{x}|\theta) \frac{\partial}{\partial \theta_c} \log \psi_c(x_c|\theta_c)
\end{aligned} \tag{2.32}$$

and if we take expectations with respect to the data distribution then we get the derivative of the ML objective function,

$$\left\langle \frac{\partial}{\partial \theta_c} \log P(\mathbf{x}|\theta) \right\rangle_{\widehat{P}(\mathbf{x})} = \left\langle \frac{\partial}{\partial \theta_c} \log \psi_c(\mathbf{x}_c|\theta_c) \right\rangle_{\widehat{P}(\mathbf{x})} - \left\langle \frac{\partial}{\partial \theta_c} \log \psi_C(\mathbf{x}_c|\theta_c) \right\rangle_{P(\mathbf{x}|\theta)} \tag{2.33}$$

The second term in this expression requires taking an expectation over all possible observations under the model distribution. This is clearly problematic in high dimensional, spaces or if the model density is not in a form which is analytically convenient. When we add the further complication of hidden variables then we must take expectations with respect to the posterior as well:

$$\left\langle \frac{\partial}{\partial \theta_c} \log P(\mathbf{v}|\theta) \right\rangle_{\widehat{P}(\mathbf{v})} = \left\langle \left\langle \frac{\partial}{\partial \theta_c} \log \psi_c(\mathbf{x}_c|\theta_c) \right\rangle_{P(\mathbf{h}|\mathbf{v},\theta)} \right\rangle_{\widehat{P}(\mathbf{v})} - \left\langle \left\langle \frac{\partial}{\partial \theta_c} \log \psi_C(\mathbf{x}_c|\theta_c) \right\rangle_{P(\mathbf{h}|\mathbf{v},\theta)} \right\rangle_{P(\mathbf{v}|\theta)} \tag{2.34}$$

where the complete set of variables x is the union of the the hidden and visible variables $x = \{v \cup h\}$. One way of computing the required expectations is to use MCMC methods to draw samples from the required distributions and then approximate the integrals. Here we give an example of how this approach has been applied to learning of the Boltzmann machine.

Example: Learning in Boltzmann Machines

As mentioned in section 2.2.2 the probability distribution of the network is Boltzmann distribution, and the local potential function relating two units s_i and s_j is just:

$$\psi_{\{i,j\}}(s_i, s_j) = e^{w_{ij} s_i s_j} \tag{2.35}$$

Where the units in the network have been partitioned into visible and hidden units, $\mathbf{s} = \{v \cup h\}$. Therefore the gradient of the expected log likelihood with respect to the weights is given by:

$$\frac{\partial}{\partial w_{ij}} \log P(\mathbf{v}|W) = \langle s_i s_j \rangle_{P(\mathbf{h}|\mathbf{v},W)} - \langle s_i s_j \rangle_{P(\mathbf{v},\mathbf{h}|W)} \tag{2.36}$$

The first expectation may be computed by clamping the visible units of the network to an observed data vector, and then sampling from the hidden variables conditioned on the clamped units. The second term of the gradient can be computed by leaving all units in the network unclamped and sampling from the joint of the hidden and the visible units(Ackley, Hinton, and Sejnowski 1985). By computing gradients in this way, the ML objective can be optimized using a steepest ascent algorithm.

The problem with this procedure is that the sampling must be done on the inside loop of an iterative optimization procedure. Not only is the sampling time-consuming, requiring many iterations to reach equilibrium, but it is only one step in the gradient optimization procedure. In addition, the sampling introduces noise into the computation of the gradients, which will introduce some random walk behaviour into the optimization. The Products of Experts model, which we will introduce in chapter 3, alleviates both of these problems by a simplification of the model structure, and a new objective function which does not require as much sampling to compute the gradients.

2.4 Dynamic Graphical Models

Since a main focus of this thesis is dynamic graphical models based on the hidden Markov model, in this section we will give an overview of graphical models for modelling sequential data and, in particular, we will review in detail the inference and learning algorithms of the HMM.

2.4.1 Hidden Markov Models

In modelling sequences we have a set of observations which are no longer independent of one another and we must account for this in the graphical model. There are two main ways of doing this. One method begins by assuming that all the information required to predict the current observation is contained in a finite number of observations from the series' history, *i.e.* we might like to model the distribution of observation y_t on the previous k observations: $P(y_t|y_{t-1}\dots y_{t-k})$. Such a model has the limitation that it can only capture dependences of length k, and the amount of history which we might need to capture all the relevant dependencies could be larger than this. In addition, as the length of our history window grows, so does the dimensionality of the conditioning space which we have to model.

The innovation of the hidden Markov (HMM) is that it introduces hidden state which controls the dependence of the current observation on the history of observations. The hidden state replaces the history in the sense that, instead of assuming that the current observation is independent of the past given the window of past observations, we assume that the current observation is independent of the past given the correct hidden state. The hidden state does not have to store all the details contained in the series' history, but only that information which is relevant to making the current prediction, and therefore it can be much more compact. In addition, the hidden state can, in theory, store information about events arbitrarily far in the past, relieving the restriction imposed by a finite-memory Markov model.

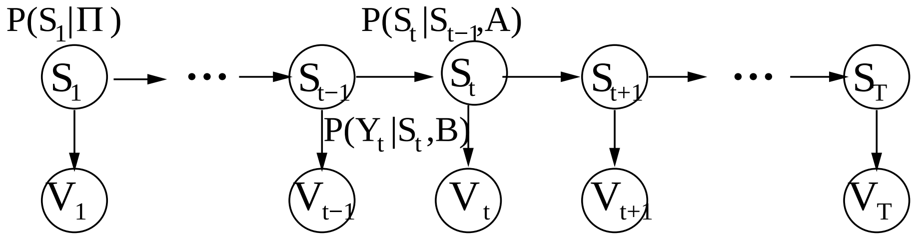

Figure 2.7: The Bayesian network for an HMM

The Bayesian network diagram for the HMM is shown in figure 2.7. From this graph we can see the fundamental property of the HMM, which is that conditioned on the hidden state at time t the current observation is independent of all the previous observations. The hidden state carries with it all information about the sequence's history.

The model has three sets of parameters

- The parameters, Π, which control the prior distribution of the hidden state, s, at time $t = 1$

- The stochastic matrix, A, whose elements, A_{ij}, give the probability of transiting from state $s_t = i$ to $s_{t+1} = j$

- The parameters which control the distribution over observables. In the case of a discrete output HMM, this is just another stochastic matrix, B. However, for a continuous output HMM, this might be a Gaussian with a separate mean, μ_i, and covariance matrix, Σ_i, for each hidden state.

The joint probability distribution of the hidden states and observations in the HMM is just the product of all the local conditional probabilities. For a discrete HMM this is:

$$\begin{aligned} P(s_{1:T}, y_{1:T}) &= P(s_1) \prod_{t=2}^{T} P(s_t|s_{t-1}) \prod_{t=1}^{T} P(y_t|s_t) \\ &= \prod_{i} \Pi_i^{\delta_{s_1,i}} \prod_{t=2}^{T} \prod_{i,j} A_{ij}^{\delta_{s_{t-1},i}\delta_{s_t,j}} \prod_{t=1}^{T} \prod_{i,j} B_{ij}^{\delta_{s_t,i}\delta_{y_t,j}} \end{aligned}$$

where we have used the Kronecker delta function, δ, as an indicator of when a discrete event occurred, such as the hidden state at time t was i: $\delta_{s_t,i}$.

2.4.2 Inference in HMM's: the Forward-Backward algorithm

By introducing hidden variables in the model we now have to deal with the problems of inferring a distribution over the hidden states conditioned on a set of observations. There are three main inference problems in HMM's: filtering, smoothing and prediction. Each of these can be interpreted as computing a distribution over the hidden variables conditioned on a particular subset of the observed variables: for filtering we would like to compute the distribution $P(s_t|y_{1:t})$, for k-step ahead prediction we compute the distribution $P(s_t|y_{t-k})$ and for smoothing we would like to compute the probability $P(s_t|y_{1:T})$. In addition, we would like to be able to compute the probability of a sequence under the model, $P(y_{1:T})$, summing out all hidden variables, and we will see that this result falls out of the inference computations.

The filtering and smoothing operations can be thought of as complementary parts of the same algorithm. A forward pass of information through the model will give us filtering distributions at each time, and a backward pass will compute the smoothing distributions. Starting with the forward pass, we show that this can be computed efficiently in a recursive manner by imagining that we have solved the problem of computing $P(s_t, y_{1:t-1})$ at some time step t. In the standard notation of HMM's this is denoted $\alpha(s_{t-1})$ (Rabiner and Juang 1986). Now we can compute the predictive distribution for the next time step t by computing the conditional joint of s_{t-1} and s_t and summing out s_{t-1}:

$$P(s_t, y_{1:t-1}) = \sum_{s_{t-1}} P(s_t|s_{t-1})P(s_{t-1}, y_{1:t-1}) \tag{2.37}$$

Next, we incorporate the likelihood of the observation at time t:

$$\alpha(s_t) = P(s_t, y_{1:t}) = P(y_t|s_t)\alpha(s_{t-1}) \tag{2.38}$$

We have now advanced the operation by one time step, computing $\alpha(s_t)$, using a recursive computation. The message passing is initialized by computing, $\alpha(s_1)$, the initial state prior weighted by the likelihood of that observation:

$$\alpha(s_1) = P(s_1|y_1) = P(s_1)P(y_1|s_1) \tag{2.39}$$

Thus, by induction, we see that we can compute $P(s_t, y_{1:t})$ for the entire series. Similarly, in a backward pass we can compute, $\beta(s_t)$, the probability of the rest of the observations conditioned on the current state $P(y_{t+1:T}|s_t)$. Assuming that we already have $\beta(s_{t+1}) =$

$P(y_{t+2:T}|s_{t+1})$ then the backward step is as follows:

$$\beta(s_t) = P(y_{t+1:T}|s_t) = \sum_{s_{t+1}} \beta(s_{t+1})P(y_{t+1}|s_{t+1})P(s_{t+1}|s_t) \tag{2.40}$$

Using Bayes' rule the two sets of distributions can be combined to compute the posterior marginal:

$$P(s_t|y_{1:T}) = \frac{P(s_t, y_{1:t})P(y_{t+1:T}|s_t)}{\sum_{s_t} P(s_t, y_{1:t})P(y_{t+1:T}|s_t)} = \frac{\alpha(s_t)\beta(s_t)}{\sum_{s_t} \alpha(s_t)\beta(s_t)} \tag{2.41}$$

The probability of a sequence under the model can be found by marginalizing out s_T from the final step of the forward pass:

$$P(y_{1:T}) = \sum_{s_T} P(s_T, y_{1:T}) \tag{2.42}$$

While these formulas give the correct forward and backward recursions, when the algorithm is implemented in practice, it cannot be implemented exactly as it is originally formulated. The two passes, consist of a series of multiplications of probabilities which are all less than one. Thus, even for short series numerical underflow may become a problem. Therefore, a scaling step is usually introduced into the algorithm. Rather than computing the joint of s_t and $y_{1:t}$, we divide by a scaling factor, c_t, which renormalizes the joint to be the conditional distribution $P(s_t|y_{1:T}$. The multiplication and marginalization steps of the forward pass go through exactly as stated above, except that we are now computing the conditional distribution at each time. The scaling factor turns out to be $c_t = P(y_t|y_{1:t-1})$, so that at the end of the forward pass, the probability of the sequence is just the product of all the c_t's, or the log-probability is the sum of their logs. During the backward pass we rescale the β's by the same constant c_t that was used in the forward pass so that, in the combination step when we compute the posterior marginal for $P(s_t|y_{1:T})$, the c_t's cancel out to get the correct result.

Once the backward pass has been computed an additional pass over the model can compute the posterior joint probabilities:

$$P(s_t, s_{t+1}|y_{1:T}) = \alpha(s_t)P(s_{t+1}|s_t)P(y_t|s_t)\beta(s_{t+1}) \tag{2.43}$$

This distribution is needed for updating the parameters controlling the transition probabilities, and it will also be needed for efficiently sampling from the posterior as we will see in a moment.

The forward-backward algorithm is based on the distributivity of sums and products but another operation which is distributive over addition is the max operation and it may be substituted into the forward-backward algorithm to obtain, not the posterior over hidden states, but the most likely hidden state path. This algorithm was first proposed by Viterbi (Viterbi 1967). As the forward-backward algorithm is a special case of belief propagation in singly-connected networks, similarly the generalization of Viterbi's algorithm to singly-connected Bayesian networks is the "max-product" algorithm developed by Dawid (Dawid 1992).

A further operation which we will need to do with HMM's is sampling from the posterior distribution over hidden state paths. The posterior distribution of the HMM is a chain, so this may be done in a single sweep through the hidden state variables. First we draw a sample from the posterior marginal of the initial state, $P(s_1|y_{1:T})$. From the forward-backward algorithm, we have the distribution of $P(s_1, s_2|y_{1:T})$, which we can condition on our draw of s_1 to get a distribution for s_2: $P(s_2|s_1, y_{1:T})$. We can repeat this process, drawing a sample for each state conditioned on the previous until we have a complete sample from the space of hidden state paths.

2.4.3 Learning the Parameters: Gradients of an HMM and the Baum-Welch Algorithm

The derivatives of a hidden Markov model can be computed using the property of latent variable models, that the derivative of the log-likelihood is equal to the derivative of the expected complete data log-likelihood under the posterior (eq. 2.28). For an HMM the complete data log-likelihood is given by:

$$\mathcal{L} = \sum_i \delta_{s_1,i} \log \Pi_i + \sum_{t=2}^{T} \sum_{i,j} \delta_{s_{t-1},i} \delta_{s_t,j} \log A_{ij} + \sum_{t=1}^{T} \sum_{i,j} \delta_{s_t,i} \delta_{y_t,j} \log B_{ij} \tag{2.44}$$

For convenience we use a softmax parameterization of the the discrete probabilities in the model: $A_{ij} = \sigma(a_{ij})$, $B_{ij} = \sigma(b_{ij})$ and $\Pi_i = \sigma(\pi_i)$, where

$$\sigma(a_{ij}) = \frac{e^{a_{ij}}}{\sum_k e^{a_{ik}}}.$$

If we were to optimize the transition and emission probabilities directly, it would require us to enforce the constraint that the parameters lie on a high dimensional simplex (positive, and sum to one). While there are methods for performing constrained gradient optimization on such constrained surfaces (Platt 1989), the softmax parameterization allows us to do unconstrained gradient optimization. The derivatives of the complete data log-likelihood for the various parameters are given in Appendix A. As described in section 2.3.2, we get the correct derivative for the log-likelihood by taking expectations of the total log-likelihoods. In essence, we substitute the hidden state probabilities in place of the counts that we would have if we had actually observed the hidden state occupancies. For example, the derivative of the transition probability parameters are:

$$\frac{\partial}{\partial a_{kl}}\mathcal{L} = \sum_{t=2}^{T}\left[\left\langle \delta_{s_{t-1},k}\delta_{s_t,l}\right\rangle_{P(s_{t-1},s_t|y_{1:T})} - \left\langle \delta_{s_{t-1},k}\right\rangle_{P(s_{t-1}|y_{1:T})}\sigma(a_{kl})\right] \tag{2.45}$$

where the angle brackets indicate the expectation under the posterior of transitioning from state $s_{t-1} = k$ to $s_t = l$ and the posterior probability of being in state $s_{t-1} = k$, respectively. Thus, every time we compute the gradient of the model we need to call on the forward-backward algorithm to compute the required posterior probabilities, which are then substituted into the derivative formulas. If we were to solve this equation directly for A_{ij} then we would get a fixed point equation for the new estimate of the transition probability. Iterating the process of computing the posterior distribution and re-estimating the probabilities yields an EM algorithm for maximizing the likelihood of HMM's which was originally discovered as the Baum-Welch algorithm (Baum, Petrie, Soules, and Weiss 1970). In this thesis, however, we will be using gradient methods for optimizing the parameters of HMM's.

2.4.4 Extensions of HMM's

The development of the graphical model formalism and the realization that the HMM and other well known time series models (such as the Kalman filter) are special cases of singly-connected graphical models (Smyth, Heckerman, and Jordan 1997; Murphy 1999), have aided the exploration of more complex models of dynamic data. The dependency rules for graphical models and the propagation algorithms for performing inference in them indicate in which models inference may be tractably performed. Armed with the theory of graphical models and a set of general algorithms for performing inference and the EM for learning, it becomes possible to consider extensions to the HMM by adding variables or links to capture more of the structure of the data.

Figure 2.8a) shows the dependency graph of a hidden Markov model. The major attraction of the HMM is that the inference algorithm together with the Baum-Welch re-estimation formulas can efficiently fit the HMM to data (Baum, Petrie, Soules, and Weiss 1970; Dempster, Laird, and Rubin 1977). The standard method for performing

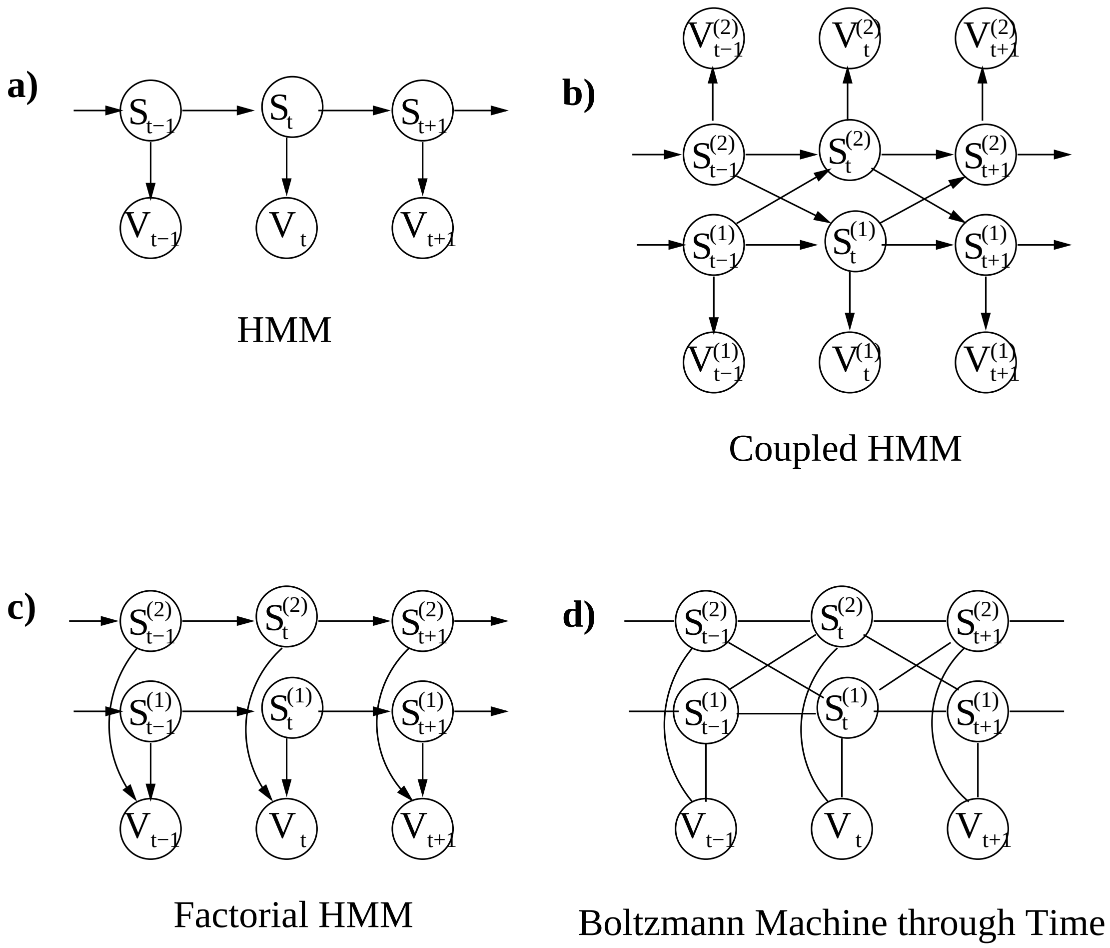

Figure 2.8: A taxonomy of hidden Markov models.

inference in HMM's, the forward-backward algorithm, (Rabiner and Juang 1986) has complexity $O(TK^2)$ in the length of the sequence T and the size of the state variable K.

However, this model has a couple of important limitations. While the HMM is in theory capable of capturing high order dependencies in a sequence, in practice the ability of the model to retain information about past events is related to the mixing time of the hidden state Markov chain (Bengio and Fransconi 1995). Also, the HMM cannot deal well with hierarchical structure in time series, or multivariate time series with interacting structure at multiple time scales (Saul and Jordan 1995). People have tried to address these limitations by modifications to the dependency structure of an HMM.

There are a few ways in which the HMM might be made more complicated: by increasing the order of Markov dependence for example, or by adding more hidden variables. However, these modifications usually move us into the realm of intractable inference, or intractable learning. In the next section I will describe some variations of the HMM which have been proposed and how intractability, when it arises, can be dealt with by approximations.

If the data which is being modelled has hierarchical structure, such as the grammar of a natural language, an HMM whose hidden state has hierarchical structure may be used. This increases the complexity of the inference algorithm, but the computations can still be done in time which is polynomial in the length of the series and the size of the hidden state. Stochastic context free grammars (SCFGs) with the corresponding inside-outside algorithm for performing inference (Lari and Young 1990), and hierarchical HMM's (Fine, Singer, and Tishby 1998) are both examples of such approaches.

If the sequences are multivariate and contain interacting structure, or structure at multiple time scales then coupled hidden Markov models may be appropriate. As shown in figure 2.8b), a coupled hidden Markov model uses interconnected hidden state chains to represent the interactions among the observed sequences (Brand 1996; Saul and Jordan 1995). If the interactions occur at different time scales then the model might include internal chains with different time resolutions, or interchain connections which have different lags (Rezek, Sykacek, and Roberts 2000). Inference can be performed by aggregating variables at adjacent time steps and performing inference in the cross product space. However, the complexity of inference grows exponentially in, M, the number of chains, $O(TK^{2M})$, and if the lags in the network are greater than first order then the number of variables which must be aggregated becomes impractical. Otherwise, approximate methods of doing inference may be used (Brand 1996). One approximate method is to choose the most probable path (the Viterbi path) of each hidden state chain conditioned on all the others, and then circularly visit each chain, performing the same procedure until it converges. Such a *chainwise* Viterbi scheme only requires $O(TM^2K^2)$ operations but it is only guaranteed to find a local maximum of the posterior, not the global MAP configuration (Jordan, Ghahramani, and Saul 1997; Saul and Jordan 1999b).

A different way of extending the HMM is to incorporate several hidden state chains whose output densities combine to model a single observed sequence. The important thing about latent-variable time-series models is that the hidden variable encapsulates all the relevant information about the series' history *i.e.* conditioned on the hidden variable the future is independent of the past. Such a model is more powerful than a fully observed Markov model where all the required information about the past is assumed to reside in the last few observations. An HMM may carry forward information about observations which occurred arbitrarily long in the past, however it is limited by the efficiency with which this information can be expressed in it's hidden state. Where an HMM with a single K state multinomial hidden variable can represent $\log_2 K$ bits of information about a sequence a model with M hidden state chains can transmit $M \log_2 K$

bits of information. Another way of saying this is that the number of configurations of the hidden state variables, at a given time step, grows exponentially in the number of hidden variables.

Incorporating multiple chains into an HMM may be done in a couple of ways. One is to add several hidden state variables which have a *causal* effect on the observed variables in the model. This is known as a Factorial HMM (FHMM) (Ghahramani and Jordan 1997) and is shown in figure 2.8c). In a causal belief network each local probability distribution can be independently re-estimated if we can compute the posterior of the latent variables. However, it is exponentially expensive to compute this posterior distribution exactly because observing the visible variables induces dependencies among the hidden variables. Ghahramani and Jordan handle this problem by approximating the posterior with a factored, variational distribution. In another example of a model with an extended hidden state representation, Williams and Hinton (Williams and Hinton 1991) used a mean-field Boltzmann machine, a non-causal, undirected graphical model (figure 2.8d). In this case, the hidden state also consists of multiple hidden variables, but the variables are fully interconnected from one time step to the next. Due to the full interconnection of the hidden state variables, inference in this model is intractable. Because it is an undirected graphical model with the corresponding partition function there is no exact update step for the parameters and learning must be done through gradient ascent in the log likelihood. The authors deal with the intractability of inference using a fully-factored, mean field approximation to compute the posterior distribution.

In the next chapter we will introduce a model which has some of the flavour of both the Factorial HMM and the Boltzmann machine through time models.

Chapter 3

Products of Experts

3.1 Introduction

In the previous chapter we reviewed the theory of both directed and undirected graphical models, and we looked at the problem of inference and learning in both. Directed graphical models have the convenient property that if we can perform inference and collect the sufficient statistics of the hidden variables, then learning can be done in a straightforward manner using the EM algorithm. In undirected graphical models both the problem of inferring the distribution over hidden state and estimating the parameters is difficult.

Hinton (2001) introduced a particular combination of directed and undirected graphical models called a Product of Experts (PoE). The PoE is a way of combining the densities of many latent variable models. PoE's are defined by the following formula:

$$P(\mathbf{v}|\Theta) = \frac{\prod_{m=1}^{M} P(\mathbf{v}|\theta^{(m)})}{\sum_{\mathbf{u}\in\mathbf{v}} \prod_{m=1}^{M} P(\mathbf{u}|\theta^{(m)})} \tag{3.1}$$

where the probability distribution of the individual experts may be of any type. While the experts might be fully visible models, it is more interesting to consider models with hidden variables. The graphical model for a general product model is shown in figure 3.1. The figure shows each expert as a single hidden node in the graph, but each of them may have many hidden variables. The internal structure of an expert could be as simple as a single binary variable or it could have much more complicated structure. The connections between the experts and the visible units indicate that each expert defines a potential function over itself and the hidden units. According to equation 3.1 this potential takes the form of a normalized probability distribution. While this is not required in general — the experts could consist of unnormalized potential functions — we will always take

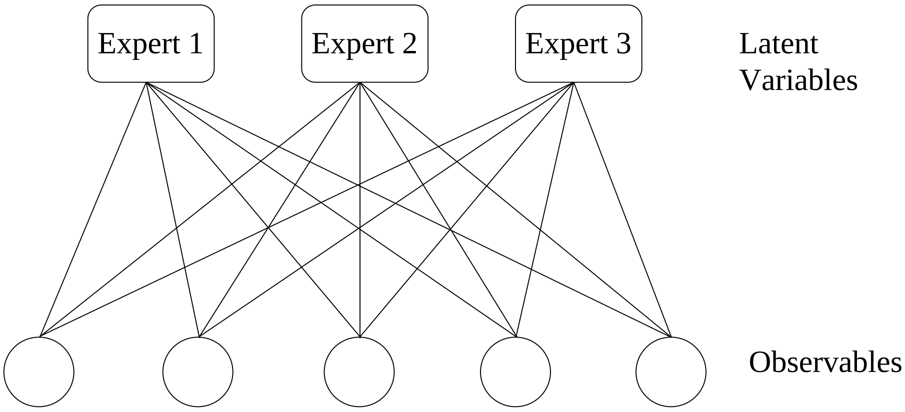

Figure 3.1: The graphical model of a general Product of Experts.

our constituent experts to be proper densities.

The key property of the PoE is that it is an undirected graphical model with no direct connections among the hidden variables of different experts[1], the experts only interact with each other indirectly, via the observed variables. This Markov blanket for each expert consists of the visible variables, which means that conditioned on the visible variables the hidden variables of all the experts are rendered independent of one another. The advantage of an architecture such as this is that when we are performing inference on the hidden units, we can do inference in each of the experts independently. So if inference in each of the constituent models is tractable then inference in the product is also tractable. However, when the states of the visible variables are not observed then the probability distributions defined by the experts interact and induce distributions over the visible units which are much more complicated than the distributions of any of the experts by itself.

In contrast to the PoE, a different way of combining many density models is as a *mixture of experts*. We can better understand the difference between a PoE and a mixture by considering the generative model of each. In a mixture model we generate a data point by picking one of the elements of the mixture and then generating from that mixture component. Only one element of the mixture is responsible for generating an observation, and when we do inference in a mixture model, we compute the posterior probability that each mixture component was the one that generated the data. In a product model, on the other hand, *all* of the experts in the model are responsible for generating the data point. A naive way of generating an observation from a product model would be to generate from each of the experts in the PoE, and then if they all generated the same point then we accept it, otherwise we reject it an start again until we

[1]Formally, the graphical model forms a bi-partite graph.

generate an observation which is acceptable to all the experts. While this is an unrealistic way of actually generating from a PoE, it illustrates the point that all of the experts in the model have some influence on the generated data, and so when we do inference in a PoE, we are not just inferring the probability that any one of them generated the data point, but the hidden state that they would all have to take on in order to generate the given observation.

So by a simple counting argument, if each component in a mixture of experts has K hidden states, and there are M experts then the mixture has a total of MK states available to represent the data. By contrast, in a product model all of the experts contribute to generating a data point. Each one can take on its own state, and the number of states for the product model is K^M. Thus, using many experts in collaboration to generate the data yields a combinatorial hidden state representation.

3.2 Learning

In order to learn the parameters of a product model we will draw on some of the concepts of learning graphical models that were reviewed in chapter 2. To fit the model described in equation 3.1 to data we would like to maximize the likelihood of a data set under the model. The maximum likelihood learning criteria is equivalent to the minimization of the Kullback-Liebler divergence between an empirical data distribution, $\widehat{P}$, and the distribution represented by our model, P. Taking the gradient of the negative KL divergence,

$$-KL(\widehat{P}||P) = \langle \log P(\mathbf{v}|\Theta) \rangle_{\widehat{P}(\mathbf{v})} - \left\langle \log \widehat{P}(\mathbf{v}) \right\rangle_{\widehat{P}(\mathbf{v})} \tag{3.2}$$

with respect to the parameters of our model, gives us a direction in which to move to improve the model. The PoE is similar to an undirected graphical model in that it contains a partition function. As we saw with Markov networks in section 2.3.3, taking the gradient of a PoE with respect to the parameters of an individual expert gives an expression which is the difference of two expectations:

$$\frac{\partial}{\partial \theta^{(m)}} KL(\widehat{P}(\mathbf{v})||P(\mathbf{v}|\Theta)) = \left\langle \frac{\partial}{\partial \theta^{(m)}} \log P(\mathbf{v}|\theta^{(m)}) \right\rangle_{\widehat{P}(\mathbf{v})} - \left\langle \frac{\partial}{\partial \theta^{(m)}} \log P(\mathbf{v}|\theta^{(m)}) \right\rangle_{P(\mathbf{v}|\Theta)} \tag{3.3}$$

The first term is the expected gradient of the expert under the empirical distribution and the second is the expected gradient of the expert under the model's distribution. If we have chosen experts in which we can tractably perform inference, then the first term can

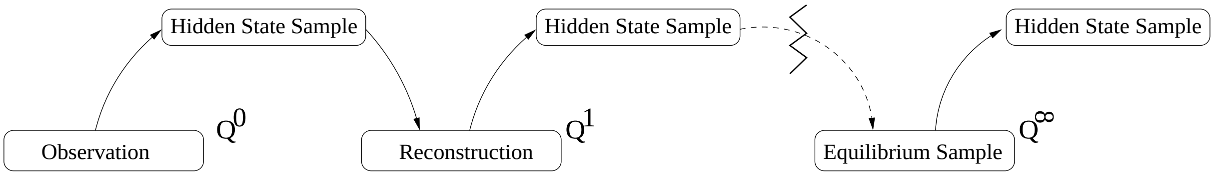

Figure 3.2: Gibbs sampling from a Product of Experts.

also be computed tractable. Unfortunately, the second term is difficult to compute even if the experts are simple.

Since we cannot compute this derivative analytically, we could compute approximately by Gibbs sampling from the model to get a set of samples to approximate the expectation (more typically, just one is used). Usually in Gibbs sampling (Neal 1993), we sample sequentially from the distribution of each variable conditioned on all other variables in the model. However, the PoE model has some independence properties which make this procedure simpler. The hidden variables are conditionally independent given the observed variables and, providing that dependencies between the observed variables are intermediated by the latent variables, the observed variables are conditionally independent given the hidden variables[2]. This makes it easy to alternately sample from the two distributions $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$. While this makes sampling in blocks from the visible and hidden variables possible, we must repeat this procedure until equilibrium is reached. The result is that sampling in order to estimate a single gradient step is still a very slow procedure. Moreover, the resulting gradients are stochastic and noisy, particularly at the start of learning when the model is unrefined and we really only need a crude estimate of the gradient to improve, and at the end of learning when we expect the differences between $\left\langle \frac{\partial}{\partial \theta^{(m)}} \log P(\mathbf{v}|\theta^{(m)}) \right\rangle_{\widehat{P}(\mathbf{v})}$ and $\left\langle \frac{\partial}{\partial \theta^{(m)}} \log P(\mathbf{v}|\theta^{(m)}) \right\rangle_{P(\mathbf{v}|\Theta)}$ to be small relative to the sampling noise.

3.2.1 Contrastive Divergence

An alternative learning procedure has been proposed by Hinton (Hinton 2001), in an attempt to alleviate these problems. The gradient computation is based on the above sampling procedure, but it is modified in two ways. Figure 3.2 gives a schematic illustration of the Gibbs sampling. The standard Gibbs sampling procedure for a PoE starts from a sample from some initial distribution, and then alternately samples from the visible and hidden units until the equilibrium distribution is reached. The new algorithm

[2]This is not true in general, since it is possible to imagine a product of experts in which there are direct dependencies among the observed variables. The PoE framework in and of itself does not rule this out.

differs from the standard method in two ways. First, in Gibbs sampling, the variables in a model may be initialized from any distribution. In terms of getting a correct sample from the equilibrium distribution of the Markov chain, the starting distribution of the sampler does not matter. In fact, the equilibrium distribution is only reached when the chain has reached a distribution which is independent of the initial configuration. In the new learning procedure, instead of initializing from some random distribution, we initialize the visible variables of the model with a sample from our data. Thus the initial distribution of the visible variables is the empirical distribution of the data. Here we change our notation slightly. As seen in figure 3.2, the initial distribution of the sampler, the data distribution, is called Q^0, and the sequence of distributions over the observed variables moves from Q^0 to Q^1 through to the equilibrium distribution, which is now denoted Q^∞. We choose this notation because it makes explicit the fact that the sample at each step of Gibbs sampling is drawn from a distribution which gets successively closer and closer to the equilibrium distribution (Goutsias 1991).

Changing the initial distribution of the Gibbs sampler does not in itself change the learning algorithm. The second alteration of the learning procedure is that rather than sampling to equilibrium, we stop after just one iteration of Gibbs sampling. The new learning rule is modified to use this one-step sample from Q^1 instead of a sample from Q^∞:

$$\Delta\theta^{(m)} = \left\langle \frac{\partial}{\partial\theta^{(m)}} \log P(\mathbf{v}|\theta^{(m)}) \right\rangle_{Q^0(\mathbf{v})} - \left\langle \frac{\partial}{\partial\theta^{(m)}} \log P(\mathbf{v}|\theta^{(m)}) \right\rangle_{Q^1(\mathbf{v}|\Theta)}. \tag{3.4}$$

This modified learning rule is no longer the gradient of the ML criteria, but may be thought of as the approximate gradient of another objective function,

$$KL(Q^0(\mathbf{v})||Q^\infty(\mathbf{v}|\theta)) - KL(Q^1(\mathbf{v}|\theta)||Q^\infty(\mathbf{v}|\theta)),$$

called the *contrastive divergence* (Hinton 2001).

We can see how this learning rule falls out of the contrastive divergence criteria by looking at the derivatives. First we take the derivative of the first term of the contrastive divergence criteria:

$$\frac{\partial}{\partial\theta} KL(Q^0(\mathbf{v})||Q^\infty(\mathbf{v}|\theta)) = -\int Q^0(\mathbf{v}) \frac{\partial}{\partial\theta} \log Q^\infty(\mathbf{v}|\theta)\, d\mathbf{v} = -\left\langle \frac{\partial}{\partial\theta} \log Q^\infty(\mathbf{v}|\theta) \right\rangle_{Q^0(\mathbf{v})} \tag{3.5}$$

The first term of the derivative is particularly simple because, $Q^0(\mathbf{v})$, is independent of the model parameters and the derivative pulls through the expectation in the KL divergence. The second part of the derivative is more complicated because it is an expectation under

$Q^1(\mathbf{v}|\theta)$ which is a function of the model parameters. By the chain rule we get the following expression:

$$\begin{aligned}
&\frac{\partial}{\partial\theta} KL(Q^1(\mathbf{v}|\theta)||Q^\infty(\mathbf{v}|\theta)) \\
&= \frac{\partial KL(Q^1(\mathbf{v}|\theta)||Q^\infty(\mathbf{v}|\theta))}{\partial Q^\infty} \cdot \frac{\partial Q^\infty(\mathbf{v}|\theta)}{\partial\theta} + \frac{\partial KL(Q^1(\mathbf{v}|\theta)||Q^\infty(\mathbf{v}|\theta))}{\partial Q^1} \cdot \frac{\partial Q^1(\mathbf{v}|\theta)}{\partial\theta}
\end{aligned} \tag{3.6}$$

Taking just the first of these terms we can see that it gives the desired expectation:

$$\begin{aligned}
\frac{\partial KL(Q^1(\mathbf{v}|\theta)||Q^\infty(\mathbf{v}|\theta))}{\partial Q^\infty(\mathbf{v}|\theta)} &= -\int Q^1(\mathbf{v}|\theta)\frac{\partial}{\partial Q^\infty}\log Q^\infty(\mathbf{v}|\theta)\; d\mathbf{v} \cdot \frac{\partial Q^\infty}{\partial\theta} \\
&= -\int Q^1(\mathbf{v}|\theta)\frac{1}{Q^\infty(\mathbf{v}|\theta)}\; d\mathbf{v} \cdot \frac{\partial Q^\infty}{\partial\theta} \\
&= -\int Q^1(\mathbf{v}|\theta)\frac{\partial}{\partial \mathbf{v}}\log Q^\infty(\mathbf{v}|\theta)\; d\theta \\
&= -\left\langle \frac{\partial \log Q^\infty(\mathbf{v}|\theta)}{\partial\theta} \right\rangle_{Q^1(\mathbf{v}|\theta)}
\end{aligned} \tag{3.7}$$

Reassembling the pieces we see that this gradient will almost give us the learning rule in equation 3.4, except for the second term in 3.6. However, as argued in (Hinton 2001) this term is typically small, and usually in the same direction as the rest of the gradient, so it may safely be ignored. Thus we get the final, approximate gradient:

$$\begin{aligned}
&\frac{\partial}{\partial\theta} KL(Q^0(\mathbf{v})||Q^\infty(\mathbf{v}|\theta)) - KL(Q^1(\mathbf{v}|\theta)||Q^\infty(\mathbf{v}|\theta)) \\
&\approx -\left\langle \frac{\partial}{\partial\theta}\log Q^\infty(\mathbf{v}|\theta) \right\rangle_{Q^0(\mathbf{v})} + \left\langle \frac{\partial \log Q^\infty(\mathbf{v}|\theta)}{\partial\theta} \right\rangle_{Q^1(\mathbf{v}|\theta)}
\end{aligned} \tag{3.8}$$

Flipping the sign to go downhill in the contrastive divergence objective function gives us the update rule in equation 3.4.

The primary advantages of this learning rule are that it provides a quicker, less noisy gradient than with full Gibbs sampling procedure. In addition to the theoretical justification given above, there are several ways of motivating this learning rule. One is that it is an approximation to the full Gibbs sampling procedure, because the expectation under Q^1 is closer to the equilibrium distribution than any distribution we choose to initialize our sampler — although it may still be a long way from it.

A second way of thinking about this learning rule is by imagining the data to lie on a manifold in some high dimensional space. We would like our model to put high probability on the data and low probability everywhere else. Contrastive divergence does this by measuring the expected derivative of the data, and taking a step to increase this probability. Then it takes one Gibbs sampling step away from the data, measures the expected derivative of the reconstruction under the model, and takes a step to decrease this probability. The result is to make it less likely for the model to wander away from the data manifold during Gibbs sampling.

This learning rule has the disadvantage that it is a stochastic gradient rule which is subject to noise, and therefore precludes the use of more sophisticated optimization, such as second order or conjugate gradient methods. Simple gradient descent methods cannot compete with the convergence properties of fixed-point algorithms used to fit the parameters of tractable models. For example the EM algorithm for mixtures of Gaussians will converge much faster than the contrastive divergence algorithm for PoE's.

Chapter 4

Products of Hidden Markov Models

4.1 Introduction

Section 2.4.4 described a number of different ways of extending the hidden state representation of hidden Markov models, including the use of multiple hidden state chains which interact via causal connections to the observed variables. In this section we introduce a model which extends the power of HMM's in a very different way — a product of experts which uses HMM's as the basic experts.

There are several reasons why we would want to use a product model constructed out of many HMM's. As mentioned in the previous chapter, a model consisting of multiple hidden state chains has a much more powerful representation of the data than can be achieved with a single K-valued state variable. There are two potential benefits to this.

First, this makes the model ideal for data which is caused by multiple underlying influences. If each of the HMM's in the product of hidden Markov models (PoHMM) can capture some aspect of the data and constrain it, then the combination of the models will create a very sharp distribution over the data. Notice, that we have described these as interacting "influences" and not causes, because the PoHMM is a descriptive rather than a causal model of the data.

Secondly, the HMM is notoriously poor at learning long range structure in time series. Bengio and Fransconi (1995) analysed this problem and determined that in order to store information from the past the model must learn to segment the state space into disjoint sets of states, and by constraining its dynamics to just one of these subsets, it remembers something about where it was in previous time steps (*i.e.* the hidden states are not ergodic). The corollary to this is that to store a single bit of information, the HMM must split its hidden state into two disjoint subsets. To store more bits of information requires

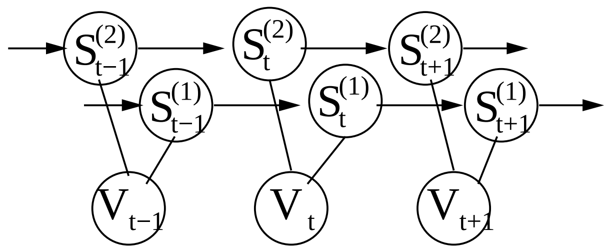

Figure 4.1: Graphical model of a Product of Hidden Markov Models

an exponentially increasing number of partitions of the state space. The PoHMM helps to rectify this by allowing each model to remember a different piece of information about the past. Thus, hopefully, one model will retain interesting information about the past while other models are left to fit the intervening structure in the data.

The chapter has three parts. Section 4.2 will present the theory of the PoHMM, and describe how contrastive divergence learning can be applied to it. Section 4.3 presents a series of experiment which demonstrate the advantages of a PoHMM relative to a regular HMM. Finally, section 4.4 discusses some remaining issues relevant to the application of PoHMM's.

4.2 Theory of a PoHMM

The graphical model diagram of a product of hidden Markov models is shown in figure 4.1. Here, we have shown a PoHMM with two hidden state chains for clarity, but it need not be restricted to just two, the architecture could be extended to many hidden state chains. Where the Factorial HMM consists wholly of directed links between the nodes in the model, the PoHMM is a combination of directed and undirected graphical models. The hidden states are connected in a directed way, but their relations with the visible units are undirected. This causes the PoHMM to have quite different conditional independence relationships among its variables than an FHMM. As discussed in the previous section on PoE's, some of these properties are convenient in terms of inference and sampling from the model.

As with other PoE models the product of HMM's (PoHMM) is defined by multiplying together the densities of its, M, experts and renormalizing:

$$P(V_{1:T}|\Theta) = \frac{\prod_{m=1}^{M} P^{(m)}(V_{1:T}|\theta^{(m)})}{Z(T,\Theta)}, \tag{4.1}$$

$$Z(T,\Theta) = \sum_{U_{1:T}\in\mathcal{V}} \prod_{m=1}^{M} P^{(m)}(U_{1:T}|\theta^{(m)}) \tag{4.2}$$

where Z is the sum over all possible observed sequences, ensuring that the density function sums to 1, and Θ is the set of all parameters in the product model $\{\theta_1, \ldots, \theta_M\}$.

In this case the density of each expert m is an HMM. The joint distribution of the visible and hidden variables in an HMM is given by:

$$P^{(m)}(V_{1:T}, S_{1:T}^{(m)}|\theta^{(m)}) = \\ P(S_0^{(m)}|\pi^{(m)}) \prod_{t=2}^{T} P(S_t^{(m)}|S_{t-1}^{(m)}, A^{(m)}) \prod_{t=1}^{T} P(V_t|S_t^{(m)}, B^{(m)}). \tag{4.3}$$

and the distribution over the visible variables can be found by summing out the hidden variables:

$$P^{(m)}(V_{1:T}|\theta^{(m)}) = \sum_{\mathcal{S}^{(m)}} P^{(m)}(V_{1:T}, S_{1:T}^{(m)}|\theta^{(m)}). \tag{4.4}$$

This can be done efficiently using the forward backward algorithm described in section 2.4.2. Note that we have given the general formulation for an HMM where we have not specified the HMM output densities. They may be either continuous or discrete. If all of the HMM's in a PoHMM have the same number of states, K, and there are M of them then we will denote it as PoHMM(M,K).

The log-likelihood of a sequence under a PoHMM is given by:

$$\log P(V_{1:T}|\Theta) = \sum_{m=1}^{M} \log P^{(m)}(V_{1:T}|\theta^{(m)}) - \log Z(T, \Theta) \tag{4.5}$$

We see that it is just the sum of log-likelihoods of the sequence under the individual HMM's less the log of the partition function. While it is possible to compare the log-likelihood of two sequences under a single PoHMM, because the log of the partition function will cancel out, we can not directly score the relative log-likelihood of a sequence under two different PoHMM's. However, there is a way around this problem by discriminatively estimating the difference in the log partition functions. Such an approach will be used in the experiments reported later in the chapter.

4.2.1 Conditional Independence

The conditional independence properties of PoHMM are quite different from those of a Factorial HMM. Because the FHMM is a directed graph, with converging links on the observed variables (as shown in figure 2.8c), then by the rules of d-separation in directed graphical models, conditioning on the visible variables of the model induces dependencies

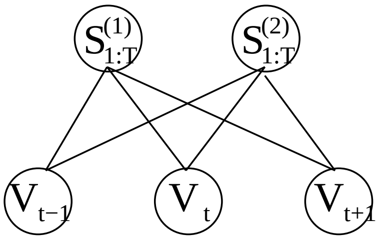

Figure 4.2: A different view of PoHMM independence properties

among the hidden variables. The PoHMM network has the opposite property of a directed graphical model, in that, conditioned on a set of observations, the hidden state chains are independent. So exact inference can easily be performed by using the forward-backward algorithm in each chain separately. This is made clearer by a different version of the PoHMM graphical model. Figure 4.2 shows a graph which has all the same dependencies of the PoHMM show in figure 4.1, except that the hidden state variables have been grouped into two single nodes. By hiding the internal structure of the hidden variables we can see the bi-partite nature of this graph, such that observing the visible variables separates the hidden variables according to the rules of Markov graphs. Even though inference in a PoHMM is easy and efficient, learning of the local probability functions is still complex, because the local distributions are all linked by a global partition function.

4.2.2 Contrastive Divergence Learning for the PoHMM

Given the learning algorithm for the general case of PoE's in the previous section, we turn our attention to the specific case of PoHMM's. In order to implement the contrastive divergence learning algorithm for a PoHMM, we need the gradient of the model parameters for a given sequence from our data set. We also need to sample from the one step reconstruction and compute the gradient from this sequence. As in the general case, computing the gradient of model parameters for the PoHMM consists of computing the gradient for the parameters of each model separately.

To sample from the one step reconstruction of the PoHMM, we must first draw a sample from the posterior distribution of hidden state *paths* of each HMM. It is important to note that the forward-backward algorithm computes the posterior marginals for particular variables, and the posterior marginal distributions for pairs of hidden state variables ($P(S_t|V_{1:T})$, $P(S_t, S_{t+1}|V_{1:T})$). It is not enough to sample independently from $P(S_t|V_{1:T})$, as this will not generate a correct sample from the distribution of paths. For example, sampling independently from $P(S_t|V_{1:T})$ and $P(S_{t+1}|V_{1:T})$ might draw values for variable S_t and S_{t+1} which are probable under their posterior marginal, but when taken together, are inconsistent because there is small or zero probability of transitioning from S_t to S_{t+1}. To generate a correct sample from the posterior distribution of hidden state paths, we use the procedure outlined in section 2.4.2, whereby we draw sequentially from the posterior distribution of each hidden state variable, conditioned on the sample of the preceding variable.

Given a sample path from the posterior distribution of each HMM, $\{s_{1:T}^{(1)}, s_{1:T}^{(2)}, \ldots, s_{1:T}^{(M)}\}$, we can see from figure 4.2 that the distributions over the visible variables are independent. We can sample from each of these by multiplying together the emission distributions of the HMM conditioned on the hidden state configuration at a particular time step, and renormalizing. The form of this distribution depends on whether the HMM has discrete or continuous emission distributions. In the case of a discrete output HMM using conditional probability tables (CPT's) to represent the distribution, we can do an element-wise multiplication of the rows of the CPT of an HMM corresponding to the hidden variable sample from that HMM at that time step.

$$P(V_t = i|\{s_t^{(1)}, \ldots, s_t^{(M)}\}) = \frac{1}{Z} \prod_{m=1}^{M} \prod_{i,j} B_{ij}^{\delta_{i,V_t}, \delta_{j,s_t^{(m)}}} \tag{4.6}$$

where Z is the sum of that quantity over all values, i, of V_T.

If we use a continuous valued HMM, with multivariate Gaussian outputs, then each state, j, of each HMM, m, has its own mean, $\mu_j^{(m)}$ and covariance matrix, $\Sigma_j^{(m)}$. The product of these is also a multivariate Gaussian where the means and variances are given as follows:

$$\Sigma_{V_t|\{s_t^{(1)},\ldots,s_t^{(M)}\}} = \left(\sum_{m=1}^{M} \left(\Sigma_{s_t^{(m)}}^{(m)}\right)^{-1} \right)^{-1} \tag{4.7}$$

$$\mu_{V_t|\{s_t^{(1)},\ldots,s_t^{(M)}\}} = \Sigma_{V_t|\{s_t^{(1)},\ldots,s_t^{(M)}\}} \left(\sum_{m=1}^{M} \left(\Sigma_{s_t^{(m)}}^{(m)}\right)^{-1} \mu_{s_t^{(m)}}^{(m)} \right) \tag{4.8}$$

So, given a set of samples of the hidden states of the HMM's at a particular time step we can compute the mean and variance of the resulting conditional Gaussian distribution over the visible variables and draw a sample from it. Now we can compute the gradient of each of the models of the HMM with respect to this reconstructed sequence to get the negative part of the contrastive divergence update rule.

To summarize, the contrastive divergence algorithm for training a PoHMM has the following steps:

1. Calculate each model's gradient $\frac{\partial}{\partial \theta^{(m)}} P(V_{1:T}|\theta^{(m)})$ on a data point using the forward-

backward algorithm.

2. For each model take a sample from the posterior distribution of paths through state space.

3. At each time step, multiply together the distributions over symbols specified by the chosen paths in each HMM. Renormalize to get the reconstruction distribution at that time step.

4. Draw a sample from the reconstruction distribution at each time step to get a reconstructed sequence, $\tilde{V}_{1:T}$. Compute each model's gradient on the new sequence $\frac{\partial}{\partial\theta^{(m)}} P(\tilde{V}_{1:T}|\theta^{(m)})$

5. Update the parameters:

$$\Delta\theta^{(m)} \propto \frac{\partial}{\partial\theta^{(m)}} \log P(V_{1:T}|\Theta) - \frac{\partial}{\partial\theta^{(m)}} \log P(\tilde{V}_{1:T}|\Theta)$$

4.3 Experiments

To demonstrate the relative merits of a product of HMM's versus a single HMM, we have applied them to a series of tasks. The first of these is a toy language modelling task and the second a character string modelling task. Further, to try to establish the relative merits of using a PoHMM as opposed to an HMM with an equivalent number of parameters we have conducted a more thorough set of experiments on two synthetic datasets. One which compares the the two models on data which has componential structure, and a second data set which displays long range structure.

These comparisons should tell us a couple of things. First, given a fixed number of parameters is it beneficial to use a model which has this product structure and the concomitant power of hidden state representation, versus the simple mixture model structure of an HMM. Second, the HMM has a tractable fixed point algorithm for fitting its parameters, whereas the PoHMM has a stochastic gradient descent algorithm, which is obviously less efficient. Does the difference in efficiency of the training algorithms, outweigh the benefit of a richer hidden state structure?

4.3.1 Modelling Simple Sentences

In this task, matching the models for the number of parameters, we use a single HMM with 32 states and a product of 10, 6 and 7 state HMM's to model a set of English sentences of the form, “Yes I am _” (with a terminating blank denoted by _) or “No she is not”. There are 14 legal sentences in the grammar, including all combinations of yes and no with the pronouns (I,you,he,she,it,we,they) and their corresponding conjugation of the verb “to be”. The sentences feature two kinds of agreement. There is short range agreement between the subject and the verb which are always adjacent, and there is longer range agreement between the “no” and “not” or “yes” and the null symbol which appear at the beginning and end of the sentence, respectively. To test whether the two types of models could capture these correlations, we created two sets of ungrammatical sentences in which either the verbs were wrong or the ending of the sentence did not match the beginning. We compared relative log likelihoods of these sentences under each model, and the results are shown in figure 4.3. Both models can discriminate the ungrammatical sentences where short range structure is corrupted, but the single HMM cannot discriminate the cases where the longer range structure is corrupted.

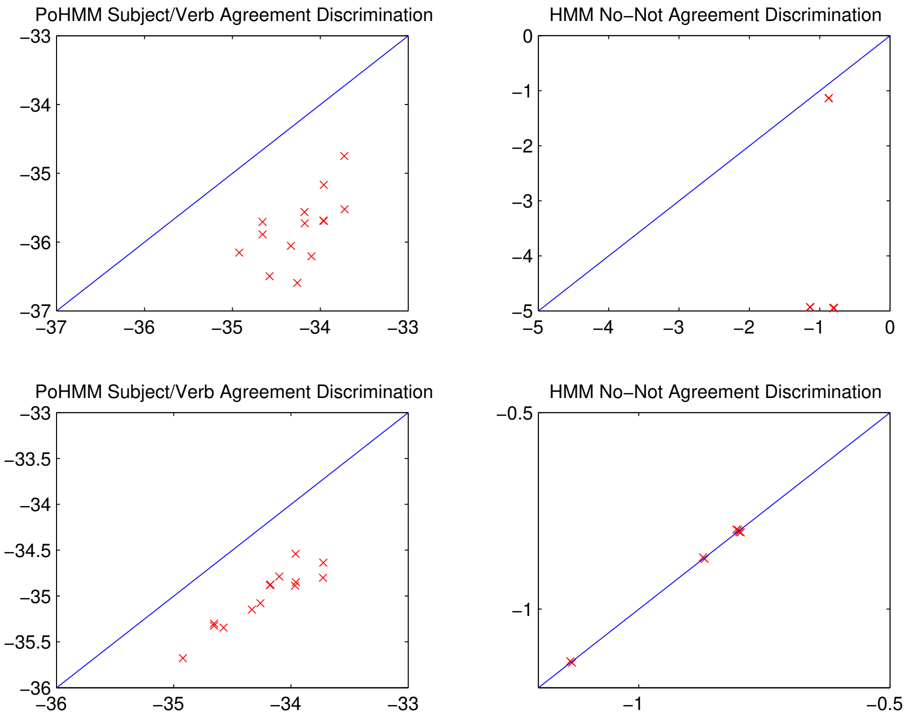

Figure 4.3: Discrimination diagrams of the correct and two incorrect sentence sets under each model. x’s below the line indicate that the model assigns higher probability to the correct sentence than the corrupted sentence. x’s on the line indicate that the model cannot discriminate the two. (Note there is considerable overlap of the x’s in the HMM plots.)

4.3.2 Modelling Character Strings

The next experiment involved modelling character strings from a corpus of English text. The problem was slightly modified to better demonstrate the advantages of a product model. Rather than training the model on a single case, or mixed case text, we trained it on data in which the characters in a particular sentence were either all upper case or all lower case. Thus there really are independent factors underlying this sequence: the binary decision of upper case or lower case and the statistics of the letters.

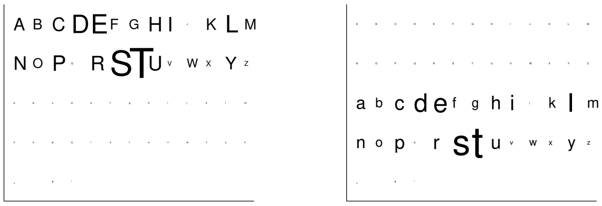

Figure 4.4: An 'eye-chart' diagram of the output distributions of the 2-state HMM in the PoHMM. Each chart corresponds to a single state's output distribution and the size of each symbol is proportional to the probability mass on that symbol.

We used 8600 sentences[1] and converted them to all upper and all lower case to yield over 17,000 training sentences. 56 symbols were allowed: 4 symbols for space and punctuation, 26 upper and 26 lower case letters. We compared a single HMM with 32 hidden states against a product of a 2 state and a 30 state hidden Markov model. In the product model the 2 state HMM learns to differentiate upper and lower case. It "votes" to put probability mass on the upper or lower case letters respectively (Fig. 4.4), and it enforces the continuity through its transition matrix. We can see also that it does a little more than just learn upper or lower case, it puts more density on common letters like 'S', 'T', 'D' and 'E'. So it is capturing some of the structure in character frequencies as well as the upper and lower cases. Then the 30-state HMM need only learn the case-independent statistics of the characters and the fact that the upper and lower case characters are analogous, by placing proportional amounts of probability mass on the two halves of the symbol set. In Fig. 4.5 we see an example of some of the big HMM's 30 hidden states. Its output distributions are symmetric over the upper and lower case letters (except for states which have learned to represent punctuation or numbers shown by a hash symbol), indicating that it has left the modelling of case to the smaller 2-state HMM model.

By contrast, the single HMM has to partition its data space into two parts, one each for upper and lower case. In effect it has to model the caseless letter statistics with a much

[1] from Thomas Hardy's "Tess of the d'Urbervilles" available from Project Gutenberg (http://www.gutenberg.net)

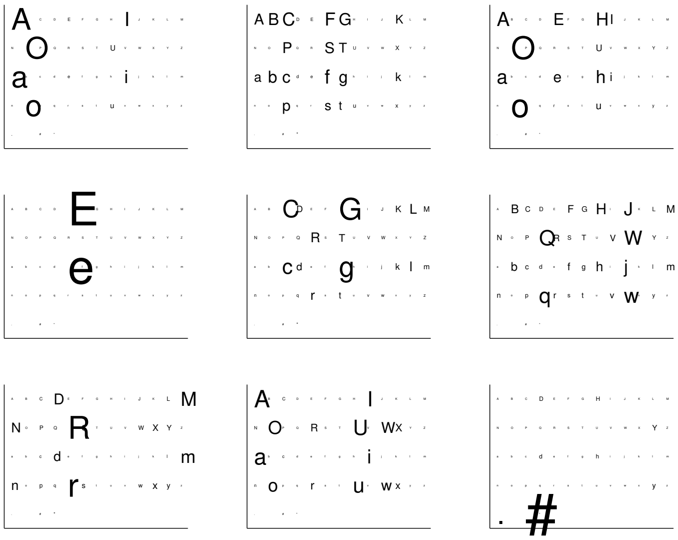

Figure 4.5: Eye-chart diagrams of the output distributions of nine of the states of the 30 state HMM. They put proportional mass on the same letters in upper case (top half of the plot) and lower case (bottom half of the plot).

smaller number of hidden states. This can be seen in Fig. 4.6a) where the observation distributions of the 32 states fall into 3 categories: punctuation, upper case, and lower case. Similarly we can see in the transition matrix (Fig. 4.6b) that the upper case states only transition to upper case states and likewise for the lower case states.

While we cannot easily compute the log likelihood of a string under the PoHMM we can compute the probability of a single symbol conditioned on the other symbols in a sentence. This leads to a simple, interesting test of the models which we refer to as the "symmetric Shannon game". In the original Shannon game, a prediction of the next symbol in a sequence is made given the previous N symbols. In the symmetric Shannon game the model is given both past and future symbols and is asked to predict the current one. We can compute this distribution exactly since we need only normalize over the missing symbol and not all strings of symbols. For models based on directed acyclic graphs, such as an HMM, it is easy to compute the probability of the next symbol in a sequence given the symbols so far. Somewhat surprisingly, this is not true for undirected models like a PoHMM. If the data after time t is missing, the posterior distribution over paths through each HMM up to time t depends on how easily these paths can be extended in time so as to reach agreement on future data.

Table 4.1 shows a comparison of several PoHMM models with a single large HMM. They were scored on a set of 60 hold-out sentences with an equal number of upper and lower case. The product of a 2-state and 30-state HMM with 2728 parameters, while

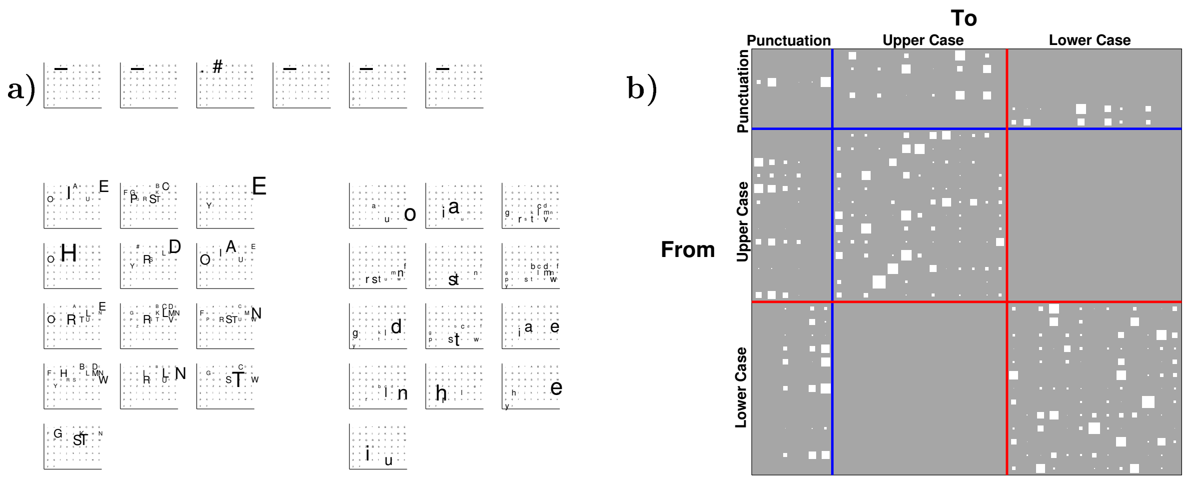

Figure 4.6: The 32 state HMM a) the observation probabilities of the HMM b) a diagram of the transition matrix where the area of the square indicates the probability of going to a state.

capturing the componential structure we were hoping for, does not outperform a single 32 state HMM which has been roughly matched for the number of parameters (2848 parameters). This is mainly an optimization problem, because if we train a 2-state model alone and a 30-state model on uni-case text, and then use their parameters to initialize the PoHMM then it does much better than the single HMM. If we use a product of many, simple HMM's then the optimization problem is eased. A PoHMM(10,4), which has still fewer parameters (2440), performs as well as a hand initialized product of 2 HMM's. Increasing, the number of HMM's in the product provides further improvements while the parameters and computation time scale linearly with the number of HMM's in the model.

Model	Sym. Shannon (bits)
PoHMM(40,4)	1.96
PoHMM(20,4)	2.06
PoHMM (10,4)	2.13
PoHMM (2-state and 30-state, pre-initialized)	2.14
32 State HMM	2.46
PoHMM (2-state and 30-state random initialization)	2.73

Table 4.1: Symmetric Shannon scores for several PoHMM models and a single large HMM on held out test data.

a	**a**	**1**
a	**b**	**2**
a	**c**	**3**
b	**a**	**4**
b	**b**	**5**
b	**c**	**6**
c	**a**	**7**
c	**b**	**8**
c	**c**	**9**

Figure 4.7: The mapping of symbols from the two sequences to the cross-product alphabet.

4.3.3 Factorial Time Series

This experiment uses a data set originally used by Williams and Hinton (1991). It is a sequence classification task in which the sequences are explicitly designed to have the kind of combinatorial structure which we expect the PoHMM to be able to find. Imagine that we have an HMM with the following structure. It has three hidden states, which it cycles through with a very small probability of jumping out of the correct sequence order. Similarly, each state outputs one of three symbols, 'a', 'b', or 'c', with a small probability of generating an incorrect symbol. We generate two sequences by starting the HMM with equal probability in each of the states. This yields two sequences of abc's which have rare anomalies and which may not be aligned. The two sequences are then combined into a new sequence whose alphabet is the cross-product of the original alphabets. The new alphabet has nine symbols in it. For this example, the mapping is show in figure 4.7. The resulting sequences now have a much larger number of symbols, and the sequence will skip through the symbols in a seemingly complicated way unless the underlying componential structure is identified. Another way of thinking of the construction is that the two HMM's represent digits in a ternary number. One controls the low order digit and the other controls the high order digit.

We can make the problem clearer by giving an example of how a product of two three-state HMM's might solve this problem. Figure 4.8 shows the parameters of a product of two HMM's trained on the factorial sequences. Both HMM's have converged to a solution where they have equal probabilities of starting in each of the three states, and they then cycle through these three states. Each state of the first HMM has allocated its emission probabilities by grouping the nine symbols into three groups of three and then putting equal mass on the first, second or third symbol of each group. The states of the second HMM have also divided the symbols into three groups of three, but each state puts all its mass equally on all the symbols in a group. We can see that these two

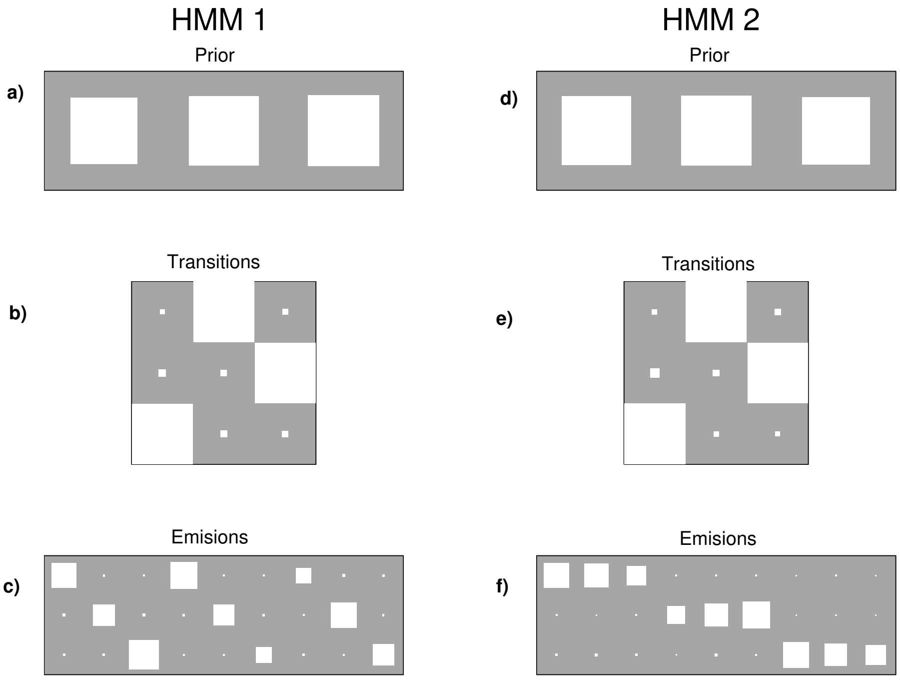

Figure 4.8: Hinton diagrams of the parameters of a PoHMM. a+d) show the initial state priors, b+e) the transition probabilities and c+f) the emission probabilities.

HMM's working together can capture the structure of the problem. One codes for the 'low order digit' and the other the 'high order digit' of our ternary number. It should be said that while this is an intuitive and optimal solution, there are other optimal solutions to the problem which the PoHMM is able to find.

To get a numerical comparison of the PoHMM and an equivalently sized HMM on data with such structure, a set of repeated experiments on a task involving the classification of two data sets with factorial structure. The first data set was generated as described above, with underlying Markov models that generate sequences of 'abc' with high probability. The second data set used underlying chains which generated 'acb' sequences with high probability. Also, the complexity of the problem was adjusted by using datasets with different numbers of underlying chains and, hence, large numbers of symbols in the observed alphabet. For example, if two chains are used then we have $3^2 = 9$ output symbols, if three chains then 27 symbols, and if 4 chains then 81 symbols.

The PoHMM's were trained using fixed step size gradient descent on the contrastive divergence objective function, and the HMM's were trained using the Baum-Welch algorithm. One PoHMM and one HMM was trained on each class of sequences. After unsupervised learning, a post-processing step was performed on the trained models. As mentioned before, we can use the unnormalized log-likelihood scores of the PoHMM (the sum of the log-likelihoods of the constituent HMM's) as input to another classifier, and treat the unknown log partition function as a bias term in the classifier. Once the two PoHMM's were trained unsupervised on the two classes of data then a set of PoHMM 'scores' were computed for each sequence in the training set and a simple logistic regression was performed to find the best linear classifier on these scores. For the HMM, if we assume that the data was generated by an HMM, then the best classification rule is to simply take the larger of the log-likelihoods of a sequence under the class 1 HMM and the class 2 HMM. However, we know that this data did not come from a simple HMM, so for symmetry of treatment, a similar post-processing classifier was fit for the HMM log-likelihoods.

Twelve experiments were performed datasets generated using 2, 3, and 4 underlying HMM's versus PoHMM's of size (2,3), (3,3), (4,3) and (5,3) were used. Against each of these PoHMM's we compared a simple HMM with an equivalent number of parameters. Tables 4.9 – 4.11 show number of parameters of the PoHMM, the number of states of the simple HMM and the number of parameters of the simple HMM in each of the 12 experiments.

Symbols PoHMM size	(2,3)	(3,3)	(4,4)	(5,4)
9	78	117	156	195
27	186	279	372	465
81	510	765	1020	1275

Figure 4.9: Number of PoHMM parameters for each experiment.

The results of the experiments are shown in the following figures. In each figure there are 12 box plots showing the results of 10 runs with 3 different sizes of problem and 4 different model sizes. In each row of the the chart 2, 3 and 4 HMM's were used to *generate* the data, and in each column PoHMM's with 2, 3, 4 and 5 HMM's were used and compared against a similar sized HMM. In all cases 1000 sequences of each class were generated. Figure 4.12 shows the error rate of the resulting classifiers and figure 4.13 shows the negative log likelihood of the classifiers on the data. We report both sets

Symbols PoHMM size	(2,3)	(3,3)	(4,4)	(5,4)
9	6	7	9	10
27	6	8	10	12
81	6	9	11	14

Figure 4.10: Number of states in the simple HMM for each experiment.

Num Symbols PoHMM size	(2,3)	(3,3)	(4,4)	(5,4)
9	96	119	171	200
27	204	288	280	480
81	528	819	1023	1344

Figure 4.11: Number of HMM parameters in each experiment.

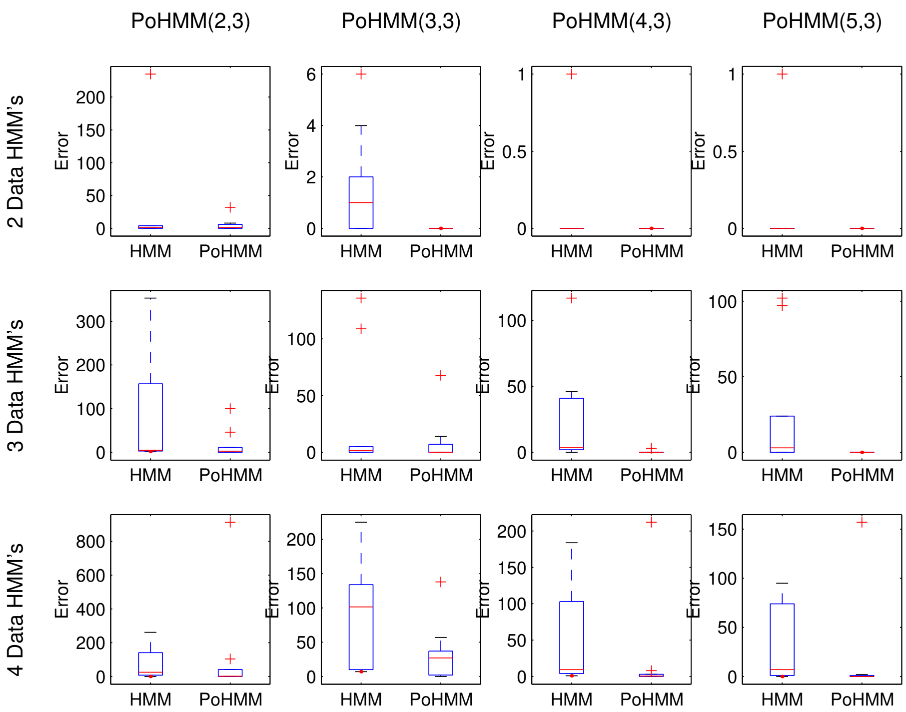

Figure 4.12: Classification error of the PoHMM and the HMM. Each graph is the result of a single experiment. Data complexity increases from top to bottom and model complexity increases from left to right.

of results because, the negative log likelihood is a real valued analog of the error rate. In addition to telling us the number of correct classifications made it tells us something about the margin of error made by the classifier — the lower the negative log likelihood the more confident the classifier is in its prediction.

In both figures we see that the HMM and PoHMM perform similarly on the data with two HMM's and 9 output symbols, but as the problem complexity increases there is a definite trend for the PoHMM to outperform the HMM in both error rate and negative log likelihood of the resulting classifier.

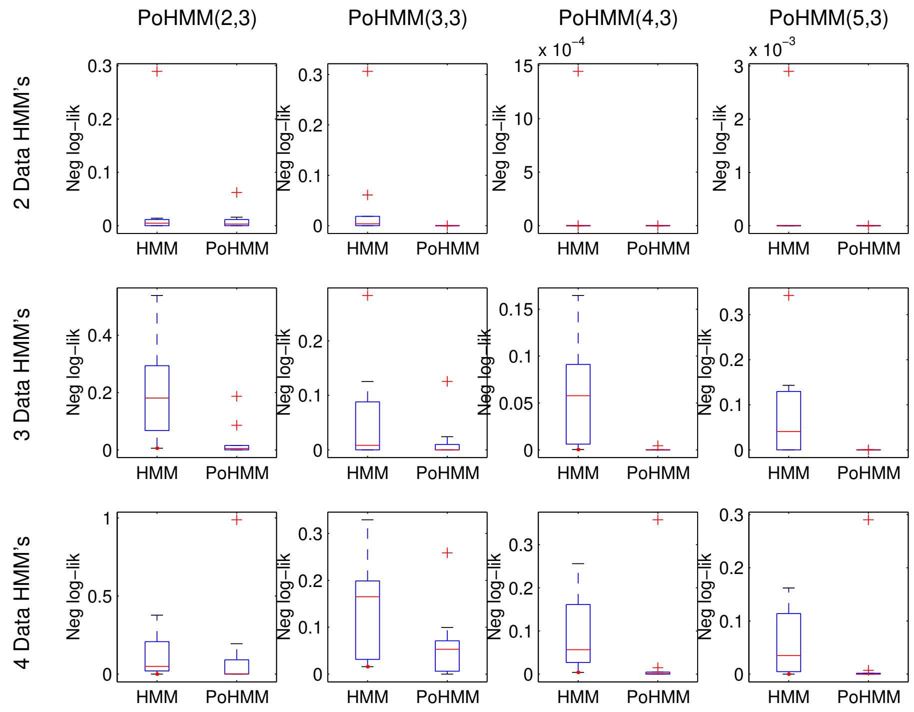

Figure 4.13: Negative log likelihood of the classifier based on the PoHMM and the HMM. Each graph is the result of a single experiment. Data complexity increases from top to bottom and model complexity increases from left to right.

4.3.4 Long Range Structure

To test the theory that PoHMM's are better able to capture long range structure in the data, a second artificial data set was generated that could easily be altered to vary the length of dependencies. Beginning with a set of strings resembling the 'abc' strings from the previous task, we then added additional , correlated, symbols at the beginning and end of each string. The correlations resembled the traditional xor problem, extended in time, using the first and second symbols of our artificial alphabet. We will call them '1' and '2'. In one class of strings if the sequence began with a '1' then it would end with '1', and if it begins with '2' then it ends in '2'. In the second class of strings, if the start symbol is a '1', then the ending symbol is a '2', or vice-versa. The intervening symbols have complex componential structure, but this structure is the same for both classes of sequences, and so it cannot be used to discriminate between them. Therefore, in order to learn a model capable of differentiating between these two classes, the model must be able to capture the long range correlations, with out being distracted by the irrelevant structure in between. The difficulty of the problem can be altered by changing the length of the intervening structure involved. The longer this middle section of the sequence, then the harder it is for the model to retain the disciriminating information about the inital symbol.

Multiple trials were performed using models of varying size and sequences of different length. Sequences of length 6, 8 and 10 were generated, and on each problem 3 different sizes of PoHMM were trained. The PoHMM's all used 5 hidden states, but the number of chains varied according to the length of the sequence. For example, if the sequence was length 6 then PoHMM's of size (6,5), (8,5) and (10,5) were used. Similarly, in the other experiments PoHMM's with M equal to the sequence length +0, +2 and +4 were used. This was done witht the belief that more chains would be required to carry information about the first symbol until the last symbol. In all cases HMM's with an equivalent number of parameters were also trained. Tables 4.14 – 4.17 show the sizes of the models and the number of parameters of the models used in each experiment. All the experiments were repeated 10 times from random initializations in order to judge the variance of the performances.

Sequence Length			
6	6	8	10
8	8	10	12
10	10	12	14

Figure 4.14: Number of 5 state HMM's in PoHMM.

Sequence Length			
6	450	600	750
8	600	750	900
10	750	900	1050

Figure 4.15: Number of PoHMM parameters in each experiment.

Sequence Length			
6	17	20	23
8	20	23	26
10	23	26	28

Figure 4.16: Number of HMM states in each experiemnt.

As with the previous experiment, after training, a simple linear classifier was trained on the 'scores' of the training data under each of the models, and the results of this classification are reported below.

Sequence Length			
6	459	600	759
8	600	759	936
10	759	936	1064

Figure 4.17: Number of HMM parameters in each experiment.

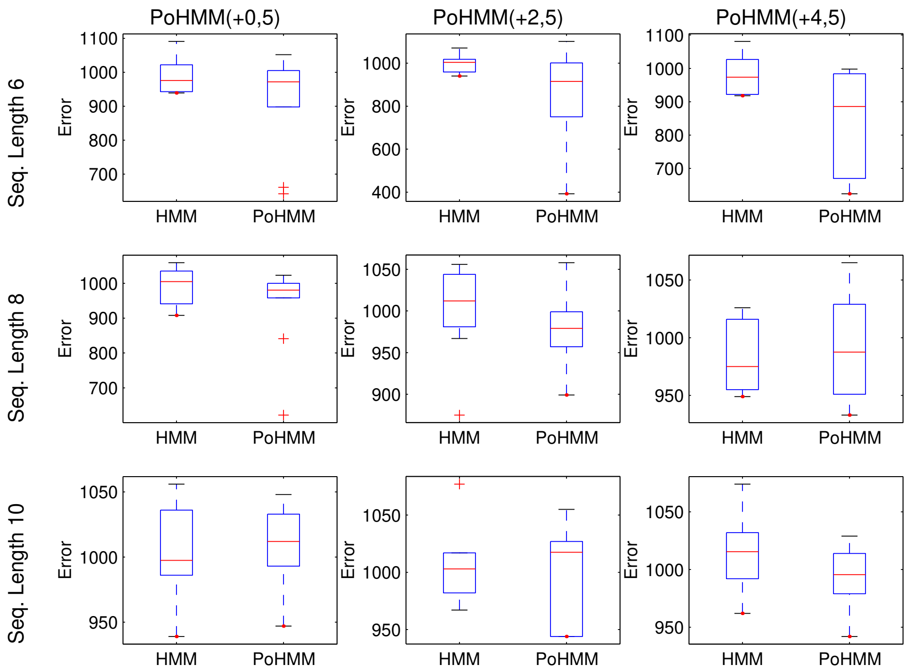

Figure 4.18: Classification error of the PoHMM and the HMM. Each row of the chart has the results of experiments with progressive data complexity from top to bottom. Each column shows progressive model complexity.

The results may be seen in the following figures. Figure 4.18 shows the error rate of the two classifiers, and figure 4.19 shows the negative log likelihood of the classifiers. From the error rates it can be seen that this is a difficult classification problem as neither classifier does much better that chance. Also, the noisy error rates don't seem to prefer one classifier over another. In figure 4.19, however, we can see a clear trend in the negative log likelihood of the two classifiers. The negative log likelihood is a real valued measure of the probability that each classifier assigns to the right class, and it seems to indicate that the PoHMM based classifier is better at assigning higher probability to the correct class, especially, as the length of the sequences grows.

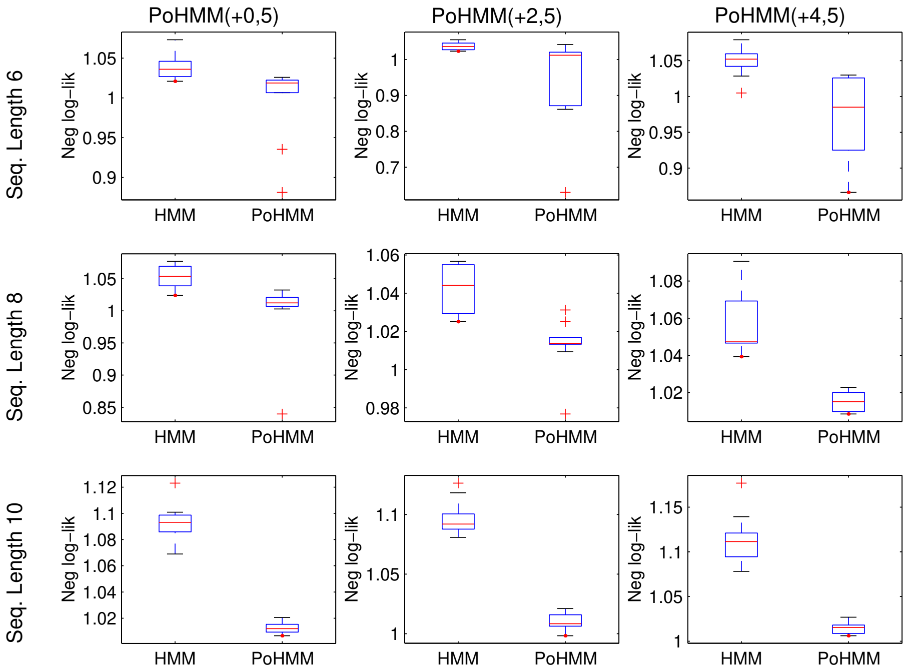

Figure 4.19: Negative log likelihood of the classifier based on the PoHMM and the HMM. Each row of the chart has the results of experiments with progressive data complexity from top to bottom. Each column shows progressive model complexity.

4.4 Discussion

In this chapter we have introduced the PoHMM, described its conditional independence properties and adapted the contrastive divergence learning algorithm to it. In addition we have presented a series of experiments which show that using a PoHMM has advantages over an equivalently sized HMM when modelling sequences with componential or long range structure. However, despite these advantages, we note that there are several additional issues which must be considered when training a PoHMM.

In some of the examples presented here, a very specific 'division of labour' must occur for the PoHMM to get the best solution and it is possible to get stuck in local minima as a result. One HMM grabbing structure too soon, can prevent other HMM's from getting the correct complementary structure. *i.e.* when using a PoHMM with a little and a big HMM in the upper-lower case character problem, if the big HMM learns to model upper and lower cases before the little HMM then a sub-optimal solution is the result. Naturally, you can use random restarts to repeat the learning and hopefully fall into the correct optimum, but in most problems its not so clear that the model has found an optimal solution. However, as demonstrated, this problem can be alleviated by using many HMM's so that there are sufficient degrees of freedom in the optimization procedure to allow it to find a path to a good solution.

On the other hand, there can be a problem in discrete output spaces when we use too many HMM's. The reconstructions can be quite good even if the output distributions are random. If we have enough random histograms to choose from then we can select a subset of these which, when multiplied together, will magnify the density on exactly the right output symbol. This is a problem, because contrastive divergence relies on information about errors in the reconstruction to create the learning signal. If the reconstructions are nearly perfect then the gradient computed by contrastive divergence will always be lmost zero. This problem can be somewhat alleviated by initializing the HMM's so that they have high entropy transition and output matrices with only enough random variation to break the symmetry, but this approach also slows learning.

Chapter 5

Discriminatively Training Networks of HMM's

5.1 Introduction

So far we have focused exclusively on unsupervised learning methods for the PoHMM. In this chapter we see how the idea of many small HMM's working in collaboration can be extended to a supervised learning framework. The following sections will present a way of discriminatively training a set of HMM's with reference to the particular application area of speech. In addition to describing a new set of architectures, we will benchmark them on a speaker classification problem.

To do classification using an unsupervised learning procedure we take labelled training data and fit a model to each of the classes we would like to discriminate. We then form a decision rule by comparing the probability of a datapoint under all the individual class models, weighted by the prior probability of each class, and assigning it the label of the model under which it scores highest. If the data comes from the same distribution as our learned class models then this decision rule will be optimal (Duda, Hart, and Stork 2001). We will refer to this as unsupervised classification, even though it is not purely unsupervised. We have given some supervision in the guise of separating the training data according to their class labels, but the fitting of the class-conditional models is done in an unsupervised way.

The problem with the semi-supervised approach is that if the modeller's assumptions about the form of the data distribution are incorrect then this decision rule may be sub-optimal. It is possible to compensate for this if we take the same class models in a supervised framework, and use both positive and negative examples of a class to modify the models so that they form better decision boundaries even if they are not well matched to their class conditional densities. For this reason we would expect a classifier trained discriminatively to outperform one which is trained in an unsupervised fashion. Having said this there is one possible catch. It is difficult to directly compare supervised and unsupervised methods. In one we are simply modelling the conditional probability of the class label, C, given the observation $\mathbf{x}$, $P(C|\mathbf{x})$. While in the other we are modelling $P(\mathbf{x}|C)$ for each value of C. We would expect a complete model of the input data to require more parameters than the conditional model of the class lables, and so in a comparison of similar sized unsupervised and supervised models the unsupervised model should do worse. To do a fair comparison, we somehow have to adjust the number of parameters in the unsupervised model to account for the complexity of the modelling task, but it is not clear how to do this. In the experiments described later in this chapter, we avoid this problem, because we are only comparing disciminatively trained models.

5.2 Discriminative Learning of HMM's

This trend has been true of applications of HMM's to speech. The original method of using HMM's for classification in speech was to train one per class using the Baum-Welch algorithm and then do classification by computing the log-probability of a speech segment under each HMM. However, the hidden Markov model is a poor model of the speech production process. Speech is generated by a complex system of articulators, which to model correctly would require a continuous dynamical system with many degrees of freedom. The HMM, at best, clusters the spectral information in each frame of speech and captures constraints on the transitions between these clusters.

More recently, discriminative learning has been used to help overcome this problem. It has been shown to give improved performance to automatic speech recognition systems based on HMM's. One method of training HMM's discriminatively used the mutual information between the density model and the class label to get a gradient of the HMM parameters which increased this objective function (Bahl, Brown, de Souza, and Mercer 1986). It was later shown by Bridle (1990) that this procedure could be viewed as a type of neural network (see figure 5.1[1]) in which the inputs to the network are the log-probability scores $\mathcal{L}(\mathbf{x}_{1:T}|\theta)$ of the sequence under hidden Markov model with parameter set θ normalized by the sequence length. In such a model there is one HMM per class,

[1]Note that this diagram shows the signal flow through the network, it it not a Bayes net diagram and does not have conditional independence semantics

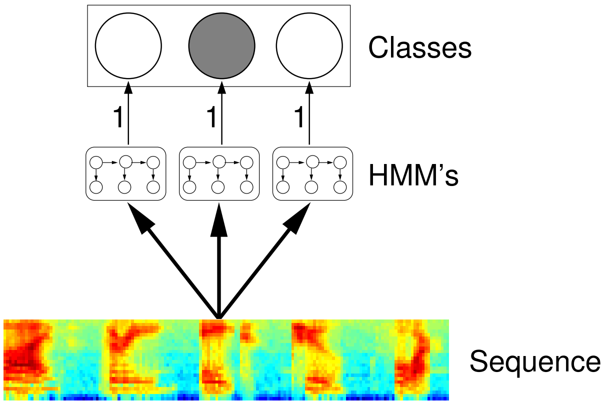

Figure 5.1: An Alphanet with one HMM per class. Each computes a score for the sequence and this feeds into a softmax output layer.

and the output is a softmax nonlinearity(Bishop 1995).

$$P(C = k|\mathbf{x}_{1:T}, \theta_1, \ldots, \theta_K) = y_k = \frac{\exp(\frac{1}{T}\mathcal{L}(\mathbf{x}_{1:T}|\theta_k))}{\sum_{j=1}^{K} \exp(\frac{1}{T}\mathcal{L}(\mathbf{x}_{1:T}|\theta_j))} \tag{5.1}$$

where $\theta_1, \ldots, \theta_K$ are the parameters of HMM's 1 through K. Bridle showed that by first training a set of HMM's using Baum-Welch and then discriminatively adapting their parameters using gradient descent the classification performance of the model could be improved (Carey, Parris, and Bridle 1991). Such an architecture has been coined an "alphanet" because it may be implemented as a recurrent neural network which computes the likelihood of an HMM by mimicking the forward pass of the forward backward procedure – computing the alpha-values.

5.3 Discriminative Training of Many HMM's

The obvious application of this same idea to PoHMM's is to replace the HMM's in the alphanet with PoHMM's.

$$P(C = k|\mathbf{x}_{1:T}, \Theta_1, \ldots, \Theta_K) = y_k = \frac{\exp(\frac{1}{T}\sum_{m=1}^{M} \mathcal{L}(\mathbf{x}_{1:T}|\theta_k^{(m)}) - Z(\Theta_m))}{\sum_{j=1}^{K} \exp(\frac{1}{T}\sum_{n=1}^{M} \mathcal{L}(\mathbf{x}_{1:T}|\theta_j^{(n)}) - Z(\Theta_n))} \tag{5.2}$$

where the function, $\mathcal{L}(\cdot|\theta_k^{(m)})$, is the log-probability of the mth HMM in the PoHMM modelling class k, and Θ_k is the set of all parameters, $\{\theta_k^{(1)}, \ldots, \theta_k^{(M)}$ in that PoHMM. Since the log-probability of a PoHMM is the sum of log-probabilities of the constituent HMM's less the log-partition function then this may be implemented as a neural net

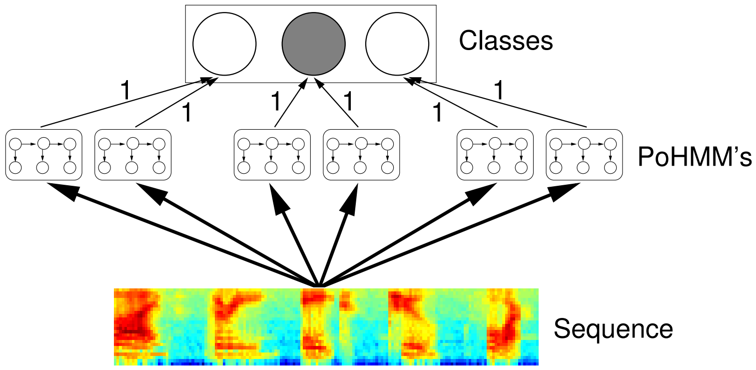

Figure 5.2: An alphanet style architecture, where we have substituted the HMM's with PoHMM's

with with a softmax output function and a fixed weight of 1 from each HMM in the kth PoHMM to the output unit corresponding to the kth class, as illustrated in figure 5.2. We can now compute the derivative of the PoHMM's, with respect to the classification objective function, by backpropagating from the output of the classifier through the network. The problem of not being able to easily compute the partition function arises, but if we add bias terms into the output units of the model then we can discriminatively estimate these as we optimize the parameters.

Even though the input to each output unit is a sum of the log-probabilities of the HMM's in each PoHMM, the product structure is retained because the softmax at the output exponentiates the input, so it is effectively multiplying together the distributions over sequences. The benefit of such an architecture is that it is no longer limited to a one-to-one relationship between classes and HMM's. We can use more than one HMM to model each class. However, when considering this architecture further it becomes clear that there is no need to partition the HMM's among the classes either. A more flexible architecture might make better use of the HMM's in the model, and for inspiration we turn to the standard feedforward neural network.

5.3.1 Backpropagation Networks as Density Comparators

A multi-layer feedforward network is usually thought of as a flexible non-linear regression model, but if it uses the logistic function non-linearity in the hidden layer, there is a different interpretation of the operation it is performing. Given a mixture of two Gaussians where we know the component priors $P(\mathcal{G})$ and the component densities $P(\mathbf{x}|\mathcal{G})$ then the posterior probability that Gaussian, $\mathcal{G}_0$, generated an observation $\mathbf{x}$, is a logistic function whose argument is the negative log-odds of the two classes (Jordan 1995):

$$\begin{aligned} P(\mathcal{G}_0|\mathbf{x}) &= \frac{P(\mathbf{x}|\mathcal{G}_0)P(\mathcal{G}_0)}{P(\mathbf{x}|\mathcal{G}_0)P(\mathcal{G}_0) + P(\mathbf{x}|\mathcal{G}_1)P(\mathcal{G}_1)} \\ &= \frac{1}{1 + \exp\left\{-\log\frac{P(\mathbf{x}|\mathcal{G}_0)}{P(\mathbf{x}|\mathcal{G}_1)} - \log\frac{P(\mathcal{G}_0)}{P(\mathcal{G}_1)}\right\}} \end{aligned} \tag{5.3}$$

If the class conditional densities in question are multivariate Gaussians

$$P(\mathbf{x}|\mathcal{G}_k) = |2\pi\Sigma|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu_k)^T \Sigma^{-1}(\mathbf{x}-\mu_k)\right\} \tag{5.4}$$

with equal covariance matrices, Σ, then the posterior class probability may be written in this familiar form:

$$P(\mathcal{G}_0|\mathbf{x}) = \frac{1}{1+\exp\{-(\mathbf{x}^T\mathbf{w}+\mathbf{b})\}} \tag{5.5}$$

where,

$$\mathbf{w} = \Sigma^{-1}(\mu_0 - \mu_1) \tag{5.6}$$

$$\mathbf{b} = (\mu_1 + \mu_0)^T\Sigma(\mu_1 - \mu_0) + \log\frac{P(\mathcal{G}_0)}{P(\mathcal{G}_1)} \tag{5.7}$$

Thus, the multi-layer perceptron can be viewed as taking pairwise comparisons between Gaussians in the input space, computing their posteriors, and combining these decisions in the output layer.

5.3.2 Relative Density Nets

This view of a backpropagation network leads us to consider substituting other kinds of density models in place of Gaussians in the input layer. In particular, we would like to introduce HMM's into the network. In the proposed network, instead of performing pairwise comparisons between Gaussians the hidden layer units perform pairwise comparisons between HMM's. For this reason we refer to it as a relative density network (RDN). For a given sequence the log-probability of of a sequence under each HMM is computed and the difference in log-probability is used as input to the unit. This is equivalent to computing the posterior responsibilities of a mixture of two HMM's with equal prior probabilities. In order to maximally leverage the information captured by the M HMM's we use $\binom{M}{2}$ hidden units so that all possible pairs are included.

$$P(C = k|x_{1:T}, \theta_1, \ldots, \theta_M) = y_k$$

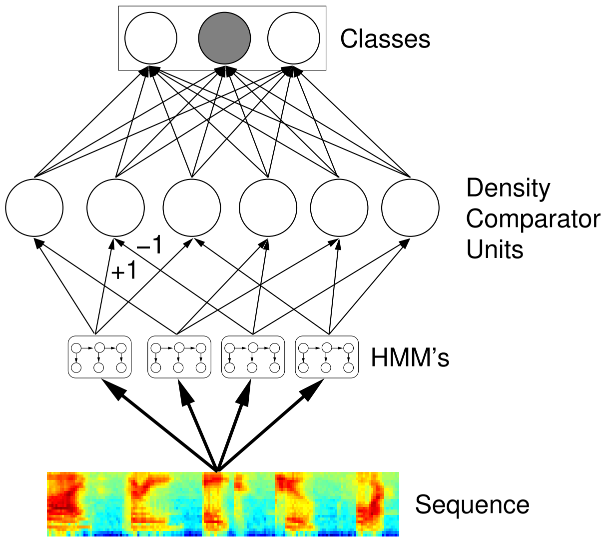

Figure 5.3: A multilayer density net with HMM's in the input layer. The hidden layer units perform all pairwise comparisons between the HMM's.

$$= \quad \text{softmax}\left(\sum_{(i,j)\in\binom{M}{2}} W_{(i,j)k}\sigma(\mathcal{L}(\mathbf{x}_{1:T}|\theta_i) - \mathcal{L}(\mathbf{x}_{1:T}|\theta_j))\right) \tag{5.8}$$

where we have used $\sigma(\cdot)$ as shorthand for the logistic function and $W_{(i,j)k}$ is the weight connecting the hidden unit which compares HMM's i and j to the output unit representing class k. The resulting architecture is shown in figure 5.3.

In contrast to the alphanet which allocates one HMM to model each class, or the PoHMM-alphanet which uses a cluster of HMM's to fit each class, this network does not require a one-to-one (or in the case of the PoHMM, a many-to-one) alignment between HMM's and classes and it gets maximum discriminative benefit from the HMM's by comparing all pairs. It makes sense to squeeze as much information as possible from the relatively expensive operation of performing the forward-backward algorithm on the HMM's. This architecture also allows us to use more HMM's than there are classes. The semi-supervised approach to training HMM classifiers was problematic because it depended on the assumption that a single HMM is a good model of the data and, in the case of speech, this is a poor assumption. Training the classifier discriminatively alleviated this drawback and the relative density net goes even further in this direction by allowing many HMM's to be used to learn the decision boundaries between the classes. The intuition here is that many small HMM's can be a far more efficient way to characterize sequences than one big HMM. Brown and Hinton (2001) show that when many small HMM's cooperate to generate sequences, the mutual information between different parts of a sequence scales linearly with the number of HMM's and only logarithmically with the number of hidden nodes in each HMM.

5.3.3 Training Algorithm

Given the definition of the RDN it is straightforward to derive the gradients for optimizing the parameters. Conditioned on the observation the probability, y_k, of a class is given by equation 5.8. If we define a vector of indicator variables t_k which is zero everywhere except in the position of the correct class, then the likelihood is

$$P(C = k|\mathbf{x}_{1:T}) = p_k = \prod_{k=1}^{K} y_k^{t_k} \tag{5.9}$$

Taking logs yields

$$\log p_k = \sum_{k=1}^{K} t_k \log y_k \tag{5.10}$$

and the derivative of this expression with respect to the predicted probabilities, y_k is

$$\frac{\partial}{\partial y_{k'}} \log p_k = \frac{t_{k'}}{y_{k'}} \tag{5.11}$$

The softmax function is defined as

$$y_k = \frac{exp(a_k)}{\sum_{j=1}^{K} exp(a_k)} \tag{5.12}$$

and its derivative with respect to the inputs is

$$\frac{\partial}{\partial a_{k'}} y_k = y_{k'} \delta_{k'k} - y_{k'} y_k \tag{5.13}$$

Now we can compute the derivative with respect to the weights in the relative density net as

$$\frac{\partial}{\partial a_k} \log p_k = \sum_{k'} \frac{\partial p_k}{\partial y_{k'}} \frac{\partial y_{k'}}{\partial a_k} = t_k - y_k \tag{5.14}$$

Having computed the derivative of the outputs with respect to the inputs we are now in a position to compute the derivative of the weights.

$$\frac{\partial}{\partial W_{(i,j)k}} p_k = a_k o_{(i,j)} \tag{5.15}$$

where $o_{(i,j)}$ is the output of the hidden unit which compares HMM's i and j. In a similar way the derivative may be backpropagated from the hidden units to the HMM's at the input. First the gradient of the likelihood with respect to the input of each sigmoidal hidden unit, $b_{(i,j)}$, is backpropagated from the softmax output units,

$$\frac{\partial}{\partial b_{(i,j)}} p_k = o_{(i,j)}(1 - o_{(i,j)}) \sum_k a_k W_{(i,j)k} \tag{5.16}$$

where $o_{(i,j)}(1 - o_{(i,j)})$ is the derivative of the logistic non-linearity of the hidden units. The gradient of the likelihood with respect to a particular HMM is just the accumulation of the gradients at the input of the hidden units of each hidden unit which includes that HMM in its comparison. If the input of each hidden unit is the difference of the normalized log-probability of HMM i and j then the gradient for a particular HMM, m, is

$$\frac{\partial}{\partial \mathcal{L}(\mathbf{x}_{1:T}|\theta_m)} p_k = \frac{1}{T} \left(\sum_{(i,j) \in \binom{M}{2}} b_{(i,j)} \delta_{i,m} - \sum_{(i,j) \in \binom{M}{2}} b_{(i,j)} \delta_{j,m} \right) \tag{5.17}$$

Now the derivative of the likelihood with respect to the parameters is just the product of these two partial derivatives

$$\frac{\partial}{\partial \theta_m} p_k = \frac{\partial p_k}{\partial \mathcal{L}(\mathbf{x}_{1:T}|\theta_m)} \frac{\partial \mathcal{L}(\mathbf{x}_{1:T}|\theta_m)}{\partial \theta_m} \tag{5.18}$$

where the calculation of the gradient of an HMM with respect to its parameters is discussed in section 2.4.3 and in Appendix A.

5.3.4 Speaker Recognition

In section 5.4 we will use a simple speech application to compare the relative density network with other architectures for discriminative HMM learning. Here we give a brief overview of the task.

Where the goal of most speech recognition systems is to recognize what is being said independent of the characteristics of the particular speaker, "speaker recognition" aims to do the exact opposite, which is to identify the parts of the speech that are characteristic of the speaker.

There are a few variations on this basic problem. First there is speaker verification versus speaker identification. Speaker verification is a two class classification problem in which we would like to decide, with high accuracy, whether or not a speaker matches the identity of a known user. Speaker identification, on the other hand, is a multi-class problem where we want to identify the speaker from a database of potentially many users. In a typical speaker recognition task we typically have a very small amount of training data from each of the users of the system. For this reason, in standard speaker recognition systems, very simple models are employed usually consisting of a single static Gaussian or a static mixture of Gaussians (Bimbot, Magrin-Chagnoleau, and Mathan 1995; Reynolds, Quatieri, and Dunn 2000).

In the experiments which follow, we will be looking at a simplified version of the speaker identification problem in which there are fewer users and we have much more data from each user, allowing us to use more sophisticated temporal speech models. One of the design choices of a speaker recognition system is whether to model the speaker saying a fixed phrase (known as text-dependent speaker recognition) or to model the speaker independent of what they are saying. In the first case we can model the time course of the utterance by putting left-to-right structure in the state transitions of the model. In the second, it is better to use an transition matrix in which every transition has a non-zero probability.

Another consideration is what type of pre-processing to use. Current methods use short time spectral analysis on frames of speech, followed by dimensionality reduction to distill the waveform into a handful of informative coefficients which hopefully capture the relevant information about the signal. These usually include the coefficients from linear predictive coding (LPC) or cepstral coefficients (Rabiner and Juang 1993) and their first differences. These techniques are intended to be a compact description of the entire spectrum of a window of speech. Such features have been applied to the speaker recognition task (Bimbot, Magrin-Chagnoleau, and Mathan 1995; Carey, Parris, and Bridle 1991; Li, Juang, Lee, Zhou, and Soong 1999; Reynolds, Quatieri, and Dunn 2000).

However, a recent trend in speech recognition has been to use features based on sub-bands of the frequency spectrum. There are several motivations for this. While ASR systems perform acceptably well under ideal speaking conditions, error rates increase dramatically at noise levels which have little effect on human speech recognition performance (Lippmann 1997). It has been observed that much of relevant information about a segment of speech is replicated in sub-bands across the frequency spectrum. Multi-band speech analysis treats sub-bands of the frequency spectrum as independent sources of information, and learns models on each sub-band. The class decisions of each of these sub-band models are then combined by various schemes, the simplest being a majority vote, to arrive at an aggregate decision for the class membership of a speech segment. Also, naturally occurring noise is frequently limited in frequency. It is clear that in the face of band-limited noise this system will be more robust than an ASR system which uses whole-spectrum templates as input features (Bourlard and Dupont 1997; Besacier and Bonastre 2000). In addition, a multi-band approach may be well matched to the way that humans recognize speech. Psychophysical evidence shows that human recognition performance is insensitive to frequency selective distortions (Allen 1994).

The relative density net seems well matched to a multi-band approach to speech analysis. Unlike the alphanet, in which one HMM must model all the structure in the speech, the HMM's in the RDN can learn to pay attention to particular structure in the spectral features by adjusting the precision of the Gaussian output densities. For these reasons we would like to use features which maintain the frequency structure of the speech, allowing the HMM's to specialize on particular parts of the frequency or time domain. When computing the cepstral features, the cosine transform is applied, and this blurs this spectral structure among the cepstral coefficients.

5.4 Experiments

The experiments consist of a series of comparisons between the RDN and a standard alphanet. In addition several control experiments were performed on architectures related to the RDN. The first of these uses a two layer decision network, like the RDN, but the weights of both layers are adjustable parameters, unlike the RDN. Similarly, a network with a single layer of weights between the HMM's and the output units was tried. Finally, a network similar to an alphanet, but with the HMM replaced with an equivalent sized static mixture of Gaussians (MoG) model was used. This was done to verify that the temporal aspect of the model was indeed useful. These three networks will be referred to as the 2-Layer MLHMM, 1-Layer MLHMM and MoGnet respectively.

5.4.1 Database

The data for the experiment was taken from the CSLU Speaker Recognition data set (Cole, Noel, and Noel 1998), which contains multiple sessions of the same speakers saying the same phrases several times per session. We chose to use recordings of 12 different speakers uttering 6 different sequences of 4 digits such as "5-3-8-4-2". Each speaker repeated the utterance four times over the course of twelve sessions for a total about 288 recordings per speaker (although not every speaker had a full complement of recordings). For each speaker the recordings were split into 10 distinct sets.

The recordings are telephone quality speech, and one of the interesting things about the data is that no effort was made to control the telephone used for the recordings or the ambient noise conditions. This makes the data particularly realistic and challenging for benchmarking speaker recognition systems. However, in processing the data we randomized the order of the recordings before generating the 10 splits of the data, so that there was a high probability that, for a given test recording a recording from the same session appeared in the training data. This ensured that the sequences used for testing contained recordings taken under circumstances at least similar to those appearing in the training data. This was done to decrease the variance of the algorithms' performance on the test data and provide a more accurate comparison of relative performance of the two methods applied to this problem.

The amount of data used here is large for a realistic speaker recognition problem. Normally, speaker recognition systems have many more speakers and few training utterances with which to fit the model, and much simpler models are used. Also, a purely supervised approach makes it difficult to add new speakers to the system. These experiments are simply a proof of concept, showing the relative advantage of the RDN architecture over a standard model.

5.4.2 Pre-Processing

As mentioned earlier, the application of networks of many HMM's to speech was partly motivated by the natural analogy to multi-band spectral methods, and for this reason we would like to stay as close to the original spectrogram as possible. So in our preprocessing we do not do the rotation of the cosine transform which is standard practice to get the cepstral coefficients. Instead we simply filter the original signal with a small set of band-pass filters.

The recordings were sampled at 8kHz, pre-emphasized, and then Fourier transformed into spectral frames using a Hanning window 256 samples in length (32ms) at a framerate of 80 samples (10 ms). The frequency of the speech is then non-linearly scaled. This is a standard processing technique which reflects the psychological evidence which shows that human perception of pitch is logarithmically related to the actual pitch. This means that a pitch difference at low frequencies is more noticeable than the same pitch change at higher frequencies. Two re-scalings which are typically used are known as the Mel and the Bark scales(Rabiner and Juang 1993). In this case we used the Mel-scale frequency transformation. The frames were then filtered using 24 triangular, band-pass filters.

5.4.3 The Models

Four experiments were performed comparing the 5 different kinds of networks described above. In each experiment an RDN was compared against the other 4 networks, with the structures chosen so as to match for the number of parameters as closely as possible. For the alphanet and the MoGnet the number of HMM's and the number of output units is fixed to be equal to the number of classes, so the only adjustable variable is the number of hidden states in each HMM or MoG. For the 1-Layer MLHMM, we matched the number of HMM's to the number used in the RDN, and adjusted the number of states so that the models were roughly the same size. Finally, for the two layer MLHMM, we used the same number and size of HMM's as the RDN, but adjusted the number of hidden units in the decision network. The RDN has a large number of hidden units $M(M-1)$, but the first layer of the network does not have adjustable parameters. The 2-Layer MLHMM, has fewer hidden units to account for the fact that it has more free parameters. Each of the experiments was repeated 10 times using 9 of the data subsets for training and one subset for testing. In each case we compared an RDN using many 4-state HMM's to an alphanet where the number of states was selected so as to match the two models for the number of parameters. The configurations are summarized in Table 5.1. For each of the architectures used in the experiments the table lists the number of HMM's in the network, the size of the relevant parameter — the number of states in the HMM or MoG, or the number of hidden units in the case of the 2-layer MLHMM — and finally the number of parameters in the model.

	RDN			Alphanet			MoGnet		
	HMM's	States	Params	HMM's	States	Params	MoG's	States	Params
Expt. 1	12	4	3348	12	5	3240	12	6	3468
Expt. 2	16	4	4844	12	7	4704	12	8	4620
Expt. 3	20	4	6532	12	9	6264	12	11	6348
Expt. 4	24	4	8412	12	12	8784	12	15	8652

	2-Layer MLHMM			1-Layer MLHMM		
	HMM's	# Hidden Units	Params	HMM's	States	Params
Expt. 1	12	32	3356	12	5	3396
Expt. 2	16	50	4854	16	5	4524
Expt. 3	20	69	6529	20	6	6852
Expt. 4	24	90	8430	24	6	8220

Table 5.1: Network configurations

In all cases the networks were optimized using up to 90 iterations of a conjugate gradient learning procedure. To initialize the class-specific HMM's in the alphanets and

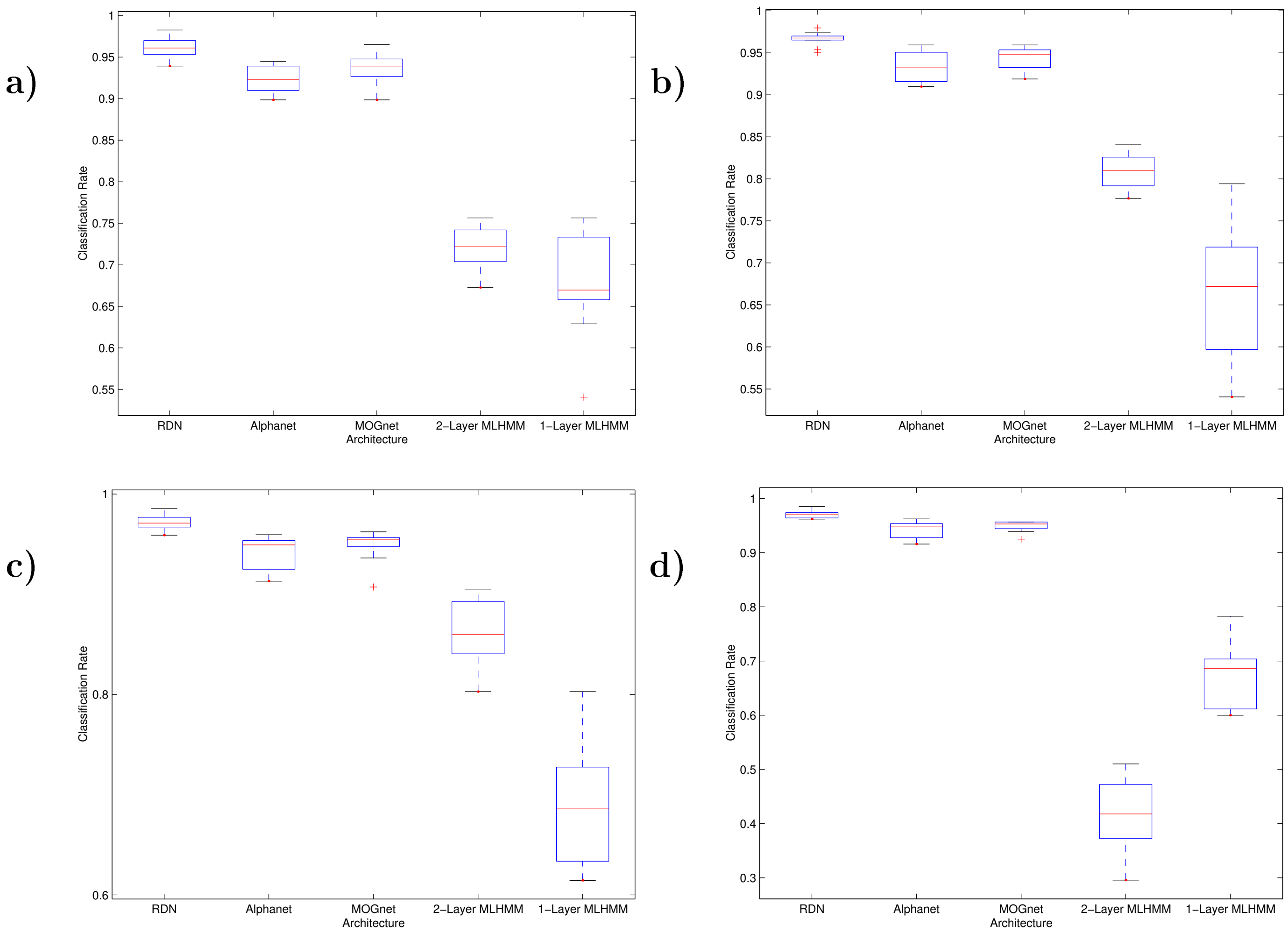

Figure 5.4: Boxplots of the test set classification rate of the experiement using: a) a 12 HMM RDN, b) a 16 HMM RDN c) a 20 HMM RDN and d) a 24 HMM RDN

the mixture of Gaussians network we did hard clustering using the k-means algorithm on data from the relevant class to get the means of the output models and then fitted a diagonal, non-isotropic covariance matrix to each cluster to get the initial variances of each output model on each data dimension. The RDN and the other two multi-layer networks were initialized by running the k-means algorithm with random restarts on all of the data. The transition matrices of the HMM's were initialized to a full matrix with probabilities perturbed from uniform by small random values.

5.4.4 Results

Figure 5.4 shows bar graphs of the relative performance of the two algorithms. In all the experiments there is a statistically significant improvement in using an RDN network as opposed to an alphanet with the same number of parameters. In all four experiments, in a paired t-test, the RDN's classification performance on the out of sample test data was superior to that of the alphanet at the 0.01 level of significance.

Somewhat worringly the MoGnet seems to perform similarly well to the alphanet with the same architecture and a mixture of Gaussians substituted for the HMM. This would seem to indicate that the shape of the spectral features is more important than the actual temporal ordering of the features. Many approaches to automatic speaker recognition treat the speech features as unordered i.i.d. samples (Li, Juang, Lee, Zhou, and Soong 1999; Besacier and Bonastre 2000; Reynolds, Quatieri, and Dunn 2000), however, these applications typically focus on very short segments of recorded speech. It was felt that because we are using relatively long speech samples in this experiment the temporal structure might prove important. The fact that we have chosen text independent speech samples (*i.e.* each speaker says several different sentences, although all the speakers say the same sentences) might counteract any consistency in the temporal ordering of the features. Nevertheless, this does not refute the basic result that it is better to perform classification use many simple density models relative to a single complex density model to fit each data class.

Also, the performance of the 1-layer and 2-layer MLHMM networks is dissappointing, as we expected these networks to perform similarly to the RDN. The numerical optimization of these architectures seems to encounter difficulties compared to the other models. There is no conclusive explanation for this, although one possibility is that the large negative outputs from the HMM's cause scaling problems in the search space. Because the hidden units of the RDN are hard coded to compute the difference of log-likelihoods between two models the input to these units will be smaller than the absolute log-probability. Also, the meaning of the model parameters are quite different. The architecture might be more sensitive to a change in the weights of the output layer than it is to a change in the parameters of an HMM resulting in very small gradients in some directions. Further work needs to be done to verify what causes the poor performance in optimization.

The experiments suggest that when doing discriminative training, there is an advantage to using many small HMM's as opposed to one HMM per class. In addition to the performance improvement, there may be a slight speed benefit to using many smaller HMM's compared to a few big ones. Computing the probability of a sequence under an HMM is order $O(TK^2)$, where T is the length of the sequence and K is the number of hidden states in the network. Thus, smaller HMM's can be evaluated faster. However, this is somewhat counterbalanced by the quadratic growth in the size of the RDN's hidden layer as M increases.

While the results are encouraging, further work needs to be done to really understand how this method is working. First of all it would be useful to understand whether the small HMM's really are specializing on isolated features of the spectrogram. One way of analyzing this would be to redo the above experiments, but first corrupt sub-bands of the spectral features in the training and testing sets with noise. If there is an advantage to the RDN then it may be able to generalize better to the corrupted test data. Alternatively, it would be nice if there were a reliable way of measuring exactly what time-spectral features of a signal a particular HMM responds to.

Chapter 6

'Spiking' Boltzmann Machines

In this chapter we will introduce another product model for time series, but one which has a different form than the PoHMM. The model is based on a form of product model for static data which uses binary hidden units called a restricted Boltzmann machine (RBM)(Hinton 1999). We will look first at the definition of an RBM before explaining the particular way in which we propose extending it to dynamic data.

6.1 The Restricted Boltzmann Machine

The graphical model of an RBM is illustrated in figure 6.1. It is a bi-partite graph with the bottom row of vertices representing the visible variables, and the top row representing the hidden variables. All the variables are binary and we will denote a visible variable, v_i, and a hidden variable h_j. The probability of a configuration of the network is defined in terms of an energy function which is the same as the Boltzmann machine energy function (see section 2.1.4). First we will establish some notation. We use $\mathbf{s}$ to represent a particular configuration of all the units in the network, and $\mathcal{S}$ to indicate the space of all such configurations, hence $\mathbf{s} \in \mathcal{S}$. Similarly, we will use $\mathbf{h}$, $\mathbf{v}$, $\mathcal{H}$ and $\mathcal{V}$ to represent particular configurations and the space of configurations for the hidden and

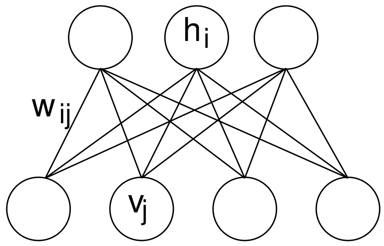

Figure 6.1: Connectivity of a restricted Boltzmann machine

visible variables respectively. Therefore $S = \{\mathcal{V}, \mathcal{H}\}$. Using this notation the probability function for the network becomes:

$$P(\mathbf{s}) = \frac{1}{Z} \exp\big(\sum_{ij} w_{ij} h_i v_j\big) \tag{6.1}$$

where w_{ij} is the weight connecting hidden unit i and visible unit j.

Unlike the Boltzmann machine which allows all units to be connected to each other (except for self-connections), the connections in an RBM are only between the two layers. This restriction on the architecture simplifies the distribution which it represents, because the distribution is now separable in terms of the hidden variables (and, by symmetry, in terms of the visible variables). If we separate the distribution of the visible variables in terms of the hidden variables and then sum them out the product nature of the RBM is made more explicit:

$$\begin{aligned} P(\mathbf{v}) &= \sum_{\mathbf{h}} P(\mathbf{s}) \qquad (6.2) \\ &= \frac{1}{Z} \sum_{\mathbf{h}} \exp\left(\sum_{ij} w_{ij} h_i v_j\right) \\ &= \frac{1}{Z} \sum_{\mathbf{h}} \prod_{i} \exp\left(\sum_{j} w_{ij} h_i v_j\right) \\ &= \frac{1}{Z} \prod_{i} \left(\sum_{h_i} \exp\left(\sum_{j} w_{ij} h_i v_j\right)\right) \end{aligned}$$

where the third step in the derivation relies on the fact that there are no dependencies among the hidden variables. It is now clear that the RBM is a product distribution with each hidden unit in the network defining its own probability distribution over the visible variables.

Given that the RBM is a product model, we may use the contrastive divergence learning rule to improve the likelihood of the model (section 3.2.1). The algorithm works in the same way: upon presentation of an example pattern we sample from the hidden units conditioned on that example, compute the positive gradient, re-sample the visible units conditioned on the hidden units, and finally compute the negative gradient. Sampling in an RBM is similar to sampling in a Boltzmann machine. The conditional probability of a hidden unit conditioned on the visible units is just the logistic function of the total input from the visible units:

$$P(h_i|\mathbf{v}) = \sigma(\sum_j w_{ij} v_j) \tag{6.3}$$

The gradient of an RBM is much simpler than an HMM, requiring only the pairwise sufficient statistics of the hidden and visible units, yielding the following update rule:

$$\Delta w_{ij} \propto < h_i v_j >_{Q^0(\mathbf{v})} - < h_i v_j >_{Q^1(\mathbf{v}|W)} \tag{6.4}$$

where $< \cdot >_{Q^0(\mathbf{v})}$ and $< \cdot >_{Q^1(\mathbf{v}|W)}$ are the expectations under the empirical distribution and the one-step reconstruction distribution respectively.

6.2 Temporal Extension of the RBM

One of the ways of extending a graphical model such as this to account for dynamic data is simply to 'spatialize' time, which means to treat a block of observations occurring across many time steps as a single observation and present them all at once to the model. Unfortunately, this approach loses the temporal continuity between time steps. For example, in modelling images, each hidden unit, i, of the RBM would have a weight, w_{ij}, to each of the pixels in the image. In order to model a sequence of images, then we have a weight to each pixel at each time, w_{ijt}. The problem with this approach is that it does not directly address the temporal relationship between pixels adjacent in time; w_{ijt} and $w_{ij(t+1)}$ are entirely different parameters. We could reorder time in some arbitrary way and present all our image sequences in that order, and the model would work equally well[1]. For the problem of modelling images, in particular, we would like to capture the idea that changes in pixel intensity from one moment to the next tend to be smooth. Also, if we spatialize time in this way then the hidden units have no temporal meaning. They do not indicate when an event occurred. They are capable of representing a pattern, but not the temporal relationship among the patterns.

Here we introduce a model which addresses these two problems. First, we would like the model to have dynamics in the hidden units. This gives a temporal ordering to the signals which the hidden units are representing. Secondly we would like the model to incorporate something of the smoothness inherent in the data. To do this we propose

[1]We could equally well make the same argument about the spatial relationships among the pixels. For this reason, image models which use spatial filters, or weight sharing are often used. We will not address this issue in the model presented in this chapter.

introducing a low-pass filter which smooths out the binary activations of the hidden units. This model has its motivations in a problem from neuroscience.

6.2.1 Coding Probabilities Using Neuron Spike Times

One of the unsolved puzzles of neuroscience is the code used by neurons to communicate with one another using spikes. It is commonly believed that neurons are communicating real values to one another which are encoded in one of two ways: through a rate code which depends on the instantaneous rate of the spikes being sent, or through the precise time of the spikes which are transmitted. While the rate code theory is thought to be the simpler of the two coding schemes, because most proposals for codes based on spike timing require complicated mechanisms for the neurons to measure the intervals between spikes, the theory proposed in Hinton and Brown (2000) does not require such complicated mechanisms to measure the value represented by the timing of a spike. The essential observation of this model is that while a spike is an extremely short event it can have a temporally extended effect on its neighbours via its excitatory post-synaptic response (EPSP). The initial rise time of an EPSP after a spike is quick, but it has a slowly decaying tail which can indicate something about how long ago a spike fired. Moreover, the EPSP is cumulative, so that a neuron which emits a rapid series of spikes can build up the post-synaptic effect which it has on neighbouring neurons.

This gives us a mechanism for measuring spike timing. Next we propose the semantics of spike timing. We imagine that the brain is trying to infer properties about the world from the raw sensory input that it receives and that the neurons are encoding, in some feature space, the current beliefs about these inferred properties. If each neuron represents a point in that feature space, and the EPSP encodes the strength of that neuron's belief about the accuracy of the value it is representing then a population of neurons could precisely specify a value by combining their predictions of the value, weighted by the strength of their beliefs. In a probabilistic setting, each neuron represents a mean value and the strength of its EPSP is the precision of its distribution about the mean. A neuron which is silent, has very low precision and hence a very broad distribution about its mean. A neuron which is rapidly spiking builds up its post-synaptic potential and specifies a very sharp distribution about its mean. If a population of such neurons combine their distributions multiplicatively then the combined distribution will have a mean value which is the average of the means, weighted by the precision with which each neuron specifies its value. Figure 6.2a shows how energy functions which are combined additively in the logarithmic domain are multiplied in the probability domain and the

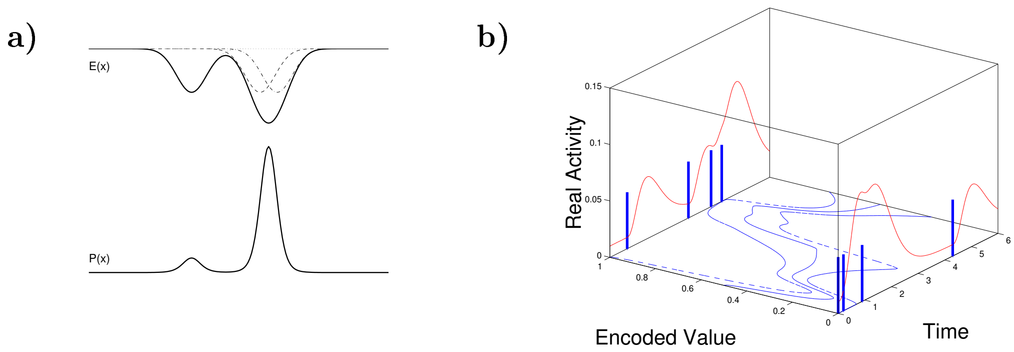

Figure 6.2: a) Flat tailed functions combined additively in the logarithmic 'energy' domain combine to form sharply tuned distributions in the probability domain. b) A schematic of how a population code might represent the evolution of a distribution over a real valued quantity. (see text)

resulting distribution can be more precise than either of the original probability distributions. Contrast this with the behaviour of a mixture model in which distributions are combined additively in the probability domain, and the resulting distribution will never be sharper than the sharpest component.

Figure 6.2b illustrates how this might work through time. Imagine, two neurons which each code for a value on the real axis: 0 and 1. Inactivity corresponds to each of them having a vague distribution. When a neuron spikes (shown by an impulse in the vertical plane) its EPSP rises (shown as a smooth curve in vertical plane). This increases the sharpness of the distribution (shown in the horizontal plane) and pulls the prediction toward the spiking neuron. As the EPSP decays then the precision of the predicted value relaxes to its rest position. The ratio of firing between the two neurons controls the mean of the distribution and the magnitude of the firing rates controls the precision about the mean.

The advantages of this scheme are twofold. First, it has the nice property that a neuron need only fire when it has information to convey; uncertainty is represented by silence. Second, the incorporation of the smooth post-synaptic response allows spikes to encode real valued probabilities. The next two sections develop a model which implements this scheme as a restricted Boltzmann machine with the addition of a smoothing kernel to achieve the desired temporal behaviour.

6.2.2 The Spiking Boltzmann Machine

While the idea of spikes modulated by smooth temporal response functions is an appealing biological model, for the reasons described above, it is also an interesting model for computational reasons. In this section we introduce the spiking Boltzmann machine (SBM), an architecture analogous to the one just described, but which defines a probability distribution over binary inputs, using an extension of the restricted Boltzmann machine. The main difference is in the interpretation of uncertainty in the two types of distributions. In the schematic diagrams above (figure 6.2) we illustrated the principle of using the smooth response function to represent precision in a Gaussian, because the mean and variance of a Gaussian are clearly distinct, but in an RBM the mean and variance are related, so that maximum uncertainty also corresponds to zero mean.

The SBM differs from a spatialized RBM in a couple of ways. First, unlike a spatialized model of time, the hidden units have a temporal instantiation; we unroll them in time, and denote the activity of hidden unit i at time t as h_{it}. Similarly, the visible units are denoted v_{jt}. Second, the visible units at each time step influence the hidden units, not only at the same time step, but also at succeeding time steps, and they influence them in a specific way. There is a weight, w_{ij}, from each visible unit, j, to each hidden unit, i, but this is not time dependent. The effect of a hidden unit at time t on a visible unit at time j is modulated by a smooth temporal kernel $r(t)$, whose form is shown in figure 6.3. $r(t)$ is zero at $t = 0$ so that that the effect of the current input is not felt until one time step later. It has a sharp rise time and a slow decay. Also, we assume $r(t)$ is a causal filter, meaning that its response is zero for all times prior to $t = 0$. Thus, we

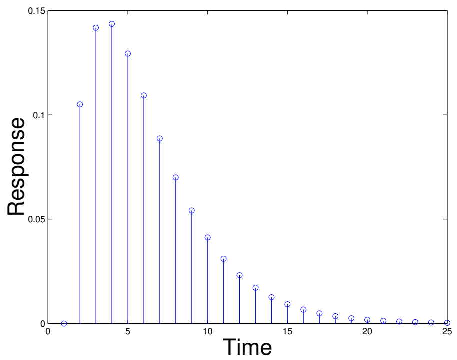

Figure 6.3: The form of the temporal kernel.

compute the input from the visible layer to the hidden layer by first convolving the time course of the visible units with r and then multiplying by the matrix of weights:

$$P(h_{it} = 1|\mathbf{v}_{1:T}) = \sigma(\sum_j w_{ij}\tilde{v}_{jt}) \tag{6.5}$$

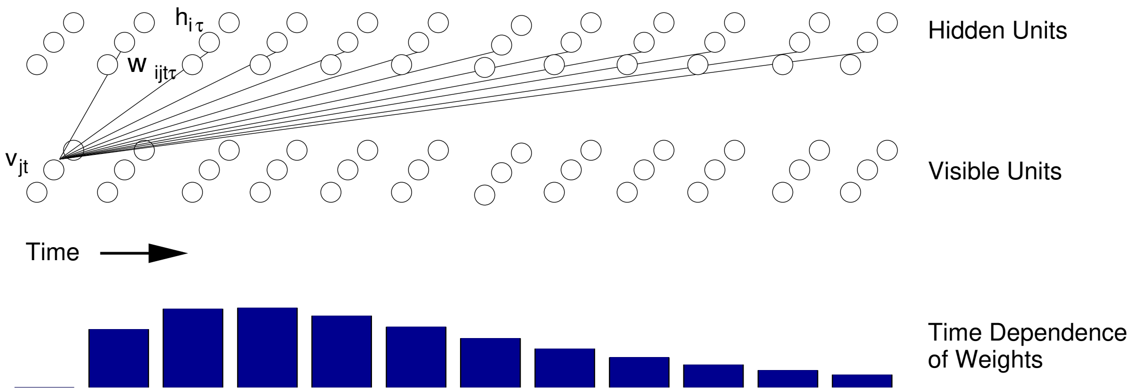

Figure 6.4: The graphical model for a spiking belief network unrolled in time. The bar chart beneath the diagram shows how the influence of visible units on the hidden units is modulated by the temporal kernel. (We show the weights over time for a single visible unit and a single hidden unit.)

where $\tilde{v}_{jt}$ is the convolved input signal:

$$\tilde{v}_{jt} = \sum_{\tau=0}^{T} r(\tau) v_{j(t-\tau)} \tag{6.6}$$

We assume that the signals are finite and zero-pad them beyond their ranges.

Another way of thinking of the combined effect of the weights and the temporal kernel is that each visible unit $v_{j\tau}$ is connected to hidden unit h_{it} at succeeding time steps by a time dependent weight $w_{ijt\tau} = w_{ij} r(t-\tau)$. Then equation 6.5 may be equivalently written as:

$$P(h_{it}|\mathbf{v}_{1:T}) = \sigma(\sum_{j\tau} w_{ijt\tau} v_{j\tau}) \tag{6.7}$$

This interpretation of the SBM is illustrated in figure 6.4.

We can see from this figure that the SBM is still an RBM. It is a bi-partite graph, with connections between each of the visible units and hidden units at later time steps. We can apply the contrastive divergence learning rule as with the RBM, with one caveat. We must be careful in the reconstruction step to use the same weight from the hidden to the visible unit, that was used going up from the visible to the hidden. This means that the top down input is not the posterior sample, convolved with $r(t)$, but the posterior sample *cross-correlated* with $r(t)$. The difference between the two operations is simply a sign change in the time shift summation.

$$P(v_{jt} = 1|\mathbf{h}_{1:T}) = \sigma(\sum_{i} w_{ij} \tilde{h}_{it}) \tag{6.8}$$

where $\tilde{h}_{it}$ is computed as:

$$\tilde{h}_{it} = \sum_{\tau=0}^{T} r(\tau) h_{i(t+\tau)} \tag{6.9}$$

Note the change of sign as opposed to equation 6.6. This ensures that the weights connecting each hidden unit with each visible unit are symmetric, however it has the flaw that the reconstruction step is non-causal, because it depends on knowing the future values of the hidden states in order to compute their influence on the current visible unit's reconstruction. However, we only require knowledge of the future up to the width of the temporal kernel.

The SBM contrastive divergence learning rule is similar to that of the regular RBM. The RBM weight update rule in equation 6.4 is the difference in expected correlations of the hidden and visible units under the data and the one step reconstruction distribution. The SBM is the same expectations taken between the hidden and visible units at different time lags, weighted by the temporal kernel:

$$\Delta w_{ij} \propto \sum_{t=0}^{T} \sum_{\tau=0}^{T} r(\tau) \left(\langle v_{j(t-\tau)} h_{it} \rangle_{Q^0(\mathbf{v}_{1:T})} - \langle v_{j(t-\tau)} h_{it} \rangle_{Q^1(\mathbf{v}_{1:T}|W)} \right) \tag{6.10}$$

As with contrastive divergence in the RBM, we approximate the expectations in the learning rul by taking samples from the relevant distributions. The steps of the SBM learning algorithm are as follows:

1. Convolve the input signal with $r(t)$ and compute the posterior probability of the hidden units using equations 6.5 and 6.6.

2. Sample from the posterior of each hidden unit at each time step.

3. Convolve the posterior samples with the reverse filter and compute the one-step reconstruction distribution as in equations 6.8 and 6.9.

4. Sample from the reconstruction distribution of the visible units at each time step.

5. Once again sample from the posterior distribution of the hidden units conditioned on one-step reconstruction of the visible units.

6. Compute the weight update using equation 6.10, substituting samples for the expectations.

6.2.3 Additional Issues

One problem with the SBM as it is described above is that hidden units at one time step have no memory of their states at previous time steps; they only see the data. If we were to add undirected connections between hidden units at different time steps, then the architecture would return to a fully connected Boltzmann machine in which the hidden units are no longer conditionally independent given the data. A useful trick borrowed from Elman nets is to allow the hidden units to see their previous states, but to treat these observations like data that cannot be modified by future hidden states (Elman 1990). Thus, the hidden states may still be inferred independently without resorting to Gibbs sampling. The connections between hidden layer weights also follow the time course of the temporal kernel. They act as a predictive prior over the hidden units. It is important to note that these forward connections are not required for the network to reconstruct a sequence, but only for the purposes of extrapolating into the future. Now the posterior probability of a hidden unit given the states of the visible units is,

$$P(h_{it} = 1|\mathbf{v}_{1:t}, \mathbf{h}_{1:t-1}) = \sigma\left(\sum_j w_{ij}\tilde{v}_{jt} + \sum_k w_{ki}\tilde{h}_{kt}\right) \tag{6.11}$$

where w_{ik} is the weight connecting hidden unit k to i. Note that the fact that $r(0) = 0$ comes into play here to ensure that there are no self dependencies in the graphical model. The learning rule for the intra-layer connections is of the same form as the learning rule for the visible to hidden layer connections (equation 6.10

6.2.4 Implementation Issues

Here we make mention of a an issue which arises in the implementation of this network. For arbitrary shapes of kernel, the whole history of the of the hidden and visible states needs to be stored in order to be convolved with the filter. However, a kernel with the right shape can be constructed from a simple recursive filter. In particular, a kernel with the fast onset and slow decay, which we would like to have, can be constructed from the difference of two decaying exponentials. Assuming the existence of two intervening variables to keep track of the decaying exponentials this can be implemented with no other memory of the previous state of the variables.

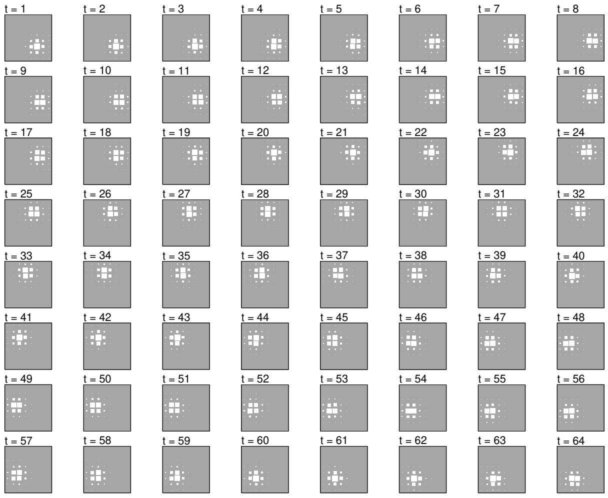

Figure 6.5: A set of 64 images from the image sequence of a blob moving along a circular path.

6.3 Results

We trained this network on a set of sequences of 8x8 synthetic images of a round blob moving along a circular path. A sample of one of these sequences is shown in figure 6.5. The figure shows a set of 64 images from the sequence. The intensity of the pixel is represented by the area of the white patch. In this sequence the period of the blob is 80 time steps.

In the following diagrams we display the time sequence of images as a matrix, with time along the y-axis and the pixels of the image along the x-axis. Each row of the matrix represents a single image with its pixels stretched out into a vector in scanline order, and each column is the time course of a single pixel. Figure 6.3a shows a segment (200 time steps) of the time series which was used in training.

The model was trained using 20 hidden layer units with several hundred iterations of the contrastive divergence learning rule. Figure 6.3b shows how the trained model reconstructs the data after we sample from the hidden layer units. Once we have trained the model it is possible to do forecasting by clamping visible layer units for a segment of a sequence and then doing iterative Gibbs sampling to generate future points in the sequence. Figure 6.3c shows that given 50 time steps from the series, the model can predict reasonably far into the future, before the pattern dies out. A problem with these simulations is that we are treating the real valued intensities in the images as probabilities. While this works for the blob images, where the values can be viewed as the probabilities of pixels in a binary image being on, this is not true for more natural images.

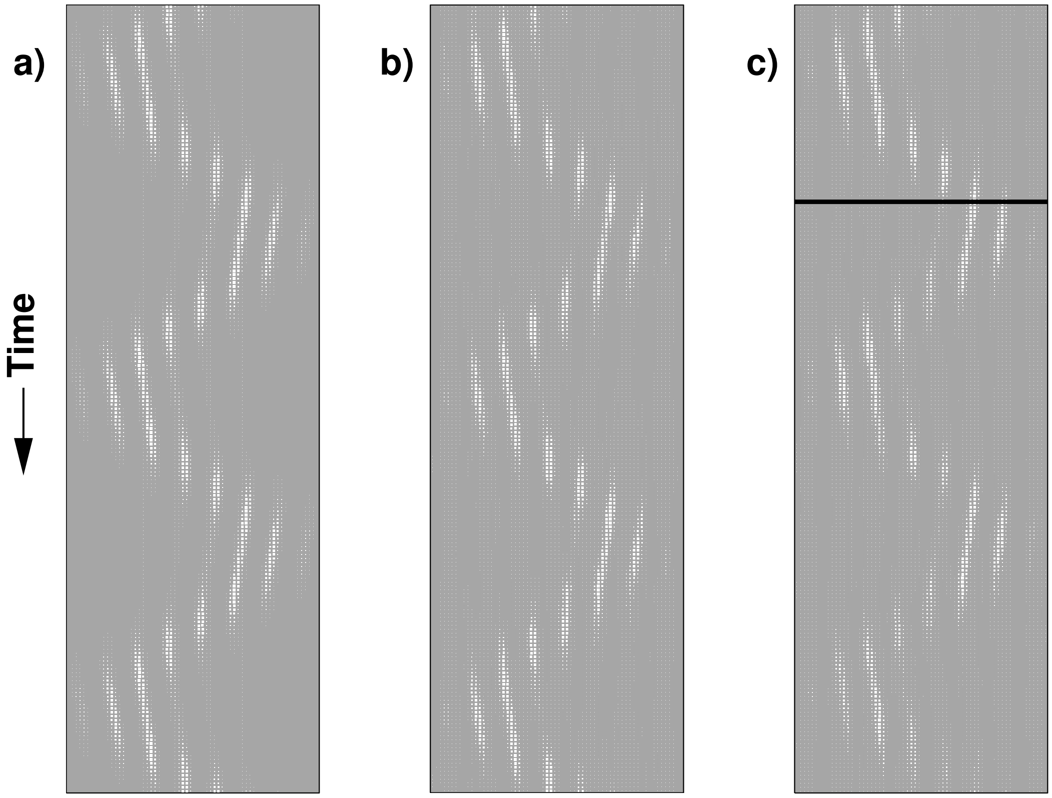

Figure 6.6: a) The original data, b) one-step reconstruction of the data, and c) prediction of the data given 50 time steps of the sequence. The black line indicates where the prediction begins.

6.4 Discussion

The model described in this chapter has some nice computational features. By the use of many binary units in the hidden layer we achieve a distributed representation of the data. This is desirable for the same reasons that using many HMM's is beneficial. It increases the power of the hidden state to represent the patterns in the input. Also, inference in the product model is very efficient. It can be done in one forward pass.

The biological interpretation of the model is spoiled somewhat by the fact that there is no simple answer to the question of where the one-step reconstructions are stored. These values have to computed by the input units based on the posterior sample from the hidden units, and there is no obvious place where they would be stored so that the learning update may be made. Although if there were a place for these values to be stored the update rule is a simple anti-hebbian term based on the value of the posterior sample and the reconstruction.

Likewise, the probabilistic interpretation is spoiled by the addition of the forward connections in the hidden layer. If we were to treat these connections in a proper probability setting then the model would actually be a fully connected Boltzmann machine in the hidden layer with the associated difficulties of learning and inference. This architecture would be much like the time series architecture used by Williams and Hinton (1991) which was described in section 2.4.4. The primary difference between the SBM and the temporal Boltzmann machine used by Williams and Hinton is the addition of the smooth temporal kernel. Also, that architecture used a full mean-field settling step to compute the statistics required for learning, making the learning procedure much slower. However, it is possible that an intermediate approximate inference scheme could be used which takes into consideration only a short window of future observations. Koller and Boyen (1999) demonstrate that, depending on the amount of stochasticity in the model, information which is propagated back from future time steps has increasingly less effect as the time difference grows. They also show that theoretically the error from such an approximation is bounded, and empirically, inference using a short window of future observations does not significantly hurt the accuracy. So it is possible that such a short range inference scheme could be used in the SBM to better account for the

Despite these problems, the work provides insight into the difficulties which arise in trying to construct a fully probabilistic model of a time series in a real-time context. Some of the operations which are done in an HMM to compute the correct posterior given a set of data, simply cannot be performed in an online fashion because they require passing information about the future state of the system arbitrarily far back. While this is practical in situations where an entire sequence can be stored and accessed in an arbitrary way, for continuous online learning it is more difficult.

One potential area of future work would be to develop a non-negative version of the spiking Boltzmann machine. In terms of biologically plausibility this is desirable because neurons in the brain communicate using signals that are all of the same sign, either positive or negative. Also, such networks have different computational properties, and the features which they find for representing the data tend to be more local (Downs, MacKay, and Lee 2000; Seung and Lee 2001).

Chapter 7

Conclusions and Future Work

7.1 Conclusions and Contributions

This thesis has presented three new time series models all tied together by the central idea of learning to model data using many simple models in collaboration. The resulting models allow for distributed hidden state representations without the usual concomitant problems of slow inference.

The introductory chapters of the thesis provide a useful background to anyone who would like to understand the PoE in the context of graphical models. In particular chapter 2 covers both directed and undirected graphical models, and the different challenges which each presents to doing inference and learning, so that it becomes clear in chapter 3 what the advantages of the PoE and the contrastive divergence algorithm are.

The work presented here has applied the PoE framework to the problem of modelling time series using two distinctly different models. In chapter 4, the PoHMM, uses the HMM as its expert. We have adapted the contrastive divergence algorithm to these more complicated models. The two language experiments (the toy English sentence domain, and the character modelling experiment) give an nice intuitive demonstration of what kinds of structure a PoHMM can in the data. It is especially good at modelling independent structure, as in the case of character statistics and capitalization, where the two things are essentially independent of one another. This is also shown in the synthetic examples, where the PoHMM is better, in the log-probability sense, at disambiguating the product sequences, and also in the case of the synthetic sequences with long range structure, the PoHMM is clearly better than a similar sized HMM.

This work on PoHMM's leads to the question of whether a set of small HMM's, trained discriminatively, will also have benefits in terms of capturing complex structure in time series. Chapter 5 introduced the relative density network. Inspired by the interpretation of the hidden units in a backpropagation network as computing the posterior distribution of a mixture of two Gaussians, we can construct analogous networks whose hidden layers compute the posterior of mixtures of other types of distributions. While we could use many different density models in a relative density network, the RDN in chapter 5 compares the relative densities of HMM's. The experimental evidence from a speaker recognition problem indicates that using many HMM's and allowing the network to learn how to use the HMM features has better performance over modelling each class with a single HMM.

The final model which we have seen is the second kind of PoE time series model. Its latent variable representation is quite different from that of the PoHMM. In this case the model is a restricted Boltzmann machine extended through time. It has the added feature that the weighted connections between the units in the model are modulated by a smooth temporal kernel. A demonstration of the model on a simple sequence of images shows that the model can learn the statistics of a domain which has smoothly varying dynamics.

7.2 Future Directions

In addition to the points already mentioned in sections 4.4, 5.4.4 and 6.4, there are a number of areas of further research and possible improvements to the PoHMM and the supervised RDN models.

7.2.1 Improvements to the PoHMM Learning Algorithm

One of the things which is currently lacking from the PoHMM is a means of computing the exact likelihood of sequences under the model. This means that we need a way of approximating the value of the partition function. There are a couple of possibilities for doing this. For small PoHMM models, a PoHMM with only two HMM's for example, it should be possible to implement the junction tree algorithm exactly. This would allow us to compute the exact marginals of the visible variables under the model, and thereby compute the summation in equation 4.2. Another potential method of doing this is by sampling from the model, possibly through Gibbs sampling. Having drawn a set of, N, sample sequences, $\{S^1_{1:T}, \dots, S^N_{1:T}\}$ then the partition function could be estimated using:

$$Z(T, \Theta) \approx \frac{1}{N} \sum_{n=1}^{N} \prod_{m=1}^{M} P^{(m)}(S^n_{1:T} | \theta^{(m)}) \tag{7.1}$$

Generating enough samples through Gibbs sampling to do this accurately might prove to be slow. The annealed importance sampling method of Neal (1998) could also be used to estimate the partition function.

Being able to compute the exact log-likelihood of the PoHMM would also facilitate model comparison between the PoHMM and Factorial HMM. One of the advantages of the FHMM, is that it is possible to compute a lower bound on the log-likelihood of the model, whereas we have not yet implemented such a measure for the PoHMM. Furthermore, if we could compute or estimate the marginal probability of data under a PoHMM, even for a limited class of PoHMM's, then we could compare the contrastive divergence algorithm to performing true maximum likelihood in the PoHMM. For example, computing the marginal likelihood of a sequence under a PoHMM with two HMM's would allow us to compute exactly the second term of the maximum likelihood learning rule as it applied to PoHMM's:

$$\left\langle \frac{\partial}{\partial \theta_m} \log P(S_{1:T}|\Theta) \right\rangle_{\widehat{P}(S_{1:T})} = \left\langle \frac{\partial}{\partial \theta_m} \log P(S_{1:T}|\theta_m) \right\rangle_{\widehat{P}(S_{1:T})} - \left\langle \frac{\partial}{\partial \theta_m} \log P(S_{1:T}|\theta_m) \right\rangle_{P(S_{1:T}|\Theta)} \tag{7.2}$$

One of the problems with the PoHMM learning algorithm is the slowness of convergence with the fixed step size stochastic gradient algorithm. Despite the better modelling that we might get from using a PoHMM architecture compared to a single regular HMM, it can take considerably longer to train. There are some methods of speeding up stochastic gradient descent using adaptive step sizes(Schraudolph 1999). These algorithms work by estimating which parameters are most important to the model and which parameters exhibit the most noise. It would be worthwhile to try some of these approaches with the PoHMM.

Another line of inquiry, which would circumvent the use of stochastic gradient descent altogether, would be to adopt a Bayesian approach to model fitting. Rather than optimizing to find a single set of parameters, we could assume a prior distribution for their values and then set up a Markov chain to sample from the parameters(Neal 1993). The appropriate posterior for the transition, and emission probability parameters of the HMM's would be Dirichlet distributions, although in the softmax domain which we currently parameterize the model a Gaussian distribution over the PoHMM parameters would be appropriate(MacKay 1998a). Furthermore, we would not need to use the approximation of contrastive divergence. Although, we can not efficiently compute the model's normalizing constant or the gradient thereof, here again, an importance sampling method such as annealed important sampling (Neal 1998) could be used to get samples from the posterior over parameters. While we would not expect the Bayesian approach to be faster than optimizing the parameters of a PoHMM, it would require a similar amount of time to draw samples from the posterior, and the Bayesian approach has many other computational advantages. For example, by drawing multiple samples from the posterior and using an ensemble approximation to the predictive distribution of the PoHMM, we might avoid the problems of local minimum that were mentioned in section 4.4.

Another problem with PoHMM's is that it might learn a set of parameters which allow it to make good reconstructions (and thus minimizing the contrastive divergence) but still not be a very good model of the data density. An example of this type of problem is mentioned in section 4.4, where by having too many HMM's in the product can make it extremely easy for the model to reconstruct the data well even with quite random parameters. This adversely affects the contrastive divergence learning algorithm because, if there are no or few mistakes in the reconstruction, we no longer have a learning signal. Another approach would be to optimize the symmetric Shannon score which we can efficiently compute. This would promote the ability of the model to predict the held out symbol in addition to making good reconstructions and this would be much harder to do without having learned something of the temporal structure of the sequence. While we refer to the conditional distribution of a single held out variable in the model conditioned on all the others as the symmetric Shannon score, this distribution is also known commonly as the *pseudo-likelihood*, and the maximizing this is known as the pseudo-likelihood method. In many probabilistic models, even if we cannot compute the global partition function of the model, computing distributions of each observed variable conditioned on the other observed variables is still possible, if the local normalization is easy. Instead of maximizing the global likelihood we maximize the pseudo-likelihood which is just the product of this collection of conditional densities. Such optimization methods have been shown to work with other intractable models (McLachlan and Peel 2000, p335) and recent work by Hinton has showed that the pseudo-likelihood method works well for training certain types of product models, so it may work well with PoHMM's.

7.2.2 Further Work on Speech Applications

There is also much work that could be done to further explore the possibilities of this thesis work in the speech domain. For example, it might be beneficial to use unsupervised PoHMM's to preprocess speech signals. While the PoHMM examples presented in chapter 4 deal exclusively with discrete valued sequences, some preliminary investigations were done on speech samples using a PoHMM with continuous Gaussian outputs on high resolution spectrograms (Hinton and Brown 2001). After training, the resulting PoHMM can be analysed by taking a new spectrogram, performing inference to get the posterior distribution of the individual HMM's, and then looking at the Gaussians of the most probable states. In high dimensional speech data the posterior distribution over state is generally highly peaked and there will almost certainly be only a single state with any significant mass. So, looking at the most probably state is not much different than examining the entire posterior. So given the most probable states at each time step we can tag the time-frequency bins of the spectrogram according to which HMM has the most probable posterior state with the lowest variance Gaussian emission density. In essence we are trying to find which HMM had the most influence in reconstructing the observed spectrogram at that time and frequency. An example of a spectrogram tagged in this way is shown in figure 7.1. The spectrogram is a single utterance from the TIMIT database (Garofolo, Lamel, Fisher, Fiscus, Pallett, and Dahlgren 1993) and the PoHMM contained 10 models of 10 states each. The figure was made by assigning a unique colour to each of the 10 HMM's and colouring the spectrogram according to which of the HMM's has the lowest variance in each of the time-frequency bins. The resulting tagged spectrogram, shows that HMM's in the model seem to focus on particular features in the spectral domain. This suggests the possibility that the PoHMM could be used as a pre-processor for a speech recognition system by first training a continuous PoHMM in an unsupervised way and then learning another model on top of that using the posterior HMM distributions or the most probable paths as input features.

As for the RDN, the speaker recognition problem which we applied it to used only a limited number of speakers. Not only were the number of speakers small, but the number of HMM's used in the RDN were also limited[1]. With such a small number of speakers, we would expect the RDN to pick up on features of the speech which were quite speaker specific. One of the problems with supervised training of a set of HMM's to perform speech or speaker recognition is that adding new classes to the model requires retraining the entire model, and this would be especially true if all the HMM's were focussed on characteristics which are specific to a couple of speakers. However, if we could perform the same speaker recognition experiment with many more speakers and many HMM's then we might hope that the HMM's would learn quite general speaker-discrimination features which could be reused. Specifically, we might expect that for an RDN trained on a large number of speakers, upon adding a new speaker, no retraining of the input layer HMM's would be required, only retraining of the output layer of weights, or possibly even just training of a single set of weight to the new output unit.

As a final proposition, all the supervised networks described in chapter 5 use the

[1]This limitation was due mostly to the amount of time which was available to train the models

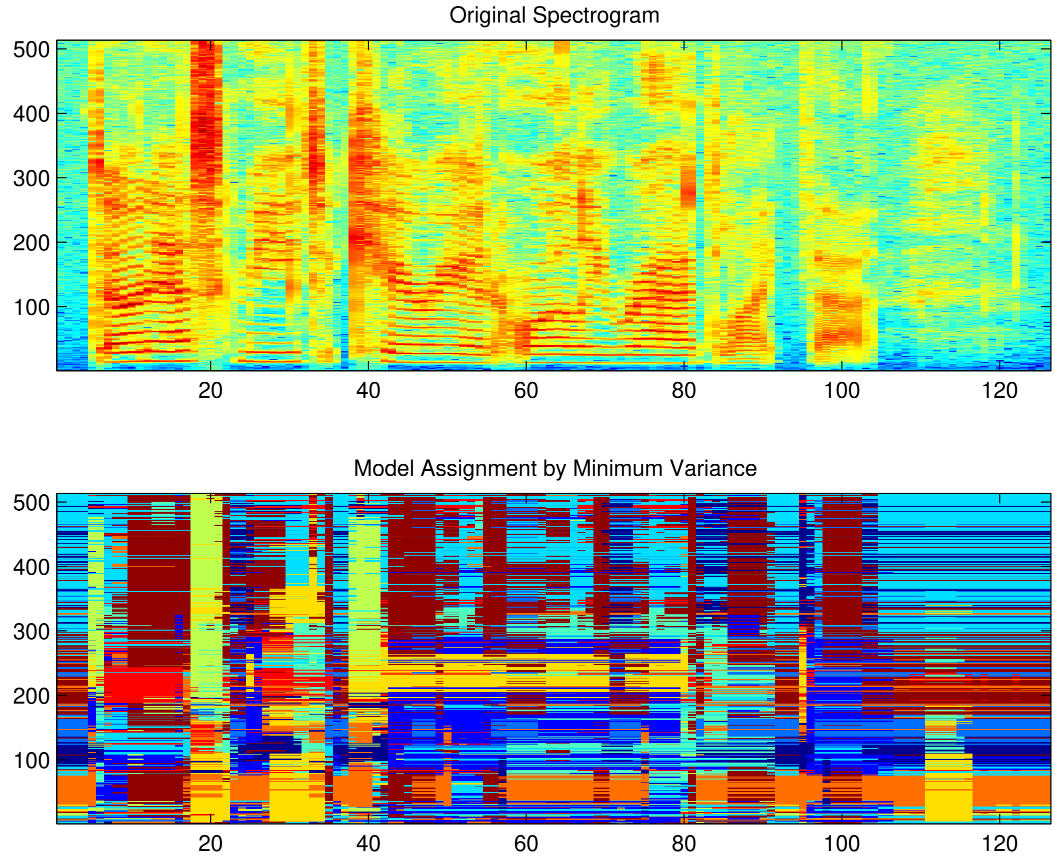

Figure 7.1: Spectrogram tagged by minimum variance HMM. The upper figure is the original spectrogram. The lower is the same spectrogram tagged according to which HMM had the lowest variance in that frequency bin after computing the reconstruction distribution.

normalized log-probability of an entire sequence as input to the decision layer of the network. Thus we are training the HMM's to recognize whole-sequence features. A useful extension to these architectures, would be a network which integrated evidence locally in time on short sub-sequences of the data. This would allow the network to learn about structure in pieces of the training sequences instead of just the entire sequence. In order to do this we would have to extract a vector of posterior probabilities from an HMM which are local in time. It is not entirely clear how this might be done efficiently, because the forward backward algorithm integrates information from the whole length of the sequence. We could imagine doing forward backward over short overlapping subsequences. This would be equivalent to simply dividing the data into short overlapping segments, and creating an ordered sequence of log-probabilities. The problem now becomes how to integrate these time-series measurements over the length of the sequence to get a final classification decision. However, deconstructing our probabilistic models in this way begins to take on the flavour of time-delay neural networks (Waibel, Hanazawa, Hinton, Skikano, and Lang 1989). It would be better to have a fully probabilistic, hierarchical time series model which looks at sequences at multiple time scales, eventually resulting in a single probability measure for the sequence, and then use this as input to the decision layer of an RDN. The whole system could then be trained in the usual supervised manner.

Appendix A

HMM Derivatives

We will parameterize the HMM as follows: the transition probabilities are parameterized as a matrix A, where $A_{ij} = P(s_t = j | s_{t-1} = i)$ and the initial state prior Π is given by $\Pi_i = P(s_1 = i)$. In the case of a discrete output HMM, the output probability matrix, B, is given by $B_{ki} = P(y_t = k | s_t = i)$. For an HMM with continuous Gaussian outputs, one Gaussian per hidden state, the mean of the ith Gaussian is μ_i and the covariance matrix is Σ_i.

For the discrete probabilities we parameterize the elements of each matrix as a softmax function

$$\sigma(a_{ij}) = \frac{e^{a_{ij}}}{\sum_k e^{a_{ik}}}.$$

where $\sigma(a_{ij})$ is the element in row i, column j and the rows of the matrix are normalized to sum to 1. This eliminates the necessity of enforcing positivity and sum-to-one constraints on the parameters. Thus the matrix entries as a function of the softmax parameters are: $A_{ij} = \sigma(a_{ij})$, $B_{ij} = \sigma(b_{ij})$ and $\Pi_i = \sigma(\pi_i)$. We also make use of the fact that the derivative of a softmax function is:

$$\frac{\mathrm{d}}{\mathrm{d}a_k}\sigma(a_i) = \sigma(a_i)\delta_{i,k} - \sigma(a_i)\sigma(a_k)$$

A.1 Discrete Output

For a discrete output HMM the joint probability is:

$$P(S_{1:T}, Y_{1:T}) = P(S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})\prod_{t=1}^{T} P(Y_t|S_t)$$

$$= \prod_i \Pi_i^{\delta_{S_1,i}} \prod_{t=2}^{T} \prod_{i,j} A_{ij}^{\delta_{S_{t-1},i}\delta_{S_t,j}} \prod_{t=1}^{T} \prod_{i,j} B_{ij}^{\delta_{S_t,i}\delta_{Y_t,j}}$$

and the total log-likelihood is:

$$\mathcal{L} = \sum_i \delta_{S_1,i} \log \Pi_i + \sum_{t=2}^{T} \sum_{i,j} \delta_{S_{t-1},i}\delta_{S_t,j} \log A_{ij} + \sum_{t=1}^{T} \sum_{i,j} \delta_{S_t,i}\delta_{Y_t,j} \log B_{ij}$$

Initial State Prior

$$\begin{aligned}
\frac{\partial}{\partial \pi_k}\mathcal{L} &= \sum_i \delta_{S_1,i} \frac{\partial}{\partial \pi_k} \log \sigma(\pi_i) \\
&= \sum_i \delta_{S_1,i} [\delta_{ik} - \sigma(\pi_k)] \\
&= \delta_{S_1,k} - \sigma(\pi_k) \qquad \text{(A.1)}
\end{aligned}$$

Transition Matrix

$$\begin{aligned}
\frac{\partial}{\partial a_{kl}}\mathcal{L} &= \sum_{t=2}^{T} \left[\sum_{i,j} \delta_{S_{t-1},i}\delta_{S_t,j} \frac{\partial}{\partial a_{kl}} \log \sigma(a_{ij}) \right] \\
&= \sum_{t=2}^{T} \left[\sum_{i,j} \delta_{S_{t-1},i}\delta_{S_t,j}\delta_{ki} [\delta_{lj} - \sigma(a_{il})] \right] \\
&= \sum_{t=2}^{T} \left[\delta_{S_{t-1},k}\delta_{S_t,l} - \sum_j \delta_{S_{t-1},k}\delta_{S_t,j}\sigma(a_{kl}) \right] \\
&= \sum_{t=2}^{T} \left[\delta_{S_{t-1},k}\delta_{S_t,l} - \delta_{S_{t-1},k}\sigma(a_{kl}) \right] \qquad \text{(A.2)}
\end{aligned}$$

Output Matrix

$$\begin{aligned}
\frac{\partial}{\partial b_{kl}}\mathcal{L} &= \sum_{t=2}^{T} \left[\sum_{i,j} \delta_{S_t,i}\delta_{Y_t,j} \frac{\partial}{\partial b_{kl}} \log \sigma(b_{ij}) \right] \\
&= \sum_{t=2}^{T} \left[\sum_{i,j} \delta_{S_t,i}\delta_{Y_t,j}\delta_{ki} [\delta_{lj} - \sigma(b_{il})] \right] \\
&= \sum_{t=2}^{T} \left[\delta_{S_t,k}\delta_{Y_t,l} - \sum_j \delta_{S_t,k}\delta_{Y_t,j}\sigma(b_{kl}) \right] \\
&= \sum_{t=2}^{T} [\delta_{S_t,k}\delta_{Y_t,l} - \delta_{S_t,k}\sigma(b_{kl})] \qquad \text{(A.3)}
\end{aligned}$$

A.2 Continuous Gaussian Output

For a continuous output HMM with a single Gaussian per state then the formula for the complete data probability is:

$$
\begin{aligned}
P(S_{1:T}, Y_{1:T}) &= P(S_1) \prod_{t=2}^{T} P(S_t|S_{t-1}) \prod_{t=1}^{T} P(Y_t|S_t) \\
&= \prod_i \Pi_i^{\delta_{S_1,i}} \prod_{t=2}^{T} \prod_{i,j} A_{ij}^{\delta_{S_{t-1},i}\delta_{S_t,j}} \prod_{t=1}^{T} \prod_i \left[|2\pi\Sigma_i|^{\frac{1}{2}} \exp\{-\frac{1}{2}(Y_t-\mu_i)^T {\Sigma_i}^{-1}(Y_t-\mu_i)\right]^{\delta_{S_t,i}}
\end{aligned}
$$

and the corresponding total log-likelihood is:

$$
\begin{aligned}
\mathcal{L} = &\sum_i \delta_{S_1,i} \log \Pi_i \\
&+ \sum_{t=2}^{T} \sum_{i,j} \delta_{S_{t-1},i}\delta_{S_t,j} \log A_{ij} \\
&- \frac{1}{2} \sum_{t=1}^{T} \sum_i \delta_{S_t,i} \left[-\frac{d}{2} \log 2\pi - \frac{1}{2} \log |2\pi\Sigma_i| - \frac{1}{2}(Y_t-\mu_i)' {\Sigma_i}^{-1}(Y_t-\mu_i) \right]
\end{aligned}
$$

where d is the dimensionality of, Y_t, the vector of observations. The derivatives of this likelihood with respect to the transition matrix and the initial state priors are identical to those of the discrete outpu HMM. The derivatives of the means and variances of the output Gaussians are as follows:

Means

$$
\frac{\partial}{\partial \mu_j} \mathcal{L} = \sum_{t=1}^{T} \delta_{S_t,j} (Y_t - \mu_j)^T \Sigma_j^{-1} \tag{A.4}
$$

Covariance

To compute the derivative with respect to the covariance matrix we reformulate the the quadratic term as a trace, by using the fact that taking the trace of a scalar leaves it unchanged, and also $\mathrm{tr}(ABC) = \mathrm{tr}(BCA)$ (providing that the matrices commute).

$$
\begin{aligned}
\frac{\partial}{\partial \Sigma_j} \mathcal{L} &= \sum_{t=1}^{T} \delta_{S_t,j} \left[\Sigma_j^{-1} + \frac{\partial}{\partial \Sigma_j} \mathrm{tr} \left(\Sigma_j (Y_t-\mu_j)(Y_t-\mu_j)^T \right) \right] \\
&= \sum_{t=1}^{T} \delta_{S_t,j} \left[\Sigma_j^{-1} - \Sigma_j^{-1}(Y_t-\mu_j)(Y_t-\mu_j)^T \Sigma_j^{-1} \right] \qquad (A.5)
\end{aligned}
$$

This derivation depends on the matrix derivatives $\frac{\partial}{\partial X}|X| = |X| {*} X^{-T}$ and $\frac{\partial}{\partial X}\mathrm{tr}(AX^{-1}B)) = -(X^{-1}BAX^{-1})^T$.

Bibliography

Ackley, D. H., G. E. Hinton, and T. J. Sejnowski (1985). A learning algorithm for Boltzmann machines. *Cognitive Science 9*, 147–169.

Allen, J. B. (1994, October). How do humans process and recognize speech? *IEEE Transactions on Speech and Audio Processing 2*(4), 567–577.

Bahl, L. R., P. F. Brown, P. V. de Souza, and R. L. Mercer (1986). Maximum mutual information of hidden Markov model parameters for speech recognition. In *Proceeding of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 49–53.

Barber, D. and P. Sollich (2000). Gaussian fields for approximate inference in layered sigmoid belief networks. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing Systems*, Volume 12, Cambridge, MA, pp. 393–399. MIT Press.

Baum, L. E., T. Petrie, G. Soules, and N. Weiss (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics 41*(1), 164–171.

Bengio, Y. and P. Fransconi (1995). Diffusion of context and credit information in Markovian models. *Journal of Artificial Intelligence Research 3*, 249–270.

Besacier, L. and J.-F. Bonastre (2000). Subband architecture for automatic speaker recognition. *Signal Processing 80*, 1245–1259.

Bimbot, F., I. Magrin-Chagnoleau, and L. Mathan (1995). Second-order statistical measures for text-independent speaker recognition. *Speech Communication 17*, 177–192.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

Bourlard, H. and S. Dupont (1997). Subband-based speech recognition. In *Proceedings* of *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1251–1254.

Brand, M. (1996). Coupled hidden Markov models for modeling interacting processes. Technical Report 405, MIT Media Lab.

Bridle, J. (1990). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. Touretzky (Ed.), *Advances in Neural Information Processing Systems*, Volume 2, San Mateo, CA, pp. 211–217. Morgan Kaufmann.

Brown, A. D. and G. E. Hinton (2001). Products of hidden Markov models. In T. Jaakkola and T. Richardson (Eds.), *Proceedings of Artificial Intelligence and Statistics 2001*, pp. 3–11. Morgan Kaufmann.

Carey, M. J., E. S. Parris, and J. S. Bridle (1991, May). A speaker verification system using alphanets. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 397–400.

Cole, R., M. Noel, and V. Noel (1998). The CSLU speaker recognition corpus. In *Proceedings of ICSLP*, Sydney, Australia.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence 42*((2-3)), 393–405.

Cover, T. M. and J. A. Thomas (1991). *Elements of Information Theory*. New York: John Wiley and Sons.

Cowell, R. G., A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter (1999). *Probabilistic Networks and Expert Systems*. Springer.

Dagum, P. and M. Luby (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence 60*, 141–153.

Dawid, A. P. (1992). Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing 2*, 25–36.

Dayan, P., G. E. Hinton, R. Neal, and R. S. Zemel (1995). The Helmholtz machine. *Neural Computation 9*, 1022–1037.

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

Downs, O. B., D. J. C. MacKay, and D. D. Lee (2000). The nonnegative Boltzmann machine. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing*, Volume 12, pp. 428–434. MIT Press.

Duda, R. O., P. E. Hart, and D. G. Stork (2001). *Pattern Classification* (2nd ed.). New York: John Wiley and Sons.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science 14*, 179–211.

Fine, S., Y. Singer, and N. Tishby (1998). The hierarchical hidden Markov model. *Machine Learning 32*, 41–62.

Frey, B. (1997). *Bayesian Networks for Pattern Classification, Data Compression and Channel Coding*. Ph. D. thesis, University of Toronto.

Frey, B. J. and D. J. C. MacKay (1998). A revolution: Belief propagation in graphs with cycles. In M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.), *Advances in Neural Information Processing Systems*, Volume 10, Cambridge, MA, pp. 479–486. MIT Press.

Garofolo, J. S., L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren (1993). DARPA TIMIT acoustic-phonetic continuous speech corpus CD-ROM.

Ghahramani, Z. (1996). The em algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, Department of Computer Science, University of Toronto.

Ghahramani, Z. and G. E. Hinton (1998). Switching state space models. Submitted for publication.

Ghahramani, Z. and M. I. Jordan (1997, November–December). Factorial hidden Markov models. *Machine Learning 29*(2/3), 245–273.

Goutsias, J. K. (1991). A theoretical analysis of Monte Carlo algorithms for the simulation of gibbs random field images. *IEEE Transactions on Information Theory 37*(6), 1618–1628. November.

Hinton, G. E. (1989). Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation 1*(1), 143–150.

Hinton, G. E. (1999). Products of experts. In *ICANN 99: Ninth International Conference on Artificial Neural Networks*, Edinburgh, pp. 1–6.

Hinton, G. E. (2001). Training products of experts by minimizing contrastive divergence. In press.

Hinton, G. E. and A. D. Brown (2000). Spiking Boltzmann machines. In S. A. Solla, T. K. Leen, and K.-R. Müller (Eds.), *Advances in Neural Information Processing*, Volume 12, pp. 122–128. MIT Press.

Hinton, G. E. and A. D. Brown (2001, April). Training many small hmm's. In *To appear in Proceedings of the Workshop on Innovation in Speech Processing*.

Hinton, G. E. and Z. Gharamani (1997). Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society* **B** *352*, 1177–1190.

Hinton, G. E. and T. J. Sejnowski (1986). Learning and relearning in boltzman machines. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing*, Volume 1, pp. 282–317. Cambridge, MA: The MIT Press.

Hofmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning 42*(1), 177–196.

Jensen, F. V. (1996). *Introduction to Bayesian Networks*. Springer-Verlag.

Jensen, F. V., S. L. Lauritzen, and K. G. Olesen (1990). Bayesian updating in recursive graphical models by local computations. *Computational Statistical Quarterly 4*, 269–282.

Jordan, M. I. (1995, August). Why the logistic function? A tutorial discussion on probabilities and neural networks. Technical Report Computational Cognitive Science, Technical Report 9503, Massachusetts Institute of Technology.

Jordan, M. I., Z. Ghahramani, and L. K. Saul (1997). Hidden Markov decision trees. In M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.), *Advances in Neural Information Processing Systems*, Volume 9, pp. 501–507. The MIT Press.

Koller, D. and X. Boyen (1999). Approximate learning of dynamic models. In M. Kearns, S. Solla, and D. Cohn (Eds.), *Advances in Neural Information Processing (NIPS)*, Volume 11, pp. 396–402. MIT Press.

Lari, K. and S. J. Young (1990). The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Compter Speech and Language 4*, 35–56.

Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B 50*(2), 157–224.

Li, Q., B.-H. Juang, C.-H. Lee, Q. Zhou, and F. K. Soong (1999, March). Recent advancements in automatic speaker recognition. *IEEE Robotics & Automation 6*(1), 24–33.

Lippmann, R. P. (1997). Speech recognition by humans and machines. *Speech communication 22*, 1–15.

MacKay, D. J. C. (1998a). Choice of basis for Laplace approximation. *Machine Learning 33*(1), 77–86.

MacKay, D. J. C. (1998b). Learning in graphical models. In M. I. Jordan (Ed.), *Introduction to Monte Carlo Methods*. Dordrecht: Kluwer Academic Publishers. http://wol.ra.phy.cam.ac.uk/mackay/erice.ps.gz.

McLachlan, G. J. and T. Krishnan (1996). *The EM Algorithm and Extensions*. John Wiley and Sons.

McLachlan, G. J. and D. Peel (2000). *Finite Mixture Models*. New York: Wiley.

Murphy, K. P. (1999, September). Filtering, smoothing and the junction tree algorithm. Technical report, Department of Computer Science, University of California, Berkeley. http://www.cs.berkeley.edu/ murphyk/Papers/smooth.ps.gz.

Murphy, K. P., Y. Weiss, and M. I. Jordan (1999). Loopy belief propagation for approximate inference: an empirical study. In K. B. Laskey and H. Prade (Eds.), *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, San Francisco. Morgan Kaufmann.

Neal, R. M. (1990). Learning stochastic feedforward networks. Technical Report CRG-TR-90-7, Dept. of Computer Science, University of Toronto.

Neal, R. M. (1993). Probabilistic inference using markov chain monte carlo methods. Technical report, Department of Computer Science, University of Toronto.

Neal, R. M. (1998). Annealed importance sampling. Technical Report Technical Report No. 9805, Department of Statistics, University of Toronto. http://www.cs.utoronto.ca/ radford.

Neal, R. M. and G. E. Hinton (1998). A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan (Ed.), *Learning in Graphical Models*, pp. 355–368. Dordrecht: Kluwer Academic Publishers.

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

Peterson, C. and J. R. Anderson (1987). A mean field learning algorithm for neural networks. *Complex Systems 1*, 995–1019.

Platt, J. (1989). *Constraint Methods for Neural Networks and Computer Graphics.* Ph. D. thesis, California Institute of Technology, Pasadena, CA.

Rabiner, L. and B.-H. Juang (1993). *Fundamentals of Speech Recognition.* Englewood Cliffs, NJ: Prentice Hall.

Rabiner, L. R. and B.-H. Juang (1986, January). An introduction to hidden Markov models. *IEEE ASSAP Magazine 3*, 4–16.

Reynolds, D. A., T. F. Quatieri, and R. B. Dunn (2000). Speaker verification using adapted Gaussian mixture models. *Digital Signal Processing 10*, 19–41.

Rezek, I., P. Sykacek, and S. J. Roberts (2000, May). A comparison of Bayesian and maximum likelihood learning of coupled hidden Markov models. Technical report, Robotics Research Group, Dept. of Engineering Science, University of Oxford.

Saul, L. K. and M. I. Jordan (1995). Boltzmann chains and hidden Markov models. In G. Tesauro, D. S. Touretzky, and T. K. Leen (Eds.), *Advances in Neural Information Processing Systems*, Volume 7, Cambridge, MA. MIT Press.

Saul, L. K. and M. I. Jordan (1996). Explointing tractable substructures in intractable networks. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo (Eds.), *Advances in Neural Information Processing Systems*, Volume 8, Cambridge, MA. MIT Press.

Saul, L. K. and M. I. Jordan (1999a). A mean field learning algorithm for unsupervised neural networks. In M. I. Jordan (Ed.), *Learning in Graphical Models*, pp. 541–554. MIT Press.

Saul, L. K. and M. I. Jordan (1999b). Mixed memory Markov models: Decomposing complex stochastic processes as mixture of simpler ones. *Machine Learning 37*(1), 75–87.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. In *Proceedings 9th International Conference on Artificial Neural Networks*, London, pp. 569–574. IEE. http://www.inf.ethz.ch/∼schraudo/pubs/#smd/.

Seung, H. S. and D. D. Lee (2001). Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Diettrich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13, pp. 556–562. MIT Press.

Smyth, P., D. Heckerman, and M. I. Jordan (1997). Probabilistic independence networks for hidden Markov models. *Neural Computation 9*(2), 227–269.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory IT-13*(2), 260–269.

Waibel, A., T. Hanazawa, G. E. Hinton, K. Skikano, and K. J. Lang (1989). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech and Signal Processing ASSP-37*, 328–229.

Williams, C. K. I. and G. E. Hinton (1991). Mean field networks that learn to discriminate temporally distorted strings. In D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton (Eds.), *Connectionst models: Proceedings of the 1990 summer school*, San Francisco, CA, pp. 18–22. Morgan Kaufmann.

Yedidia, J. S., W. T. Freeman, and Y. Weiss (2001). Generalized belief propagation. In T. K. Leen, T. G. Dietterich, and V. Tresp (Eds.), *Advances in Neural Information Processing Systems*, Volume 13, pp. 689–695. MIT Press.