Speech Recognition with Dynamic Bayesian Networks

by

Geoffrey G. Zweig

B.A. (Universtity of California, Berkeley) 1985
M.A. (University of California, Berkeley) 1989
M.S. (University of California, Berkeley) 1996

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA, BERKELEY

Committee in charge:

Professor Stuart J. Russell, Chair
Professor Nelson Morgan
Professor Jitendra Malik
Professor John J. Ohala

Spring 1998

The dissertation of Geoffrey G. Zweig is approved:

Chair Date

Date

Date

Date

University of California, Berkeley

Spring 1998

Speech Recognition with Dynamic Bayesian Networks

by
Geoffrey G. Zweig

Abstract

Speech Recognition with Dynamic Bayesian Networks

by

Geoffrey G. Zweig
Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Stuart J. Russell, Chair

Dynamic Bayesian networks (DBNs) are a powerful and flexible methodology for representing and computing with probabilistic models of stochastic processes. In the past decade, there has been increasing interest in applying them to practical problems, and this thesis shows that they can be used effectively in the field of automatic speech recognition.

A principle characteristic of dynamic Bayesian networks is that they can model an arbitrary set of variables as they evolve over time. Moreover, an arbitrary set of conditional independence assumptions can be specified, and this allows the joint distribution to be represented in a highly factored way. Factorization allows for models with relatively few parameters, and computational efficiency. Standardized inference and learning routines allow a wide variety of probabilistic models to be tested without deriving new formulae, or writing new code.

The contribution of this thesis is to show how DBNs can be used in automatic speech recognition. This involves solving problems related to both representation and inference. Representationally, the thesis shows how to encode stochastic finite-state word models as DBNs, and how to construct DBNs that explicitly model the speech-articulators, accent, gender, speaking-rate, and other important phenomena. Technically, the thesis presents inference routines that are especially tailored to the requirements of speech recognition: efficient inference with deterministic constraints, variable-length utterances, and online inference. Finally, the thesis presents experimental results that indicate that real systems can be built, and that modeling important phenomena with DBNs results in improved recognition accuracy.

Professor Stuart J. Russell
Dissertation Committee Chair

Contents

List of Figures

List of Tables

Acknowledgements

This work benefited from interactions with numerous people during the course of my graduate study. In addition to my committee members, I would like to thank Richard Karp, Steve Glassman, and Mark Manasse. Working with Richard Karp on computational biology impressed on me the importance of addressing practical problems. Steve Glassman and Mark Manasse at DEC SRC introduced me to the world of large-scale computing, and taught me that every bit counts.

The speech group at the International Computer Science Institute provided an exceptionally supportive and pleasant atmosphere to work in. I fondly acknowledge all the members: Nelson Morgan, Steven Greenberg, Dan Ellis, Jeff Bilmes, Eric Fosler-Lussier, Daniel Gildea, Adam Janin, Brian Kingsbury, Nikki Mirghafori, Michael Shire, Warner Warren, and Su-Lin Wu.

My ideas on Bayesian networks benefited from discussions with Nir Friedman, Kevin Murphy, Paul Horton, and especially Stuart Russell.

Chapter 1

Introduction

1.1 Motivation

1.1.1 Probabilistic Models for Speech Recognition

The problem of automatic speech recognition (ASR) consists of writing computer programs that are able to examine a speech waveform and emit the same sequence of words that a person would hear when listening to the sound. Essentially, this requires defining an association between the acoustic features of sounds and the words people perceive; ASR further imposes the constraint that the association must be defined so precisely that it can be evaluated by a computer. In the course of the last quarter century, probabilistic models have become the predominant approach to defining the association between sounds and words, and have been used to model the processes of both speech perception and speech generation.

These models work in terms of linguistic units that represent the different kinds of sounds that are encountered in a language. As an example, syllables are representative and intuitive. Taken together, the set of syllables spans the range of sounds used to produce words, and syllabic word representations can be found in any dictionary.

A perceptual model makes the association between sounds and words in a bottom-up fashion, and can be thought of as two black boxes. The first takes acoustic features from a short period of time as its input, and produces a probability distribution over the the set of possible linguistic units. The second black box takes this stream of disjoint probabilities,

and incorporates information about which sequences of linguistic units constitute acceptable words, in order to find an interpretation that makes sense over a long span of time. The association between sounds and linguistic units is most often made with artificial neural networks (ANNs) (Robinson & Fallside 1988; Waibel *et al.* 1989; Robinson & Fallside 1991; Bourlard & Morgan 1994). The linkage between subword linguistic units and complete word models is made with a stochastic finite state automaton (SFSA) that defines a distribution over the possible pronunciations of a word.

A generative model works the other way around, and starts with a word hypothesis. It first relates this hypothesis to a sequence of linguistic units, and then relates the linguistic units to sounds. The linkage between words and subword units is made with the same SFSA that the perceptual approach uses, and the association between subword units and sounds is established with a simple lookup table or equivalent functional representation. Hidden Markov models (HMMs) encompass both aspects of this process, are the most commonly used generative models.

Despite a great deal of success, as indicated by current commercial products from companies such as IBM and Dragon Systems, current systems have significant problems (Makhoul & Schwartz 1995; Young 1996). These problems are caused by a number of different factors, including coarticulation (the modification of a sound in the context of surrounding sounds), rate-of-speech variability, speaker accent, and ambient noise conditions. Although there are techniques for addressing these problems within the current frameworks of speech recognition, they are limited by the fact that the basic representational unit is the subword linguistic unit, and there is no explicit causal representation of either generative or perceptual processes.

In the HMM framework, the subword linguistic unit is atomic, and there is not typically any explicit representation of the physical process by which sound is generated or modified by noise. In the ANN framework, below the level of the linguistic unit there does exist a highly distributed representation of the perceptual process — in the form of a multiplicity of artificial neurons — but no explicit meaning is assigned. The contributions of this thesis are to apply a new probabilistic modeling framework, Bayesian networks, to the problem of speech recognition, and to show how it can be used to address these problems.

1.1.2 A Next Step

Computational power has doubled roughly every 18 months since the late 1960s, and this trend is expected to continue for another ten to twenty years. This increase in computing power makes it feasible to move beyond the simple representational framework of current ASR systems, and Bayesian networks provide an ideal framework in which to formulate probabilistic models that are simultaneously expressive, precise, and compact.

A Bayesian network (Pearl 1988) has the ability to represent a probability distribution over arbitrary sets of random variables. Moreover, it is possible to factor these distributions in arbitrary ways, and to make arbitrary conditional independence assumptions. The combination of factorization and conditional independence assumptions can vastly reduce the number of parameters required to represent a probability distribution. Because of Ockham's razor, this is desirable on general principles; on a more practical level, it allows the model parameters to be estimated with greater accuracy from a limited amount of data (Zweig 1996; Ghahramani & Jordan 1997).

As a simple example of the application of a factored probabilistic model, consider a doctor with a patient who is complaining of headaches and blurred vision. Suppose the doctor knows that the person has a family history of diabetes, and also that the person is a programmer who stares long hours at a computer screen, and is under stress at work. The doctor is interested in evaluating the probability that the patient has diabetes. The set of relevant variables is:

$$\{headaches, blurredVision, programmer, stress, familyHistory, diabetes\},$$

and the doctor wants to evaluate

$$P(diabetes = true | headaches = true, blurredVision = true,$$
$$programmer = true, stress = true, familyHistory = bad).$$

In order to do this, it is necessary to be able to compute

$$P(headaches, blurredVision, programmer, stress, familyHistory, diabetes)$$

for every possible combination of variable values, and one way to do this is to maintain a full representation of the joint probability distribution. However, in situations where there

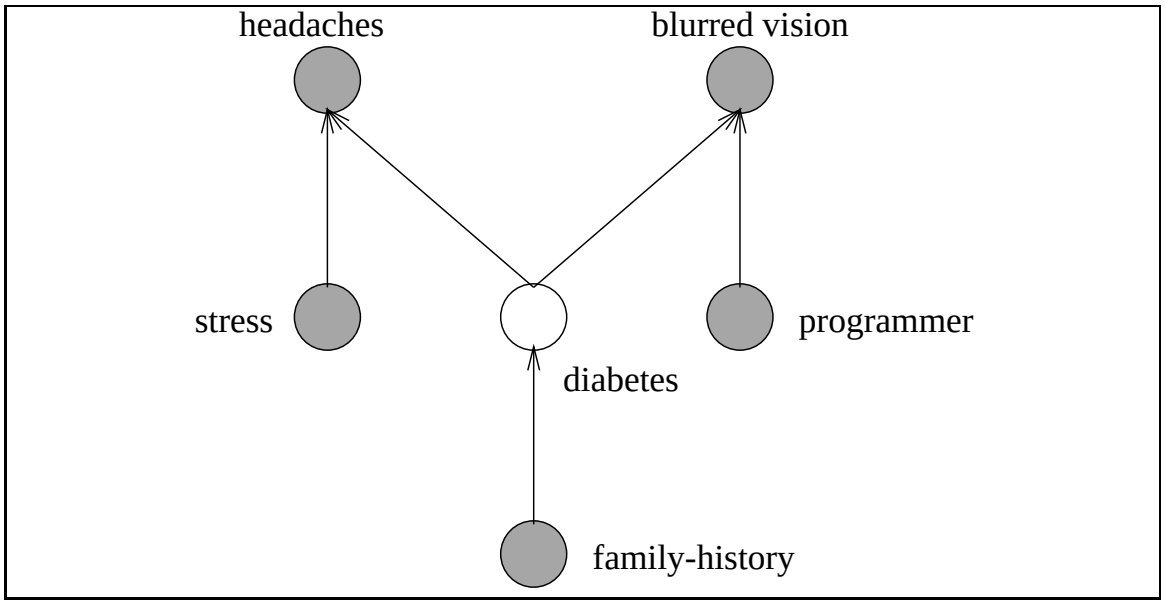

Figure 1.1: A Bayesian network for a simple medical situation. The shaded variables have known values, while the unshaded variable does not.

are a large number of variables, there are an excessive number of combinations, and the scheme ignores the fact that not everything is relevant to everything else. A more reasonable approach might be to factor the full joint distribution as:
$P(programmer)P(stress)P(familyHistory)$
$P(headaches|familyHistory, stress)P(blurredVision|familyHistory, programmer)$.
This factoring expresses a good deal of intuition, e.g. that both family medical history and stress at work are relevant to the existence of headaches, but also that being a programmer is irrelevant to the family medical history. With a factored model like this, the doctor can look up a small number of probabilities, multiply them together, and get the answer.

Perhaps, however, being a programmer is not irrelevant to stress, and a factorization such as
$P(programmer)P(stress|programmer)P(familyHistory)$
$P(headaches|familyHistory, stress)P(blurredVision|familyHistory, programmer)$
would be more appropriate.

The primary strength of a Bayesian network system is that once the program is written, it is extremely easy to switch from one model to another and evaluate different ideas. There can also be savings in the number of model parameters, and it should be noted

	/T/	/EH/	/N/	/K/	/AE/	/T/	/S/
Lips	0	0	0	0	0	0	0
Tongue Blade	1	0	1	0	0	1	3
Tongue Body	0	9	0	1	10	0	0
Nasality	1	1	2	1	1	1	1
Larynx	2	1	1	2	1	2	2

Figure 1.2: An articulatory model of the pronunciation of "ten cats," adapted from Deng and Sun, 1994. The linguistic units are shown along the top row. The numbers in the chart represent target articulator positions that correspond to these linguistic units. The shaded boxes represent the range of variability in articulator positions from utterance to utterance and person to person. While Deng and Sun use rules to determine the possible ranges, this kind of information can be encoded probabilistically in a Bayesian network.

that both of the factorizations presented above require fewer parameters than the unfactored representation. Finally, Bayesian networks have the advantage that the mathematics which they express is also simple to represent in graphical form. The first factorization is shown in Figure 1.1. The second differs simply by the addition of an arc.

Dynamic Bayesian networks (DBNs) extend the Bayesian network methodology to address temporal processes. DBNs are used to model discrete time processes that evolve over fixed time intervals. To do this, a set of variables is associated with each time interval, and the joint probability distribution over assignments of values to these variables is specified with a set of conditional independence assumptions as in a static Bayesian network. DBNs have the same strengths as static networks, and allow for arbitrary sets of variables, and arbitrary conditional independence assumptions.

The ability to model arbitrary sets of variables is highly desirable in ASR because it enables the construction of explicit models of speech generation and perception. There are several ways in which the expressive power of Bayesian networks can be used to model

speech generation, perhaps the most attractive of which is in the construction of articulatory models. In contrast to conventional models, where the atomic representational unit is the subword linguistic unit, articulatory models maintain an explicit representation of speech articulators such as the lips, tongue, jaw, velum, and glottis. These models have been used previously, e.g (Deng & Erler 1992), and have the advantage of naturally modeling pronunciation variability as a causal process.

As an example, consider Figure 1.2, which is adapted from (Deng & Sun 1994). This shows the expected positions of several speech organs as the phrase "ten cats" is pronounced. This type of information can be conveniently expressed in a Bayesian network by modeling a set of variables corresponding to the articulators. Once programs for inference and learning are written, it is easy to test different model structures, and learn model parameters.

1.2 Goals and Accomplishments

This thesis has both theoretical and computational goals. The main theoretical goal is to demonstrate how to structure DBNs in a way that is appropriate for speech recognition. This requires first creating DBNs that are able to achieve the functionality of HMMs, and then showing how to extend them to address the problems with current systems. The challenge of emulating an HMM is that the dynamic programming procedures that an HMM uses to consider all possible partitionings of a speech signal into subword linguistic units must be encoded declaratively in terms of variables and conditional probabilities. This is significantly different from the usual imperative way of accomplishing the task (see Chapter 6). The problem is exacerbated by the fact that interpretations of the speech signal must respect known facts about word pronunciations.

Once the basic machinery for emulating an HMM is in place, it is straightforward to address many current problems in ASR. A second theoretical contribution of this thesis is to show how to address speaking-rate variability, accent, gender, coarticulation, noise, and combined generative and perceptual models in a unified Bayesian network framework.

In order to build a working Bayesian network system for ASR, it is necessary to solve a number of algorithmic challenges, and the final theoretical contribution of the thesis is to present inference algorithms that are especially tailored to the speech recognition

application. This application imposes the following constraints:

- The routines must be extremely efficient, and in particular able to handle deterministic relationships between variables (i.e. cases where a variable's value is uniquely determined by its parents' values). This constraint stems from the necessity of encoding the deterministic constraints of pronunciation models.
- The routines must be able to efficiently handle variable length training utterances.
- The routines must be able to do online recognition, where words are recognized before an utterance is completed.

These issues are resolved in Chapter 3.

The main computational goal of this work was to implement a general Bayesian network system for ASR, and test it on a challenging database. This goal was achieved, and the thesis presents results that show that it is in fact practical to base an ASR system on Bayesian network technology, and that significant benefits accrue from using the technique. Chapter 7 presents results that indicate that modeling acoustic and articulatory context with a DBN reduces the word error rate by between 10 and 30%, depending on the exact conditional independence assumptions. The flexibility of the methodology is further demonstrated by presenting results with multiple context variables, and for networks that perform unsupervised utterance clustering.

1.3 Outline

This thesis is divided into two main parts. Chapters 2 through 4 present Bayesian networks as a general tool for modeling stochastic processes. Chapter 2 describes a range of current methods for stochastic modeling, and places Bayesian networks in that context. Chapter 3 describes in detail the algorithms that are necessary for inference and learning in Bayesian networks. In this chapter, special attention is given to the requirements imposed by the specific application of speech recognition, including the ability to do online recognition. Chapter 4 presents a set of experimental results that illustrate the benefits of using factored representations of probability distributions.

The second half of the thesis applies Bayesian networks specifically to the problem of speech recognition. Chapter 5 presents background material on speech recognition, and illustrates the use of the standard stochastic modeling techniques introduced in Chapter 2 in this area. Chapter 6 shows how to structure Bayesian networks specifically to do speech recognition. The chapter begins by showing how to encode finite state pronunciation models of the type used by HMMs and ANN/HMM hybrids in Bayesian networks. Then we present a set of Bayesian network structures designed to address different issues in ASR. Chapter 7 presents experimental results that indicate that significant benefits can accrue from more detailed models of speech generation.

Chapter 2

Probabilistic Models for Temporal Processes

2.1 Overview

The purpose of this chapter is to describe the main approaches that have been used to model stochastic processes in the past, and to describe the use of Bayesian networks in this context. The stochastic processes we will be concerned with generate a sequence of observable quantities, or observations, as they evolve over time in a non-deterministic way. Although stochastic processes occur in a large range of application areas, there are a number of common themes, and the modeling methods fall into a well-developed taxonomy. The principal distinctions are:

- Continuous-time vs. discrete-time processes. Continuous time processes occur naturally in many models of physical systems. Discrete time models can be used as approximations to continuous models, and also occur naturally in many areas of economics, communications and computer science.

- Use of hidden state. Many time-series modeling techniques work exclusively with observable quantities. More complex techniques posit the existence of a hidden underlying state, whose value determines in some way the observed quantities.

- Continuous-state vs. discrete-state processes. Again, when modeling physical sys-

tems, continuous state variables are often most appropriate, whereas discrete state variables are more appropriate in other areas.

- Continuous vs. discrete observables. This distinction is analogous to the dichotomy in hidden state types.

An example of a completely continuous stochastic process is the trajectory of a ball when thrown in the air and buffeted by the wind. An example of a completely discrete process is the sequence of dice rolls in a backgammon game. In this chapter, we will be concerned with hidden-state, discrete-time modeling techniques that are applicable to systems with both discrete and continuous variables. The standard techniques we will examine are hidden Markov models, Kalman filters, and neural networks.

A key feature of the methods we will study is that they all have probabilistic interpretations, and may be used to generate one or more of the following:

- The likeliest hidden state value(s) at each point in time.
- A marginal posterior distribution over the hidden state values.
- The probability of an observation sequence.

Furthermore, the models make the first-order Markovian assumption that the present state is conditionally independent of the entire past given the immediately preceding state. This enables a factorization of the joint distribution as the product of localized factors, each of which involves variables from no more than two time-slices.

A final key point of similarity is that the modeling techniques all have associated learning procedures by which their parameters can be adjusted. This adjustment is usually done according to the principle of maximum likelihood: the model parameters Θ are adjusted to maximize the probability that a collection of data D was generated by the model: $\Theta^* = \arg\max_\Theta P(D|\Theta)$, where Θ^* is the optimal set of parameter values. Neural networks differ from the other approaches in this respect because there are a variety of different criteria that are used.

We turn now to a more specific discussion of the different techniques.

2.2 Hidden Markov Models

Hidden Markov models (HMMs) are a powerful modeling technique for discrete state processes (Baum *et al.* 1970; Baker 1975; Jelinek 1976; Rabiner & Juang 1986). The basic idea of a hidden Markov model is that the observation sequence **o** is generated by a system that can exist in one of a finite number of states. At each time-step, the system makes a transition from the state it is in to another state, and the emits an observable quantity according to a state-specific probability distribution. More precisely, a hidden Markov model is defined by the following things:

1. A set of possible states $\mathcal{Q} = \bigcup_i q_i$.
2. A state transition matrix A where a_{ij} is the probability of making a transition from state q_i to state q_j.
3. A prior distribution over the state of the system at an initial point in time.
4. A state-conditioned probability distribution over observations. That is, a specification of $P(o|q_i)$ for every state and all possible observations.

The observation sequence modeled by the HMM may be either discrete or continuous in nature, but because of the transition matrix, the state space is required to be discrete. Hidden Markov models have been used in a wide variety of application fields, with great success. Examples include gene prediction, protein secondary-structure prediction, handwriting recognition, and speech recognition (Hu *et al.* 1996; Bengio *et al.* 1995; Karplus *et al.* 1997; Krogh *et al.* 1994; Levinson *et al.* 1983; Kulp *et al.* 1996).

The use of HMMs is well-illustrated by a (simplified) example from computational biology: the problem of predicting whether a region of DNA codes for a gene. The DNA in the chromosome of a higher animal falls into one of two categories: it either codes for a protein, and can be used by a cell as a template for constructing that protein, or it is extraneous with respect to protein coding. The former regions are referred to as exons, and the latter as introns. Introns are "spliced out" of a DNA strand in the process of transcription. The ability to recognize exons is significant to biologists because it allows them to identify and study regions of biological significance. An HMM can be used to model this distinction by assuming that the DNA sequence is generated by a system that

essentially acts like a typist. The system can either be in the state of "typing out" a gene, or of "typing out" a non-coding region. When in the gene-producing state, base pairs from the set $\{A, C, T, G\}$ are emitted with characteristic frequencies. When in the intron state, the characteristic frequencies are different. The HMM is "trained" to learn these characteristic frequencies, and the probability of switching from one region to another, with examples of DNA where the coding and non-coding regions are known. Using this information, the HMM can find the likeliest partitioning of an unknown sequence into coding and noncoding regions. A more sophisticated approach that has been found to produce good results in practice can be found in (Kulp *et al.* 1996).

2.2.1 Variations

The HMM methodology has been quite successful, and this is indicated by a large number of variations that have been explored. One approach, used by researchers at IBM, is to associate output distributions with transitions, rather than states. Ostensibly, this has the effect of squaring the number of output distributions; in fact, the two approaches are formally equivalent (Jelinek 1997).

The assumption of time-invariant transition probabilities implies an exponentially decreasing a-priori distribution over durations, but in cases where this is undesirable, it is possible to explicitly model the state durations. A particularly elegant parametric representation based on the gamma distribution is discussed in (Levinson 1986); the gamma function looks like a skewed Gaussian, and is defined for positive durations. Related work is presented in (Russell & Moore 1985). Although more sophisticated transition probabilities can give a better fit to the data, it is often the case that the behavior of the model is dominated by the observation probabilities.

Another important variation deals with the modeling of autoregressive observation sequences. The assumption behind autoregressive HMMs (Poritz 1982) is that it is reasonable to model the output y_t at time t as a linear combination of the immediately preceding values. The precise assumption is that the observation stream is real-valued, and $y_t = \sum_1^k a_k y_{t-k} + u_t$. The term u_t represents a normally distributed error term, and the a_i are autoregressive coefficients. Essentially, this model tries to predict the current observation from the past k observations. Since the errors are assumed to be normally

distributed with some standard deviation σ, the probability of a particular error can be computed as $\frac{1}{\sqrt{2\pi\sigma}} exp(u_t/2\sigma^2)$. The errors are also assumed to be independent and identically distributed, so that the probability of a sequence of observations can be computed from the product of their individual probabilities. The idea behind an autoregressive HMM is to associate a set of predictor coefficients with each state, and compute the observation probability from the prediction errors. This type of model is extended to autoregressive mixtures in (Juang & Rabiner 1985).

This sampling of HMM variations shows that one must be careful in defining an HMM. That said, in this thesis, the term HMM will be used to refer to the "plain vanilla" kinds of HMMs described in early papers (Baum *et al.* 1970; Baker 1975; Jelinek 1976), and now defined in standard texts (Rabiner & Juang 1993; Deller *et al.* 1993; Lee 1989; Jelinek 1997). The only significant difference between the formulations found in these sources is the question of state vs. transition emissions, which is universally agreed to be irrelevant.

2.2.2 Algorithms

We now turn to the algorithms that are available for use with HMMs. Denote a fixed length observation sequence by $\mathbf{o} = (o_1, o_2, \ldots, o_n)$ and a corresponding state sequence by $\mathbf{q} = (q_1, q_2, \ldots, q_n)$. An HMM defines a joint probability distribution over observation sequences as follows:

$$\begin{aligned} P(\mathbf{o}) &= \sum_{\mathbf{q}} P(\mathbf{q})P(\mathbf{o}|\mathbf{q}) \\ &= \sum_{\mathbf{q}} P(q_1)P(q_2|q_1)\cdots P(q_n|q_{n-1})P(o_1|q_1)P(o_2|q_2)\cdots P(o_n|q_n) \\ &= \sum_{\mathbf{q}} P(q_1)P(o_1|q_1)\prod_{i=2}^{n} P(q_i|q_{i-1})P(o_i|q_i) \end{aligned}$$

The value of $P(q_i|q_{i-1})$ is specified in the state transition matrix, and the value of $P(o_i|q_i)$ is specified by the observation distributions associated with the HMM. We denote the assertion that the state of the system at time t was q_i by $Q_t = q_i$. There are efficient algorithms (Rabiner & Juang 1986) for computing the following quantities:

1. $P(\mathbf{o}) = \sum_{\mathbf{q}} P(\mathbf{o}, \mathbf{q})$: the probability of an observation sequence.

2. $\arg\max_{\mathbf{q}} P(\mathbf{o}, \mathbf{q})$: the likeliest hidden state sequence given an observation sequence.

3. $\arg\max_{\Theta} P(\mathbf{O}|\Theta)$: the optimal model parameters in the maximum likelihood sense.

4. $P(Q_t = q_i|\mathbf{o}),\ \forall t, i$: the marginal distribution over states at time t given an observation sequence.

Since the algorithms themselves are well known, we do not present them here, and note only that the running time is in all cases proportional to $n|\mathcal{Q}|^2$.

2.3 Kalman Filters

Kalman filters were developed in the 1960s to address the problem of estimating the state of a process with continuous hidden state variables. The paradigmatic use of Kalman filtering is to infer the position of an airplane from a sequence of imperfect radar measurements. A Kalman filter (Kalman 1960) assumes a stochastic process of the following kind:

$$\mathbf{q}_{t+1} = A\mathbf{q}_t + \nu_{\mathbf{t}}$$

$$\mathbf{o}_t = C\mathbf{q}_t$$

The state and observation variables, $\mathbf{q}_t$ and $\mathbf{o}_t$ are real-valued vectors, while A and C are real-valued matrices; A governs the evolution of the state variables, while C relates the state variables to the observations. The quantity $\nu_{\mathbf{t}}$ is assumed to be drawn from a Gaussian noise source with zero-mean and a fixed variance. There are many variations on the exact mathematical formulation, e.g. those found in (Goodwin & Sin 1984; Anderson & Moore 1979).

To give the flavor of the matrices associated with Kalman filters, $\mathbf{q}$ might consist of entries for position, velocity, and acceleration; in this case, A would encode the Newtonian equations relating successive values for these quantities over a small time increment. Kalman filters have found widespread use in fields as diverse as engineering and economics (Dattellis & Cortina 1990; Blanchet *et al.* 1997; Manohar Rao 1987). In contrast to HMMs, Kalman filters are most relevant when the hidden state is most naturally described by continuous variables. Kalman filters are also distinctive because they factor the hidden state into a combination of quantities, as indicated by the vector nature of the hidden state variable.

Algorithms exist for computing the same quantities that can be computed with a HMM (Goodwin & Sin 1984; Anderson & Moore 1979). Moreover, the algorithms are highly efficient with running times proportional to the amount of time required to multiply two d by d matrices, where d is the larger of the state and observation dimensions. Although it is not immediately obvious that Kalman filters can be applied to problems with a discrete-state component, we will see in chapter 5 that such an extension is possible.

2.4 Neural Networks

Over the course of the last decade, neural networks have found widespread use in time-series modeling (Haykin 1994; Haykin 1996; Hertz *et al.* 1991). In the following sections, we will describe three examples. The first two are based on extensions to multi-layer perceptrons (MLPs) using sigmoidal nonlinearities, while the third is based on the use of Gaussian radial basis functions. In order to understand the application of MLPs to temporal processing, we will begin with a brief description of their functioning for problems of static input-output mappings. For further details of these algorithms, the reader is referred to the original articles, or to (Haykin 1994), which provides thorough summaries, and on which the next few sections are based.

2.4.1 Multi-Layer Perceptrons

The basic unit of an MLP is the *perceptron*. A perceptron can be thought of as a nonlinear function, or processing unit, taking k real-valued inputs and producing a real-valued output. We will refer to the inputs by the vector $\mathbf{y}$, and the output as v. The output of a perceptron is given by $v = \varphi(\mathbf{w} \cdot \mathbf{y})$, where $\mathbf{w}$ is a real-valued "weight" vector, and $\varphi(x)$ is a nonlinear function, typically sigmoidal: $\varphi(x) = \frac{a}{1+exp(-x)}$, where a is a constant and often 1. In the case that there are several different perceptrons each receiving the same input, we will denote them by $v_i = \varphi(\mathbf{w_i} \cdot \mathbf{y})$; note that each has its own weight vector.

An MLP consists of several "layers" of perceptrons (see Figure 2.1). Each layer consists of a collection of perceptrons, and the output of one layer forms the input to the next. Thus, a single layer can be thought of as a function mapping an input vector $\mathbf{y}$ into a k-dimensional output vector $\mathbf{v}(\mathbf{y})$. The function is given by $\mathbf{v}(\mathbf{y}) = (v_1(\mathbf{y}), v_2(\mathbf{y}) \ldots, v_k(\mathbf{y}))$. In the case that there are several different layers, we will denote them by $\mathbf{v}_i(\mathbf{y})$, each of

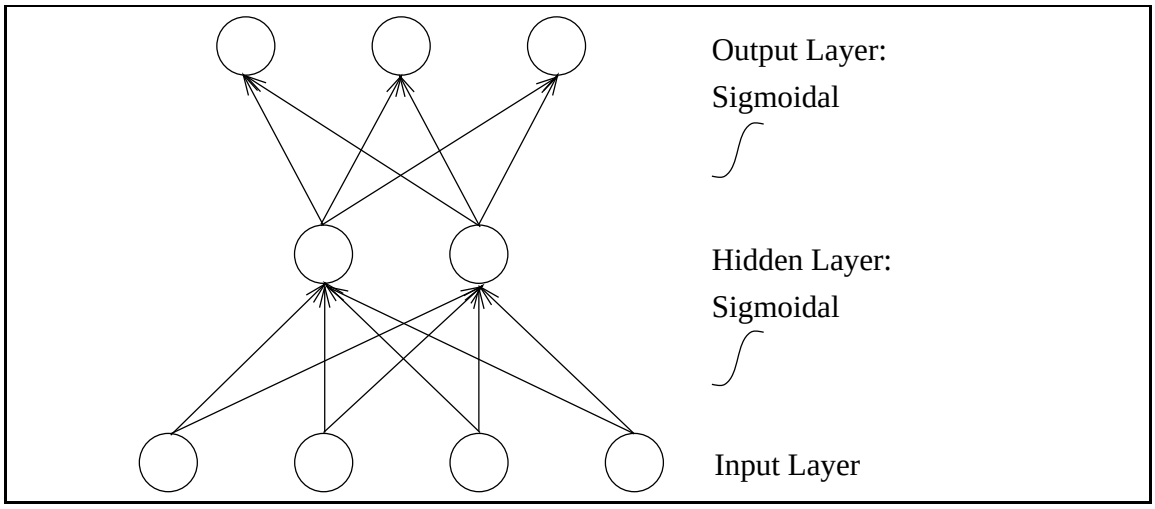

Figure 2.1: A MLP with one hidden layer. The nodes are typically fully interconnected between layers.

which defines a distinct function. A f layer MLP relates its input $\mathbf{y}$ to its output $\mathbf{z}$ by the composition of the functions defined by each of its layers: $\mathbf{z}(\mathbf{y}) = \mathbf{v}_f(\mathbf{v}_{f-1}(\cdots(\mathbf{v}_1(\mathbf{y}))))$.

The specification of an MLP consists of the number of layers, the number of perceptrons in each layer, and the weight vectors of each perceptron. Training consists of adjusting the weight vectors so as to minimize the discrepancy between the network output and some desired output for a set of training examples. Let z_e^i be the value of the ith output perceptron on the eth example, and let t_e^i be the target value. There are two commonly used measures of discrepancy or error E:

1. Least squares: $E = \sum_e \sum_i (z_e^i - t_e^i)^2$.
2. Cross entropy: $\sum_e \sum_i t_e^i log(\frac{t_e^i}{z_e^i}) + (1 - t_e^i) log(\frac{1-t_e^i}{1-z_e^i})$. This is appropriate when the target values represent a probability distribution.

The standard learning techniques (Haykin 1994; Hertz *et al.* 1991; Bishop 1995) work by doing gradient descent in weight space so as to minimize the error.

2.4.2 Finite Impulse Response MLPs

Finite impulse response MLPs, or FIR-MLPs, are a simple generalization of MLPs in which a standard MLP is presented with a succession of input vectors, and each of the

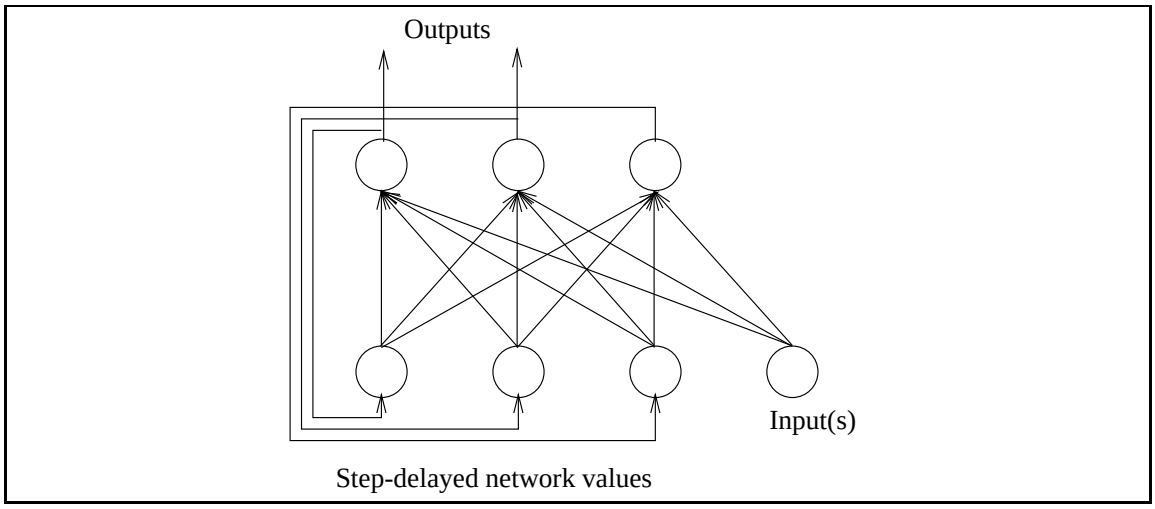

Figure 2.2: A RTR-NN.

constituent perceptrons is endowed with a memory of its previous input values (Wan 1990; Haykin 1994). A temporal interpretation of this model results when successive input vectors correspond to successive instants in time.

More specifically, each perceptron remembers the last p input values presented to it, so that the input at time t is effectively the concatenation of $y_{t-p}, y_{t-p+1}, \ldots y_{t-1}, y_t$. The weight matrix is correspondingly enlarged. Algorithms for training FIR-MLPs can be found in (Wan 1990; Haykin 1994).

FIR-MLPs are of interest because they are a general form of temporal modeling that, in restricted form, has found widespread use in speech recognition. Examples of this kind of network include the time-delay neural networks of (Lang & Hinton 1988; Waibel *et al.* 1989), and the MLPs described in (Bourlard & Morgan 1994). We will discuss these applications in more detail in Chapter 5.

2.4.3 Recurrent NNs

The second broad class of neural networks for temporal processing are real-time recurrent neural networks (RTR-NNs) of the type first described in (Williams & Zipser 1989). Restricted versions of this kind of network have also found important application in speech recognition (Robinson & Fallside 1991). The basic idea of a RTR-NN is to maintain a single layer of hidden nodes with sigmoidal nonlinearities; some of these nodes represent

the output of the network, and others are used solely to encode state information. The input to the hidden layer consists of an input vector concatenated with the values of the hidden nodes at the previous time step. This scheme is illustrated in Figure 2.2. The method of (Robinson & Fallside 1991) is the same, except that the output nodes do not feed back into the network.

RTR-NNs are similar to Kalman filters in that they maintain a real-valued hidden state vector, whose value at time t is a function both of its previous value and some new input to the system. RTR-NNs are significantly different, however, because whereas Kalman filters were developed to model a "dynamic system excited by an independent Gaussian random process," (Kalman 1960), the input vector to a RTR-NN is not assumed to be Gaussian noise, and the state at time t is related in a highly nonlinear way to both its previous value and the current input vector.

2.4.4 Radial Basis Function Networks

A radial basis function neural network (RBF-NN) works along completely different lines from the sigmoid-based networks, but can also be used for temporal modeling. The distinguishing feature of RBF-NNs is that they are based on localized nonlinear functions, typically Gaussians. The idea is to form a two-layer network (see Figure 2.3) in which the nodes in the first layer take an input vector and compute Gaussian activations (Moody & Darken 1989):

$$v_i(\mathbf{y}) = \frac{\frac{1}{\sqrt{2\pi\sigma_i}} exp(-(\mathbf{y} - \mu_\mathbf{i})^2/(2\sigma_i^2))}{\sum_j \frac{1}{\sqrt{2\pi\sigma_j}} exp(-(\mathbf{y} - \mu_\mathbf{j})^2/(2\sigma_j^2))}.$$

The values of the nodes in the second layer form the output of the network; each node in this layer simply computes a linear combination of the values of the first-layer outputs:

$$z_j(\mathbf{y}) = \sum_i V_i(\mathbf{y}) w_{ij},$$

where $z_j(\mathbf{y})$ is the jth output and w_{ij} is a scalar weighting factor.

Intuitively, the operation of a RBF-NN is easy to understand. The input space is partitioned into regions by the Gaussian centers, and stereotypical output values for each region of the input space are stored by the weights connecting the hidden nodes to the output nodes. To the extent that more than one Gaussian is activated, the input vector belongs in more than one region, and the final output is determined by linear interpolation.

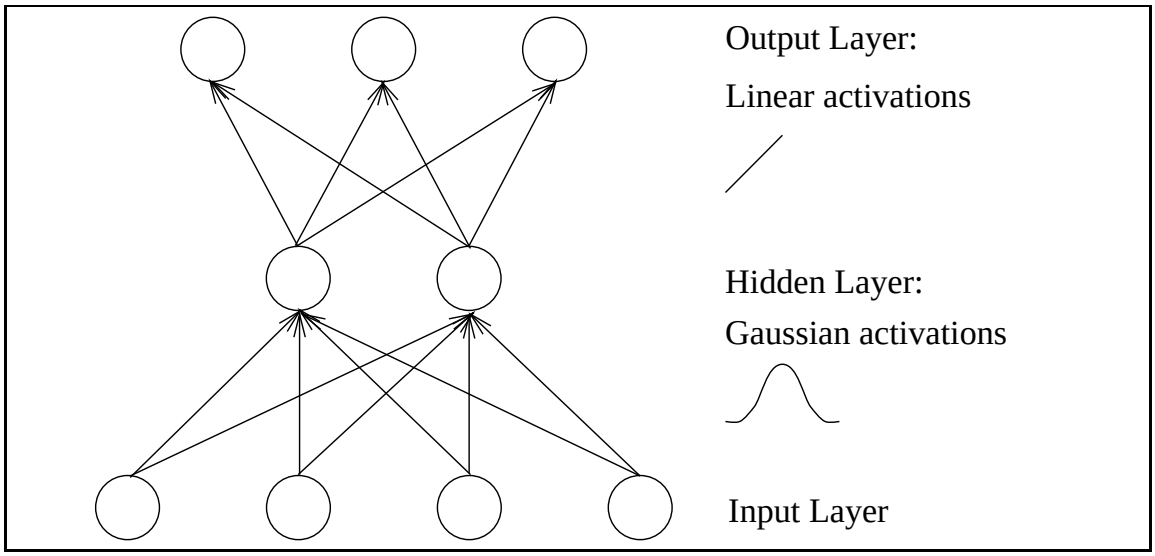

Figure 2.3: A radial basis function neural network. The first layer computes Gaussian activations while the second layer is linear.

One of the main advantages of RBF-NNs is the ease with which they can be trained. The means and variances of the Gaussians can be found by EM (Bishop 1995), and the weights on the second layer can be found by setting up and solving (in the least squares sense) a system of simultaneous equations that relate the Gaussian activations to the target outputs for each of the input vectors (Haykin 1994; Bishop 1995).

RBF-NNs can be used for time-series modeling by treating each of the sequences in the training data as a multiplicity of static training examples. For example, consider the time-series $x_1, x_2, \ldots, x_n$, and suppose we want to predict x_k from $x_{k-p} \ldots x_{k-1}$. We can turn this into something amenable to a RBF-NN by creating a distinct training pattern from each point-in-time: $((x_{j-p}, \ldots, x_{j-1}), x_j), n \geq j > p$. The network can be trained to predict these static patterns, and then used to predict unknown values from unseen segments of a similar time series. This method has been used by (Moody & Darken 1989) to predict chaotic time-series, and improved on by (Stokbro *et al.* 1990).

2.5 Dynamic Bayesian Networks

Before turning to Bayesian networks, we pause to consider the methods for temporal processing discussed so far. While all the methods maintain a hidden state represen-

tation and operate in the discrete time domain, there are very significant differences and limitations. It is convenient to consider these along the axes of linearity, interpretability, factorization, and extensibility.

Linearity. The Kalman filtering technique is fundamentally linear: it assumes that successive states are related by a linear transform, and that the state and observation variables are related by a linear transform. Although various schemes have been developed to model nonlinear systems with Kalman filters (Anderson & Moore 1979), they tend to be complex and of limited applicability. In contrast, both HMMs and NNs are naturally suited to model nonlinear processes. In HMMs, this capability derives from the arbitrary conditional probabilities that can be associated with both the transmission and emission matrices, or with functional representations thereof. In the case of NNs, it derives from the use of nonlinear activation functions.

Interpretability. The Kalman filter is probably the most interpretable of the modeling techniques we have discussed. In many applications, the matrices involved are designed by hand to reflect known physical laws. The parameters associated with HMMs are interpretable in so far as they are clearly labeled as "transition" or "emission" probabilities, but the states of an HMM do not always have a clear interpretation, especially after training. Neural networks are the least interpretable because often the hidden units are not assigned any meaning, either before or after training. There are exceptions, however, see e.g. (Towell & Shavlik 1993).

Factorization. There is wide variation in the degree of factorization imposed by the different modeling techniques, and the variability is increased by the degree to which one is willing to modify "plain vanilla" systems. The simplest case to deal with is Kalman filters, where the vectorized state and observation representation are inherently factored. To the extent that the matrices are sparse, the factorization also leads to a reduced number of parameters.

Basic neural networks are factorized in the sense that state is represented in a distributed fashion by a large number of nodes; but, if there is complete interconnection between the nodes in successive layers, the number of parameters is quadratic in the number of states, and scalability is severely limited. (Pruning techniques and weight-decay can be used to counteract this: see, e.g. (Le Cun *et al.* 1990; Scalettar & Zee 1988).) A greater degree of structure can be imposed by breaking a large network into a combination of

smaller networks. For example, a system to recognize handwritten digits (Le Cun *et al.* 1989) decomposes the units in the hidden layers into several separate groups, and does not use a complete interconnection between layers. Moreover, the weights of different groups are constrained to be the same (i.e. there is parameter tying), further reducing the number of free parameters. Hierarchical network construction algorithms (Frean 1990; Fahlman & Lebiere 1990) achieve a factored representation by carefully building a hierarchical structure in which the nodes in successive layers are carefully added to correct the mistakes of the previous layer; again, complete interconnection is avoided. The mixture of experts structure presented in (Jacobs & Jordan 1991; Jacobs *et al.* 1991) is similar: small neural networks can be trained as local "experts," and their outputs combined in a principled way to form the output of the entire system. In (Jordan 1992), this scheme is extended to hierarchically organized networks of experts.

In the field of speech recognition, factored neural net approaches have been used by a number of researchers. In (Morgan & Bourlard 1992), a method is presented for factoring a neural net so that it computes $P(A, B|C)$ as

$$P(A, B|C) = P(A|C)P(B|A, C).$$

A separate neural net is used to compute each of the factors, and this scheme reduces the number of parameters in the output layer, without requiring statistical independence assumptions. This method is extended and applied to a large-scale speech recognition task in (Cohen *et al.* 1992); clearly, a factorization into more than two components is also possible. The work of (Fritsch 1997) uses a hierarchy of ANNs to represent a probability distribution in a factored way. These schemes indicate that parameter-reducing factorization techniques can be applied to neural networks.

In the standard definitions, HMMs are fundamentally unfactored: if the state of the system consists of a combination of factors, it cannot be represented concisely this in the methodology. With effort, however, it is again possible to create HMM systems in which the states implicitly represent the combination of multiple distinct pieces of information. This is the case in, for example, HMM-decomposition (Varga & Moore 1990) which implicitly models both a noise source and a speech source, and in the articulatory HMMs of (Deng & Erler 1992). It should be noted that although these schemes achieve a parameter reduction, there is no corresponding reduction in computational requirements.

Extensibility. Neural networks are extremely extensible, and can be proven to be universal function approximators; a simple explanation for this can be found in (Lapedes & Farber 1988). Kalman filters are also quite extensible because the state and observation variables are vectors; thus system complexity can be increased by increasing the dimensionality of these vectors. This flexibility is modulated, however, by the underlying assumption of linearity. Hidden Markov models are somewhat limited in their extensibility by the fact that the main way of increasing their complexity is simply to increase the number of states. This can be awkward when the overall state of the system is actually composed of a combination of separately identifiable factors.

In the following sections, we will see that Bayesian networks, and their temporal counterparts, dynamic Bayesian networks, combine most of the advantages of HMMs, Kalman filters, and NNs, while avoiding many of their limitations.

2.5.1 Bayesian Networks

In recent years, probabilistic or Bayesian networks (Pearl 1988) have emerged as the primary method for representing and manipulating probabilistic information in the AI community. These networks can be used to represent either static events, such as the co-occurrence of a set of diseases and symptoms, or to represent temporal processes such as the motion of an automobile in traffic.

A probabilistic network represents the joint probability distribution of a set of random variables $\{X_1, \ldots, X_n\}$. Denoting the assignment of a specific value to a variable by a lower-case letter, the probability of a joint assignment of values is specified with the chain rule and a set of conditional independence assumptions as: $P(x_1, \ldots, x_n) = \prod_i P(x_i | Parents(X_i))$. Here $Parents(X_i)$ refers to a subset of the variables $X_1 \ldots X_{i-1}$; given values for its parents, X_i is assumed to be conditionally independent of all other lower-indexed variables. The conditional probabilities associated with each variable are often stored in tables referred to as CPTs. A Bayesian network has a convenient graphical representation in which the variables appear as nodes, and a variable's parents are specified by the arcs leading into it, see Figure 2.4.

As an example of a Bayesian network, consider Figure 2.4. This network relates asbestos exposure to medical symptoms that can be observed, through two underlying dis-

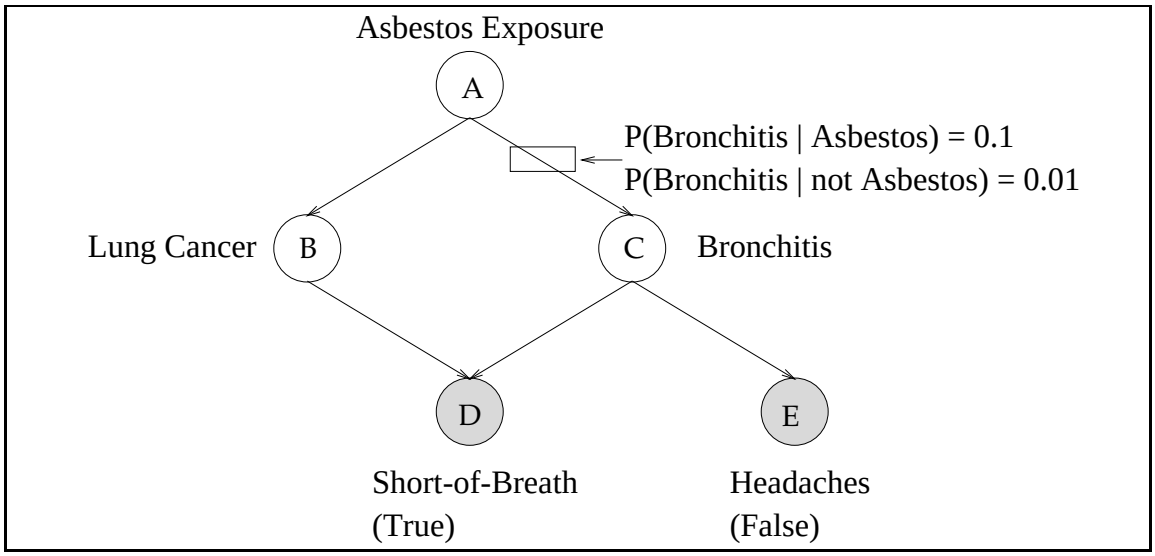

Figure 2.4: A Bayesian network. The shaded nodes represent variables whose values are observed. Each variable has an associated conditional probability table (or equivalent functional representation) that specifies a distribution over values, conditioned on the values of the variable's parents.

eases. The set of variables in this case is: "asbestos exposure," "lung cancer," "bronchitis," "shortness-of-breath," and "headaches." These are all binary variables, though in general the variables can take many values or be continuous. For referential convenience, the variables have also been given single-letter abbreviations. The factorization that this network encodes is:

$$P(a, b, c, d, e) = P(a)P(b|a)P(c|a)P(d|b, c)P(e|c).$$

As with the other techniques we have discussed, there are well-known algorithms for computing with Bayesian networks (Pearl 1988; Heckerman 1995), and these procedures will be covered in more detail in Chapter 3.

It is usually the case that knowledge of a variable's parents does not completely determine the value of the variable; we refer to such variables as *stochastic*. There are important exceptions, however, where a variable's parents completely determine its value, and we refer to such variables as *deterministic*. When this is the case, large gains in efficiency can result from using a sparse encoding of the conditional probabilities; this will emerge as an important issue in the application of Bayesian networks to speech recognition in Chapter 6.

2.5.2 Dynamic Bayesian Networks

In the dynamic case, a probabilistic network models a system as it evolves over time (Dean & Kanazawa 1988). At each point in time, a set of variables $X_1, \ldots, X_n$ are of interest. For example, to model car-driving, lane-position and speed are relevant. A DBN uses a set of variables X_i^t to represent the value of the ith quantity at time t. DBNs are also time-invariant so that the topology of the network is a repeating structure, and the CPTs do not change with time. The joint probability distribution is then represented as $\prod_{i,t} P(x_i^t | Parents(X_i^t))$. In networks with the first-order Markov property, the parents of a variable in timeslice t must occur in either slice t or $t-1$. The conditional distributions within and between slices are repeated for all $t > 0$, so that DBNs can be specified simply by giving two slices and the links between them. When applied to an observation sequence of a given length, the DBN is "unrolled" to produce a probabilistic network of the appropriate size to accommodate the observations.

The top of Figure 2.5 illustrates a generic DBN unrolled to show five time steps. The variables are divided into state variables, whose values are unknown, and observation variables, whose values are known. The state variables evolve in time according to a model encoded in their CPTs, which we refer to as the state evolution model. The state variables are related to the observation variables by the CPTs of the observation nodes. The bottom of Figure 2.5 shows a more realistic DBN in which the hidden state has been factored.

In this example, the observation stream is conditioned on the hidden state variable(s), and the same model could be expressed implicitly, but with less computational efficiency, as a factored HMM. It should be noted that this is not always the case; for example, with a DBN it is possible to condition the hidden state variables on the observations, thus resulting in a fundamentally different model.

2.5.3 Strengths of DBNs

Dynamic Bayesian networks are ideally suited for modeling temporal processes. In terms of the qualities mentioned earlier, DBNs have the following advantages:

1. Nonlinearity. By using a tabular representation of conditional probabilities, it is quite easy to represent arbitrary nonlinear phenomena; moreover, it is possible to do efficient

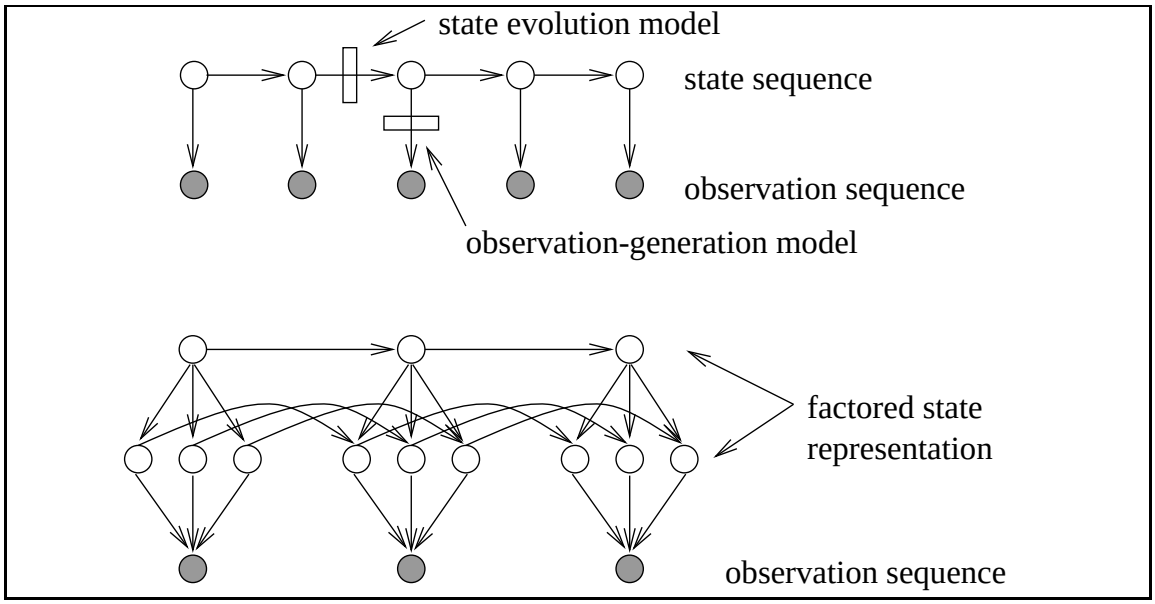

Figure 2.5: Top: A simple DBN, "unrolled" to show five time steps. Bottom: A DBN with a factored state representation. The factored representation can describe the evolution of an equal number of total states with exponentially fewer parameters.

computation with DBNs even when the variables are continuous and the conditional probabilities are represented by Gaussians - see, e.g. (Shachter & Kenley 1989).

2. Interpretability. Each variable represents a specific concept.

3. Factorization. The joint distribution is factorized as much as possible. This leads to:

   - Statistical efficiency. Compared to an unfactored HMM with an equal number of possible states, a DBN with a factored state representation and sparse connections between variables will require exponentially fewer parameters.
   - Computational efficiency. Depending of the exact graph topology, the reduction in model parameters may be reflected in a reduction in running time.

4. Extensibility. DBNs can handle large numbers of variables, provided the graph structure is sparse.

Finally, DBNs have a precise and well-understood probabilistic semantics. The combination of theoretical underpinning, expressiveness, and efficiency bode well for the future of DBNs in many application areas.

2.6 Discussion

This chapter has presented thumbnail sketches of several important techniques for modeling stochastic processes. For comprehensibility, it is useful to present somewhat stereotyped descriptions of the different methods, thus exaggerating their differences. It is also important to realize that the methodologies are neither mutually exclusive, nor completely distinct. In its simpler forms, the DBN framework merges into the HMM framework; conversely, as implicit factorization is added to the HMM framework, it blends with the DBN methodology. Hybrid approaches are also possible, for example one might encode the conditional probabilities required by a DBN with a small neural network; the resulting system would then consist of a large number of relatively small neural networks organized with a coherent large-scale structure, and endowed with a natural probabilistic semantics. This is similar to the approach proposed in (Fritsch 1997) for phone classification in speech recognition.

Chapter 3

Inference and Learning with DBNs

In this chapter, we present inference and learning algorithms for DBNs, with special attention to the requirements imposed by the speech recognition task. These requirements, which will be discussed more fully in Chapter 6 are:

1. that the procedures be extremely efficient in networks that include both stochastic and deterministic variables, and
2. that variable-length observation sequences are dealt with efficiently.

Although there are well-known procedures for doing inference in Bayesian networks, e.g. (Pearl 1988; Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990), we present algorithms in some detail because

- the special requirements of speech recognition have not been dealt with before, and
- the simple dynamic programming formulation we present is more appropriate for implementation than the usual message-passing formulation.

The algorithm we use for inference in a tree is an improvement on that presented in (Peot & Shachter 1991). Inference in general graphs is most often done by clustering together groups of variables in the original graph into "cliques." These cliques are then joined together into a tree structure known as a clique tree, and a special set of inference routines are derived (Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990). In this chapter, we present a novel derivation of the clique tree procedure that retains the simple tree-inference

algorithm, and uses a change of variables to convert an arbitrary Bayesian network into an equivalent tree-structured one. The derivation proceeds to specific algorithms from an axiomatic statement of the requirements that must be satisfied for two Bayesian networks to represent the same probability distribution.

The main points of this chapter are:

1. A simple statement of inference in a tree in terms of dynamic programming. This is lacking in the literature.
2. A novel derivation of clique tree inference in terms of a change of variables.
3. A novel method for propagating the constraints of deterministic variables through a clique tree so that inference with deterministic variables is highly efficient. In contrast to previous methods, this procedure exploits evidence on a case-by-case basis. Triangulation routines that enable the procedure are presented.
4. Novel procedures for handling variable length observation sequences are presented. These procedures are useful for offline inference and learning.
5. A novel method for online inference is presented. This scheme can easily be combined with beam-search to maintain a small set of highly likely hypotheses in an online manner.

Where relevant, we present a short comparison with analogous issues in HMMs; this comparison brings the issues into sharper relief, and clarifies the differences between the methodologies.

3.1 Inference on a Tree

This section presents an inference algorithm for the case where the variables in a Bayesian network are connected together in a tree-structured graph.

3.1.1 Definitions

For each variable X_i we define three mutually exclusive sets of assignments: $\mathbf{e}_i^0, \mathbf{e}_i^-$, and $\mathbf{e}_i^+$. $\mathbf{e}_i^0$ is the observed value of X_i in the case that X_i is an evidence variable. $\mathbf{e}_i^-$ is the

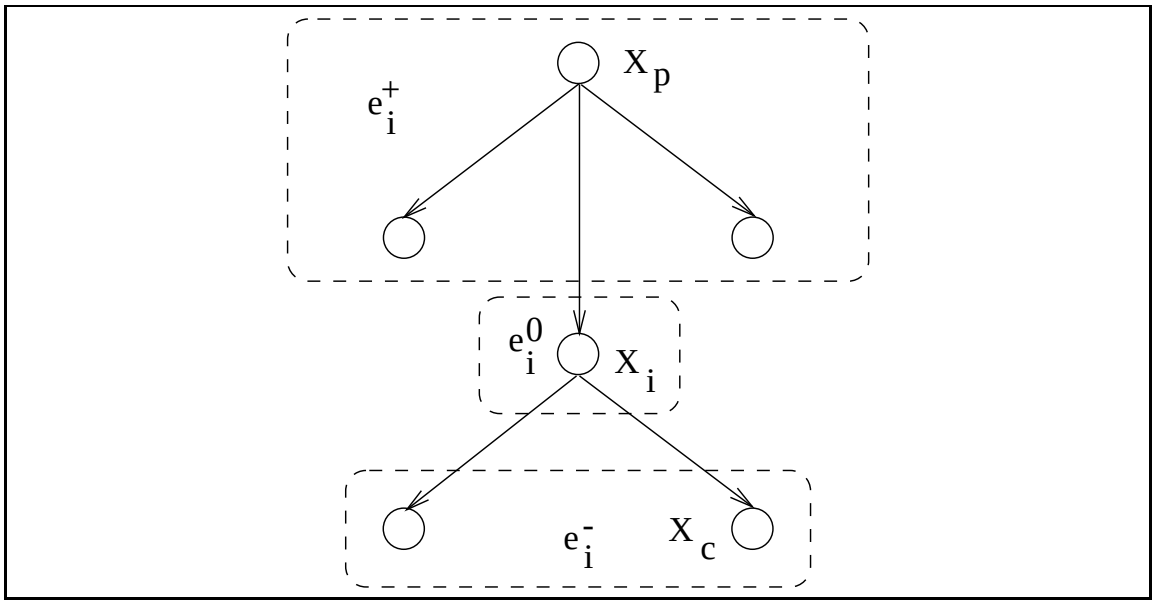

Figure 3.1: A tree of variables. The partitioning of the evidence is shown for X_i.

set of observed values for the evidence variables in the subtrees rooted in X_i's children. $\mathbf{e}_i^+$ is the set of observed values for all other evidence variables. The partitioning is shown in Figure 3.1. In the case that $\mathbf{e}_i^0 \neq \emptyset$, an assignment $X_i = j$ is consistent with the evidence if it matches the assignment in $\mathbf{e}_i^0$, and all other assignments are inconsistent. If $\mathbf{e}_i^0 = \emptyset$, we say that all values of X_i are consistent with the evidence. We denote the set of values for X_i that are consistent with the evidence by $CON(i)$.

Note that

$$\begin{aligned} P(\mathbf{e}, X_i = j) &= P(\mathbf{e}_i^0, \mathbf{e}_i^-, \mathbf{e}_i^+, X_i = j) \\ &= P(\mathbf{e}_i^+, X_i = j) P(\mathbf{e}_i^-, \mathbf{e}_i^0 | \mathbf{e}_i^+, X_i = j) \\ &= P(\mathbf{e}_i^+, X_i = j) P(\mathbf{e}_i^-, \mathbf{e}_i^0 | X_i = j) \end{aligned}$$

If $X_i = j$ is inconsistent with the evidence, i.e. contradicts $\mathbf{e}_i^0$, then $P(\mathbf{e}_i^-, \mathbf{e}_i^0 | X_i = j) = 0$. In the inference procedure, the following two key quantities will be calculated for each variable X_i:

- $\lambda_j^i = P(\mathbf{e}_i^-, \mathbf{e}_i^0 | X_i = j)$
- $\pi_j^i = P(\mathbf{e}_i^+, X_i = j)$.

It follows from these definitions that

- $P(Observations) = \sum_j \lambda_j^i * \pi_j^i, \ \forall i.$
- $P(X_i = j|Observations) = \frac{\lambda_j^i * \pi_j^i}{\sum_j \lambda_j^i * \pi_j^i}, \ \forall i.$

Hence, once the λs and πs are calculated, we can determine the probability of the observations, and the marginal posterior probabilities for all of the variables.

3.1.2 Algorithm

The λ probabilities will be calculated in a bottom up pass over the tree, and then the π probabilities will be calculated from the λs in a top-down pass. The algorithm is given in Figure 3.2.

The likeliest assignment of values to the variables can be found with a simple modification to this algorithm. In the bottom-up pass, the λs are computed as before, except that the sum over f is replaced by a maximization over f. The maximizing f value is stored for each of the children and for each λ_j^i. Then the πs are calculated for the root as in the first step of the top-down pass. At this point, the likeliest assignment to the root variable X_r can be computed as $\arg\max_j \lambda_j^r * \pi_j^r$. By starting at the root and recursively looking up the maximizing values for the children, the remainder of the variables can be assigned values.

There are a couple of things to note. First, in the calculation of the λs, the sum over f can be modified to read $sum_{f \in CON(c)}$. This is because $\lambda_f^c = 0, \ \forall f \notin CON(c)$. This is advantageous since no values need to be stored for the λs that are inconsistent with the evidence. Secondly, π_j^i needs only be stored for $j \in CON(i)$, and can be implicitly assumed to be 0 otherwise. This will have no effect on any results because 1) in computations involving X_i itself, π_j^i will be multiplied by λ_j^i, which is 0 for $j \notin CON(i)$; and, 2) in the recursive computation of πs for X_i's children, inconsistent values are ignored. Again, a space savings can result from doing this.

3.1.3 Comparison with HMM Inference

Recall that HMMs deal with a set of states $\mathcal{Q} = \bigcup_i q_i$ and an observation sequence $o_1, o_2, \ldots o_n$. In HMM inference, the following quantities are computed (Rabiner & Juang

Algorithm **Inference**()

for each variable X_i in postorder

if X_i is a leaf

$\lambda_j^i = 1,\ \forall j$ *consistent with the evidence*;

$\lambda_j^i = 0,$ *otherwise*.

else

$\lambda_j^i = \prod_{c \in children(X_i)} \sum_f \lambda_f^c * P(X_c = f | X_i = j),\ \forall j$ *consistent with the evidence*;

$\lambda_j^i = 0,$ *otherwise*.

for each variable X_i in preorder

if X_i is the root

$\pi_j^i = P(X_i = j)$

else

let X_p be the parent of X_i

$\pi_j^i = \sum_{v \in CON(p)} P(X_i = j | X_p = v) * \pi_v^p * \prod_{s \in siblings(X_i)} \sum_f \lambda_f^s * P(X_s = f | X_p = v)$

Figure 3.2: Inference in a tree.

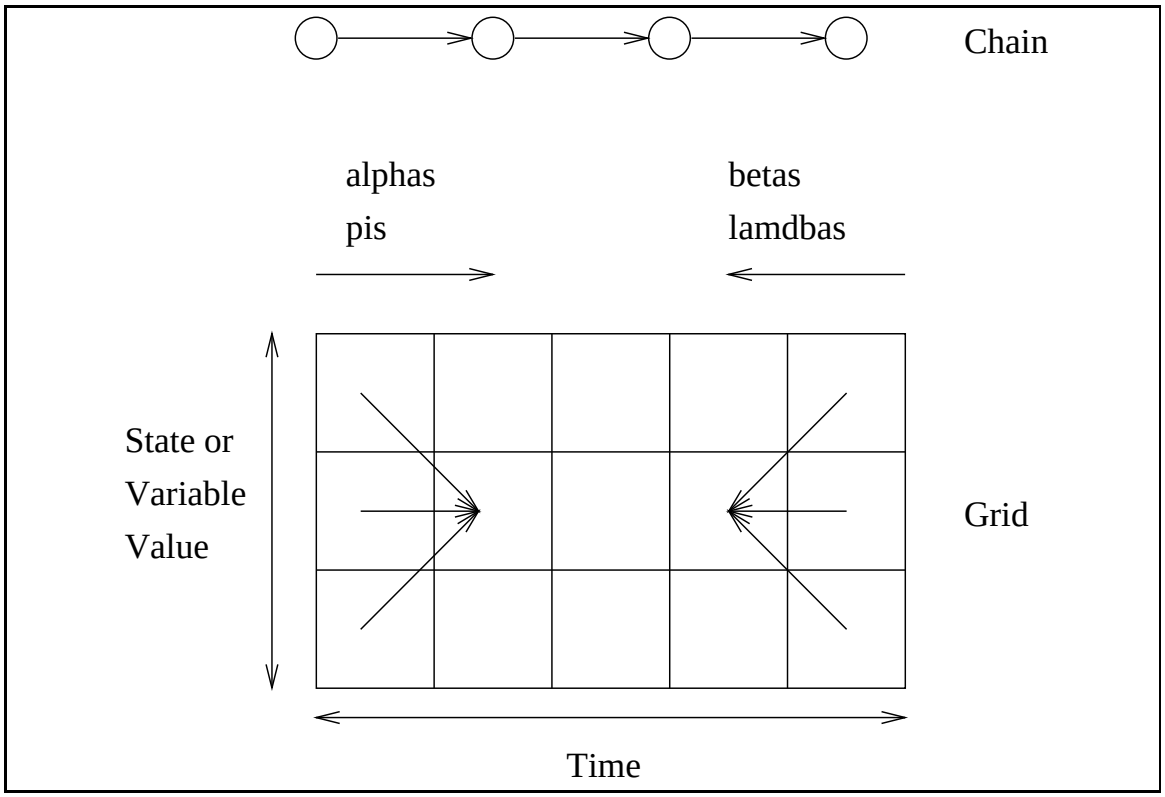

Figure 3.3: A chain structured graph. A two-dimensional grid is an adequate data structure for computing the λs and πs for a chain. In this case, the λs are analogous to HMM βs and the πs are analogous to αs. The diagonal arrows in the grid show the values that are used to compute the λ and π values for a particular cell.

1986):

$$\alpha_t(i) = P(o_1, o_2, \ldots o_t, q_t = i)$$

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots o_n | q_t = i).$$

These are analogous to the πs and λs respectively, with the difference that we have associated the evidence at time t with the λ rather than the π; this makes the Bayes net derivations slightly simpler, and is otherwise irrelevant. The bottom-up computation of the λs is analogous to the backwards β-recursion in HMMs, and the top-down computation of the πs is analogous to the forwards α-recursion. The analogy becomes precise for chain-structured Bayesian networks, when $\mathbf{e}_t^+$ corresponds with $o_1, o_2, \ldots, o_{t-1}$, $\mathbf{e}_t^-$ corresponds with $o_{t+1}, o_{t+2}, \ldots o_n$, and $\mathbf{e}_t^0$ corresponds with o_t. The inference procedures for Bayesian networks are essentially identical to those for HMMs when the underlying graph is a chain; when the graph is a real tree, however, and has side-branches emerging from the main backbone, the two are substantially different. The analogy between HMM and DBN inference is illustrated in Figure 3.3.

Viterbi decoding in an HMM is usually done with a modified forward recursion. In chain-structured DBNs, a similar procedure can be used. In general, however, the modified backward procedure is necessary. This is because the forward recursion in a tree-structured graph cannot proceed without already knowing the λs. The root cause of this is that for a variable "hanging" off the main backbone of a chain, $\mathbf{e}_i^+$ includes evidence from *all* timeslices. Therefore, the πs for this variable cannot possibly be computed without looking into the future. The implications of this are discussed further in Section 3.7.

3.1.4 A Speedup

Let X_p denote the parent of X_i. If we define τ_v^i as

$$\tau_v^i = \sum_f \lambda_f^i * P(X_i = f | X_p = v)$$

we may rewrite λ_j^i for non-leaf variables as

$$\lambda_j^i = \prod_{c \in children(X_i)} \tau_j^c$$

and π_j^i as

$$\begin{aligned}\pi_j^i &= \sum_{v \in CON(p)} P(X_i = j | X_p = v) * \pi_v^p * \prod_{s \in siblings(X_i)} \sum_f \lambda_f^s * P(X_s = f | X_p = v) \\ &= \sum_{v \in CON(p)} P(X_i = j | X_p = v) * \pi_v^p * \lambda_v^p / \tau_v^i\end{aligned}$$

Hence if the τ-factors contributing to each λ are stored in the bottom up pass, the products over siblings can be constructed with a division.

In the case that $\tau_v^i = 0$ in the calculation of π_j^i, no update is necessary. This is in contrast to the original algorithm of (Peot & Shachter 1991) which computes the product over siblings from scratch in this case. This change makes the code slightly simpler, and improves the running time when this case occurs.

3.1.5 Proof of Speedup

The proof of correctness is inductive. We begin by noting that a π computed by X_i is only used by X_i and its children, and therefore only relevant to those variables. We will show that

1. The only way an error can be made is in the calculation of a child's π value.
2. Either the child's πs are correctly calculated, or the affected terms have τ-factors which themselves are 0. This implies an inductive chain in which πs can only be miscalculated for leaf variables. But since errors can only occur in a child's π, and a leaf variable has no children, the calculation is sound.

The proof is given in more detail below.

Theorem 3.1 *The omission of terms for which* $\tau_v^i = \sum_f \lambda_f^i * P(X_i = f | X_p = v) = 0$ *is irrelevant.*

Proof

Without loss of generality, consider the calculation of π_j^i:

$$\pi_j^i = \sum_{v \in CON(p)} P(X_i = j | X_p = v) * \pi_v^p * \prod_{s \in siblings(X_i)} \sum_f \lambda_f^s * P(X_s = f | X_p = v).$$

Suppose a value of v is encountered for which

$$\tau_v^i = \sum_f \lambda_f^i * P(X_i = f | X_p = v) = 0.$$

First note that $\sum_f \lambda_f^i * P(X_i = f|X_p = v) = 0$ implies $\lambda_f^i * P(X_i = f|X_p = v) = 0$, $\forall f$. In particular, $\lambda_j^i * P(X_i = j|X_p = v) = 0$, so either $\lambda_j^i = 0$ or $P(X_i = j|X_p = v) = 0$.

If $P(X_i = j|X_p = v) = 0$, the product over siblings is irrelevant because it will be multiplied by 0. Omitting the term does not result in error, and neither values associated with X_i nor its children will be affected.

If $\lambda_j^i = 0$ we must consider first the effects of miscomputing π_j^i on future calculations regarding both X_i and its children. There are no effects on future calculations regarding X_i because whenever π_j^i is used (e.g. to calculate marginals) it will be multiplied by λ_j^i.

Thus far we have shown that omitting the term can only affect future calculations regarding X_i's children, and never calculations regarding X_i itself. This implies that if X_i is a leaf, the omission is safe. Furthermore, the children can only be affected when $\lambda_j^i = 0$.

Now suppose $\lambda_j^i = 0$ and consider a child X_c computing some π value π_w^c.

$$\pi_w^c = \sum_{v \in CON(i)} P(X_c = w|X_i = v) * \pi_v^i * \prod_{s \in siblings(X_c)} \sum_f \lambda_f^s * P(X_s = f|X_i = v)$$

Since we are considering the results of a miscomputation of π_j^i, the only effect on the computation of π_w^c occurs when $v = j$. If $j \notin CON(i)$, there is no contribution to the sum, so a miscomputed π_j^i is irrelevant. Otherwise, the term involved is

$$\begin{aligned} P(X_c = w|X_i = j) * \pi_j^i * \prod_{s \in siblings(X_c)} \sum_f \lambda_f^s * P(X_s = f|X_i = j) \\ = P(X_c = w|X_i = j) * \pi_j^i * \lambda_j^i / \tau_j^c \\ \equiv P(X_c = w|X_i = j) * \pi_j^i * \lambda_j^i / (\sum_f \lambda_f^c * P(X_c = f|X_i = j)) \end{aligned}$$

Since $\lambda_j^i = 0$, we know by its definition that

$$(\sum_f \lambda_f^c * P(X_c = f|X_i = j)) * \prod_{s \in siblings(X_c)} \sum_f \lambda_f^s * P(X_s = f|X_i = j) = 0$$

Either one of the factors over siblings is 0 or the parenthesized factor is 0.

If one of the factors over siblings is 0, the miscomputed value of π_j^i is multiplied by 0, rendering it inconsequential. In the case that the parenthesized factor is 0, we have discovered a term in the calculation of π_w^c for which the τ-factor is 0. This term will be omitted, and we have already shown that omitting it can only affect X_c's children. This establishes an inductive chain in which harmless omissions are made; the chain must terminate at a leaf variable, where the omissions are again harmless.

3.2 Inference in General Graphs

The algorithm presented in section 3.1 works only in tree-structured graphs. In this section, we present a technique for doing inference in graphs with arbitrary topology. The method of attack is to use a change-of variables. We will define a new tree-structured network in terms of a new set of variables in such a way that the new network represents exactly the same joint probability distribution as the old network. We will then be able to use the simple tree-inference algorithm.

3.2.1 Equivalent Representations of Probability Distributions

Suppose we have two Bayesian networks $\mathcal{N}$ and $\mathcal{N}'$ over the sets of variables $\mathbf{X}$ and $\mathbf{X}'$ respectively. Let $\mathbf{x}_i$ denote a joint assignment of values to the variables in $\mathbf{X}$, and let $\mathbf{x}_i^{'}$ denote a joint assignment of values to the variables in $\mathbf{X}'$. Let ξ be the set of all possible joint assignments of values to the variables in $\mathbf{X}$; i.e. $\xi = \cup_i \mathbf{x}_i$. Let $\xi^{'}$ be the set of all possible joint assignments of values to the variables in $\mathbf{X}'$; i.e. $\xi^{'} = \cup_i \mathbf{x}_i^{'}$. Assume without loss of generality that $\mid \xi^{'} \mid >= \mid \xi \mid$. Let ξ^* denote a subset of $\xi^{'}$ such that $\mid \xi^* \mid = \mid \xi \mid$. We will use P to represent probabilities associated with $\mathcal{N}$, and $P^{'}$ to represent those associated with $\mathcal{N}'$.

Theorem 3.2

$\mathcal{N}'$ represents the same distribution as $\mathcal{N}$ if the following conditions are met:

1. *There is a one-to-one correspondence between the members of ξ and the members of ξ^*. For notational convenience, let $\mathbf{x}_i$ be associated with $\mathbf{x}_i^*$.*

2. *For each such pairing, $P(\mathbf{x}_i) = P'(\mathbf{x}_i^*)$.*

3. For all joint assignments $\mathbf{x}_i^0 \in \xi' \setminus \xi^*$, $P'(\mathbf{x}_i^0) = 0$.

Proof. These conditions imply that the sum or max over any subset of ξ can be computed by performing the same operation on a well-defined subset of ξ'.

3.2.2 Inference with Trees of Composite Variables

In this section, we will derive the clique tree inference algorithms (Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990) from the principles set out in section 3.2.1. We will structure the derivation by satisfying each requirement of Theorem 3.2 in turn.

Correspondence Requirement

The variables in the tree will be defined to represent subsets of the variables in the original network. Each composite variable ranges over the Cartesian product of the variables in the subset associated with it. This creates the one-to-one mapping from each joint assignment in the original network to a joint assignment in the tree.

Equal Probability Requirement

We will now add a further stipulation that each variable in the original network must occur together with its parents in at least one of the composite variables. We may thus "assign" each variable in the original network to a composite variable C_i in which it occurs with its parents. Let $\mathbf{F}_i$ represent the variables in the original network that are assigned to C_i. Let C_p and C_c denote two composite variables in a parent-child relationship in the tree. Define $P'(C_c = i|C_p = j)$ to be $\prod_{X_k \in \mathbf{F}_i} P(x_k|Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i. In the case of a composite variable C_r with no parents, define $P'(C_r = i)$ in to be $\prod_{X_k \in \mathbf{F}_i} P(x_k|Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i. In the case that $\mathbf{F}_i = \emptyset$, the conditional probabilities are defined to be 1, unless i and j imply inconsistent values for shared variables, in which case the conditional probability is 0 (see below). These definitions ensure equality between the individual factors used to compute the probability of a joint assignment in the two representations, and thus guarantee that $P'(\mathbf{x}_i^*) = P(\mathbf{x}_i)$ for all $\mathbf{x}_i \in \xi$. Operationally the required condition can be ensured by the process conventionally known as "moralization".

Zero Probability Requirement

There are more possible joint assignments to the composite variables than to the original variables. However, each of the assignments to the composite variables for which there is no analog in the original network must imply the simultaneous assignment of inconsistent values to at least one of the original variables. (This follows from the one-to-one mapping in which there occurs a $\mathbf{x}_i^*$ for *every possible* $\mathbf{x}_i$. Thus any excess assignments in ξ' must correspond to invalid assignments in the original network.)

To satisfy the final requirement, we will impose two further stipulations.

1. We will define $P(C_c = i|C_p = j) = 0$ if i and j imply inconsistent values for shared variables.
2. We will require that the members of the composite variables satisfy the running intersection property: if two variables from the original network occur in any pair of composite variables, they also occur in every composite variable along the path connecting the two.

To see that this is sufficient, it suffices to realize that all inconsistencies must involve either:

1. two occurrences of an original variable in composite variables that are in a parent-child relationship, or
2. two occurrences of an original variable in composite variables that are separated by other composite variables.

The first requirement ensures that all inconsistencies of the first variety receive zero probability; the second requirement ensures that all inconsistencies of the second type imply an inconsistency of the first kind, and therefore receive zero probability. Operationally, the running intersection requirement can be satisfied by the process of triangulation (Rose 1970).

3.2.3 Summary of Inference in a Clique Tree

The variables in a clique tree represent subsets of the original variables. Algorithmically, the most natural way to view these subsets is as new variables. Each new clique-variable can take a distinct value for every possible assignment of values to its members.

Each variable in the original network is assigned to a single clique in which it occurs with its parents (when there are several such cliques, and an arbitrary choice may be made). The inference procedure outlined in Section 3.1 works on clique trees with the following change of variables:

- $\mathbf{e}_i^0$ is the set of observed values for the evidence variables assigned to clique C_i.
- $\mathbf{e}_i^-$ is the set of observed values for the evidence variables assigned to cliques in the subtrees rooted in C_i's children.
- $\mathbf{e}_i^+$ is the set of observed values for all other evidence variables.
- $\lambda_j^i = P(\mathbf{e}_i^- | C_i = j)$
- $\pi_j^i = P(\mathbf{e}_i^+, C_i = j)$.
- $\mathbf{F}_i$ is the set of variables assigned to C_i.
- $P(C_c = i | C_p = j) = 0$ if i and j imply inconsistent values for shared variables. Otherwise, if $\mathbf{F}_i \neq \emptyset$,
  - $P(C_c = i | C_p = j) = \prod_{X_k \in \mathbf{F}_i} P(x_k | Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i.
  - $P(C_r = i) = \prod_{X_k \in \mathbf{F}_i} P(x_k | Parents(X_k))$, with x_k and $Parents(X_k)$ implied by i.
- Otherwise, ($\mathbf{F}_i = \emptyset$), any remaining conditional probability is defined to be 1.

Separators

In an implementation, it is also beneficial to introduce separator variables between each pair of parent-child cliques. These separator variables represent the variables in $\{C_p \cap C_c\}$. The value of a separator's parent uniquely defines the separator's value, and the conditional probability of a separator taking its one allowed value given its parent's value is always 1. The inference algorithm remains unchanged; the separators simply represent extra variables which have the same status as any other variables. This view of separator cliques is quite different from that taken in (Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990) where separators play a fundamental role in representing the probability distribution.

The use of separators can have a dramatic effect on running time since the work done for each clique is proportional to the number of values it can take multiplied by the number of values its parent can take, subject to the consistency constraints imposed by shared variables. For example, suppose there is a network consisting of variables A, B, C and D, which take a, b, c and d values respectively. In a clique tree consisting of the cliques $\{A, B, C\}$ and $\{B, C, D\}$, the introduction of a separator-variable representing $\{B, C\}$ reduces the work from $a * b * c * d$ to $a * b * c \ + \ b * c * d$.

3.3 Fast Inference with Deterministic Variables

3.3.1 Motivation

We will see that modeling word pronunciations with a Bayesian network requires the extensive use of deterministic variables. In contrast to regular stochastic variables, deterministic ones impose significant constraints, and in order to achieve efficient inference, these must be addressed.

There will be a further complication in that the deterministic relationships will change on a word-by-word basis. Essentially, the same network structure will be "reprogrammed" on a word-by-word basis to do the dynamic programming that is appropriate for that word model. Thus, it is not possible to determine the repercussions of the deterministic relationships just once in a network-compilation stage, as is done typically, e.g. in the *HUGIN* system (A/S 1995). The *HUGIN* approach also suffers from the serious flaw that it first tabulates and stores every possible clique value, and then throws away the ones that can be proven to have zero probability. The space required for this approach can be prohibitive in networks with a large number of deterministic relationships.

A final point is that the possible clique values will in general vary depending on the pattern of evidence observed. Thus any static compilation scheme is bound to miss some efficiencies that a dynamic scheme will be able to identify and exploit on a case-by-case basis. Therefore, our approach is to structure the network so that the possible clique values can be swiftly enumerated on demand.

3.3.2 Approach

The basic approach is to do a preorder traversal of the tree and recursively identify the values that are legal for a child clique, given the just-computed legal values of the parent. Since neighboring cliques typically share variables, it is usually the case that knowing the possible values of the parent clique significantly reduces the set of possible values for a child clique.

Examination of the inference procedures shows that all the loops are of the form "for each value of a parent clique and for each value of a child clique, do a handful of multiplications and additions." Thus, once the legal values of a child are identified for each of the possible parent values, the inference loops can be made considerably more efficient by iteration just over the subset of values that are possible, given the known facts and deterministic relationships.

Deterministic variables impose a constraint on the feasibility of this approach: in order to resolve the legal values in a single pass, it is necessary that the first time a deterministic variable appears in a clique, its parents also be present. Without this constraint, the (unique) value of a deterministic variable cannot necessarily be resolved when it is first encountered. This property is a fundamental requirement, and we restate it as follows:

Immediate Resolution Property **IRP**: In a preorder traversal of a clique tree, each deterministic variable first appears in a clique with all its parents. This is equivalent to the Strong Clique Tree property of (Jensen *et al.* 1994) for influence diagrams with decision variables.

In Section 3.4 we will present a method for constructing trees with this property. First, however, we will describe how the property can be used to efficiently enumerate the possible clique values.

3.3.3 Enumerating the Legal Clique Values

Once a clique tree satisfying **IRP** is set up, there is a simple procedure for identifying the legal values of each clique. High-level pseudocode for the procedure is given in Figure 3.4.

Algorithm **Enumerate_Legal_Values**()

for each clique C_i in preorder

if C_i is the root C_r

Enumerate all the values j for which $P(C_r = j) \neq 0$.

else

let C_p be C_i's parent

if C_i is a separator

For each legal value k of C_p which represents a distinct instantiation of the variables in $\{C_i \cap C_p\}$, store a legal value j for C_i.

Note that each parent value k maps onto a distinct child value $j = mapping(k)$, and $P(C_i = mapping(k)|C_p = k) = 1$.

else

for each legal value k of C_p

for each combination of assignments to the variables in $\{C_i \setminus C_p\}$,

Generate a candidate value j corresponding to the now complete assignment of values to C_i's members.

Store a legal value if $P(C_i = j|C_p = k) \neq 0$.

Figure 3.4: Enumerating the legal values of each clique.

The **IRP** property guarantees that if a deterministic variable is present in C_i, its value can be immediately determined.

3.3.4 Discussion of Time and Space Requirements

Enumerating Possible Clique Values

Consider two cliques C_p and C_c in a parent-child relationship. The legal values of C_c can be found by recursively instantiating the variables in $\{C_c \setminus C_p\}$, for each of the legal values of C_p, in a depth-first manner. Clearly, the space required is proportional to the number of values which are finally stored (and some negligible stack overhead). In the worst case, the running time is proportional to the total number of possible assignments to the variables in C_c.

In general, however, the actual number of values enumerated will be smaller because the legal values discovered for C_p will be a subset of the total possible values. Furthermore, if we instantiate the variables in $\{C_c \setminus C_p\}$ in topological order, the enumeration procedure can be pruned as soon as there is some variable $X_i \in \{C_c \setminus C_p\}$ for which $P(x_i|Parents(X_i)) = 0$.

Clique Tree Inference

Let d_{C_i} denote the degree of clique i; i.e. the total number of edges incident on C_i including both incoming and outgoing edges. Let s_{C_i} denote the total number of possible values C_i can take. Once the possible clique values are enumerated, an examination of the inference loops reveals that the running time of the actual inference procedure is $O(\sum_{C_i \in Non-Separators} d_{C_i} s_{C_i})$. This work is about evenly distributed between λ and π calculations.

This time bound is slightly more precise than, but in the worst case the same as, the the bound presented in (Lauritzen & Spiegelhalter 1988). This bound cannot be improved on by any procedure in which each clique transmits information to all its neighbors.

3.4 A Tree-Building Procedure

In our previous discussion, we have required clique trees to have the following properties:

1) Running Intersection Property **RIP**: If any two cliques share a variable, all the cliques along the path joining them must also contain the variable.

2) Moralization **MORAL**: Each variable in the original graph must occur in at least one clique with its parents.

3) Immediate Resolution Property **IRP**: In a preorder traversal of the tree, every deterministic variable is first encountered in a clique in which its parents are also present.

To keep the number of cliques to a minimum consistent with **RIP**, it is also useful to enforce one further requirement:

4) Maximal Clique Property **MAC**: The cliques in the clique tree correspond to the maximal cliques in a triangulation of the moralized Bayesian network.

The tree-construction routines described in, for example, (Pearl 1988; Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990) guarantee **RIP**, **MORAL**, and **MAC**, but not **IRP**. In the following section we present an algorithm for constructing a clique tree that satisfies all four properties. We will first construct a tree satisfying **RIP**, **MORAL**, and **IRP**, and then systematically transform it until it satisfies **MAC**.

The key to our algorithm is to produce a triangulated graph (Rose 1970; Pearl 1988) by using an elimination sequence in which each deterministic variable is eliminated before any of its parents. To ensure that variable length sequences can be efficiently processed, we will also require that all the variables from time-slice i be eliminated before any from slice $i-1$. This produces a clique tree that is segmented into time-slices. The triangulation routine is itself from (Pearl 1988). In the following, a variable's "neighbors" are the variables connected to it by any edge, including those introduced in the triangulation process.

3.4.1 Moralization

Form an undirected version of the Bayesian network in which edges are added between each variable's parents, if they are not already present (Pearl 1988).

Algorithm **Triangulate**(graph, order)
    For each variable X in decreasing order
        Eliminate X: add edges between all the pairs of X's lower-numbered neighbors.

Figure 3.5: The triangulation algorithm.

3.4.2 Triangulation

The triangulation algorithm is shown in Figure 3.5. It is from (Pearl 1988) and builds on original work by (Rose 1970).

3.4.3 Tree Formation

The triangulation step is followed by a tree building procedure which is a combination of those described in (Pearl 1988) and (Lauritzen & Spiegelhalter 1988). However, unlike the latter, we do not insist on working with maximal cliques. Instead, a clique is formed for every vertex in the original graph. This ensures that the resulting tree will have **RIP**. [1] In contrast to (Lauritzen & Spiegelhalter 1988), the object of this procedure is to generate a tree with *directed* edges. The procedure is shown in Figure 3.6.

As we prove in section 3.4.6, the tree produced so far has **RIP**, **MORAL**, and **IRP**. The following procedure repeatedly modifies the tree in such a way that **RIP** and **IRP** are maintained at each step, and **MAC** is guaranteed on termination. (Once the graph is triangulated, the **MORAL** property cannot be lost.)

3.4.4 Tree Reduction

The simplest possible algorithm for producing **MAC** is shown in Figure 3.7. A linear-time recursive version of **Condense**() is shown in Figure 3.8. When called with the root as the argument, it produces a fully transformed tree.

Algorithm **Clique_Tree**(triangulated-graph)

1. Form a clique for each variable and its lower-numbered neighbors.
2. Order the cliques $C_1, C_2, ..., C_n$ in increasing order according to the highest numbered vertex in each clique.
3. For each clique Y in increasing order
   - Identify the subset of its constituent variables that have occurred in lower-numbered cliques.
   - Find a lower-numbered clique X that contains this subset.
   - Make X the parent of Y.

Figure 3.6: Clique tree formation.

Algorithm **Condense**()

repeat

Identify a child that is a superset of its parent.

Contract the edge between the two and replace the parent by the child.

until no child is a superset of its parent.

Figure 3.7: Non-deterministic tree condensation.

Algorithm **Condense**(Clique C)

Initialize a queue with C's children.

while the queue is not empty

Remove a child C_i.

if C_i is a superset of C

Replace C's members with C_i's members.

Remove C_i from C's list of children.

Add C_i's children to the queue and to C's list of children.

for each child C_j of C

Condense(C_j).

Figure 3.8: A linear time algorithm for producing **MAC**.

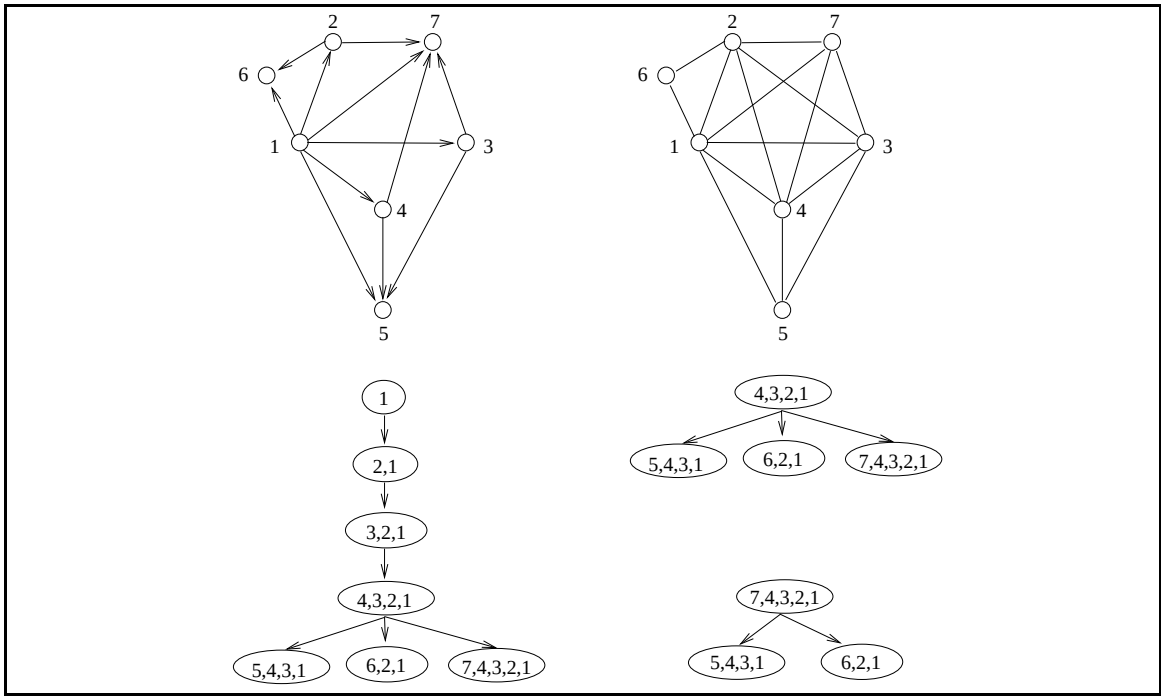

Figure 3.9: A Bayesian network and its clique tree.

3.4.5 An Example

Figure 3.13 shows a Bayesian network, a clique tree for this network, and several intermediate stages in the tree-construction process. The original network is shown in the upper-left corner. The variables are numbered for use in the elimination sequence. The triangulated graph after elimination in reverse order is shown in the upper-right corner. The cliques corresponding to the elimination of each vertex are:

$$\{1\},\{2,1\},\{3,2,1\},\ \{4,3,2,1\},\{5,4,3,1\},\{6,2,1\},\{7,4,3,2,1\}.$$

The tree produced by the initial construction procedure is shown in the lower-left corner. When **Condense** executes, the edge to $\{2,1\}$ is contracted first, followed by the edge to $\{3,2,1\}$ and the one to $\{4,3,2,1\}$. This produces the intermediate tree shown. Finally, the edge to $\{7,4,3,2,1\}$ is contracted, producing the output tree.

Note that the tree construction process will not work if the maximal cliques

$$\{5,4,3,1\},\{6,2,1\},\{7,4,3,2,1\}$$

are used directly as its input: the addition of the last clique will violate **RIP**. Hence it is necessary to construct a full initial tree, and then condense it. This issue has apparently been overlooked in the literature, e.g. (Pearl 1988; Lauritzen & Spiegelhalter 1988), perhaps because a maximum cardinality search is intended to be used to renumber the vertices **after** triangulation. In other cases, e.g. (Jensen *et al.* 1994), there is no renumbering. The maximum spanning tree algorithm presented in (Jensen & Jensen 1994) for generating a clique tree from a collection of cliques does not have this problem, but is difficult to extend to trees with **IRP**.

3.4.6 Correctness of the Tree-Building Procedure

We have established that clique trees which satisfy **RIP**, **MORAL**, and **MAC** are acceptable representations of the underlying Bayesian network. We show the correctness of our algorithms by establishing that the output trees have these properties. First we note that the moralization and elimination processes will produce a triangulated graph in which **MORAL** is satisfied. We then show that the tree building procedure satisfies **RIP** and

[1] The example in the following section shows that if maximal cliques are used, **RIP** may be violated.

furthermore that **IRP** is satisfied. We end by showing that the condensation procedure ensures **MAC** without violating **RIP** or **IRP**.

Moralization and Triangulation

It is proven in (Rose 1970) that an arbitrary elimination ordering will produce a valid triangulation. Furthermore, since moralization induces a clique on each family, the whole family is guaranteed to occur together in the clique that is generated when the first of its members is eliminated.

Proof of RIP

Lemma 3.1 *After step 2 of* **Clique_Tree**, *all the variables in a clique C which have occurred in earlier numbered cliques can be found together in a single clique other than C.*

Proof. Consider the subset of variables U that has occurred earlier. Since U induces a subgraph on a clique C, the vertices of U themselves form a clique. Consider the highest numbered vertex in U, X. The previously processed clique resulting from X's elimination will contain all the other members of U.

Theorem 3.3 *Algorithm* **Clique_Tree** *produces a tree with* **RIP**.

Proof. The proof is inductive. Consider an arbitrary variable X_i. The base case is when the first clique containing X_i is added to the tree. Clearly the property holds at this point. Now consider the addition of a new clique C_j containing X_i to the tree. By Lemma 3.1, we know that C_j can be added, and by construction it will be joined to a clique that also contains X_i. Hence **RIP** is maintained at each step.

Proof of IRP

Theorem 3.4 *Algorithm* **Clique_Tree** *produces a tree with* **IRP**.

Proof. The elimination of a variable X generates a clique C in which X is the highest numbered variable. Therefore when the cliques are ordered by their highest numbered

member, the first clique in which X occurs is the one it generated when it was eliminated. When X is a deterministic variable, the restriction on the elimination ordering ensures that this clique contains X's parents. An earlier-occurring clique will be selected as C's parent. Since all further cliques containing X must be descendants of C (in order to have as a parent a clique in which all previously seen vertices occur), C will be the first clique encountered in a preorder traversal.

Proof of Condense()

Lemma 3.2 *When* **Clique_Tree** *terminates, all the supersets of a clique C_i will be descendants of C_i.*

Proof. Let C_j be a superset of C_i. Consider X_j, the highest numbered variable in $\{C_j \backslash C_i\}$. In order for X_j not to be present in C_i, X_j must be higher numbered than any member of C_i. Hence C_j will occur after C_i in the clique ordering and be added as a descendant.

Lemma 3.3 *If a superset of clique C_i exists, then C_i has an immediate descendant that is a superset.*

Proof. Suppose there is a clique C_j that is a superset of C_i, and that C_j is a descendant of C_i, but not an immediate descendant. (By Lemma 3.2, it must be a descendant of some sort.) Let C_k be the first clique encountered on the path from C_i to C_j. If C_k is not a superset of C_i, then there exists some member of C_i which is present in C_i and C_j but not C_k. This violates **RIP**, which was established in theorem 2.

Theorem 3.5 Condense*() guarantees a tree with* **RIP**, **IRP**, *and* **MAC**.

Proof. The proof of correctness for Algorithm **Condense** proceeds as follows:

1. We show that after each step **RIP** holds.
2. We show that after each step **IRP** holds.
3. We show that upon termination **MAC** holds.

Proof of Part 1: At each step of the algorithm, a child clique C_c replaces a parent clique, C_p. Consider a path between two cliques C_x and C_y. There are four cases:

1. The path goes through neither C_c nor C_p. In this case the contraction is irrelevant.
2. The path goes through C_c only. In this case the contraction is acceptable because C_c is unchanged.
3. The path goes through C_p only. In this case the contraction is acceptable because C_p is replaced by a superset.
4. The path goes through both C_c and C_p. In this case the contraction is acceptable because any variable that was previously present in both C_c and C_p will still be present in C_c.

Proof of Part 2: Let C_p be replaced by C_c. Consider a clique C_i in which variable X_i first appears. There are three cases:

1. C_i is C_p. Here C_c will replace C_p as the first clique with X_i, but since C_c is a superset of C_p, it will still have X_i's parents.
2. C_i is C_c. This is acceptable because C_c remains the first clique encountered with X_i.
3. C_i is neither C_c nor C_p. Since all the other cliques with X_i are rooted in C_i, the preorder relationship is unaltered.

Proof of Part 3: Suppose that on termination of **Condense** there exists a clique C_j that is a superset of clique C_i. By Lemma 3.3, C_i must have an immediate descendant that is a superset, thus violating **Condense**'s termination condition.

The linear-time recursive version of **Condense** is correct because each step in the **while** loop is a legal contraction. Furthermore, when the **while** loop terminates none of C's descendants can be a superset of C without violating either the termination condition or **RIP**.

3.5 Comparison with Other Approaches

We pause here to briefly describe other approaches to inference in Bayesian networks and their relationship the scheme presented here. Historically, there have been two main approaches to doing inference: exact algorithms, and stochastic simulation algorithms.

The exact algorithms all attempt to make the required summations tractable through the use of dynamic programming. Stochastic simulation algorithms produce approximate answers by sampling a representative subset of the terms. The problem of inference in Bayesian networks is NP-hard (Cooper 1990), and recently it has been shown that even producing answers that are accurate to within a fixed fraction is equally difficult (Dagum & Luby 1993). This suggests that the choice of an inference algorithm should not be made once-and-for-all, but should be done with the particular characteristics of a specific network and task in mind.

Exact Inference

The simplest algorithms for exact inference, and the only ones whose running times are linear in the size of the underlying graph, work with graphs that induce an undirected tree. Algorithms for this kind of graph can be found in (Pearl 1988) and (Peot & Shachter 1991), and the basic algorithm of 3.1 is an expression of this explicitly in terms of dynamic programming. When loops are present in the underlying graph, the inference problem is fundamentally more difficult. As we have seen, loops are typically handled by transforming the input graph in some way so that an inference procedure for trees can be used.

The simplest method of dealing with graphs that contain loops is known as cutset conditioning (Pearl 1988). In this approach, cycles in the underlying graph are broken by assuming a known value for one of the variables in the cycle. A set of variables such that every (undirected) cycle has at least one variable in the set is known as a *cutset*. The tree algorithm is run once for every possible assignment of values to the variables in the cutset, and the results are combined. Obviously, a small cutset is necessary for this approach to be feasible.

A second method, and the most widely used, is to somehow agglomerate the underlying variables into mega-variables in such a way that the mega-variables form a tree. The join-tree algorithms (Lauritzen & Spiegelhalter 1988; Jensen *et al.* 1990) are the most widely used of this sort. The agglomeration is done through the process of moralization and triangulation. Although the subsequent calculations boil down to essentially the ones we presented in Section 3.2, the terminology and intermediate derivations differ.

Stochastic Simulation

Stochastic simulation schemes approximate a sum such as $P(\mathbf{o}) = \sum_{\mathbf{s}} P(\mathbf{o}, \mathbf{s})$ by summing over a small subset of the terms. The simplest of these schemes, logic sampling (Henrion 1988), generates full instantiations of the network by assigning values to the variables in topological order, each according to the distribution specified by the assignment to its parents. Statistics conditioned on certain events are computed by computing the desired frequencies in the samples conforming to the conditions. This scheme has the severe disadvantage that many simulations may be required before one is generated that matches the specified conditions or evidence.

Likelihood weighting (Shachter & Peot 1989) is a more refined version of logic sampling, in which evidence nodes are always instantiated to their observed values. By weighting each simulation by an appropriate quantity, unbiased statistics can be computed. Likelihood weighting is the workhorse of simulation schemes, and is described in detail by (Dagum & Luby 1997).

There are several approximation algorithms based on sampling a Markov chain in which the states represent joint instantiations of the variables (Chavez & Cooper 1990b; Chavez & Cooper 1990a; Pearl 1988). Although bounds can sometimes be provided for these and the other simulation schemes, there are always cases in which they are bound to fail. Unfortunately, the bounds break down when extreme probabilities, i.e. probabilities arbitrarily close to 0 and 1, are present in the network (Dagum & Luby 1997). These are exactly the kind of probabilities needed to express pronunciation models in speech recognition (see Chapter 5).

Another problem with simulation algorithms arises with DBNs spanning many time-slices. Intuitively, as one instantiates the network in a forward manner, it is possible to generate samples all of which, though they seen reasonable at the current time, are rendered implausible by future evidence. In general (Dagum & Luby 1997), the number of simulation runs required to achieve a fixed degree of accuracy is inversely proportional to the probability of the observations. Thus the number increases exponentially with the number of time-slices in the network. For example, the probability of a one-second utterance might be 10^{-1400} in a trained DBN. Note that the efficiency of exact inference is not affected by the probability of the evidence. Two promising approaches to ameliorating this

problem are arc reversal, in which arcs leading into evidence variables are reversed and the necessary conditional probabilities recalculated, and "survival-of-the-fittest" sampling in which samples having a low probability are replaced by likelier samples at each time-step in the simulation (Kanazawa *et al.* 1995).

3.6 Variable Length Observation Sequences

3.6.1 Motivation

In order to compute the statistics necessary to learn parameter values, both λs and πs are required (see Section 3.8). Therefore, inference and learning must be done with a Bayesian network that is as long as the observation sequence. Unfortunately, the utterances vary in length across a wide range of values; there are two obvious ways of dealing with this:

1. On demand, unroll the network to the appropriate length for an utterance, moralize it, triangulate it, and form a clique tree.

2. Precompute and store a clique tree for every reasonable utterance length.

The first solution is inefficient in terms of computing time, and the second solution hopeless on the grounds of excessive memory requirements.

In this section, we present an alternative in which a single clique tree of the maximum possible length is precomputed, and spliced down to the appropriate length for any particular utterance. The operations involved boil down to some pointer-swapping, and a sweep over the tree in which the numbering of the constituent variables of the cliques is adjusted. Note that this is not simply a question of truncating the tree. The cliques corresponding to the initial and final few slices of a fixed length Bayesian network are typically different than the cliques corresponding to intermediate slices.

The procedure is based on the observation that, despite the fact that the initial and final segments of a clique tree may have an arbitrary structure, when ties are broken in a consistent manner, the clique tree creation algorithms will produce a tree with a repeating *intermediate* structure. By identifying the beginning and ending of each of the repeated segments, it will be possible to efficiently remove segments on demand. The basic idea

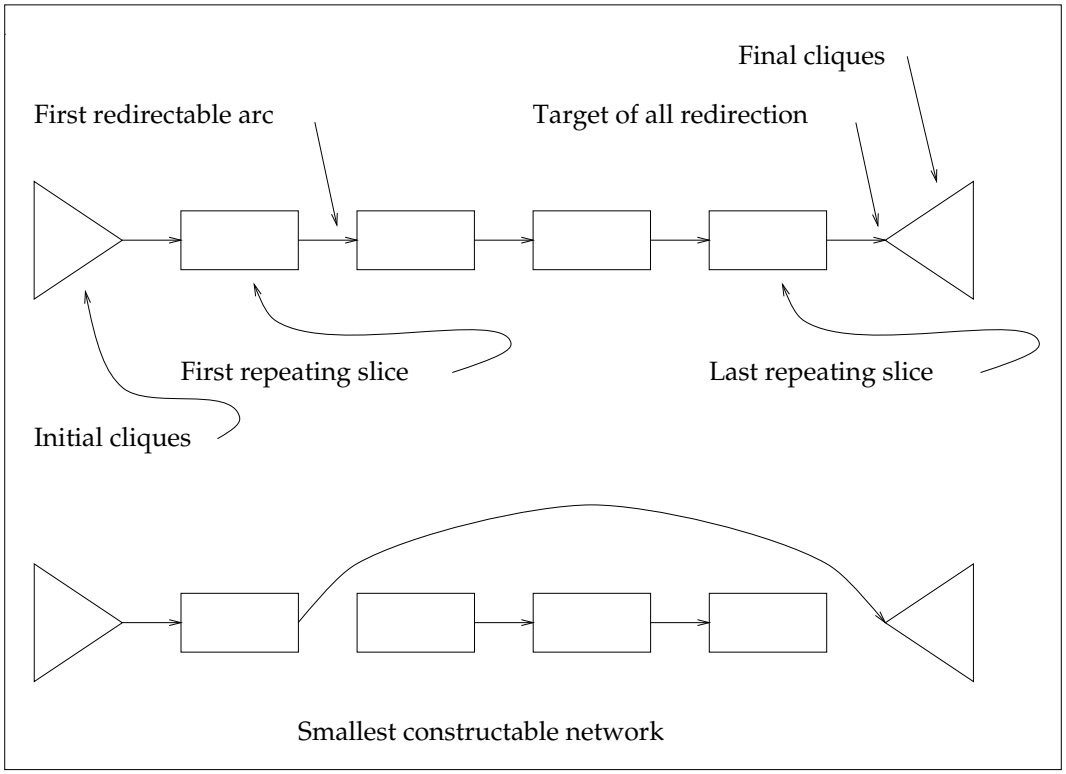

Figure 3.10: Splicing a clique tree. The triangles represent non-repeating initial and final portions of the clique tree. The rectangles represent repeating segments. Splicing is accomplished by redirecting arcs connecting repeating segments.

can be gleaned from Figure 3.10. An alternative approach to dealing with variable length sequences that focuses on generality, rather than efficiency, can be found in (Kjaerulff 1992).

Finally, we note that this section assumes that an entire utterance is available at the start of the inference procedures. That makes the technique suitable for offline learning, and the recognition of isolated words. The subject of online recognition for continuous speech is dealt with in Section 3.7.

3.6.2 Definitions and the Splicing Algorithm

In order to reason about the clique trees that are used in conjunction with DBNs, it is useful to associate each clique in the tree with a specific time-slice in the underlying Bayesian network. We will associate each clique with earliest time-slice of any of its constituent variables, and say that a clique is "from" the time-slice of this variable. For

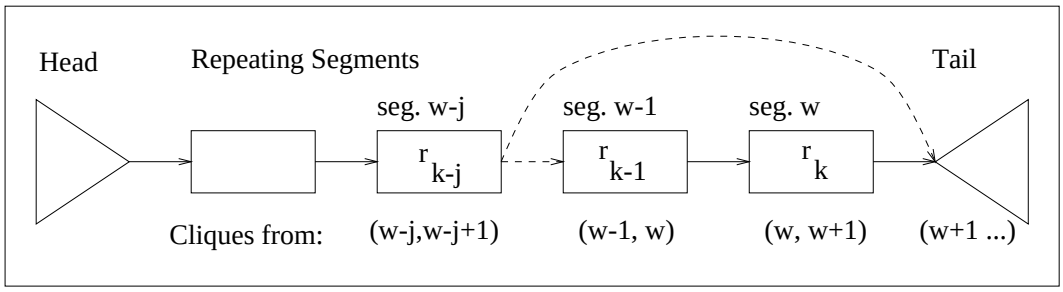

Figure 3.11: Splicing terms defined.

example, a clique containing variables from time-slices 2 and 3 will be said to be "from" time-slice 2. We refer to the set of cliques from time-slice i as a "segment," and say that the segment is from time-slice i.

Let the original Bayesian network have N slices. We will refer to the number of variables in a time-slice of the underlying network as S. We shall see that in the corresponding clique tree, any path from the root, which is from slice 1, to a clique with a member from slice N, will be uniquely defined through segment $N-2$. We refer to this unique path as the backbone of the tree. Denote the repeating units by $r_1, \ldots r_k$. We will assume that the cliques in the last repeating segment r_k are from time slice w. The repeating segments are denoted by rectangles in Figure 3.11. We will refer to the unique arc along the backbone leaving r_i as the "exit" from r_i. The clique in segment r_{i+1} that is connected to the exit from r_i is referred to as the entry clique for r_{i+1}. We will refer to the cliques that occur in the part of the tree rooted in r_k's exit as the "tail." The tail is denoted by the rightmost triangle in Figure 3.11. We will choose r_k so that it is from slice $N-2$ or earlier, thus ensuring that the exit arcs are uniquely defined. Cliques that do not lie in a repeating unit or the tail are referred to collectively as the "head." These are represented by the leftmost triangle in Figure 3.11.

The repeating segments have the following properties by definition:

1. Segment r_i contains exactly the cliques from a fixed time slice.

2. Each clique C_g from $r_i, i = 2, \ldots k$ has an analog A_g in segment r_{i-1}:

   - The variables associated with A_g are the same as those associated with C_g, except

Algorithm **Splice**(j: number of slices to remove)

- Redirect the exit of r_{k-j} to point to the exit of r_k.
- Renumber the variables occurring in the tail cliques by lowering their indices by $j * S$.
- Recalculate the conditional probabilities for all the cliques.

Figure 3.12: The splicing algorithm.

that their indices are less by S.

- The parent of C_g is analogous to the parent of A_g, except possibly for the entry clique into r_2. The restriction can be relaxed in this case because the entry arc into r_1 is never redirected to a later segment.
- The indices of the variables in $\mathbf{F}_{A_g}$ are the same as those associated with $\mathbf{F}_{C_g}$, except that their indices are less by S.

We will take the occurrence of a repeating structure for granted, subject to runtime verification. (Verification is necessary because a repeating structure is not guaranteed to exist, e.g. if ties are broken randomly in the tree-creation process.)

Algorithm

The splicing algorithm to remove j time slices is shown in Figure 3.12.

3.6.3 Proof of Splicing Algorithm

We will show the correctness of the splicing algorithm by showing that the clique tree produced is well-defined, and represents the same probability distribution as the shortened Bayesian network. The proof will proceed in several stages:

- First we will show that the arcs connecting the segments are unambiguously defined.
- Then we will show that the requirements of section 3.2.1 are satisfied.

Proof of Uniqueness of Splicing Arcs

The proof will proceed in several steps. First we will establish some general properties of clique trees. Then we will show that there is a path connecting the head to the tail, that it goes through a monotonically increasing sequence of segments, and finally that there is only one path.

Lemma 3.4 *Each pair of variables A, B that are connected by an arc in the underlying Bayesian network will appear together in a clique.*

Proof. One of the variables, say A, must be higher-indexed than the other. When A is eliminated, B will be a lower-indexed neighbor, and will be added to A's clique.

Definition 3.1 *A Bayesian network has the property of time-slice contiguousness when for every pair of variables in time-slice k there is an (undirected) path connecting them that only goes through other variables from time-slice k. A clique tree has the property of time-slice contiguousness when for every pair of cliques with any variables from time slice k, all the cliques along the path connecting them contain at least one variable from time-slice k.*

Theorem 3.6 *Time-slice contiguousness in the underlying Bayesian network implies time-slice contiguousness in the resulting clique tree.*

Proof. Consider an arbitrary pair of cliques F and G, and let the first one contain variable A from slice k and the second one contain variable B from slice k. If A and B are the same, the theorem is proved by **RIP**. Suppose A and B differ. Let A and B be linked in the underlying Bayesian network by the path $A - X_1, X_1 - X_2, \ldots, X_m - B$ where all the variables in the path are in slice k. Locate the clique that contains the pair $A - X_1$. By Lemma 3.4, the clique must exist. Proceed from F to this clique; by **RIP**, all the cliques on this path must contain A and therefore a variable from slice k. Locate the clique with the pair $X_1 - X_2$. Again, Lemma 3.4 ensures that it exists, and RIP ensures that it is connected to the last clique by a path along which X_1 (which is from slice k) occurs in every clique. Continue in this manner until the clique with $X_m - B$ is reached. Now proceed to G. This process has visited all the clique along the path connecting F to G, and they all had a variable from k.

Assumption 3.1 *The underlying Bayesian network is time-slice contiguous.*

Assumption 3.2 *All variables in time-slice $i+1$ are higher indexed (in the elimination sequence) than any variable in slice i. Since we are free to use any elimination sequence, this can be achieved.*

Assumption 3.3 *The underlying Bayesian network has the Markov property, i.e. all the parents of a variable from time-slice i are either in slice i or $i-1$.*

Definition 3.2 *The length of an edge between two variables in a Bayesian network is the difference in the time-slice indexes of the variables. For example, the length of an edge between two variables in slice i is 0, and the length of an edge between a variable in slice i and a variable in slice $i+1$ is 1.*

Theorem 3.7 *No clique contains variables from time-slices that differ by more than 1.*

Proof. In the triangulated graph, a clique containing variables from more than two time-slices must have an edge of length 2 or more. We will show that this is impossible. Consider the process of moralization and triangulation. Originally all edges have length 0 or 1, because of the Markov property. Adding edges between parents in the process of moralization can only introduce edges of length 1, again because of the Markov property. Now consider eliminating a variable from a graph whose edges are length 1 at most. Let the variable lie in slice i. It can be connected to variables in slices $i-1$, i, and $i+1$. But by assumption 3.2 and the triangulation algorithm, edges will only be introduced between variables from slices i and $i-1$. Hence only edges of length at most 1 are added, and a graph whose edges are at most length 1 results from the operation. This holds at each step, so a clique with variables spanning more than 2 time-slices cannot occur.

Assumption 3.4 *The underlying Bayesian network is connected. By* **RIP** *and Lemma 3.4, this implies that the intersection between neighboring cliques is nonempty.*

Lemma 3.5 *Two adjacent cliques are always from the same slice or from slices that differ by at most 1.*

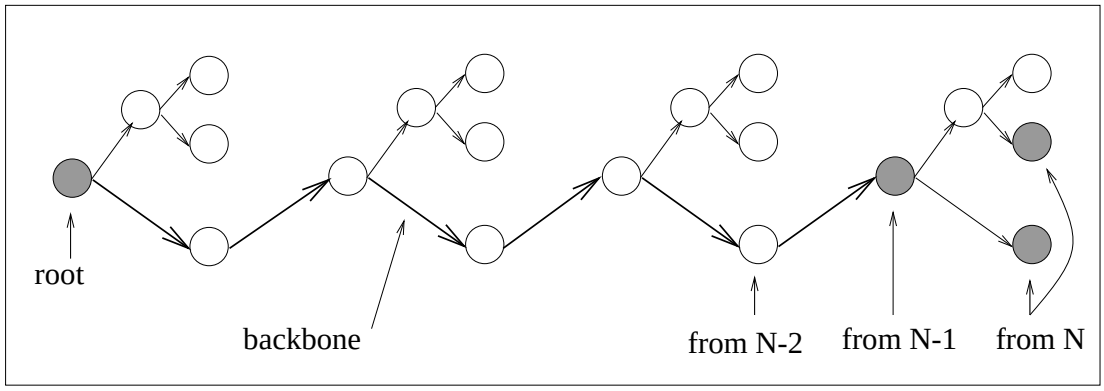

Figure 3.13: The backbone of a clique tree.

Proof. Let the cliques share a variable from time-slice i. The biggest the minimum member could be is i. The smallest the minimum member could be is $i-1$ by Theorem 3.7. Thus the greatest the difference can be is 1.

Lemma 3.6 *There is a directed path (the backbone) from the root of the clique tree, which is from time-slice* 1 *by the tree-construction process, to any clique with a member from slice* N*, and this terminating clique is from no earlier than slice* $N-1$*.*

Proof. A variable from slice N must occur in some clique, and by Theorem 3.7, no other constituent variable can be from a slice earlier than $N-1$. Since there is a directed path from the root to every other clique, the path in question exists.

Lemma 3.7 *The backbone contains cliques from all the time-slices* $1, 2, \ldots, N-1$*.*

Proof. The path must proceed from a clique from slice 1 to one from slice at least $N-1$, and by Lemma 3.5 it can only do so by steps of 1.

Lemma 3.8 *The minimum node in each clique along the backbone increases monotonically.*

Proof. Consider the first violation along a path where this is not the case. By Lemma 3.5, the decrease must be by 1. The minimum variables encountered in the cliques along this path must proceed in the sequence $i, (i+1)^+, i$. This leads to a contradiction: by Theorem

3.6, all the cliques in this part of the path would have to have variables from time-slice i, so a variable from $i+1$ could not be the minimum.

Lemma 3.9 *The backbone is uniquely defined up to the last clique encountered from $N-2$.*

Proof. The cliques with variables from slice N form a connected component by Theorem 3.6. The same holds for cliques with variables from slices $1, \ldots, N-2$. These components have no intersection by Theorem 3.7. The backbone is one path connecting the root, which is in one component, to the cliques in the other component. The existence of a second path from the root to the variables from slice N would imply a cycle.

Theorem 3.8 *The edge along the backbone connecting a clique from slice i to one from slice $i+1$, $i \leq N-2$ defines a unique point in the tree.*

Proof. The backbone is unique up to this point by Lemma 3.9, and the minimum nodes in the cliques along this path increase monotonically by Lemma 3.8.

We now turn to the question of whether the new clique tree represents the correct probability distribution.

Proof of Representation

We assume here that $j \geq 1$ slices are removed from the Bayesian network. Although it is not actually necessary to renumber the variables in the cliques from r_k, this is conceptually useful, and we imagine doing it in the proofs.

Theorem 3.9 *The correspondence requirement and* **MORAL** *are satisfied, i.e. each variable in the underlying Bayesian network from slice $1..N-j$ occurs in the new tree, and moreover they all occur at least once with their families.*

Proof. We will show that

1. All the variables from slices $1, \ldots, w-j$ occur with their families in the cliques present in the head through segment r_{k-j}.

2. All the nodes from slices $w-j+1, \ldots, N-j$ occur with their families in the renumbered tail cliques and the renumbered r_k cliques.

Since r_k's cliques have the same members and **F** sets after renumbering as r_{k-j} (by the definition of analogous cliques), all the variables and their families are present in the spliced tree. The proofs of these two parts follow:

1. All the variables from $1, \ldots, w-j$ must occur no later than r_{k-j}, by the segment definition and monotonicity (see also Figure 3.11). And they must occur with their families by **MORAL** in the original tree.

2. All the variables from slices $w+1, \ldots, N$ occur, necessarily somewhere with their families, in the cliques in r_k through the tail. After renumbering, the cliques in r_k through the tail will contain all the variables from time-slices $w-j+1, \ldots N-j$ and their families, by definition of analogous cliques. Since the renumbered r_k has cliques with the same variables and families as r_{k-j}, the new tree, which has both r_{k-j} and the tail, will have all the variables in time slices $w-j+1, \ldots, N-j$ and their families.

Theorem 3.10 RIP *is satisfied in the new tree.*

Proof. RIP holds in $head, \ldots, r_{k-j}$ because this is a connected component of the original tree. **RIP** holds in the renumbered segments $r_k, \ldots tail$ by **RIP** in the original tree combined with a uniform offset in numbering. **RIP** holds in the connection between r_{k-j} and the tail because the exit from r_{k-j} has the same variables as the exit from the renumbered r_k, and **RIP** holds between the exit of the renumbered r_k and the renumbered tail.

Theorem 3.11 IRP *is satisfied in the new tree.*

Proof. IRP holds in the original tree, and the splicing operation retains the relative ordering of all the remaining cliques.

Theorem 3.12 *The equal and zero probability requirements hold between the new underlying Bayesian network and the new clique tree.*

Proof. The required probabilities are recalculated. Since each variable's family is present, equality can be achieved. Since the new tree has **RIP**, the zero-probability requirement

can be satisfied. **IRP** ensures that deterministic variables can be handled as efficiently as before.

3.6.4 Comparison with HMMS

Variable length observation sequences are not an issue with HMMs. This is because computation is done on a homogeneous two-dimensional grid; every column of the grid is identical. Therefore it is trivial to allocate a grid suitable for the longest possible utterance, and only use the portion that is needed for any particular utterance. The comparison between DBNs and HMMs for online inference is presented in the following section.

3.7 Online Inference

In many applications, it is desirable to process a continuing stream of data in an online fashion. For example, in speech recognition it is necessary to recognize words in real time, before a person has finished speaking. In these types of applications, it is not possible to construct a complete clique tree to represent the utterance, because the total number of frames is unknown and too long a delay would be imposed by waiting until the end. Under these circumstances, it is useful to do a Viterbi decoding in an online fashion.

Under some circumstances, there is an additional constraint: the number of possible clique values may be too large to handle in real time, or with the available memory. In speech recognition, this occurs when a Bayesian network is used to encode an entire language model, and tens of thousands of words are possible at any point in time. Under these circumstances, it is necessary to prune the set of hypotheses, and keep track only of ones that are reasonably likely. Typically a beam search is used, and this is straightforward to incorporate into the forward pass of HMM inference (see, e.g. (Jelinek 1997)).

With DBNs, however, the situation is more tricky. Recall that the likeliest assignment of values to the variables is computed with a *bottom-up* pass over the tree. Since the root of the tree must be at time 0 in order to handle deterministic variables, this creates a problem: the bottom of the tree is extended as each new frame becomes available. This means that the Viterbi decoding must be recomputed from scratch for the entire utterance after each frame. Clearly, this is prohibitive.

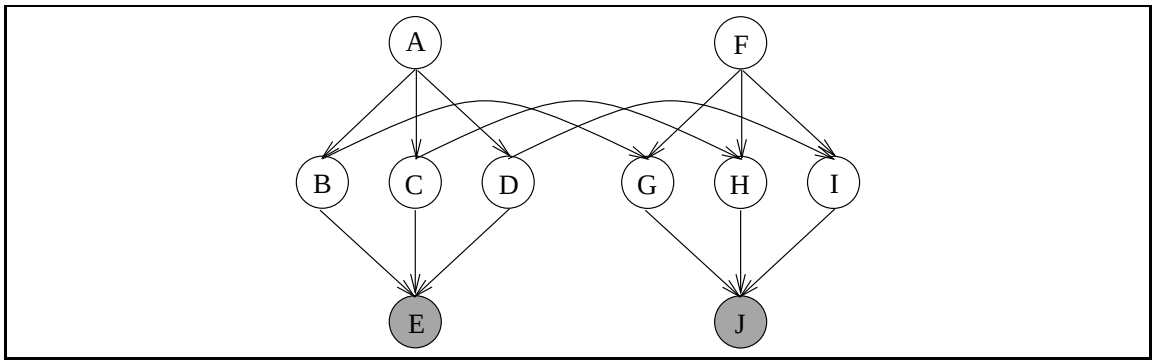

Figure 3.14: Two slices of a complex DBN; when reduced to a chain-structured tree, the computational requirements are significantly lower than if a cross-product of the state values were used in a an HMM.

There are two obvious ways around this, neither of which works:

1. Compute the likeliest assignment of values in a top-down manner, rather than bottom-up. This does not work because the πs must be computed with reference to the λs, which are not available until the bottom-up pass is complete.

2. Modify the tree-building procedure so that the bottom of the tree - rather than the root - is at time 0. This has the flaw that it becomes extremely cumbersome to keep track of the clique values that are possible, given the constraints of deterministic variables. Recall that this was done in a top-down manner, starting at the root.

We now present two methods for online decoding.

3.7.1 Chain Decoding

The first method comes from realizing that with chain-structured DBNs, it is possible to compute the likeliest values with a forward sweep. When the Bayesian network is a chain, the computation of π_j^i (for variable i with parent p) reduces to:

$$\pi_j^i = \sum_v \pi_v^p P(X_i = j | X_p = v).$$

Since there is no reference to anything that must be computed in a bottom-up pass (specifically to any λs), this quantity can be computed online in a top-down fashion. By maximizing over v, rather than summing, and keeping track of the best v for each π_j^i, the likeliest assignment of values can also be recovered online. This is exactly analogous to online Viterbi

decoding with HMMs (Rabiner & Juang 1993; Jelinek 1997). Finally, beam search can be implemented by pruning away the least likely π_j^is after each new frame. Since this is a top-down procedure and the root of the tree is still at time 0, it is straightforward to combine with the procedure for propagating deterministic constraints.

It is important to realize that in terms of computational requirements, a chain-structured DBN may be significantly more efficient than an HMM in which the state space represents the cross-product of all the variables in a timeslice (even though both can be represented by chain structures). This is because it is possible to "spread" the variables from a single slice across many cliques in the chain. Figure 3.14 shows an example of this. A valid chain of cliques - that satisfies all the requirements of the previous sections - is:

```
ABCD
BCD
BCDE
BCD
BCDFG
CDFG
CDFGH
DFGH
DFGHI
GHI
GHIJ
```

Assuming that each of the hidden variables has f values, the compute time is proportional to f^5. This compares favorably to the f^8 requirement of a cross-product HMM, which results from the necessity of considering a transition from any of f^4 states at time t to any of f^4 states at time $t+1$. In general, if there are k hidden variables in the middle layer, the compute time is f^{k+2} as opposed to f^{2k+2}. Finally, we note that this is *not* a question of parameter tying; it is an unavoidable consequence of computing with a cross-product representation.

The tree-building procedures presented in Section 3.4 generate trees, not necessarily chains. There are, however, simple ways of creating chain-structured "trees." A simple and effective technique known as the frontier algorithm is presented in (Zweig 1996), along with specialized inference routines.

3.7.2 Backbone Decoding

The second method for online decoding stems from two observations:

1. The likeliest assignment of values can be recovered from a *combination* of λ and π values.

2. The side chains hanging off the main backbone of the clique tree cannot extend more than one timeslice into the future. [2]

This suggests the following process for extending the backbone of the clique tree, one clique at a time:

1. Add the next clique along the backbone, and all the side-chains rooted in it. Since a side-chain can have variables from up to two timeslices, it may be necessary to maintain a buffer of one frame in order to be able to assign values to all its members.

2. Find the set of possible clique values by propagating forward the deterministic constraints.

3. Compute λs for the side-chain cliques:

$$\lambda_j^i = \prod_{c \in children(X_i)} \max_f \lambda_f^c * P(X_c = f | X_i = j).$$

   Store the maximizing f value.

4. Compute πs for the new backbone clique. These require only λs that are already available:

$$\pi_j^i = \max_v P(X_i = j | X_p = v) * \pi_v^p * \prod_{s \in siblings(X_i)} \max_f \lambda_f^s * P(X_s = f | X_p = v)$$

   Store both the maximizing f and v values.

Since the side chains are short compared to the entire tree, it makes sense simply to compute all the λs. Beam search can be done with the πs by retaining a subset. The likeliest

[2] Suppose there is a backbone clique A from slice i, and a clique B from slice $i+2$ on a side-chain rooted in A. Now consider a clique C on the backbone that is from $i+2$. A and B are connected via the side chain, and A and C are connected by the backbone. By timeslice contiguousness, B and C must be connected by a chain of cliques with variables from $i+2$, but Theorem 3.7 excludes a path through A. Therefore, there must be yet another path from B to C, and a cycle is implied.

assignment for the cliques on the backbone can be recovered by starting with the most likely value for the last clique on the backbone, and recursively looking up the maximizing values of its parent and siblings. The likeliest values for the side-chains can be recovered by proceeding top-down from the backbone and using the λs. This procedure has the drawback of being significantly more complicated than chain decoding. However, it is theoretically more efficient since it does not impose any constraints on the tree structure. Since online inference is not explored further in this thesis, we do not reproduce the algorithm.

3.8 Learning

There are two basic issues involved with learning Bayesian networks. The first concerns learning the structure of the network, and the second concerns learning the required conditional probabilities once the network structure has been selected (Heckerman 1995; Buntine 1994). Traditionally, these have been viewed as separate problems; early work on structure learning (Cooper & Herskovits 1992) performed a greedy search over model structures, and evaluated each candidate structure by learning optimal parameters for it. More recent work, (Friedman 1997), combines the two processes and changes the network structure dynamically as the conditional probabilities are learned. In our work, the candidate structures were generated manually to address specific issues.

There are also two main approaches to learning the conditional probabilities Θ. In both cases, parameter adjustment is done according to the maximum likelihood principle to maximize the probability of a collection of observed data, i.e. $\arg\max_\Theta P(\mathbf{o}|\Theta)$. The first method is that of gradient descent (Binder *et al.* 1997), and is applicable whenever the derivative of the data likelihood with respect to Θ can be computed. The second method is the EM algorithm (Dempster *et al.* 1977), and is applicable when the conditional probabilities are represented by distributions in the exponential family. (The definition of the exponential family is complex (Buntine 1994), but it includes many common distributions such as the Gaussian, Chi-squared, and Gamma distributions). The gradient descent techniques have the advantage of greater generality, while the EM algorithm has the advantages of simplicity and robustness. In the following sections, we will provide an overview of both approaches; in our implementation, we adopted the EM approach. One key similarity between the two is that the information they require is computed with the inference routines.

Thus inference is a crucial step in parameter adjustment. Another similarity is that in general, both are guaranteed to find only a local optimum in the parameter space.

3.8.1 Gradient Descent Techniques

An excellent review of gradient descent techniques applied to Bayesian networks can be found in (Binder *et al.* 1997). Gradient descent can be thought of as moving a point corresponding to the parameter values through parameter space so as to maximize the likelihood function. The basic questions that must be answered are:

- What direction to move?
- How far to move?

This is complicated by the fact that in many cases there are restrictions on the parameter values. For example, if conditional probability tables are used, all the entries must lie between zero and one. Furthermore, all the entries in a distribution must add to one. These constraints define a feasible region in parameter space in which Θ must lie. Thus, the problem is one of constrained optimization.

Gradient descent techniques move either in the direction of the gradient, or, in the case of conjugate gradient techniques (Price 1992), in a conjugate direction. In (Binder *et al.* 1997), a derivation is given that shows the gradient to be a simple function of N_{ijk}, the number of times that variable X_i has value k and its parents are found in the jth possible configuration. Again, this issue is complicated, because the current parameter setting may lie on the boundary of the feasible region with the gradient pointing out of the region. This can be dealt with either by projecting the gradient onto the constraint surface, or re-parameterizing the conditional probabilities so the unconstrained optimization can be performed on the new parameters (Binder *et al.* 1997).

The simplest strategy for deciding how far to move is to move a fixed amount. More sophisticated techniques perform line search to maximize the likelihood along the direction chosen. Once more, the question of constraints complicated both approaches. For example, if the likelihood is optimal and increasing at the point where the line intersects the feasible region, a decision must be made whether to recalculate the gradient at that point, or continue the line search along a projected direction.

3.8.2 EM

The questions of what direction, and how much to move are answered simultaneously by the EM algorithm. In the case of discrete CPTs, the crux of the EM algorithm is extremely simple: estimate N_{ijk}, and then estimate θ_{ijk}, the probability that $X_i = k$ given that its parents have instantiation j, as $\frac{N_{ijk}}{\sum_k N_{ijk}}$ (Lauritzen 1991; Heckerman 1995). When both a variable and its parents have observed values, N_{ijk} is obtained simply by counting. More commonly, there are some unknown values in which case inference is necessary. The calculations can be summarized as follows.

Let C_i be the clique containing x_i and its parents. Because of the **MORAL** property, we know such a clique exists. Let $\mathbf{V}^i_{jk}$ be the set of C_i's clique values corresponding to underlying variable assignments that include $X_i = j, Parents(X_i) = k$. Recall that

$$P(C_i = w|Observations) = \frac{\lambda^i_w * \pi^i_w}{\sum_w \lambda^i_w * \pi^i_w}$$

Now, N_{ijk} can be found by summing over the appropriate clique values:

$$N_{ijk} = \sum_{w \in \mathbf{V}^i_{jk}} P(C_i = w|Observations)$$

Estimating N_{ijk} from a collection of examples simply requires summing the individual estimates for each example. By maintaining the appropriate data structures, the counts for every family can be computed in a single sweep over the cliques. EM has much to recommend it. The problem of constrained parameter values is nonexistent, and line searches are unnecessary. Where applicable, it is usually the method of choice.

3.8.3 Comparison with HMMs

In general, the learning techniques for Bayesian networks are analogous to the learning techniques for HMMs. In both cases, EM is the everyday workhorse. Under some circumstances, however, gradient descent is required. For example, if a functional representation of conditional probabilities is used in a Bayesian network, it may not be possible to derive EM update equations. Similarly, when optimization criteria other than maximum likelihood are used in HMMs (such as minimum discrimination information (Ephraim *et al.* 1989) or maximum mutual information (Bahl *et al.* 1986)), it is necessary to resort to gradient descent techniques.

Chapter 4

DBNs and HMMs on Artificial Problems

4.1 Overview

This chapter presents a set of experiments demonstrating the advantages of a factored state representation. The experiments were done by generating data from a process that consists of multiple loosely interacting state and observation variables, and learning models with both factored and unfactored representations. HMMs are used to encode the unfactored representations, and DBNs are used to encode the factored representations. As expected, using a factored representation provides a significant advantage that increases as the number of underlying variables increases. In practice, the state representation in a model will never exactly match reality; therefore, experimental results are also presented that study the effect of learning with models that are either overly-simple or overly-complex. This data is adapted from (Zweig 1996), and further experiments can be found there.

4.2 Converting DBNs to HMMs

There is a simple procedure for constructing an HMM from a DBN. Recall that a discrete HMM is characterized by five quantities:

1. The number of states.

2. The number of observation symbols.
3. Transition probabilities between states.
4. Emission probabilities for observation symbols.
5. A probability distribution on the initial states.

Given a DBN, these quantities can be derived for an equivalent HMM as follows:

1. The number of HMM states is equal to the the number of ways the DBN state nodes can be instantiated. For example, if there are k binary DBN state nodes, there are 2^k HMM states.
2. The number of HMM observation symbols is equal to the the number of ways the DBN observation nodes can be instantiated.
3. To calculate the HMM transition probability from state i to state j, instantiate the DBN state nodes in a time slice t in the configuration that corresponds to HMM state i. Instantiate the state nodes in slice $t+1$ in the configuration that corresponds to j. Compute the product of the probabilities of the state nodes in slice $t+1$, given their parents. This is the transition probability.
4. Emission probabilities are calculated similarly, except that the state and observation nodes in a single time slice are instantiated.
5. To calculate the probability of being in state i initially, instantiate the DBN state nodes in time slice 1 in the configuration that corresponds to HMM state i. Compute the product of the probabilities of the state nodes in time slice 1, given their parents (if any). This is the desired prior.

Note that this is a one-way transformation. There is no known way of taking an HMM and constructing the *minimal* equivalent DBN. It is straightforward to create an equivalent DBN in which there is a single hidden variable in each timeslice, with as many possible values as there are HMM states. However, this representation is not necessarily minimal in terms of the number of parameters or computational requirements. There may be a much better equivalent factored representation.

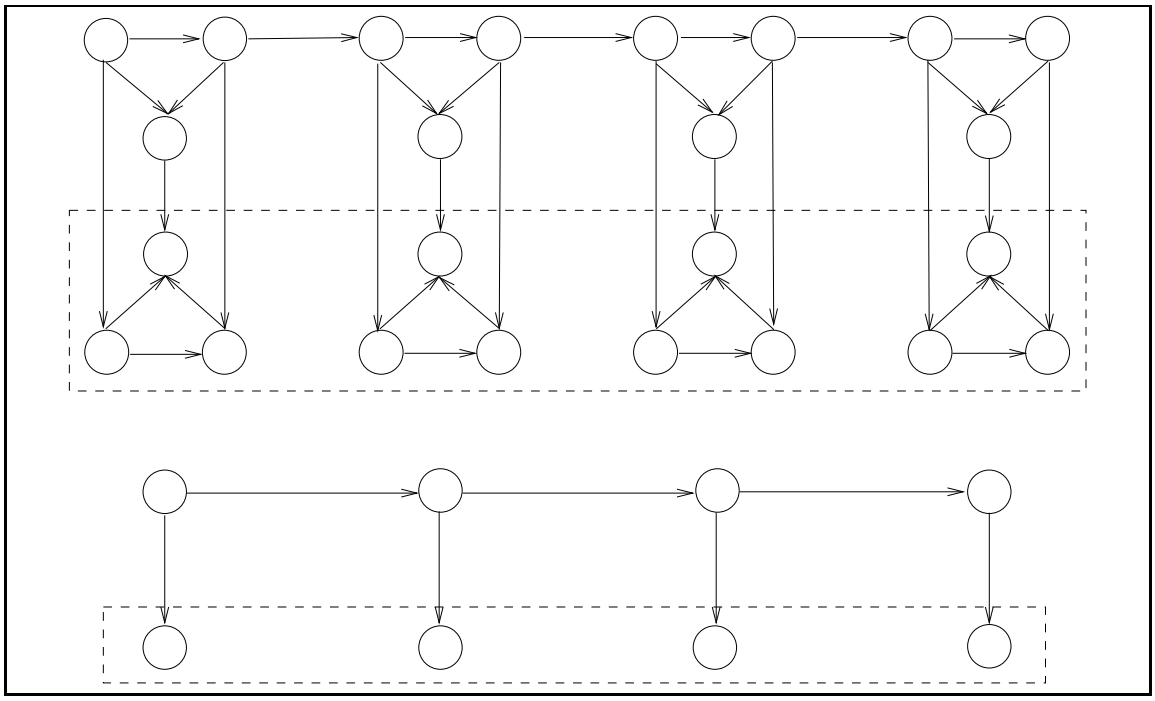

Figure 4.1: A $3-3$ DBN and an equivalent HMM. Both have been unrolled four time steps. The observation variables are boxed. The variables in the HMM can take on many more values than those in the DBN: each state variable must have a distinct value for each way the DBN's cluster of state variables can be instantiated. The same is true for the observation nodes.

4.3 Performance on a Family of Regular Graphs

In this section we compare the performance of DBNs and HMMs on a class of regular graphs. Each of these graphs has k state nodes and k observation nodes; we refer to such graphs as $k-k$ graphs. The state nodes within a time slice are connected in what would be a cycle, except that one of the arcs is reversed. The observation nodes in a time slice are connected to each other similarly. Each state node is also connected to the corresponding observation node in its time slice, and to the corresponding state node in the next time slice. A $3-3$ network is illustrated in Figure 4.1.

The results were obtained by using a $k-k$ DBN to generate training and test data. A topologically correct DBN was used to learn the distribution, and an unfactored HMM was constructed and trained with the same examples. All the variables were trinary, and the networks are initialized with random CPTs; the representations were initialized to

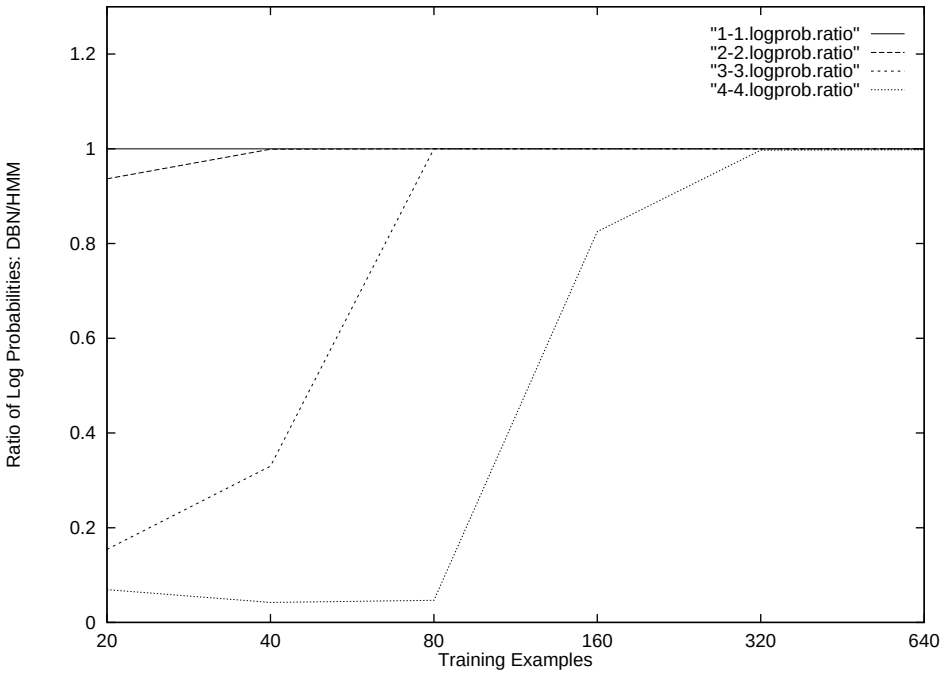

Figure 4.2: Solution quality as a function of the number of training examples. The horizontal axis is logscale. Large values represent good HMM performance.

equivalent starting points. The results reported are averages of 5 problem instances. Test cases that had 0 probability (because a particular combination of observations was never observed in the training data) were assigned the arbitrarily low log-likelihood of -1000. Note that for a $1-1$ network, there is a one-to-one correspondence between parameters in the HMM and the DBN, so identical performance is expected, and observed.

Figure 4.2 compares the test-set log-likelihood as a function of the number of training examples. The ratio of the HMM's score to the DBN's score is plotted. Since log-likelihoods are negative, low ratios indicate bad performance on the part of the HMM. These results clearly indicate the superiority of the DBN representation on $k-k$ graphs. The factored representation requires fewer parameters to represent the same distribution, and these results indicate that the factorization translates into better likelihood scores on the test data. As expected, this advantage increases rapidly with k.

Figure 4.3 compares the number of EM iterations required by the two algorithms. When few training examples are available, the DBN converges more rapidly, and to superior solutions.

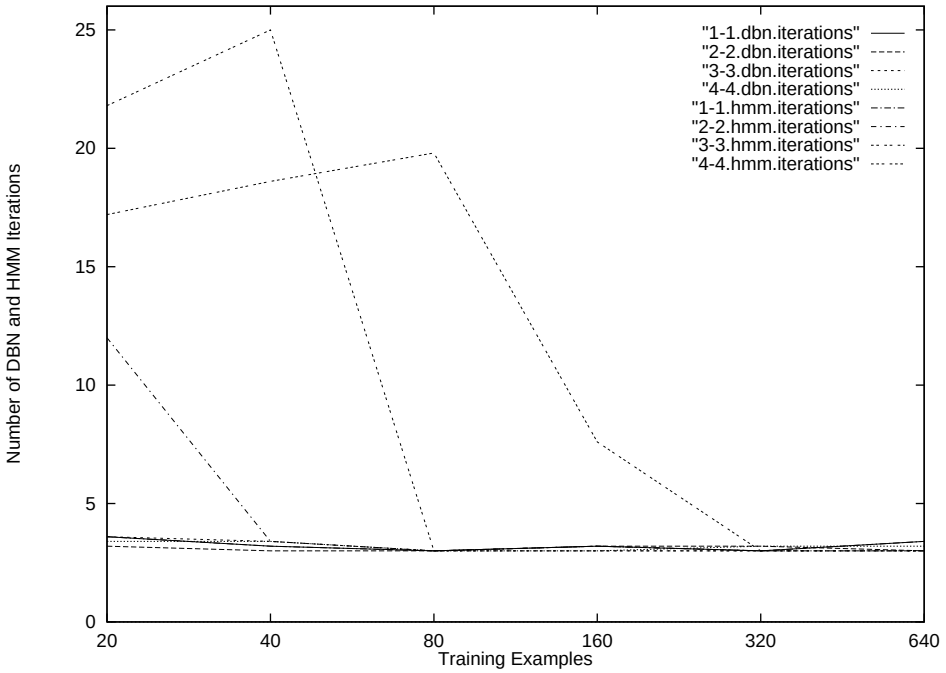

Figure 4.3: Absolute number of EM iterations required as a function of the number of training examples. The average number of iterations is about 3 except for the HMM on a $2-2$, $3-3$, and $4-4$ network, which require many more.

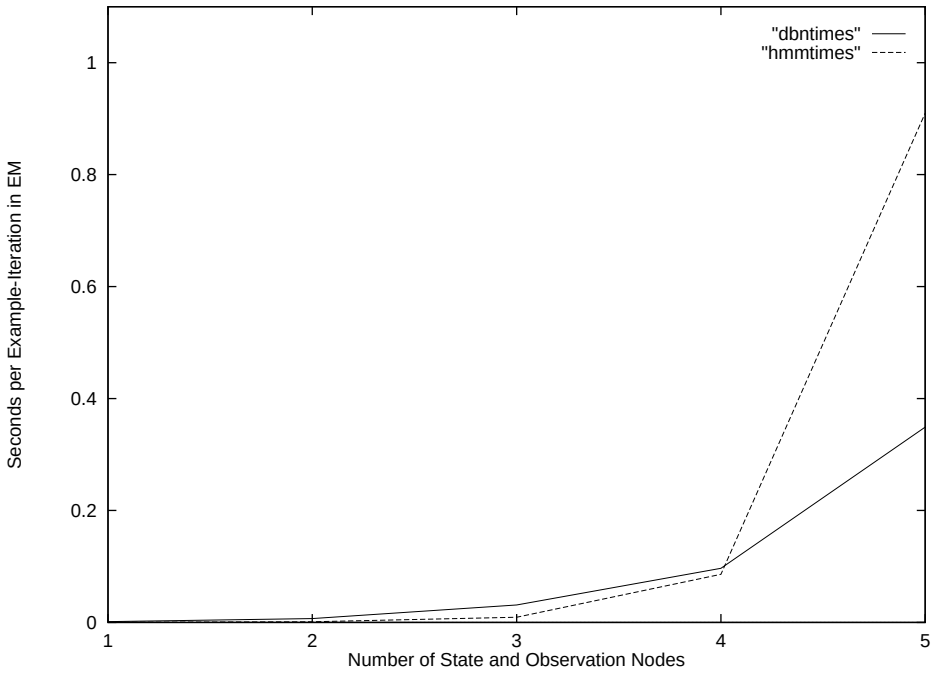

Figure 4.4: Time to process one example through one EM iteration. Times are shown for a DBN and analogous HMM. The horizontal axis shows k in a $k - k$ network.

Figure 4.4 shows the absolute amount of time required to process a single training example through one iteration of EM. The total running time is linear in the number of training examples and the number of iterations. Note that unlike the data on the number of iterations required, this data is implementation dependent.

For a $1-1$ network topology, the DBN and HMM are equivalent. Exactly the same results are generated. Since the DBN is a more general model, however, there is additional overhead, and the running time per example-iteration is about ten times greater. For a $4-4$ network, the running times are comparable, and the DBN is significantly faster on a $5-5$ network. This graph shows that the factored representation used by the DBN results in a significant decrease in the computational load, compared to an unfactored representation. As discussed in Section 3.7, this is not a question of parameter tying, and cannot be avoided with a simple modification to the HMM inference procedures.

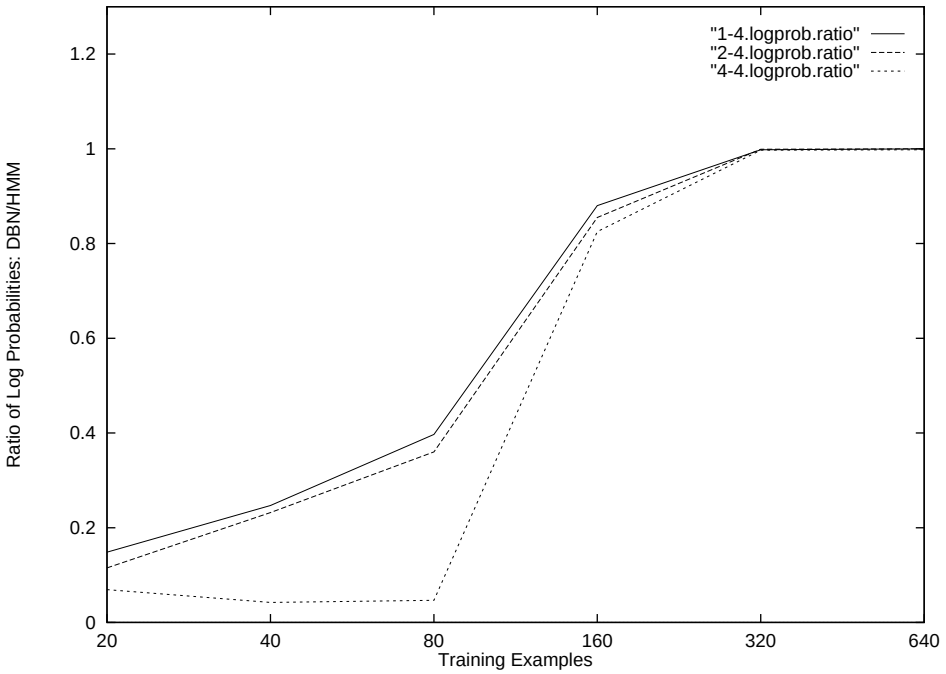

Figure 4.5: Solution quality as a function of the number of state nodes in the learned network. Note that the lines for the simpler models lie to the left of the line generated when the correct network is used. This indicates a faster increase in the HMM's performance.

4.3.1 Learning with an Incorrect Model

This section examines the effects of using an incorrect number of states in the learned model. The results show that an HMM benefits (in a relative sense) when too simple a model is used, and is harmed when too complex a model is used.

Another important issue is how the DBN's performance changes relative to that of the generating model. This can be determined by calculating the test-set log-probability with the model that generated the instances, and comparing this performance with that of the trained network. Here we find that when there are either too many or too few state nodes, performance is degraded.

Learning with too few States

Figure 4.5 shows relative solution quality as a function of training examples when too simple a training model is used. The data were generated with a $4-4$ network, and learned with $1-4$ and $2-4$ networks. In the $1-4$ network, the single state node in a time

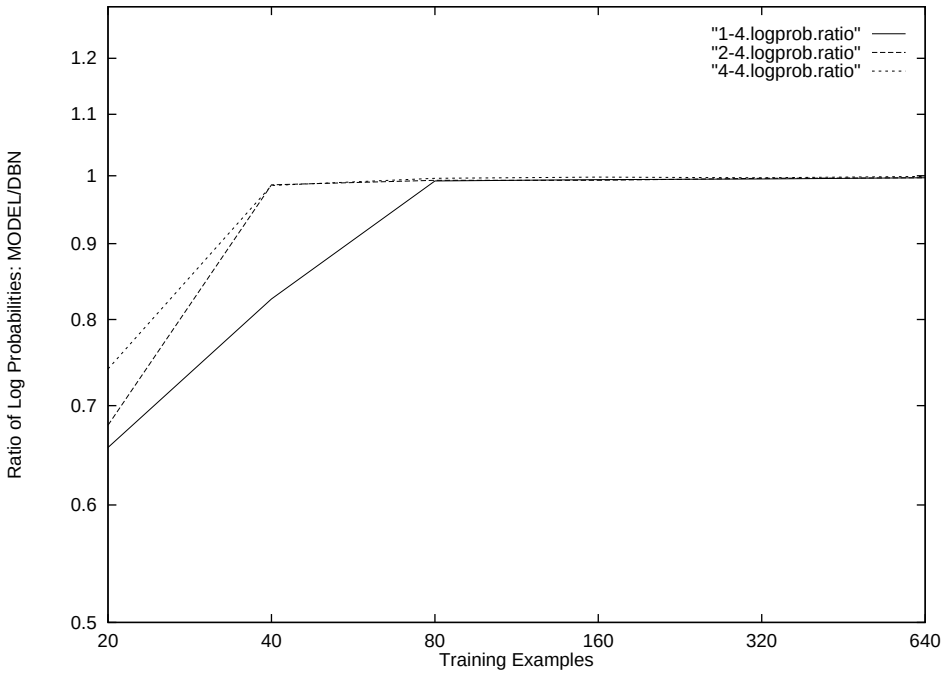

Figure 4.6: Log probability of the learned DBN model vs. log probability of the training model. The DBN's learning performance is degraded as the number of states in the learned model decreases.

slice was connected to all the observation nodes in that slice. In the $2-4$ network, each of the state nodes was connected to two observation nodes. The inter-state arcs were as in a $2-2$ network. The HMM had an equal number of states, and was started with the same transition, emission, and initial-state probabilities as the DBN.

Figure 4.5 shows that HMM performance increases more rapidly relative to DBN performance when too few states are available in the learned model. Figure 4.6 focuses on the DBN alone, and shows that learning requires more examples with too simple a network. Surprisingly, both a $1-4$ and $2-4$ network can closely approximate a $4-4$ network - but an examination of the tails of the distributions shows that their performance is never quite as good.

Learning with too many States

Figure 4.7 shows relative solution quality as a function of training examples when too complex a training model is used. Training data was generated with $1-4$ and $2-4$

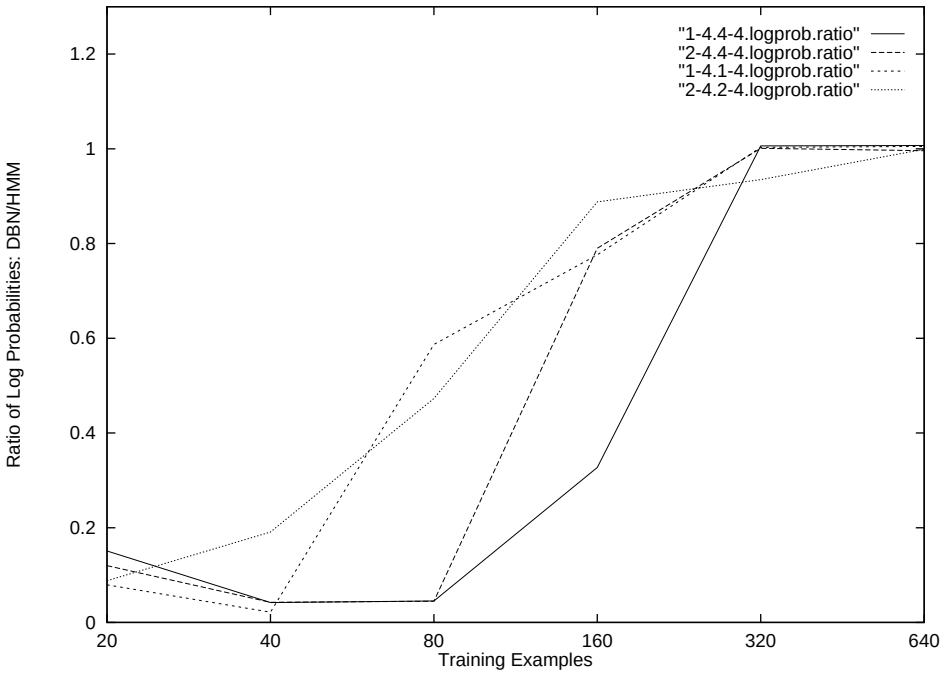

Figure 4.7: Solution quality as a function of the number of state nodes in the learned network. Note that the lines for the correct models lie to the left of the lines generated when the over-complex network is used. This indicates a slower increase in the HMM's performance.

networks, and learned with a $4-4$ network and its HMM analog. The performance impact in this case is the opposite of when too simple a model is used: the HMM takes longer to achieve equal performance.

Figure 4.8 compares the performance of the learned network with that of the generating network. Once again, an incorrect state model hampers performance. However, a close examination of the tail of the distribution shows that when sufficient training examples are available, the performance of the over-complex model equals or exceeds that of the correct model.

4.4 Discussion

These experiments demonstrate that when a system consists of several state and observation variables, a factored representation and the DBN inference procedures constitute a better modeling tool than an unfactored HMM. Although a factored HMM represen-

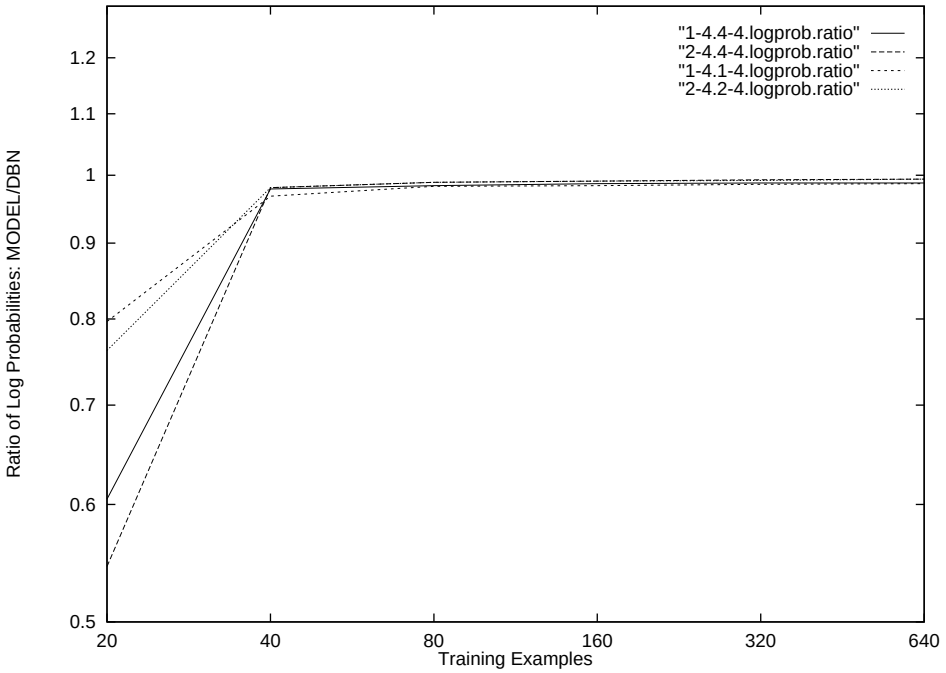

Figure 4.8: Log probability of the learned DBN model vs. log probability of the training model. Learning performance is degraded when the learned model has too many states and only a small number of training examples are available.

tation can be used to reduce the number of model parameters, inference with a cross-product representation is significantly more expensive than with DBN techniques, which are specially designed for factored representations. In our experimental results, the differences between HMMs and DBNs are most apparent when a small number of training examples are used. There is a range in which a trained DBN will produce good log-likelihood scores with respect to the generating model, but a trained HMM will do poorly. Remarkably, the HMM will also require many more EM iterations to converge.

In practice, models will have various kinds of inaccuracies, and it is important to study their effects on performance. We found that when there is too little state, an HMMs performance improves relative to a DBNs. When too many states are in the model, an HMM is harmed. In the networks we studied, both kinds of model were able to do well in either case - with a sufficient number of training examples.

Chapter 5

Speech Recognition

This chapter provides background material on automatic speech recognition (ASR). We begin with an overview of the problem and the general approaches that have been used to solve it. We then show in more detail how the techniques introduced in Chapter 2 can be applied to ASR. Finally, we describe some of the outstanding problems.

5.1 Overview

5.1.1 The Problem

Simply put, the problem of automatic speech recognition (ASR) is to program a computer to take digitized speech samples and print on the screen the words that a human would recognize when listening to the same sound. Over the course of the last fifty years, innumerable approaches to solving this problem have been developed, but despite their variety, it is possible to analyze them in terms of some common themes. There are several fundamental problems that must be overcome in any speech recognition system:

1. Acoustic representation. How will the information in the acoustic signal be represented?

2. Word representation. How will words be represented? Words are linguistic units, and as such could represent the atomic units in a recognition system. But words are also composed of syllables and phonemes, and these can also be used as the atomic units.

When a word can be pronounced in multiple ways, how will this be represented?

3. Linkage. In order to do recognition, we must link the acoustic representation of the signal to the word representations that derive from prior linguistic knowledge.

4. Training. It is generally agreed that a human cannot program a computer to recognize speech without giving the computer access to a large number of speech samples representing known utterances. Instead, the linkage between acoustic and word representations must be made by examining many examples for which the association is known. How exactly should this be done?

5. Recognition. Once the recognizer is trained, how exactly can it be used to do recognition?

In the following sections, issues are discussed in more detail.

Acoustic Representation

As a first step in ASR, the acoustic signal is processed to extract features that are higher-level than the raw sound wave itself. Although much early work was influenced by computational limitations, and therefore restricted to very simple mathematical models such as linear predictive coding (Makhoul 1975), more recently there has been an emphasis on models that are motivated by an understanding of the human auditory system (Davis & Mermelstein 1980; Ghitza 1991; Hermansky & Morgan 1994; Morgan *et al.* 1994). Since technical details are abundant in other work, e.g. (Rabiner & Juang 1993; Deller *et al.* 1993), and not central to our own, we will present only the basic ideas.

Essentially all speech representations begin by breaking the signal up into short time-frames, and computing a spectral representation of each of the frames. These frames are typically 25-30ms long, and overlap by 50-75%. The length of the analysis window is chosen to be long enough that good spectral estimates can be obtained, but at time same time short enough that each frame represents a stationary portion of the speech signal.

In practice, the spectral representation is considerably massaged before it is used. Cepstral representations are particularly useful. The simplest form of cepstral representation results from taking the cosine transform of the log of the original power spectrum. The resulting representation has many desirable properties (Noll 1964), including the fact that

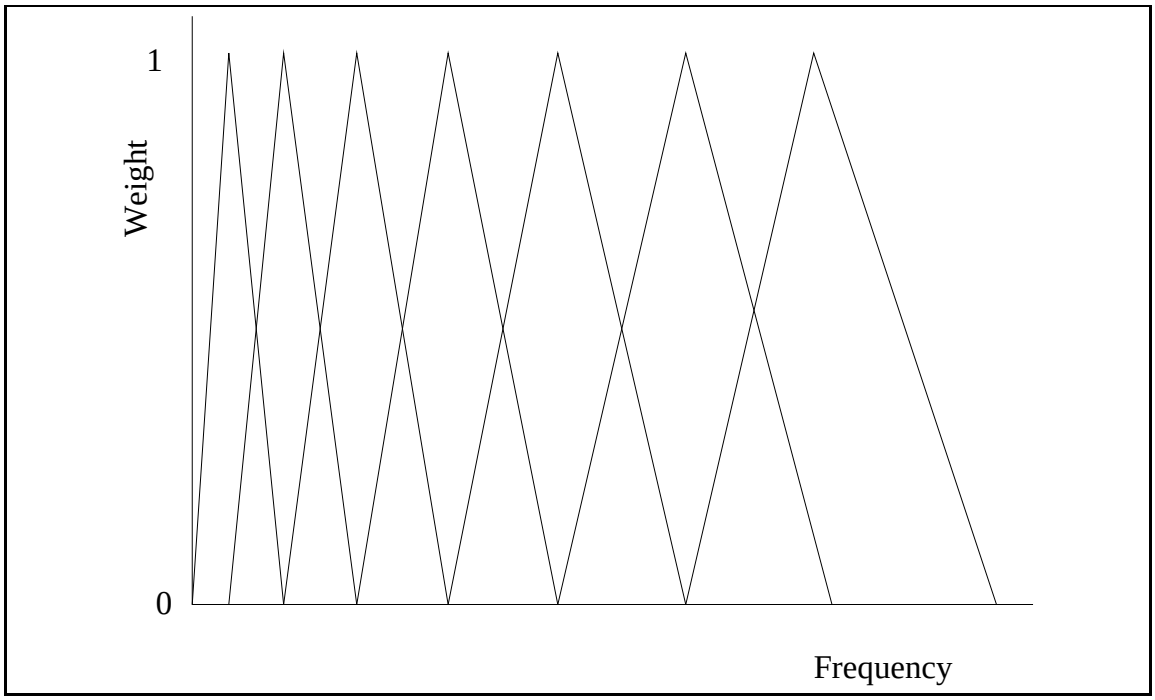

Figure 5.1: Overlapping, triangular, nonlinear MFCC-style filterbank. The peaks have a constant spacing on the mel-frequency scale. The output of each filter is a weighted sum of the sound energy in its frequency range.

the low-order coefficients tend to be correlated with the overall shape of the vocal tract, while the high-order coefficients are correlated with the presence of voicing. Additionally, by subtracting the mean value from each cepstral coefficient, it is possible to remove the effect of many telephone transmission characteristics (Mammone *et al.* 1996). Frequently, derivative features are computed from the basic cepstral representation. For example, the rate of change of the different cepstral coefficients might be computed, or even the second derivative.

There are two basic approaches to representing the spectral information extracted in this first stage of processing. In the first and most common approach, the spectral features are simply concatenated together into one long acoustic feature vector for each frame. Other systems use the method of vector-quantization instead (Linde *et al.* 1980). The basic idea of vector quantization is to find a relatively small number (e.g. 256-1024) of stereotypical spectra or cepstra. The spectrum of a frame can then be concisely represented by the index of the stereotype it is closest to, rather than by a whole vector of real-values numbers. This typically results in a factor of 40 reduction in the amount of space required to represent

a speech frame, and has the benefit of considerably simplifying all future computation by allowing for a completely discrete acoustic representation. It has the disadvantage that representation by a stereotype is inherently less precise than a full characterization. When vector-quantization is used, it is common to quantize the different spectral features (e.g. the cepstrum and its derivative) separately, and then to represent a speech frame by a combination of several vector indices (Lee 1989).

In many recent approaches to acoustic processing, an effort is made to mimic the processing of the human auditory system. Mel-frequency spectral warping (Davis & Mermelstein 1980) is illustrative. Mel-frequency cepstral coefficients (MFCCs) are computed in much the same way as ordinary cepstral coefficients, except that the power in different frequency ranges is added together to generate a "warped" spectral representation (see Figure 5.1). Approximately 20 overlapping frequency bins are used, and they are nonlinearly spaced. For example, the first frequency band might extend from 100 to 200Hz, while the last might extend from 3200 to 3600Hz (for telephone-quality speech). This nonlinear spacing emphasizes the high frequency range in a manner that is similar to the human auditory system. MFCCs have proven quite effective, and are now one of the most widely used spectral representations (Young 1996; Makhoul & Schwartz 1995).

Word Representation

The simplest way to represent words is atomically. If this is possible, the acoustics of a particular utterance are directly related to the words in the vocabulary; in essence, the system maintains an explicit model of how every word should sound. These types of systems have the advantage that they are explicit and direct, and can easily model any word-specific phenomena. However, they have the fatal disadvantages that in large vocabulary systems the parameters of the word models cannot be reliably estimated from the small number of examples that are commonly available. Therefore, their use is restricted mainly to small-vocabulary "command-and-control" applications such as navigating and automated help-systems. More sophisticated systems must use sub-word units as their atomic building blocks.

The simplest subword units that can be used are syllables. Syllables are relatively intuitive units that are often defined as consisting of a vowel and optional surrounding con-

sonants. (There is no completely agreed on precise definition, however: see, e.g. (Ladefoged 1993).) Unfortunately, there are still approximately 10,000 syllables in English (Rabiner & Juang 1993), so they suffer from some of the same drawbacks as whole word models. Despite the fact that syllable-based units are relatively rare in English-language ASR, recent work (Wu *et al.* 1997; Wu *et al.* 1998) indicates that they can be used effectively both on their own, and in combination with other schemes.

The next most atomic linguistic unit is the *phoneme*. Loosely speaking, a phoneme represents a maximal group of sounds that are similar enough to be used interchangeably. Languages can typically be described in terms of 40-100 phonemes, and taken together, the phonemes of a language cover the entire range of speech sounds. One can think of the set of phonemes as partitioning acoustic space into as few regions as possible such that the following rule is satisfied (Ladefoged 1993): every two sounds that can be used to differentiate between any pair of words are in different phonemic categories Thus, for example, /p/ and /t/ are distinct in English because the sounds distinguish "pie" from "tie." However, the /p/ at the beginning of "pop" is not sufficiently different from the /p/ at the end of "pop" that the two can differentiate between any English words. Another way of viewing the definition of phonemes in terms of "minimal word pairs." A minimal word pair is a pair of words with the same number of phonemes, and that differ only in one sound in one location, yet nevertheless have distinct meanings. An example is the pair "fine" and "vine" (Akmajian *et al.* 1995). They differ only in the initial sound, but have different meanings. The fact that the sounds /f/ and /v/ form the basis of a minimal word pair is proof that the sounds must be placed in separate phonemic categories in English.

The sounds that are grouped together in a phonemic category may have enough intra-group variation that they can be further subdivided into groups of *allophones*. Allophones are distinct enough that they can be distinguished from one another, yet not so distinct that they can form the basis of a minimal word pair. Allophones are atomic in the sense that linguists do not find further subdivisions, and these basic sounds are also referred to as *phones* (Akmajian *et al.* 1995). Table 5.1.1 lists one set of phones that is commonly used to represent the English language, adapted from (Deller *et al.* 1993).

Phonetic units based on phonemes or their allophones are attractive for use in ASR because a relatively small number of them can be combined to form all the possible words. In any reasonably sized dataset, each phoneme will occur many times, so accurate

Phone	Example	Phone	Example	Phone	Example	Phone	Example
IY	heed	V	vice	IH	hid	TH	thing
EY	hayed	DH	then	EH	head	S	so
AE	had	Z	zebra	AA	hod	SH	show
AO	hawed	ZH	measure	OW	hoed	HH	help
UH	hood	M	mom	UW	who'd	N	noon
ER	heard	NX	sing	AX	ago	L	love
AH	mud	EL	cattle	AY	hide	EM	some
AW	how'd	EN	son	OY	boy	DX	batter
IX	roses	F	five	P	pea	W	want
B	bat	Y	yard	T	tea	R	race
D	deep	CH	church	K	kick	JH	just
G	go	WH	when				

Table 5.1: The ARPAbet. This phonetic alphabet was adopted for use by ARPA, and is representative of phonetic alphabets.

models can be learned.

There are a variety of ways in which subword units can be combined to represent actual words. In the simplest schemes, a word model is represented by a single linear string of phones. For examples "dog" might be represented /D AO G/; cat might be represented /K AE T/, and so forth. At the next level of sophistication, alternative pronunciations can be represented simply by maintaining multiple linear sequences; for example, "tomato" might be represented by the word model

$$\{ \text{/T AH M EY T OW/, /T AH M AA T OW/ } \}.$$

Although simple, this method of word representation involves considerable redundancy, and in general word models are represented by directed graphs; see Figure 5.2. This representation has the advantage that it is easy to represent multiple words with concatenation: to link word A to word B, it suffices to take the two graphs, and add arcs from the terminal node of A to the initial nodes of B. This sort of representation is the standard in speech recognition.

Phoneme based units have the advantage of being few in number, but they have the significant disadvantage that the actual acoustic realization of a phoneme-based unit is highly variable, and moreover it varies in a somewhat predictable way depending on the nature of the surrounding phonemes. This effect is known as *coarticulation*, and is is a consequence of the way in which speech is generated.

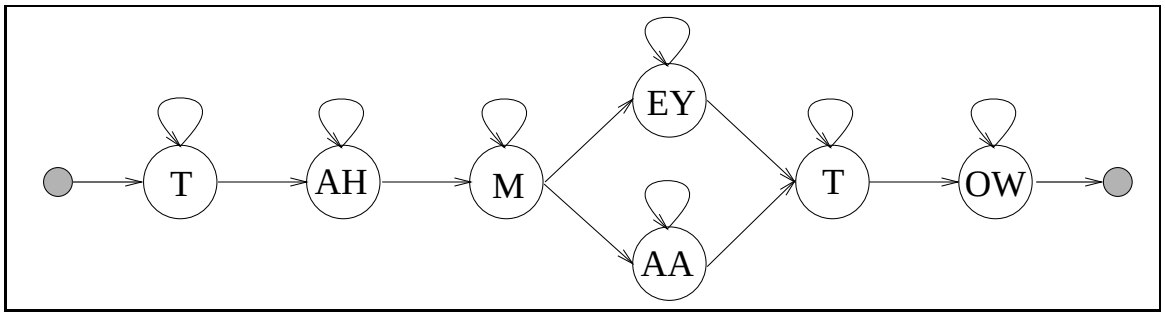

Figure 5.2: Word model for "tomato" showing two possible pronunciations.

One way of understanding speech production is as an acoustic filtering operation (Deller *et al.* 1993). In this model, a sound source forces air through the vocal tract, and the combination of the shape of the vocal tract and the type of sound source determines the sound that is produced. Fundamentally, the sound source is exhalation by the lungs, but there is an important distinction between a voiced and an unvoiced source. In voiced speech (e.g., when a vowel is uttered), periodic constriction of the vocal folds produces sharp, periodic changes in air pressure. In unvoiced speech, the vocal folds remain open, and the sound source is more chaotic in nature. The shape of the vocal tract is determined by the tongue, lips, jaw, and velum, and modulates the spectrum of the sound source. Together, the organs involved in producing and modulating speech sounds are known as the speech articulators.

It is possible to classify the different phonemes according to characteristic articulator positions and modes of excitation; see, e.g. (Ladefoged 1993; Browman & Goldstein 1992). However, the articulators are constantly in motion, and it is important to realize that they can move asynchronously and independently. It is in this context that coarticulation can be understood as occurring for at least two reasons:

1. The vocal apparatus is in a certain state after enunciating the immediately preceding unit, and cannot reach "target" positions, and

2. one or more of the articulators undergoes a modified motion in *anticipation* of an upcoming sound.

To address coarticulatory effects, speech recognition systems often use context-dependent phonetic alphabets, in which there are one or more units for each phoneme in

the context of surrounding phonemes. Several of the more common schemes are:

1. Biphones. This scheme comes in two flavors. In a left-context biphone alphabet, there is a phonetic unit for each phoneme in the context of every possible *preceding* phoneme. In a right-context biphone alphabet, there is a unit for each phoneme in the context of every possible *following* phoneme.

2. Diphones. In contrast to biphones, diphones do not represent entire phonemes. Rather, they represent the end of one and the beginning of another, and are thus transitional in nature (Schwartz *et al.* 1980).

3. Triphones. Each phoneme is represented in the context of every possible pair of surrounding phonemes.

In practice, not all combinations of phonemes are modeled in any of these schemes; only the frequently occurring ones. Also, it can be profitable to merge similar contexts, e.g. /b/ and /p/ into equivalence classes to reduce the number of possible contexts. This can be done either by hand, or through various automated clustering techniques (Lee 1989). Finally, hybrid schemes are common in which explicit word models for commonly occurring function words, e.g. "and," "of," "to," and "the" are used in combination with phone-based models for less frequent words.

Training and Recognition

The training task consists of taking a collection of utterances with associated word-level labels, and learning an association between the specified word models and the observed acoustics. Since it is impossible to be specific except with reference to a particular modeling approach, we defer a detailed discussion to the following sections, and simply point out the main difficulties:

- When the database consists of continuous speech, the word boundaries are not usually identified in the training data. Therefore, the program must either guess the word boundaries, consider all possible word boundaries, or consider some sort of weighted combination of the possible segmentations.

- Even when the database has only isolated words, the boundaries of the subword units are not usually available. Hence there is a problem analogous the missing word boundaries.

A large part of the training task consists of figuring out, implicitly or explicitly, which word model and which subword model to apply to each frame of an utterance.

In the recognition phase, the unknown utterance is compared to the various word models, and segmented in such a way that each part becomes associated with the most likely atomic speech unit. Again, one of the main problems will be to determine which word model and which subword unit to associate with each frame of the unknown utterance.

5.1.2 Approaches

We turn now to a brief description of two of the most commonly used techniques for associating word models to observed acoustic features.

Templates

The template-based approach to pattern recognition is highly intuitive: the basic idea is to store several examples of each word, and then to do recognition by comparing an unknown utterance to all the templates, and picking the one it most closely matches. In a template-based system, training can consist of simply storing all previously heard utterances. In a more sophisticated setting, one can cluster the examples of each word, and store only a small number of stereotypical examples, in a procedure analogous to vector-quantization.

In order to adopt the template approach, one need simply define a precise method for measuring the similarity of two utterances. Typically, this is done by defining the "distance" between the acoustic features associated with two frames of speech, aligning the two utterances, and adding the frame-wise distances. There are many possible ways of defining the distance between two speech frames (see, e.g. (Rabiner & Juang 1993)), but for concreteness, we may think of computing the Euclidean distance between two cepstral vectors.

The task of aligning a template pattern to a specific utterance requires some care.

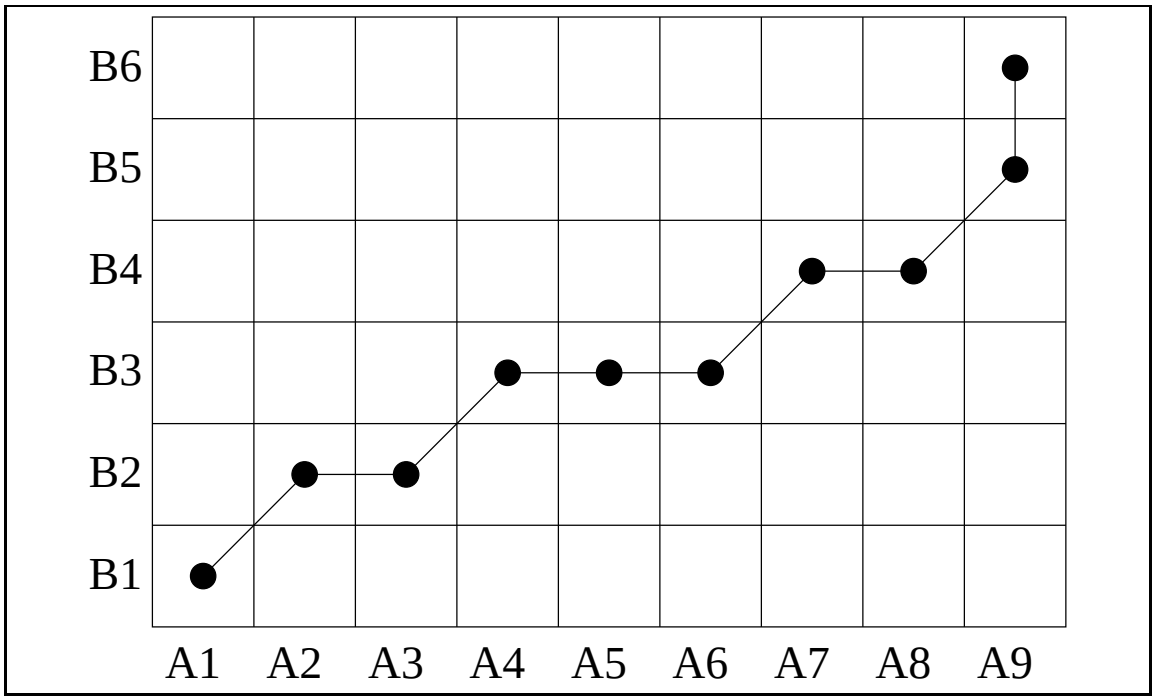

Figure 5.3: Utterance A is time-aligned to utterance B.

If the utterance and the template have the same number of frames, it is easy to pair them off and add the pairwise-distances. However, because of variations in speaking rate, this may not be the best thing to do, and because of the wide variability in the length of utterances, it is usually not possible at all. A more subtle solution stems from the use of dynamic programming (Bellman 1957), which makes it possible to quickly find the *best possible* alignment, even if the two utterances have different numbers of frames.

The method of dynamic programming, as applied to template matching, makes use of two simple data structures:

1. a two-dimensional array C in which entry $c_{i,j}$ holds the cost of pairing the ith frame of the reference pattern with the jth frame of the actual utterance, and

2. a two-dimensional array A in which entry $a_{i,j}$ holds the total cost of the best possible alignment that ends with the pairing of frames i and j.

Furthermore, there are two pieces of a-priori information that are used:

1. the first frame of the utterance must be paired with the first frame of the template,

and

2. the last frame of the utterance must be paired with the last frame of the template.

We will assume that the template has m frames, and the utterance n frames. The process of dynamic programming consists of starting with the first of these facts, and then methodically figuring out for each array position $a_{i,j}$ the least cost path from array position $a_{1,1}$ to $a_{i,j}$. We state the procedure below, with the assumption that $c_{0,0} = 0, c_{0,j} = \infty, j > 0, c_{i,0} = \infty, i > 0$; ties may be broken arbitrarily.

$$a_{i,j} = c_{i,j} + min(a_{i-1,j},\ a_{i,j-1},\ a_{i-1,j-1})\ i > 0, j > 0$$

Once the $a_{i,j}$ have been computed in this fashion, the cost of the best possible alignment is stored in $a_{m,n}$. Recognition simply consists of computing the cost of the best alignment of the utterance to each of the reference patterns, and keeping track of the best. If desired, the actual path can be reconstructed by storing a small amount of extra information as the algorithm proceeds. The alignment of an utterance to a template is illustrated in Figure 5.3. The use of template-based systems for continuous word recognition is somewhat more complicated, but proceeds along basically the same lines (Ney 1984).

Template-based recognizers formed the basis of many early speech recognition systems, but suffer from the problem that it is not possible to decompose a training utterance into sub-utterance level units. Note that this deficiency is apparent only in the *training* phase, when no sub-utterance level templates are available to begin with. For example, given a string of connected words as its input, a template-based system that does not already have word models is unable to segment the utterance into its component words. This means that the words in the training phase must be spoken in isolation, and connected word models derived by concatenating isolated word templates; this results in a poor model of inter-word coarticulatory effects. Again, this difficulty occurs only during training; once a set of atomic templates is available, long utterances can be segmented into these atomic units in an optimal way (Ney 1984). An additional problem arises when there are several qualitatively different acoustic features associated with each frame, and a single distance measure must be defined that weights them all appropriately. These problems make template-based recognizers attractive only for simple isolated-word command-and-control type applications; for example, they are currently used in many speaker-dependent voice-dialing systems.

Statistical Pattern Recognition

Statistically-based ASR systems are based on the notion that an utterance is represented by some sequence of acoustic features $\mathbf{a}$ that derives from some underlying sequence of words $\mathbf{w}$, and that the two can be probabilistically related. More specifically, the goal of a statistically based ASR system is to find

$$\arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{a}).$$

It is often beneficial to rewrite this using Bayes' rule as

$$\arg\max_{\mathbf{w}} \frac{P(\mathbf{w})P(\mathbf{a}|\mathbf{w})}{P(\mathbf{a})} = \arg\max_{\mathbf{w}} P(\mathbf{w})P(\mathbf{a}|\mathbf{w}).$$

The reason for doing this is that it breaks the problem into two subproblems, each of which can be tackled independently. The first is the problem of computing the probability of a sequence of words, $P(\mathbf{w})$; this can be done by constructing a *language model* that specifies the probability of strings of words in a language. Examples of language models include stochastic context free grammars, and bigram and trigram models (Allen 1995). The second problem consists of computing the probability of an observed sequence of acoustic features, given an assumed word sequence: $P(\mathbf{a}|\mathbf{w})$. This can be done with a generative model that explains how acoustics are generated for a given word sequence. Although this decomposition is fairly standard, it it by no means the only way of constructing a probabilistic model; as we will see, neural-net based systems decompose the problem in a somewhat different way. In the following sections, we will examine several important ASR methodologies in more detail; since they are all based on probabilistic modeling, we defer the specifics of training and recognition to those sections.

5.2 Standard Techniques

In the following sections, we describe in more detail how the temporal modeling techniques introduced in Chapter 2 can be applied to the speech recognition task. Since the generalization to connected word recognition is in all cases straightforward, we focus, for simplicity, on isolated word recognition.

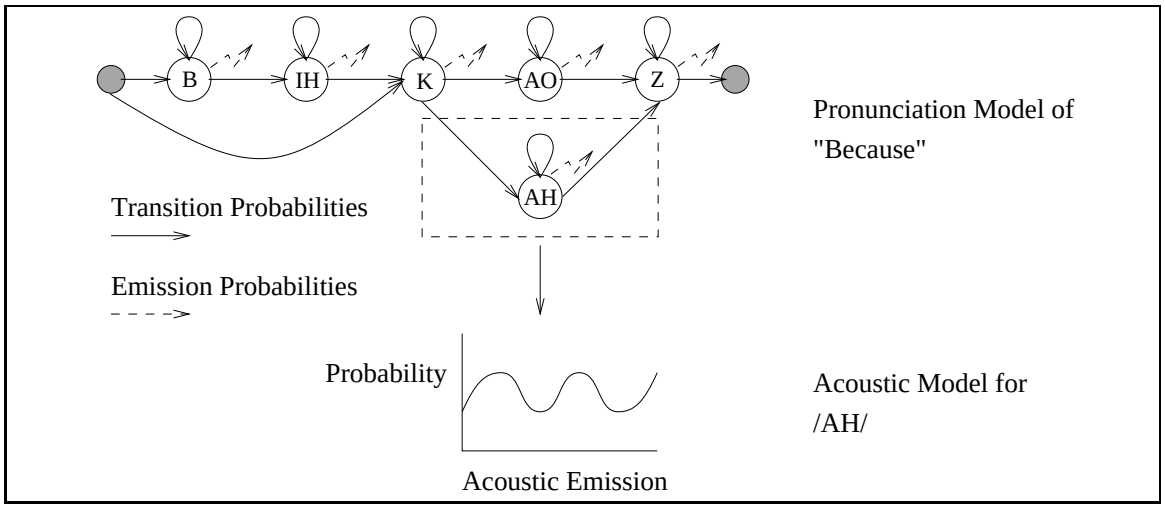

Figure 5.4: An HMM for the word "because." The transition matrix is defined graphically by the solid arcs; if there is no arc between two states, the transition probability is 0. The small shaded nodes represent artificial initial and final states.

5.2.1 Hidden Markov Models

In this section, we elaborate the use of hidden Markov models in ASR. As we have seen, an HMM consists of a set of states with associated transition and emission probabilities. HMMs are easily applied to ASR by associating the states with sub-word phonetic states, and associating the emissions with sounds. This is illustrated in Figure 5.4.

Since $P(\mathbf{w})$ is computed with a language model outside the scope of the HMM, we need only worry about computing $P(\mathbf{a}|\mathbf{w})$. (And since we are considering isolated words, we may replace $\mathbf{w}$ by w.) From Section 2.2, this corresponds to:

$$\begin{aligned} P(\mathbf{a}|w) &= \sum_{\mathbf{q}} P(\mathbf{q}, \mathbf{a}|w) \\ &= \sum_{\mathbf{q}} P(\mathbf{q}|w) P(\mathbf{a}|\mathbf{q}, w) \\ &\approx \sum_{\mathbf{q}} P(q_1|w) P(a_1|q_1, w) \prod_{i=2}^{n} P(q_i|q_{i-1}, w) P(a_i|q_i, w). \end{aligned}$$

In words, this corresponds to the sum over all the paths through the HMM of the probability of the path multiplied by the probability of the acoustics given the path.

In the training phase, the required transition and emission probabilities are determined, and in the recognition phase, each possible word hypothesis is evaluated.

5.2.2 Neural Networks

Neural-net based systems also relate the acoustics to subword phonetic units collected together in graphical word models, but they are distinctive because they use a completely different factorization of $P(\mathbf{w}|\mathbf{a})$ (Bourlard & Morgan 1994; Hennebert *et al.* 1997). The factorization is as follows:

$$\begin{aligned} P(\mathbf{w}|\mathbf{a}) &= \sum_{\mathbf{q}} P(\mathbf{q}, \mathbf{w}|\mathbf{a}) \\ &= \sum_{\mathbf{q}} P(\mathbf{q}|\mathbf{a}) P(\mathbf{w}|\mathbf{q}, \mathbf{a}) \\ &\approx \sum_{\mathbf{q}} P(\mathbf{q}|\mathbf{a}) P(\mathbf{w}|\mathbf{q}) \end{aligned}$$

The main novelty is that the state sequence is conditioned on the observation sequence rather than vice-versa; furthermore, a NN-based system estimates $P(\mathbf{q}|\mathbf{a})$ with a neural net. We will discuss this in more detail in the following two sections, and pause here only to note that the evaluation of $P(\mathbf{w}|\mathbf{q})$ is somewhat tricky.

The factor $P(\mathbf{w}|\mathbf{q})$ is more complicated than anything that occurred in the HMM specification, because it must carry the load of a language model, and is additionally conditioned on a state sequence. One way of dealing with this is through Bayes rule:

$$P(\mathbf{w}|\mathbf{q}) = \frac{P(\mathbf{w})P(\mathbf{q}|\mathbf{w})}{P(\mathbf{q})}$$

This reduces the problem to computing the probability of the word sequence with a language model as before, and computing the probability of the state sequence given the word sequence as before, and computing a prior for the state sequence. This can be approximated by $P(\mathbf{q}) = \prod_i P(q_i)$, with $P(q_i)$ estimated from Viterbi decodings of the training data. The final specification of the scheme is (Hennebert *et al.* 1997):

$$P(\mathbf{w}|\mathbf{a}) = \sum_{\mathbf{q}} P(\mathbf{w})P(\mathbf{q}|\mathbf{a}) \frac{P(\mathbf{q}|\mathbf{w})}{\prod_i P(q_i)}$$

Note that if it is assumed that there are no homonyms, then for isolated words with equal priors, we have:

$$P(w|\mathbf{a}) \propto \sum_{\mathbf{q}} P(\mathbf{q}|\mathbf{a}).$$

MLPs

One way of estimating $P(\mathbf{q}|\mathbf{a})$ is with an MLP. The most common assumption that is made (Hennebert *et al.* 1997) is that

$$P(\mathbf{q}|\mathbf{a}) = \prod_i P(q_i|a_{i-k}..a_{i+k}).$$

The probability of a state sequence is the product of factors, each of which is conditioned on a small amount of acoustic context. Typically, k might be 4, so that the probability of a phonetic state at time t is a function of about 9 frames of surrounding speech. The task is thus to estimate a distribution over phones from a small number of speech frames.

This can be done with an MLP in which the output layer has a node to represent each phonetic unit. Since sigmoidal activation functions naturally lie in the range $[0 \ldots 1]$, the output of unit i can be taken to be the probability of phone i. In a properly trained network, it will also be the case that the output activations sum to 1 (Bourlard & Morgan 1994).

In order to train an MLP-based system, there must be a set of training patterns consisting of

1. the input frames of speech and
2. a desired output distribution over phones.

Such a training set can be obtained in the following way:

1. For each training utterance, compute the marginal distribution over phones for each frame, using the current network parameters. This requires the use of a procedure analogous to the forward-backward algorithm in HMMs (Hennebert *et al.* 1997).
2. Create a training examples from each frame so labeled.

The complete procedure for training an HMM-based system consists of alternating between creating labeled training examples with the current network parameters, and re-training the MLP with the new (self-generated) examples.

Since the training procedure for MLP-based systems is somewhat more complicated than that of HMMs, one might wonder why they should be used. There are three compelling reasons:

1. The probability of a phone can be conditioned on a large amount of acoustic context. This potentially gives neural-net based systems an edge in modeling coarticulatory effects.

2. The training procedure directly maximizes $P(\mathbf{w}|\mathbf{a})$ in a discriminative way. This is in contrast to maximum likelihood based methods which maximize $P(\mathbf{a}|\mathbf{w})$ as a surrogate.

3. No assumptions of conditional independence between different acoustic features are required.

Recurrent NNs

RTR-NNs provide another way of estimating $P(\mathbf{q}|\mathbf{a})$. Again, the main potential benefit is in modeling acoustic and articulatory context. Here, however, it is not necessary to provide the context explicitly with each frame. Since the network has a long-term memory of its own, it can implicitly keep track of important previously seen features. Descriptions of these types of systems can be found in (Robinson & Fallside 1991; Robinson & Fallside 1988).

5.2.3 Kalman Filters

As we have seen, Kalman filters are ideally suited to tracking the motion of an object in a multidimensional space. To adapt the methodology to speech recognition, one can simply treat the acoustic feature-vector as an object, and track its trajectory through acoustic-feature space. In order to model subword phonetic units, the idea is to create a separate Kalman filter for each unit, and tune it so that trajectories that are typical of the phone receive a high probability according to the model.

If hand-segmented utterances are available, then the procedure is relatively simple: each phone model can be trained on acoustic trajectories known to be associated with it. Otherwise, a two-stage procedure is necessary in which the current phone models are used to generate a segmentation, and then the model parameters are retrained using this partitioning. Note the similarity to the procedures used for neural nets; the only difference is that phone probabilities are estimated using a Kalman filter rather than a neural net.

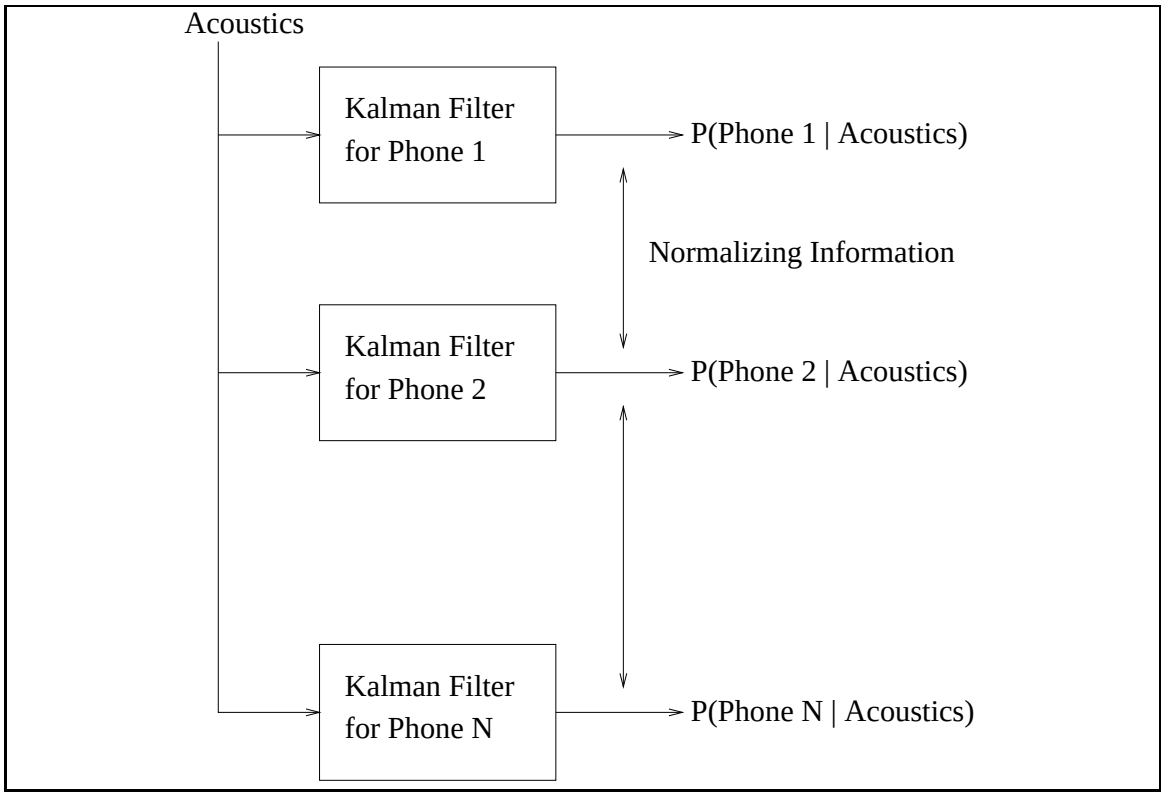

Figure 5.5: A Kalman filtering approach to ASR, loosely adapted from Anderson and Moore, 1979. The probability of a phone q_i at time t is recursively calculated from the acoustic input a_t, and all prior acoustic input, a_1^{t-1} by $P(q_i|a_1^t) = \frac{P(a_t|a_1^{t-1},q_i)P(q_i|a_1^{t-1})}{\sum_{j=1}^{N} P(a_t|a_1^{t-1},q_j)P(q_j|a_1^{t-1})}$. All the required quantities are readily available.

Figure 5.5 illustrates the way that phone probabilities are computed from the acoustic input.

Somewhat different schemes have actually been tested. In (Digalakis *et al.* 1993), a procedure for normalizing segment lengths is used in conjunction with Kalman filtering to model phonemes from pre-segmented data; the authors report good results. In (Kenny *et al.* 1990), an approach is used in which the hidden state is related to the observation vector by the identity matrix, and several frames of the past are used to predict the present. The authors applied their scheme to connected word recognition, and report that their best results came from a more conventional HMM.

5.3 Outstanding Problems

In the preceding sections, we have reviewed several standard techniques for ASR. Although they all perform well in many circumstances, there are some generally accepted problems (Rabiner & Juang 1993; Deller *et al.* 1993; Young 1996; Makhoul & Schwartz 1995):

- Coarticulation. This can be handled with biphones, triphones, diphones, syllables, or word-dependent phone models, but there are associated problems:
  1. None of these methods pays attention to the specifics of an actual utterance. In other words, they all embody *a-priori* information that can be stated without reference to any specific acoustic observations. Hence, they miss utterance-specific cues.
  2. There is a large increase in the number of parameters, and complex estimation techniques must be used.
- Sensitivity to speaking style. The expected pronunciations and acoustics of words are affected both by dynamic factors as speaking rate, and static factors such as gender, age, and accent.
- Sensitivity to the acoustic environment. Current systems can be catastrophically affected by even mildly noisy conditions, for example soft background music or room reverberation.

Much current research in ASR focuses on ways of overcoming these problems, and the following chapter will address the issues with DBNs.

Chapter 6

Speech Recognition with DBNs

This chapter describes the use of dynamic Bayesian networks in speech recognition, and shows how they can be structured to address the outstanding problems outlined in the previous chapter. Although many papers, e.g. (Smyth *et al.* 1996; Ghahramani & Jordan 1995) have mentioned the possibility, the details of implementing a working system have not been previously addressed. In order to apply DBNs to ASR, it is necessary to develop a technique for combining subword phonetic models into whole word and multiple-word models, and the chapter begins by describing the process of model composition with DBNs. This is done in such a way as to allow for parameter tying between multiple occurrences of the same phone model, efficient computation, and the generality of word models structured like arbitrary directed graphs. After describing the basic technology required for ASR with DBNs, the chapter continues with a description of several important DBN structures.

6.1 Model Composition with DBNs

6.1.1 Motivation

We have seen that standard approaches to speech recognition concatenate smaller models into larger ones: sub-phonemic units into phonemes, phonemes into words, and words into sentence structures. This procedure is by no means exclusive to speech recognition; many other temporal processes evolve through a series of distinct stages, each of which is best represented by a separate model. For example, the process of writing a word can

be decomposed into the sequential formation of its letters. Television broadcasting can be decomposed into programming and advertising segments, and driving can be decomposed into sequences of lane-changing, accelerating, braking, and similar maneuvers. When modeling these processes, it is convenient to create submodels for each stage, and to model the entire process as a composition of these atomic parts. By factoring a complex model into a combination of simpler ones, composition achieves a combinatorial reduction in the number of models that need to be learned.

Model composition raises two crucial but independent issues. The first is the *specification of legal submodel sequences*. In this chapter, we consider the use of stochastic finite-state automata (SFSAs) to describe a probability distribution over possible submodel sequences. This is a fairly standard choice in areas such as speech and handwriting recognition. The second issue is *submodel representation*, for which we use Bayesian networks to specify the behavior of each submodel.

6.1.2 Encoding an SFSA with a DBN

Why Model Composition with DBNs is Difficult

The difficulty in concatenating DBN models is best illustrated with an example. Suppose the word "no" is uttered, and there are separate models for the phonemes /n/ and /ow/. The top of Figure 6.1 shows the simplest possible such submodels. There is a hidden state variable representing articulator positions, and a variable representing the sound observed at each point in time. The state CPTs specify articulator dynamics, and the observation CPTs link the articulator positions to the sounds made. Each submodel can be duplicated for an arbitrary number of timesteps.

Now consider constructing a composite model for a specific fixed-length utterance of the word "no." This is shown at the bottom of Figure 6.1. A naive concatenation of the two models would have to explicitly partition the model into a fixed-length /n/ prefix followed by a fixed-length /ow/ suffix. In practice, however, such segmentations are unavailable. Therefore, when doing inference or learning, all reasonable partitionings of an observation sequence between the models must be considered, and possibly weighted according to some distribution. Since the number of partitionings grows exponentially with the number of submodels, this is a demanding task that must be solved in an efficient way.

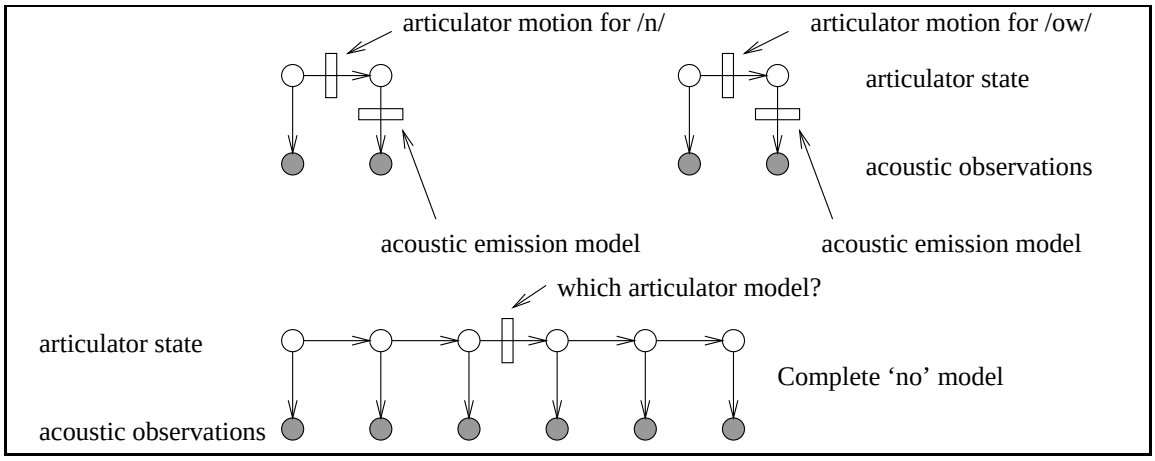

Figure 6.1: Concatenating submodels. Naive submodel concatenation requires specifying which state-evolution model to use at each point in time.

In the following sections we show how to construct a DBN that represents the distribution over partitionings specified by an arbitrary SFSA.

Encoding an SFSA

This work was first presented in (Zweig & Russell 1997). Consider the SFSA shown at the top of Figure 6.2. The nodes in this diagram represent states, and there are transition probabilities associated with the arcs. The initial and final states are shaded. We interpret the initial state to represent the history of the system prior to the first point in time that has an observation associated with it. The final state represents the future of the system after the last point for which there is an observation. The states in the automaton of relevance to the observation sequence are the unshaded nodes in Figure 6.2. We refer to the initial state as s_I, to the final state as s_F, and to an arbitrary state i as s_i. The probability of a transition from state s_g to state s_h is denoted by $P_{s_g s_h}$.

The probability of a length k path $s_1 s_2 \ldots s_k$ through the automaton is given by $P_{s_I s_1} P_{s_1 s_2} \ldots P_{s_{k-1} s_k} P_{s_k s_F}$. The DBN at the bottom of Figure 6.2 represents paths of length k through this structure in a somewhat different way. Each state variable in the DBN represents the position in the SFSA at a *specific time*. The DBN state variables $M^1 \ldots M^k$ have a distinct value for each state in the automaton, and there is a one-to-one mapping between paths of length k through the automaton and assignments of values to the variables in the DBN. So, for example, the path $1, 3, 4, 4, 7$ through the SFSA corresponds

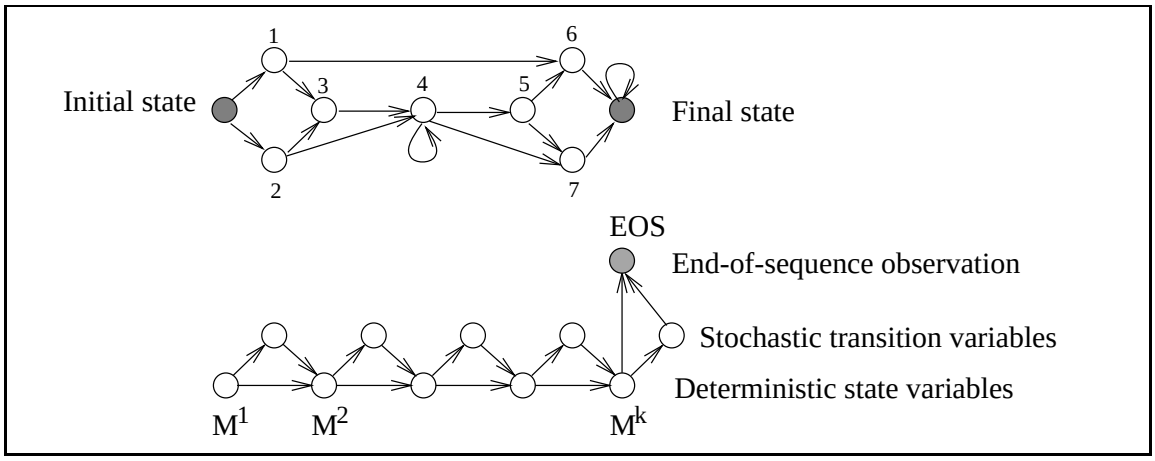

Figure 6.2: An SFSA and a DBN network representation for fixed-length observation sequences. Note that in the automaton the arcs represent transition probabilities while in the Bayesian network they represent conditional independence relations. The initial and final states of the SFSA are shaded. The shaded node in the DBN represents an artificial observation; the CPT of this variable will encode the length of the observation sequence.

to the assignments $M^1 = 1, M^2 = 3, M^3 = 4, M^4 = 4, M^5 = 7$.

The DBN transition variable encodes which arc is taken out of the SFSA state at any particular time. The number of values the transition variable can take is equal to the maximum outdegree of any of the states in the SFSA. The probability distribution over transition values is determined by the state value at that point in time, and will be used to encode the appropriate transition probabilities. For all the time slices after the first, the probability distribution over states at time $t+1$ is simply a deterministic function of the state and transition variable values at time t; it encodes the fact that if we know the SFSA state at time t, and the SFSA arc taken at that time, then we know the SFSA state at time $t+1$. The probability distribution over state values in the first time-slice is non-deterministic, and reflects the distribution over successors to the SFSA initial state. The shaded DBN node represents a binary-valued variable whose value is always observed to be 1. The CPT of this "end-of-sequence observation" will encode the fact that the observation sequence is k steps long.

The transition probabilities associated with the arcs in the automaton are reflected in the CPTs associated with the transition variables in the DBN. Denote the transition variable at time i by T^i. We will denote the index of the arc leading from state s_g to state s_h by $a_{s_g}^{s_h}$, e.g. if the second arc out of s_g leads to s_h, then $a_{s_g}^{s_h} = 2$. If the probability of

transitioning from state s_g to state s_h in the automaton is $P_{s_g s_h}$, then $P(T^t = a_{s_g}^{s_h} | M^t = s_g) = P_{s_g s_h}$ in the CPT associated with T^t. Note that this relation does not depend on t, and therefore a single CPT can be shared by all instances of the transition variable. The same is true for state variables from M^2 on. Assignments to transition variables that do not correspond to any arc, i.e. those whose value exceeds the state's outdegree, receive 0 probability.

All paths through the SFSA must start in one of the successors of the initial state s_I, and end with a transition to the final state s_F. Suppose the SFSA states are numbered $1 \ldots n$, exclusive of the initial and final states. By setting the prior distribution on M^1 to $P(M^1 = s_g) = P_{s_I s_g}$, the constraint on initial states is satisfied. By setting the conditional probability on the end-of-sequence observation EOS to $P(EOS = 1 | M^k = s_g \in predecessors(s_F), T^k = a_{s_g}^{s_F}) = 1$, and $P(EOS = 1 | M^k = s_g \notin predecessors(s_F)) = 0$, we ensure that any assignment of values to the variables which does not terminate with a transition to the final state is assigned a probability of 0. Note that s_F has a selfloop; we define $predecessors(s_F)$ to exclude s_F itself. This will ensure that 0 probability is assigned to DBN variable assignments that end by cycling in s_F. We summarize with the following Theorem.

Theorem 6.1 *Every assignment of values to the variables in the DBN either:*

1. *corresponds to a legal path through the SFSA and is assigned a probability equal to the probability of the path in the SFSA, or*
2. *corresponds to an illegal path in the SFSA and is assigned a probability of* 0.

Proof. Let the DBN consist of k time-slices. First assume the assignment corresponds to an illegal path. It must be illegal because one or more of the following are true:

1. $M^1 = s_g,\ s_g \notin successors(s_I)$.
2. There is an assignment $M^i = s_g, M^{i+1} = s_h$ for which there is no transition arc in the SFSA.
3. $M^k = s_g,\ s_g \notin predecessors(s_F)$.
4. $M^k = s_g,\ s_g \in predecessors(s_F),\ T^k \neq a_{s_g}^{s_F}$.

These cases will be assigned 0 probability because by construction

1. $P(M^1 = s_g \notin successors(s_I)) = 0.$
2. $P(T^i = k | M^i = s_g) = 0,\ k > outdegree(s_g).$
3. $P(EOS = 1 | M^k = s_g \notin predecessors(s_F)) = 0.$
4. $P(EOS = 1 | M^k = s_g, T^k \neq a_{s_g}^{s_F}) = 0.$

Now assume the assignment of values corresponds to a legal path. Let the path be $s_1 s_2 \ldots s_{k-1} s_k$. The assignment to the Bayesian network variables is:

$$M^1 = s_1,\ T^1 = a_{s_1}^{s_2},\ M^2 = s_2,\ \ldots, M^{k-1} = s_{k-1},\ T^{k-1} = a_{s_{k-1}}^{s_k}, M^k = s_k, T^k = a_{s_k}^{s_F}.$$

The probability assigned by the DBN has the factors:
$P(M^1 = s_1) = P_{s_I s_1},\ P(T_1 = a_{s1}^{s_2} | M^1 = s_1) = P_{s_1 s_2}, \ldots,$
$P(T_{k-1} = a_{s_{k-1}}^{s_k} | M^{k-1} = s_{k-1}) = P_{s_{k-1} s_k},\ P(T_k = a_{s_k}^{s_F} | M^k = s_k) = P_{s_k s_F}$
and all other factors are 1. The probability assigned by the SFSA is the product of exactly the same factors, and is therefore identical.

Although it is possible to encode the same information without using transition variables - in CPTs associated with stochastic state variables - the use of an explicit transition variable is usually much more efficient. This is because not all transitions are possible in the SFSA, and the combination of explicit transition variables with deterministic state variables compactly encodes the possibilities. This is particularly true when the maximum outdegree is small compared to the number of states.

Model Composition

Figure 6.3 illustrates the most general way in which model composition is achieved. The submodel-index variable specifies which submodel to use at each point in time. There is also a transition variable, as in the previous section; together we call these variables the control layer. The constraints on legal sequences of submodels are encoded in the CPTs of this layer, as described previously. The submodel state layer represents the hidden variables in the DBN submodels. By conditioning the submodel state variables on the submodel-index variable in the control layer, the desired switching behavior between models is achieved.

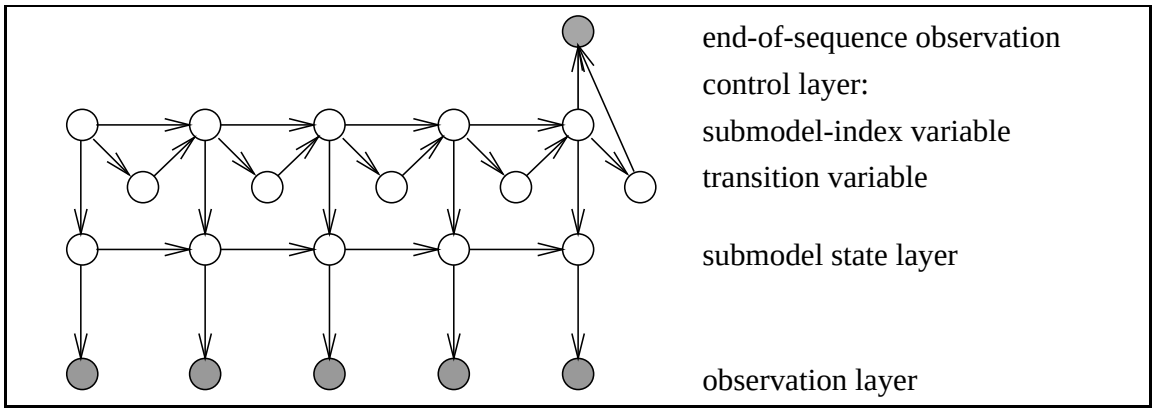

Figure 6.3: A DBN structured for model composition. The submodel-index variable specifies which submodel to use at each point in time.

This result can be stated somewhat more precisely as follows. Let $\mathbf{y}^t$ denote an assignment of values to the submodel observation and state variables at time t, and let m^t denote an assignment of a value to the submodel index variable at time t. The combination of a SFSA together with a set of DBN submodels specifies a probability distribution over sequences of $\mathbf{y}$ values: the probability of a particular sequence of $\mathbf{y}$ values is given by the weighted sum over all possible submodel sequences of the probability that each submodel sequence generates the given sequence of $\mathbf{y}$ values:

$$P(\mathbf{y}^1 \ldots \mathbf{y}^k) = \sum_{m_1 \ldots m_k} P(m^1 \ldots m^k) P(\mathbf{y}^1 \ldots \mathbf{y}^k | m^1 \ldots m^k)$$

By the Markov property of both the SFSA and the DBN submodels, we then have

$$P(\mathbf{y}^1 \ldots \mathbf{y}^k) = \sum_{m_1 \ldots m_k} P(m^1 \ldots m^k) P(\mathbf{y}^1 | m^1) P(\mathbf{y}^2 | \mathbf{y}^1, m^2) \cdots P(\mathbf{y}^k | \mathbf{y}^{k-1}, m^k)$$

The first factor in each term corresponds to the probability of a particular path through the SFSA and is determined by the CPTs of the control layer. The remaining factors correspond to the probability of the specified behavior of the submodel variables, conditioned on the submodel sequence. These factors are determined by the CPTs of the submodel state and observation layers.

Parameter Tying

In many situations, it is convenient to require that the transition behavior of two different states be the same, but for some reason the two states are *not identical*, and

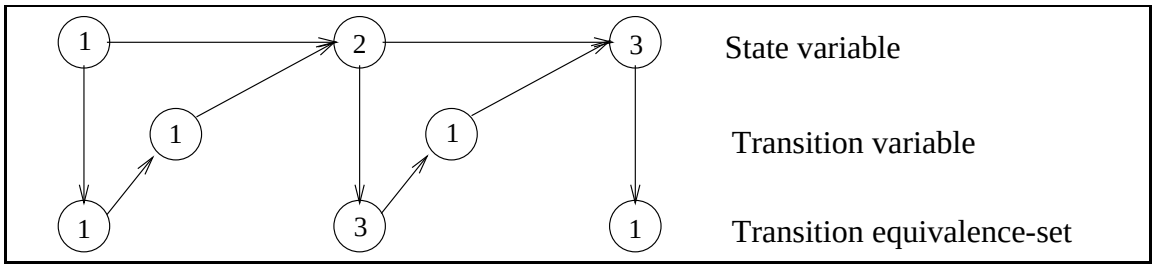

Figure 6.4: Mapping states into equivalence sets with respect to transition probabilities. The variables are labeled with one possible assignment of values. States 1 and 3 both map into the same transition equivalence set.

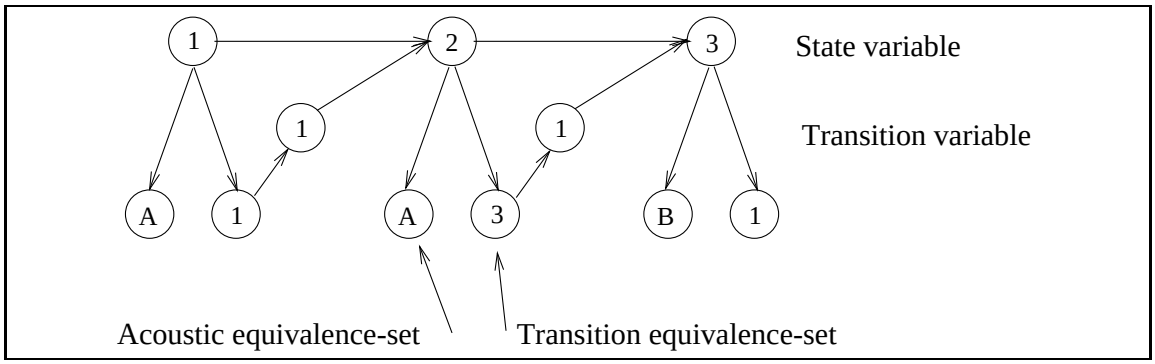

Figure 6.5: Mapping states into multiple equivalence classes. There is a transition equivalence class, and an acoustic one. The states behave differently with respect to the two.

must be distinguished. Essentially, we wish to group the states into equivalence sets based on their transition behavior, and tie the transition parameters of all the members of an equivalence set. It is straightforward to deal with this requirement in a DBN by adding another variable that represents the equivalence class to which a particular state belongs. The approach is illustrated in Figure 6.4. The value of the state variable maps into a particular equivalence set, and this is explicitly represented by the value of the equivalence set variable. The probability distribution over actual transition values is conditioned on the value of this equivalence set, rather than on the value of the state directly.

This approach can be extended to situations in which there are several important qualities associated with each state, and each state maps into a separate equivalence set with respect to each of these properties. For example, in speech recognition we wish to associate a durational distribution with a state, which is a function of its transition probabilities, and a distribution over acoustic emissions. A structure that allows for arbitrary parameter tying with respect to these two qualities is shown in Figure 6.5. In the remainder of our work,

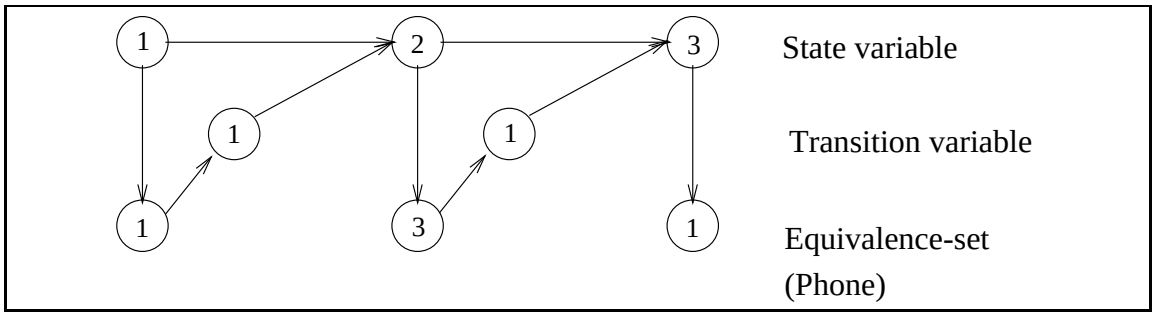

Figure 6.6: The control structure used in this work. A state maps into a phone label, and this value will determine both durational and acoustic properties.

however, we will assume that states belong to the same equivalence sets with respect to both acoustic and durational qualities. More specifically, the states correspond to phones, and we assume that all the occurrences of a particular phone behave the same with respect to both durational and acoustic qualities. This leads to the control structure used in subsequent experiments, which is shown in Figure 6.6.

This approach to parameter tying is significantly different from that used in HMMs. Parameter tying in an HMM system occurs somewhere in the implementation, in an imperative manner. In the DBN framework, it is achieved by manipulating the same representational units (variables and conditional probabilities) that are used to express every other concept.

Null States and Language Models

The SFSAs we have been dealing with have dummy initial and final states; the most straightforward way of concatenating SFSA models is to connect the final state of one to the initial states of its possible successors. This is useful in speech recognition when the SFSAs represent pronunciation models for individual words, and a multi-word utterance must be processed. A simple bigram language model results from connecting the final state of each word to the initial state of every other word, and setting the transition probability to the fraction of the time the second word follows the first.

Note that no observations are associated with SFSA initial and final states. When modeling concatenated SFSA models with DBNs, it is necessary to "skip over" the dummy states. This can be done with a DBN structured as in Figure 6.7. Dummy state skipping is

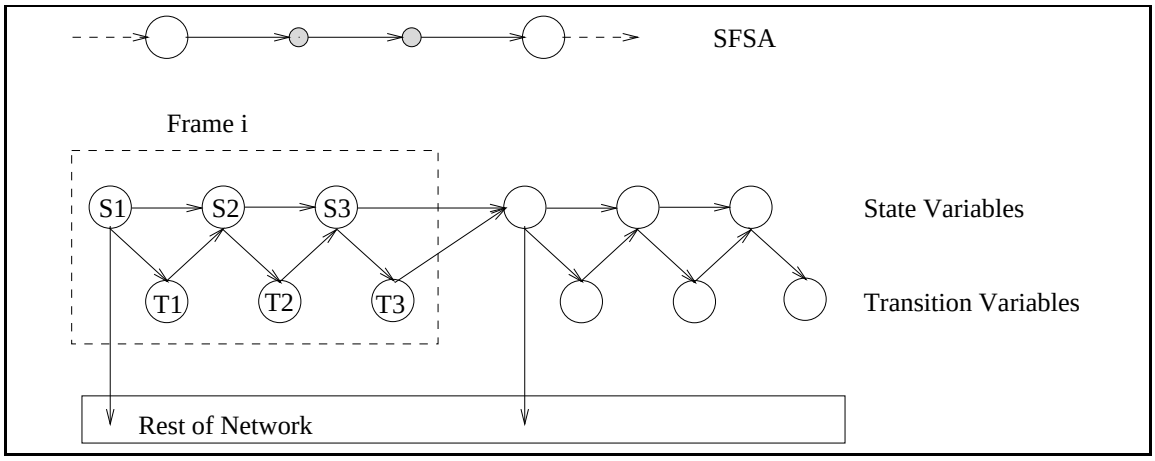

Figure 6.7: Modeling null states with a DBN. At the top is a portion of two concatenated SFSAs, showing the final state of one connected to the initial state of the next. At the bottom is a DBN with two auxiliary state and transition variables per timeslice. These allow the null states to be skipped. The state and transition variables from a single timeslice are boxed with the dashed line.

accomplished by associating three state and transition variables with each frame i. (Denote these by $S_i^1, S_i^2, S_i^3, T_i^1, T_i^2, T_i^3$.) Only the first state variable is linked to the observations. T_i^1 indicates the arc out of S_i^1, and the combination of these variables determines S_i^2 as before. If S_i^2 is a normal state, T_i^2 takes the arbitrary value 1, and S_i^3 copies S_i^2. Otherwise, S_i^2 is a null state, and T_i^2 assumes a value according to the distribution over arcs out of that state (which reflects the bigram probabilities), and S_i^3 is determined stochastically. The process repeats again with S_i^3 and T_i^3 to determine the value of S_{i+1}^1.

A more complicated SFSA structured to represent a trigram language model is shown in Figure 6.8. In this case, the DBN scheme must be extended to accommodate three dummy states in a row.

A Complete Speech Model

Figure 6.9 illustrates an example of a DBN that is structured for model composition in speech recognition in such a way as to be equivalent to a standard HMM. For clarity, we explicitly distinguish between CPTs that encode deterministic relationships and those which encode stochastic relationships. Table 6.1 summarizes the properties of each of the variables.

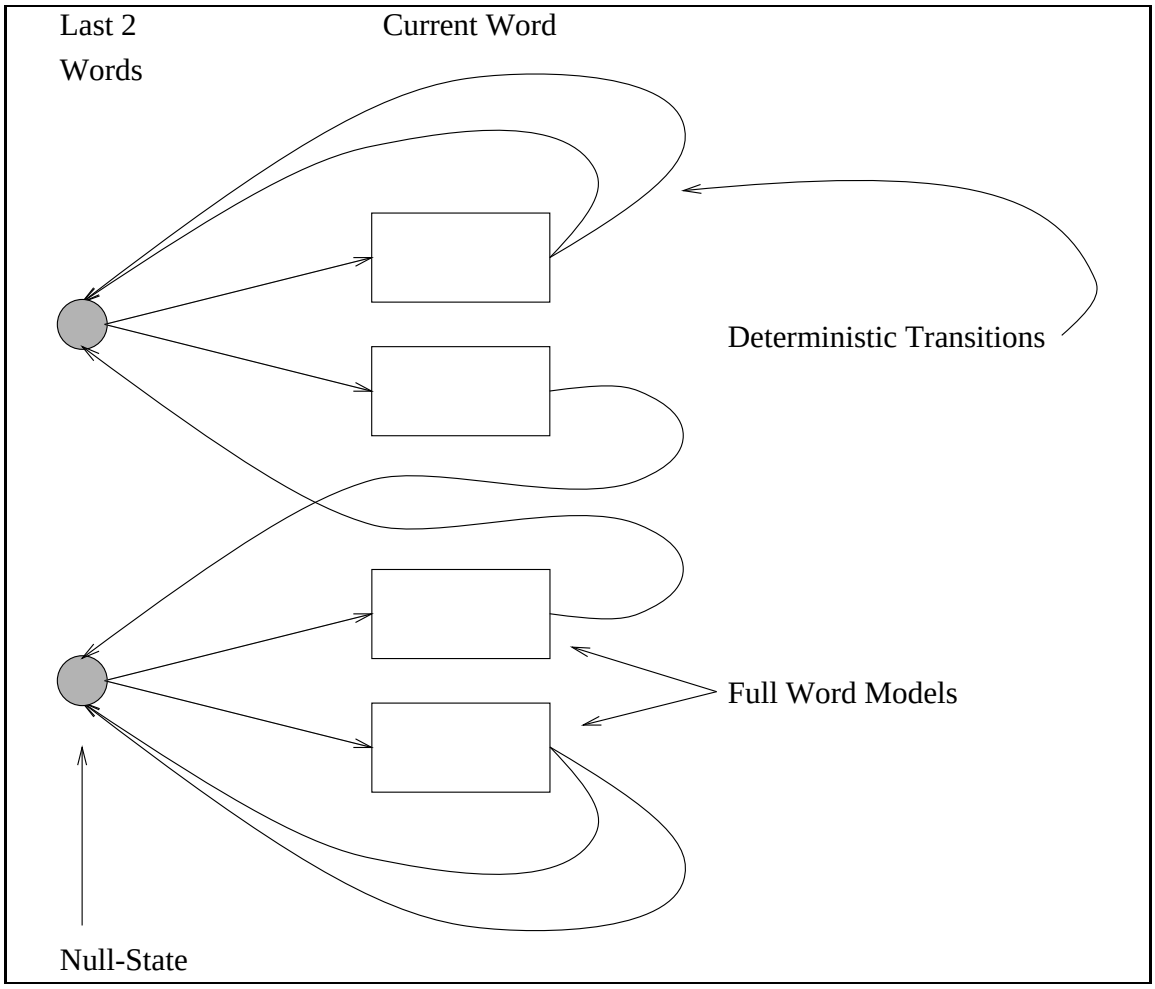

Figure 6.8: SFSA structure structured to reflect a trigram language model. The shaded circles represent dummy states; there is one for each pair of words. The rectangles represent whole word models (each with its own initial and final state). The total number of boxes is equal to the cube of the vocabulary size: there is a box for each word preceded by every possible two-word combination. Since the combination of the last two words with the current word uniquely determines the two-word context for the next word, the arcs leading out of the word models have transition probabilities of 1. The trigram probabilities are associated with the arcs from the dummy states into the word models. To avoid clutter, a only subset of the possible arcs are drawn.

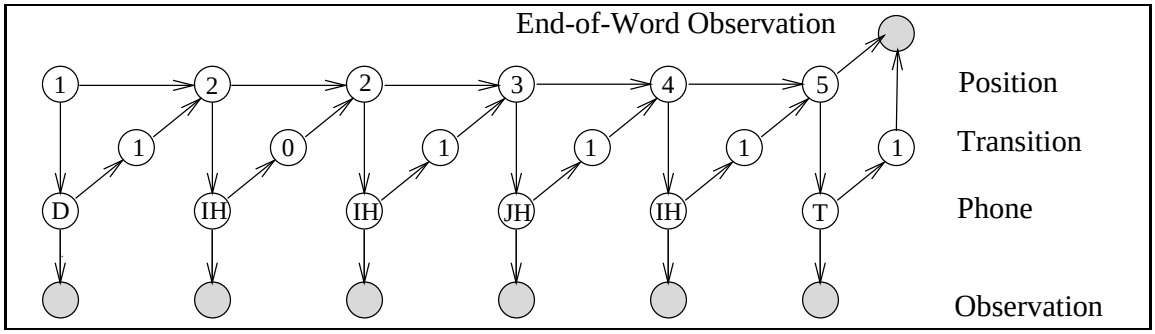

Figure 6.9: A DBN representation of a simple HMM. Nodes with fixed CPTs are fixed on a per-example basis.

Node Type	Deterministic CPTs	Example-Specific CPTs	Learned CPTs
Transition	N	N	Y
Position	Y	Y	N
Phoneme	Y	Y	N
Acoustic Obs.	N	N	Y

Table 6.1: The properties of the different variables. In this work, we use a chain-structured pronunciation model, so the value of the initial state is uniquely determined. This allows all occurrences of the index variable to be deterministic. The CPTs that are not learned are adjusted on an utterance-by-utterance basis.

6.1.3 Discussion: Write Networks not Code?

Essentially, what we have done in this chapter is to encode a dynamic programming algorithm into a network structure and its associated conditional probabilities. In the process of executing the standard procedures for probabilistic inference, the Bayesian network implicitly executes the desired program. This is extremely different from conventional approaches, where special purpose code is written for every occasion. It also sheds light on the importance of efficiently processing deterministic relationships between variables: since we are encoding a deterministic program, it is not surprising that deterministic variables play a central role.

Since a one-to-one correspondence could theoretically be made between the variables in a DBN and the circuits in a computer, there is apparently no limit on the kinds of behaviors that can be induced. Although expressive obscurity and computational inefficiency make it undesirable to exercise this capability, it can be extremely useful for limited tasks.

6.2 Model Structures for ASR

We now turn to the specific network structures required to address the problems mentioned in Chapter 5. Because DBNs can track arbitrary sets of variables, they are an ideal tool for creating precise models of the various phenomena.

6.2.1 Articulatory Modeling

Figure 6.10 illustrates a DBN structure that can explicitly model articulatory motion. It is the same as that in Figure 6.9, except that the connection from phone to observation is mediated by articulatory variables. The CPTs associated with the articulator variables describe both linguistic knowledge about the target positions of the articulators for the various phonetic units, and additionally the basic physics of the vocal apparatus; these CPTs explicitly model the way in which the constraints imposed by this physical model (e.g. inertia) modulate the target positions. The CPTs associated with the observation variables describe the sounds generated by particular physical configurations of the vocal apparatus. The precise topology and initial parameter estimates for these connections

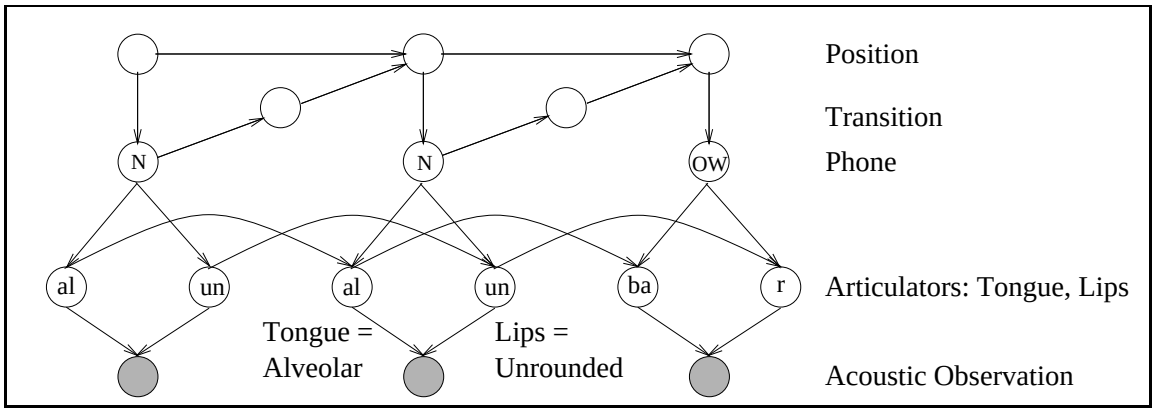

Figure 6.10: An articulatory DBN structured for speech recognition. The tongue moves from the alveolar ridge to the back of the mouth; the lips move from an unrounded to a rounded configuration. The properties of each node are shown to the right.

embody a phonological theory.

Enforcing Model Semantics

It is one thing to define a model that has the capability to track articulatory motion, and another to ensure that after "training" the variables will actually have the desired meaning. There are several ways that the correct model semantics can be encouraged:

1. Train with data in which articulatory positions are available from actual measurements. These sorts of measurements can be made using magnetic coils (Hogden *et al.* 1996), X-rays (Papcun *et al.* 1992), or radar (Holzrichter *et al.* 1996).

2. Initialize the network parameters to reflect prior linguistic knowledge. For example, Figure 6.11 relates the position of the tongue to the different vowel sounds.

3. Use Dirichlet priors to encode prior linguistic knowledge. This is discussed more fully below.

Ideally, articulatory models should be trained with known articulator positions; this is the only way of guaranteeing that the trained model will accurately reflect the articulators. Note that even if articulatory data is only available during training, and not during testing, a benefit can still be expected, because of more accurate parameter estimation. In

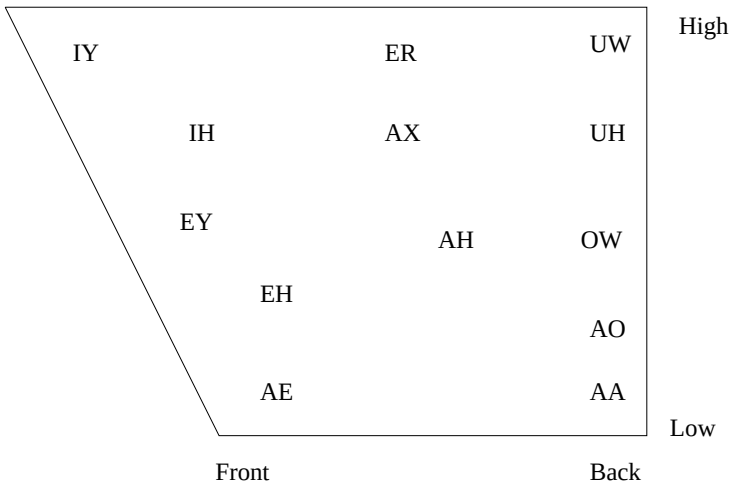

Figure 6.11: Tongue position for different vowels, adapted from Deller et al., 1993.

practice, the standard speech recognition databases do not have this information, and the use of Dirichlet priors (Heckerman 1995) is probably the next best approach.

Recall from Section 3.8.2 that parameter estimation is done is by counting the number of times an event of interest occurs, and estimating the count if necessary. For example, the conditional probability of a voicing variable having the value 1 (true) given that the speaker is in the state of pronouncing /ER/ would be estimated by counting the number of speech frames in which both assertions are true, and dividing by the total number of frames labeled /ER/. The concept of Dirichlet priors is simply to augment the actual counts with fictitious counts. So, for example, to reflect the prior knowledge that /ER/ is voiced, the tally of speech frames that are simultaneously labeled /ER/ and "voiced" might be initialized to 10,000 rather than 0. The magnitude of the fictitious counts (in relation to the actual counts) determines the confidence with which the prior information is expressed.

Articulatory HMMs: A Comparison

There has been significant previous work incorporating articulatory models into HMMs (Deng & Erler 1992; Erler & Deng 1993; Deng & Sun 1994; Erler & Freeman 1996; Deng 1996), and a comparison with the DBN approach highlights many of the advantages of DBNs. Starting in the early 1990s, Deng and Erler have explored HMM extensions that explicitly model articulator motion. The basic approach is simple to explain. First,

a deterministic mapping between phonetic units and articulator positions is established. Typically, the positions of five articulators are used. Each articulator is assumed to be in one of a discrete number of positions. The overall state of the system is thus defined by the cross-product of values assigned to the articulators. An HMM-state space is defined in which there is a distinct state for each possible articulatory configuration.

When a training word is presented (or a recognition word-hypothesis evaluated) the phonetic transcription of the word is mapped into a sequence of articulatory targets, one for each phoneme in the transcription. This defines a set of legal paths through the HMM grid. Then a series of phonological rules is applied to expand the set of legal paths through the grid. This expansion can express coarticulatory effects by modifying the expected target positions in a context-dependent way. Once the final set of legal paths is identified, training or recognition can proceed with standard techniques.

The DBN approach differs in the following important ways:

1. It is a particular instantiation of a general-purpose tool. Hence it is easy to modify to address other phenomena.

2. There is a stochastic - not deterministic - mapping between phonetic units and articulator targets.

3. This mapping can be learned, and need not be hand-coded.

4. The rules governing acceptable articulator motion are stochastic rather than deterministic.

5. The conditional probabilities governing articulator motion can be learned.

6. Prior knowledge is expressed with statistical priors, rather than rules.

7. The system represents a uniform application of statistical pattern recognition, rather than a combination of hand-coded rules with probabilistic inference.

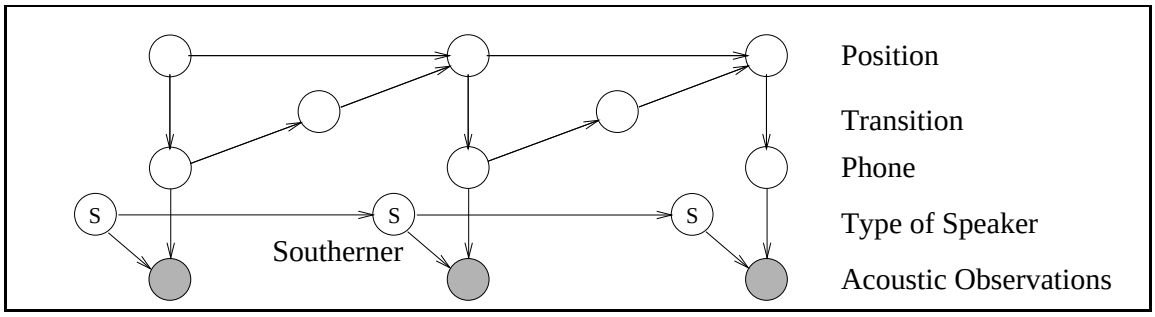

Figure 6.12: A DBN structured to model speaker-type.

6.2.2 Modeling Speaking Style

Speaker-Type

Figure 6.12 illustrates the way in which a DBN can be structured to model speaker characteristics. It is the same as the basic HMM-DBN, except that there is now an auxiliary variable representing speaker-type in each timeslice. This variable might represent, for example, whether the speaker is male or female, or the speaker's accent. Since there are more than one important characteristics, it may be beneficial to use more than one auxiliary variable.

The characteristics of a speaker do not change over time, and this fact can be encoded in the auxiliary variable by making the first occurrence of this variable stochastic, and all subsequent occurrences deterministic. The variable in the first timeslice encodes a prior over types of speaker, and the later occurrences simply "copy" the value. This is analogous to having multiple HMMs and choosing between them according to some prior, except that there is only one DBN.

A network such as that of Figure 6.12 can be trained in either a supervised or unsupervised manner. Supervised training consists of training the network using utterances labeled with the speaker's type; in this case, the auxiliary variable is an observation variable for the purposes of training, and hidden during testing. Supervised training ensures that the auxiliary variable has a well-defined meaning. In unsupervised training, the auxiliary variable is hidden during both training and testing; in this case, the network learns to group utterances together into clusters automatically. The utterances in a particular cluster will tend to have commonalities that are based on more than one concept, for example a mixture

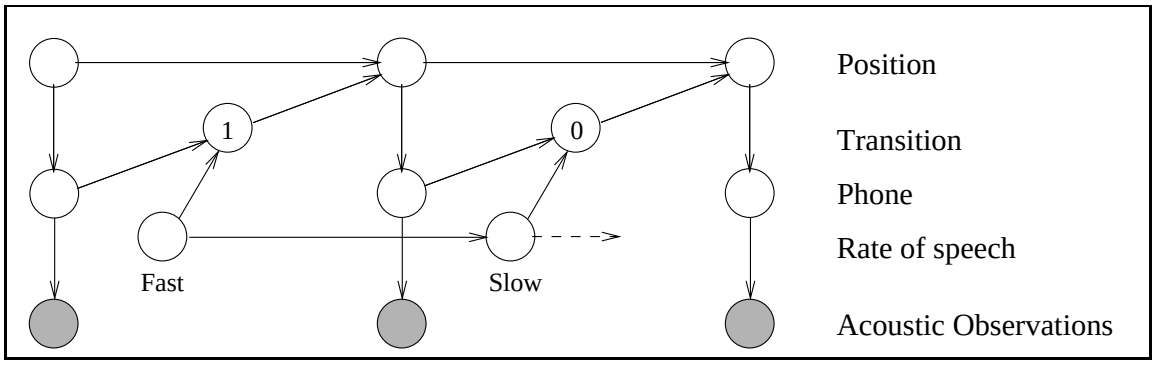

Figure 6.13: A DBN structured to model speaking-rate.

of gender and accent. Results for unsupervised training are presented in section 7.8.

Speaking-Rate

Figure 6.13 illustrates a simple way in which speaking rate can be modeled with a DBN. The auxiliary variable in this case represents the speaker's speaking rate, and the transition variables are conditioned on it. The intention is that when the speaker is talking quickly, transitions will be more likely. In recent work, (Morgan & Fosler-Lussier 1998), reliable procedures for estimating speaking rate directly from acoustic observations have been developed. If these measures are available, they can be incorporated into the network as shown in Figure 6.14.

It is also known that rate-of-speech has a more complex effect than simply changing transition probabilities. In (Fosler-Lussier & Morgan 1998), it is shown that, for a given word, the expected sequence of phonemes changes with speaking rate. Moreover, (Mirghafori *et al.* 1995; Siegler & Stern 1995) shows that the expected acoustics of a given phoneme vary with speaking rate. The first of these effects can be handled by representing word pronunciations with a more elaborate SFSA in which the possible insertions, deletions, and substitutions are explicitly represented. By conditioning the transition variable on the rate-of-speech estimator, the probability of these modifications can be appropriately adjusted. The second of these effects can be addressed by conditioning the observations on the rate variable.

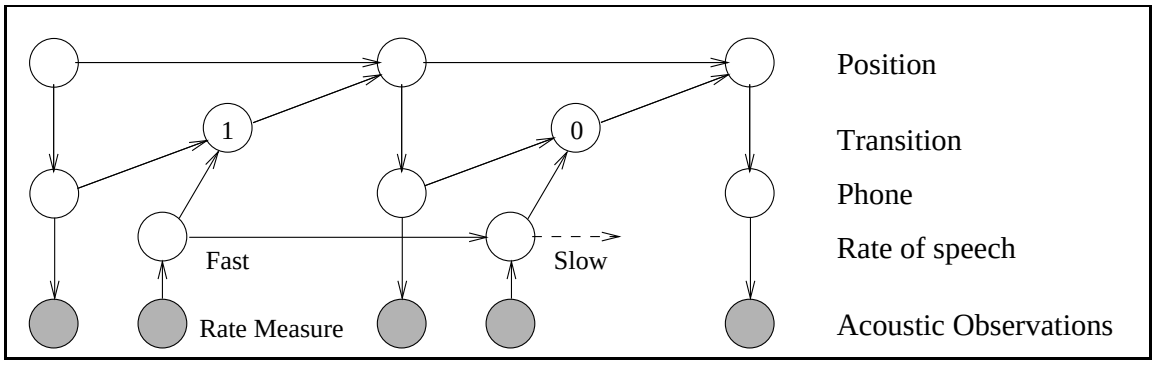

Figure 6.14: A DBN structured to model speaking-rate, with observations that are highly correlated with rate.

6.2.3 Noise Modeling

Explicit noise models can be constructed with DBNs, in a manner similar to that presented in (Varga & Moore 1990; Gales & Young 1992) in the context of HMMS. This is illustrated in Figure 6.15. In its original formulation, scheme consists of three basic parts:

1. An HMM to model speech.
2. An HMM to model noise.
3. A model of how speech and noise sounds combine into the sound that is actually heard.

In (Varga & Moore 1990; Gales & Young 1992), the two HMM models are trained separately on examples of pure speech and pure noise, and the model for sound combination is analytical. The DBN model shown in Figure 6.15 can be used in the same way, or it can be trained on a single noisy observation stream.

6.2.4 Perceptual and Combined Models

As a final example of the versatility of the DBN approach, Figure 6.16 shows a model that combines generative and perceptual aspects. The distinguishing feature of this model is that it maintains a representation of both the speaker's intention and the listener's perception, and encourages them to coincide. For clarity, the index, transition, and phone variables for both the speaker and the listener are combined into a single state variable

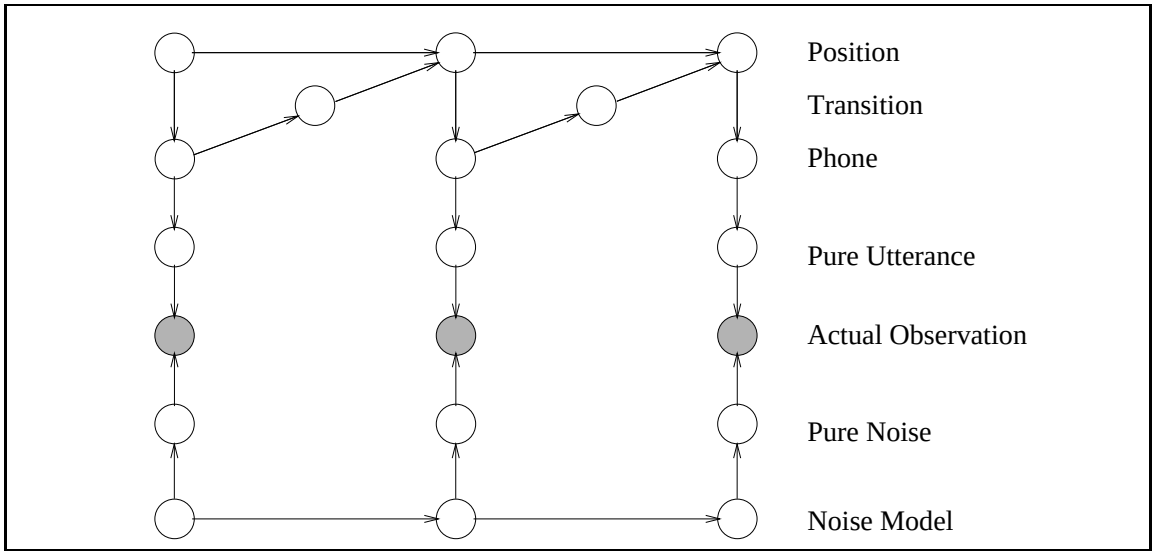

Figure 6.15: A DBN structured to model speech in a noisy environment.

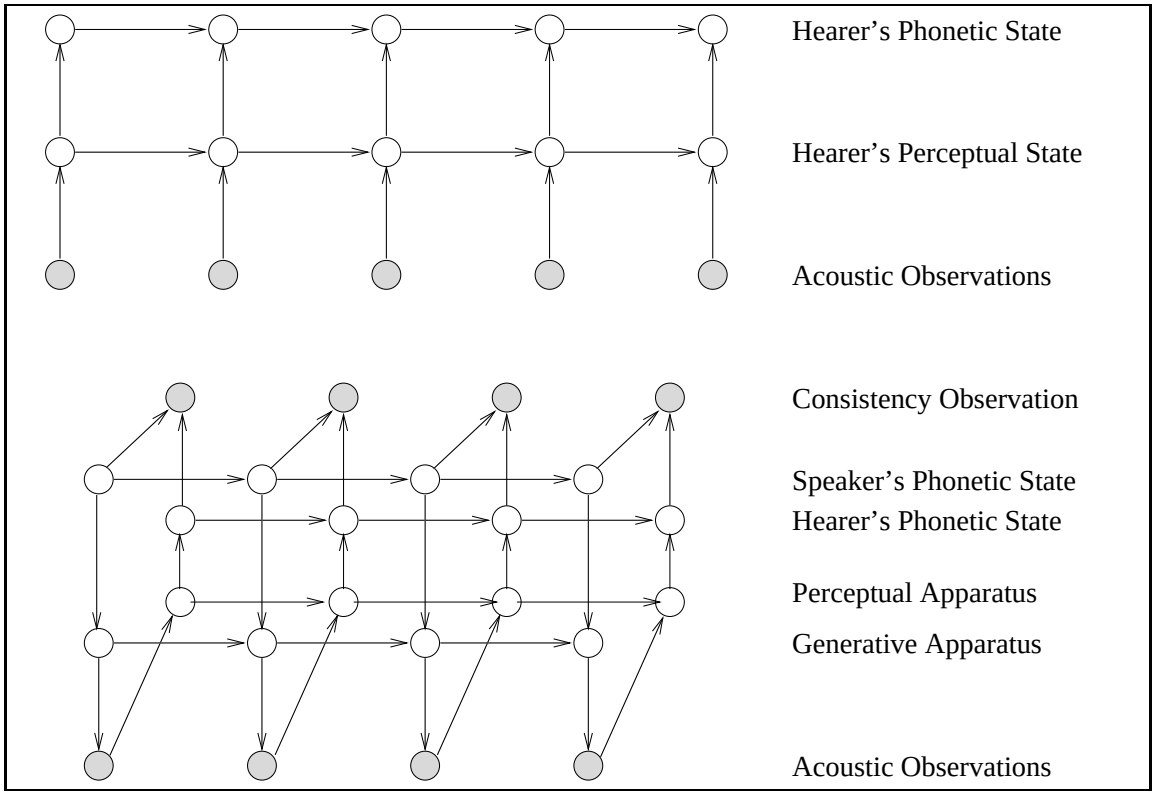

Figure 6.16: A perceptually-structured DBN (top), and a combined perceptual-generative model. For clarity, the index, transition, and phone variables are simply represented by a "phonetic state" variable.

for each person. The generative part of the model consists of the speaker's phonetic state affecting his articulators and causing sound production. The perceptual part of the model consists of the sound affecting the listener's perceptual apparatus and causing a sequence of phones to be recognized. The speaker's intention and the listener's perception are linked through a consistency observation which has a constant value and is used to encode the fact that intention and perception ought to be identical. This can be done in a rigid way by setting the conditional probability of the observed consistency value given its two inputs to 0 in the case of inconsistency. A more flexible model of mistakes can also be encoded by using a less extreme probability distribution for readily confusable phones. Finally, it is worth noting that a consistency model can also be learned, rather than hard-coded.

6.3 Discussion

In the preceding sections, we have seen that DBNs can be adapted to address the requirements of automatic speech recognition, and that they can model many of the important factors affecting the speech recognition process. We conclude with a brief summary of the advantages:

- Arbitrary sets of variables can be associated with each timeslice. This enables a highly expressive representational framework.
- There are efficient, general-purpose algorithms for doing inference, and no special-purpose algorithms need be derived for handling extensions to HMMs such as articulator models.
- Sharing variables between submodels leads to a natural way of describing transitional behavior, which is important for modeling coarticulation.
- Statistical efficiency. DBNs are factored representations of a probability distribution, and may have exponentially fewer parameters than unfactored representations such as standard HMMs. Hence these parameters can be estimated more accurately with a fixed amount of data (Zweig 1996).
- Computational efficiency. Gains in statistical efficiency are often mirrored computationally.

Chapter 7

Speech Recognition Experiments

This chapter presents experimental results for a fully implemented speech recognition system based on DBNs. Early on, computational limitations forced a choice between experimenting on a fairly small database of digits or alphabet-letters, with the ability to test relatively complex network structures, or experimenting on a more challenging database with simpler network structures. In order to get more meaningful results, we chose the latter, and selected a large-vocabulary multi-speaker database of isolated words to use as a testbed. The database is challenging enough that results are significant (unlike databases of digits or a few command words), yet because it has isolated words, it avoids many issues that complicate any continuous-speech ASR system. In short, the database is just complex enough to test some basic issues relating to the use of factored state representations in ASR. Some of the results presented in this chapter appeared in (Zweig & Russell 1998).

7.1 Database

This chapter presents results for the Phonebook database (Pitrelli *et al.* 1995). Despite its name, the database does not contain entries from a phonebook; instead, it consists of a collection of words chosen to exhibit all the coarticulatory effects found in the English language. Researchers at NYNEX compiled the list of words, and then contracted with an outside organization to obtain actual utterances. These were collected over the telephone, and thus contain a variety of transmission distortions. The database is divided into subsets, and the words in one of these subsets are reproduced in Table 7.1.

achieved	apex	arsenic	ashtray	ashwell
awe	barleycorn	beeswax	belgium	biff
bloodletting	boyish	breathes	broadview	cashways
chadwick	cheesecloth	colorfast	compromise	confession
cowling	craigs	disrespectful	echoes	egghead
exhaustion	festival	formalization	grisly	handlooms
haymarket	highman	humdinger	humphreys	immobilizing
impolite	indictments	inscribe	instincts	ivy
lavender	lawmakers	majorities	masks	mckane
mutually	mysteriously	nonstandard	noose	nothingness
overambitious	penguins	perm	plowing	porch
postmark	rustle	salesmen	sluggishness	soggy
sorceress	spokeswoman	staying	subgroups	sulfuric
swordcraft	theory	undeterred	unleashes	unnatural
vulgarity	watchful	windowless	windshield	youngman

Table 7.1: Typical words in the Phonebook database.

Word utterances were collected from a group of American speakers who were "balanced for gender, and demographically representative of geographic location, ..., income, age (over 18 years), education, and socio-economic status." (Pitrelli *et al.* 1995) Each speaker was asked to read 75 or 76 words, and the utterances were then screened for acceptable pronunciation; therefore, there are somewhat fewer than 75 words per speaker on average.

7.2 Acoustic Processing

The utterances were processed with relatively conventional acoustic processing along the lines presented in Section 5.1.1. Initial and final silence was removed using the endpoints provided with the database. Then the utterances were divided into 25ms windows and MFCCs were calculated. The analysis windows overlapped by 2/3.

Smoothed MFCC derivatives (Rabiner & Juang 1993) were computed, and three data streams were created: one for the MFCCs, one for the derivatives, and one for the combination of C_0 and delta-C_0 (which were omitted from the first two streams). Mean cepstral-subtraction (Mammone *et al.* 1996) was performed for cepstral coefficients $C_1 - C_{10}$, and speaker normalization (Lee 1989) was done for C_0. The first process removes the effects of telephone transmission characteristics, and the second subtracts the maximum C_0 value

in an utterance from each frame, in order to make the values comparable across speakers.

The data streams were vector-quantized to eight bits (256 values). The MFCCs and delta-MFCCs were quantized in separate codebooks. C_0 and delta-C_0 were quantized to four bits each, and then concatenated to form a single eight-bit stream.

7.3 Phonetic Alphabets

Results are presented for DBN models using both context-independent and context-dependent phonetic units. The Phonebook database provides phoneme-level transcriptions, and these formed the basis of both kinds of alphabet. To keep the number of parameters reasonable, and in common with other work (Dupont *et al.* 1997), we did not use the 3-way stress distinction for vowels. Additionally, occurrences of /N/, were replaced by /n/; /N/ occurs only 24 times in the database, and is not on the official list of phonemes (Pitrelli *et al.* 1995). The phoneme /L/ is on the official list, but never occurs in the data. The size of the basic phoneme alphabet was thus 41. Two additional phonetic units were used to represent initial and final silence.

7.3.1 Context Independent Alphabet

In the case of context-independent units, i.e. simple phonemes, each phoneme was replaced by a k-state left-to-right phone model. Most of the experiments report results for 4-state phone models; these have an initial and final state, and two interior states. Experimentation shows this to be a good number. In all cases, one-state models were used to represent silence.

7.3.2 Context Dependent Alphabet

To create a context-dependent alphabet, we used a variation on diphones (Schwartz *et al.* 1980). The basic idea is to create two new units for each phoneme in a transcription: one for the initial part of the phoneme in the left-context of the preceding phoneme, and one for the final part of the phoneme in the right-context of the following phoneme. Thus, for example, $/k\ ae\ t/$ becomes

$$(sil\ \mathbf{k})(\mathbf{k}\ ae)(k\ \mathbf{ae})(\mathbf{ae}\ t)(ae\ \mathbf{t})(\mathbf{t}\ sil).$$

This scheme has the advantage of addressing both left and right contexts, like triphones, while only squaring - rather than cubing - the number of potential phonetic units.

To prevent overtraining, a context-dependent unit was used only if it occurred a threshold number of times in the training data. Units that did not meet this criterion were replaced with context-independent units. We used thresholds of 250 and 125, which resulted in alphabets with sizes of 336 and 666 respectively, including a full set of context-independent units.

We found it beneficial to double the occurrence of each of the units in a context dependent transcription. Thus, the total number of phones is four times the original number of phonemes, the same number that results from four-state context-independent phoneme models. Repeating phonetic units has the effect of changing the minimum and expected state durations.

It is important to realize that context as expressed in a context-dependent alphabet is significantly different from that represented by a hidden context variable in a DBN. Context of the kind expressed in a context-dependent alphabet is based on an idealized and invariant pronunciation template; a word model based on context-dependent phones can be written down before ever seeing a sound wave, and therefore represents a-priori knowledge. The context-variable represents context as manifested in a specific utterance.

7.4 Experimental Procedure

7.4.1 Training, Tuning, and Testing

The database was divided into separate portions for training, tuning, and testing. All decisions concerning network structures and alphabets were made by training a system on the training data, and testing it on the tuning data. Decisions were not made on the basis of performance on the actual test data.

The training data consisted of the utterances found in the *a, *h, *m, *q, and *t subdirectories of the Phonebook distribution; the tuning utterances were from the *o and *y directories, and the test utterances from the *d and *r directories. This is the same partitioning used in (Dupont *et al.* 1997). This resulted in 19,421 training utterances, 7,291 tuning utterances and 6,598 test utterances, with no overlap between the speakers or words

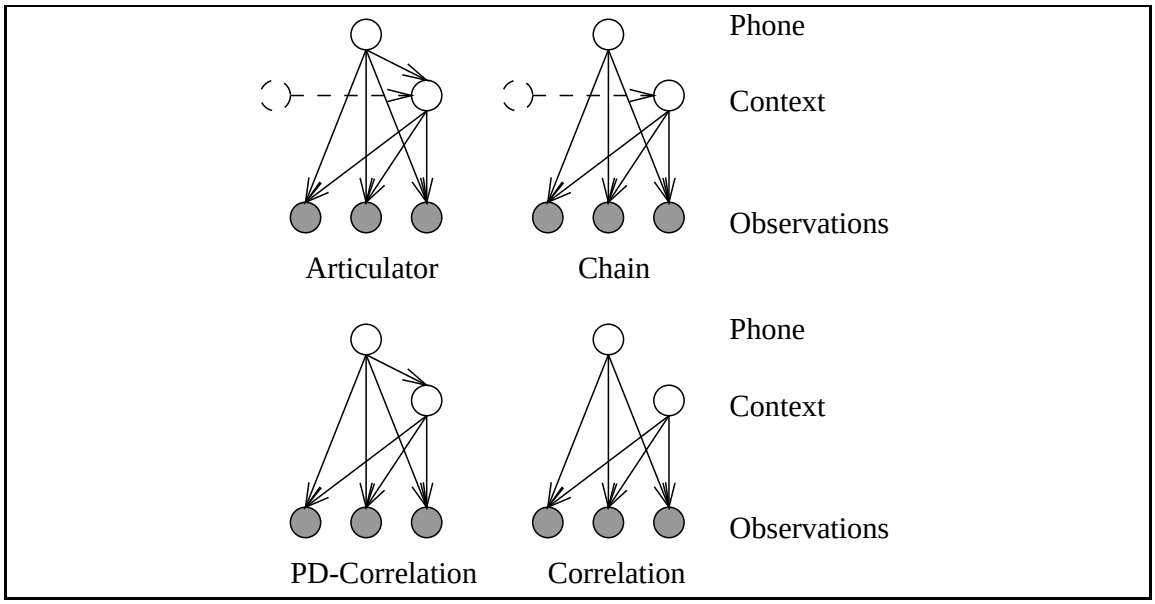

Figure 7.1: The acoustic models for four of the network topologies tested. The index and transition variables are omitted. The dotted lines indicate conditioning on the previous frame.

in any of the partitions.

The words in the Phonebook vocabulary are divided into 75 and 76-word groups, and the recognition task consists of identifying each test word from among the other words in its subset. Hence, random guessing would result in a recognition rate of under 2%.

7.4.2 Models Tested

A baseline DBN was constructed to emulate an unfactored HMM. DBNs with one or more auxiliary state variables were then designed to answer the following questions:

1. What is the effect of modeling correlations between observations within a single timeslice? Specifically,

   (a) What is the effect of modeling these correlations in a phone-independent way? This question was addressed with the "Correlation" network of Figure 7.1. This network is only capable of modeling intra-frame observation correlations in the most basic way.

   (b) What is the effect of modeling these correlations in a phone-dependent way? This

question was addressed with the phone-dependent "PD-Correlation" network of Figure 7.1. This network can directly model phone-dependent intra-frame correlations among the acoustic features.

2. What is the effect of modeling temporal continuity? Specifically,

   (a) What is the effect of modeling temporal continuity in the auxiliary chain in a phone-independent way? This question was addressed with the "Chain" network of Figure 7.1. This network results from the addition of temporal links to the context variable of the correlation network, and can directly represent phone-independent temporal correlations. The network was initialized to reflect continuity in the value of the context variable.

   (b) What is the effect of modeling temporal continuity in the auxiliary chain in a phone-dependent way? This was addressed with the "articulator" network of Figure 7.1. In this network, the context variable depends on both the phonetic state and its own past value. This can directly represent phone-dependent articulatory target positions and inertial constraints. The network was initialized to reflect voicing.

3. How does the use of a context-dependent alphabet compare to context-modeling with an auxiliary variable? This was addressed by using a context-dependent alphabet in a network with no auxiliary variable. Additionally, we tested the combination of a context-dependent alphabet with a context variable.

4. What is the effect of increasing the number of values in the auxiliary chain, and how does increasing this number compare to increasing the number of context variables? This question was answered by making the proposed changes to the chain-network.

5. What is the effect of using an unfactored state representation to represent the same process? To answer this question, a DBN was used to emulate an HMM with an unfactored representation.

6. What is the effect of doing unsupervised clustering? This question was answered by testing the network shown in Figure 6.12.

Network	Parameters	Error Rate
Baseline-HMM	127k	4.8%
Correlation	254k	3.7%
PD-Correlation	254k	4.2%
Chain	254k	3.6%
Articulator	255k	3.4%

Table 7.2: Test set word error rate for systems using the basic phoneme alphabet. All the systems had slightly different numbers of parameters. The standard error is approximately 0.25%. Results from Zweig & Russell, 1998.

7.5 Results with a Single Auxiliary Variable

Answers to the first three questions are presented in this section. Table 7.2 shows the word-error rates with the basic phoneme alphabet, for the network structures shown in Figure 7.1. The results for the DBNs with a context variable are consistently better than without a context variable. A large improvement results simply from modeling within-frame correlations, but for both the Correlation and the PD-Correlation networks, a further improvement results from the addition of temporal-continuity links. The Articulator network provided the best performance.

In terms of absolute error-rates, these results compare favorably with those reported in (Dupont *et al.* 1997). That paper reports an error rate of 4.1% for an ANN-HMM hybrid using the Phonebook transcriptions, and the same training and test sets. Worse results are reported for a conventional Gaussian-mixture HMM system. However, with phonetic transcriptions based on the CMU dictionary, (Dupont *et al.* 1997) achieved significantly improved results. Comparison with this work provides a useful check on the overall recognition rate, but it should be remembered that a vector-quantized system is being compared with a continuous-observation system. Thus differences are difficult to interpret.

7.5.1 Context Dependent Alphabet

Error rates with the context-dependent alphabets are reported in Table 7.3. These results improve significantly on the context-independent results, and (as expected) confirm the benefits of using a context-dependent alphabet. As discussed previously, context as represented in an alphabet is different from context as represented in a DBN with a context

Network	Parameters	Error Rate
CDA-HMM	257k	3.2%
CDA-Articulator	515k	2.7%
CDA-HMM	510k	3.1%

Table 7.3: Test set word error rates for systems using context dependent alphabets. The first two results use an alphabet with 336 units, and the last result uses an alphabet with 666 units. The standard error is approximately 0.20%. Results from Zweig & Russell, 1998.

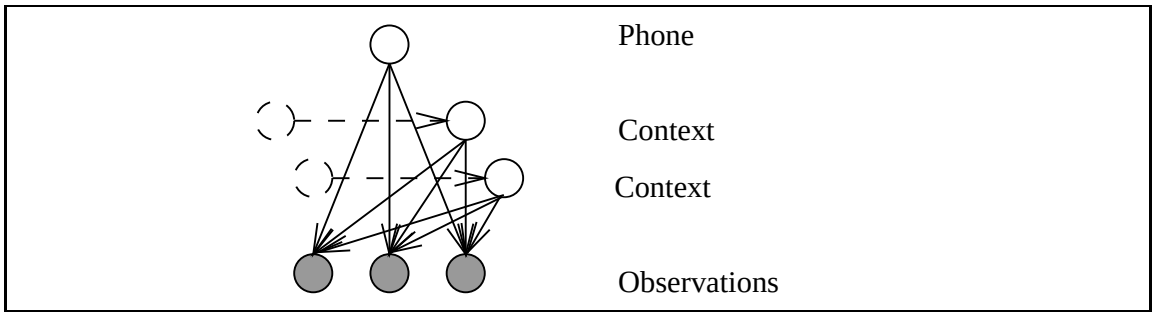

Figure 7.2: Network with two context variables.

variable. Therefore it makes sense to combine the two strategies, and this in fact produced the best overall results. Adding an extra context variable doubled the number of parameters, but as the last line in Table 7.3 indicates, doubling this number by using a bigger alphabet is not as effective.

7.6 Results With Two Auxiliary Variables

In order to evaluate the use of multiple context chains, the network shown in Table 7.2 was tested. Both of the context variables were binary. For comparison, a network with a single context variable with 3 and 4 values was tested. To cut down on memory usage and running time, and to save on the amount of disk-space needed to store conditional probabilities, 3-state phones were used in this set of experiments. The results are shown in Table 7.4.

These results indicate that increasing the amount of context state improves recognition performance. In addition, factoring the context state is beneficial. The results are somewhat worse than with the four-state phone models, which indicates that the combination of greater precision and longer minimum durations afforded by the four-state models

Network	Parameters	Error Rate
Binary-Chain	191k	4.1%
Trinary-Chain	287k	4.0%
Quaternary-Chain	383k	3.8%
Double-Chain	383k	3.6%

Table 7.4: Test results with multi-valued and multi-chain context variables; the standard error is approximately 0.25%. The double-chain network used binary variables, and thus had a total of 4 possible context values.

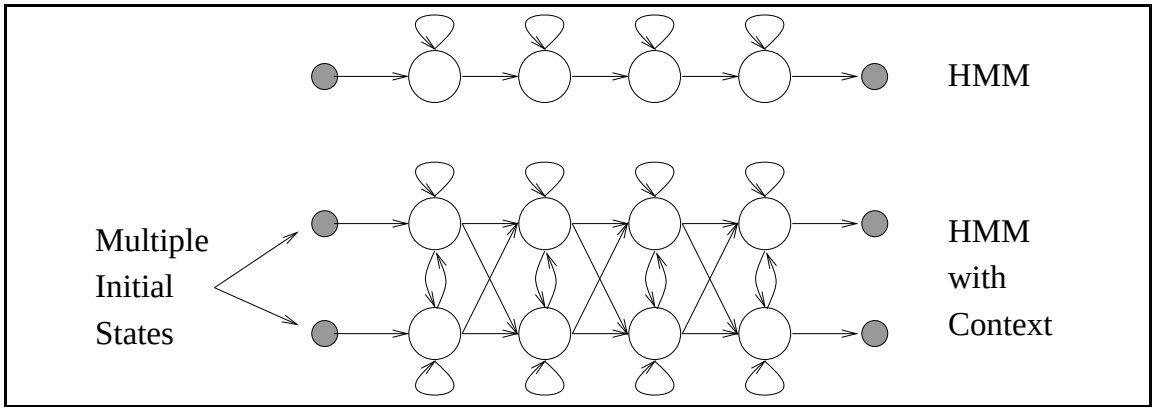

Figure 7.3: Top: a four-state HMM phone model. Bottom: the same model with a binary context distinction. There are now two states for each of the previous states, corresponding to the different combinations of phonetic and contextual state.

is important.

7.7 Cross-Product HMM

In the conventional HMM framework, the effect of a context variable can by simulated by using a more complex finite state automaton. Figure 7.3 illustrates this strategy for a binary context distinction. The idea is to create a distinct state for every possible combination of state and context. As shown at the bottom of Figure 7.3, the transition structure must be made significantly more complex, and the number of transition parameters is increased much more than in a DBN with a context variable. For example, the number of independent context and transition parameters in the articulatory DBN is three times the number of phones. In a cross-product HMM, it is six times the number of phones. This difference increases rapidly as the number of context variables and values increases. A

States per Phone	Context Values	Parameters	Initialization	Error Rate
4	2	255	Continuity	3.5
4	2	255	Voicing	3.2
3	4	386	Continuity	3.3

Table 7.5: Results for cross-product HMMs. Due to computational limitations, three states per phone were used in combination with the four-valued context distinction.

second fact to keep in mind is that the cross-product representation requires multiple initial and final states — one for each context value. This is because the context value must be retained across phonetic boundaries when the individual phone models are concatenated to form word models. Standard HMM packages, e.g. (Young *et al.* 1997), do not have this ability.

Results for a cross-product HMM with two different kinds of initialization are presented in Table 7.5. (More precisely, the results were generated with a DBN structured to be equivalent to a cross-product HMM.) The first kind of initialization was similar to that used in the Chain-DBN, and reflected continuity in the context variable value. The second kind reflected voicing, and is analogous to that used in the Articulator-DBN. For comparison with the double-chain network, we also tested a cross-product HMM with four possible context values; this network is an unfactored representation of the double-chain structure shown in Figure 7.2. The results for the cross-product HMM are actually slightly better than for the unfactored representation; the EM training procedure was able to make effective use of the extra parameters.

These results suggest that small amounts of acoustic and articulatory context can be modeled effectively with a cross-product HMM. From an engineering standpoint, this is an attractive, since it requires comparatively minor changes to the phone-models of existing HMM systems. Nevertheless, there are significant drawbacks to this approach; the most important of these is that it does not scale well: the number of transition parameters grows like the square of the number of context values. It is also inflexible, and would be difficult to modify to address different sets of variables and different conditional independence assumptions.

7.8 Clustering Results

This section presents results for a network doing unsupervised clustering. The network structure is presented in Figure 6.12. A binary-valued context variable was used, with the restriction that its value not change over the course of an utterance. This was enforced by using a stochastic context variable in the first timeslice, and then using a deterministic context variable from the second timeslice on to copy the value determined in the first frame. The network was trained as usual, and then during testing the likeliest value for the cluster variable was determined.

There are at least two dimensions along which one might expect clustering to occur: the type of speaker (e.g. male vs. female, adult vs. child), and the type of word (e.g. consonant-initial vs. vowel initial). [1] The degree to which such clustering occurs can be measured by looking at the degree to which utterances with a particular characteristic are classified together in a single cluster. Figures 7.4 and 7.5 show that both speaker and word clustering are observed. Since there are more cross-validation utterances than test utterances, the histograms are based on that subset (no tuning was involved).

Figure 7.4 shows the consistency with which utterances from a single speaker were classified together, and what would be expected at random. Clearly, utterances from individual speakers are being grouped together with high frequency.

Figure 7.5 shows the same information for particular words. The fact that the occurrences of a single word tend to be clustered together indicates that word characteristics, as well as speaker characteristics, are being modeled by the auxiliary variable.

In terms of overall error-rate, the clustering technique did not do as well as the other augmented networks; the test-set error rate of 4.5% was midway between the 4.8% rate of the baseline network and the 3.6% score for the chain network. This is expected, since the cluster-network has more state, and therefore modeling power, than the baseline network, but not as much expressiveness as the chain-network, where the context variable can "flip-flop" between values.

It is possible to make sense of the value assigned to the cluster variable, both in terms of speaker-type and word-type. The mutual information between the cluster variable and the gender of the speaker is 0.24 bits, indicating a strong correlation between cluster and

[1] Thanks to Jeff Bilmes for pointing out the duality between speakers and words.

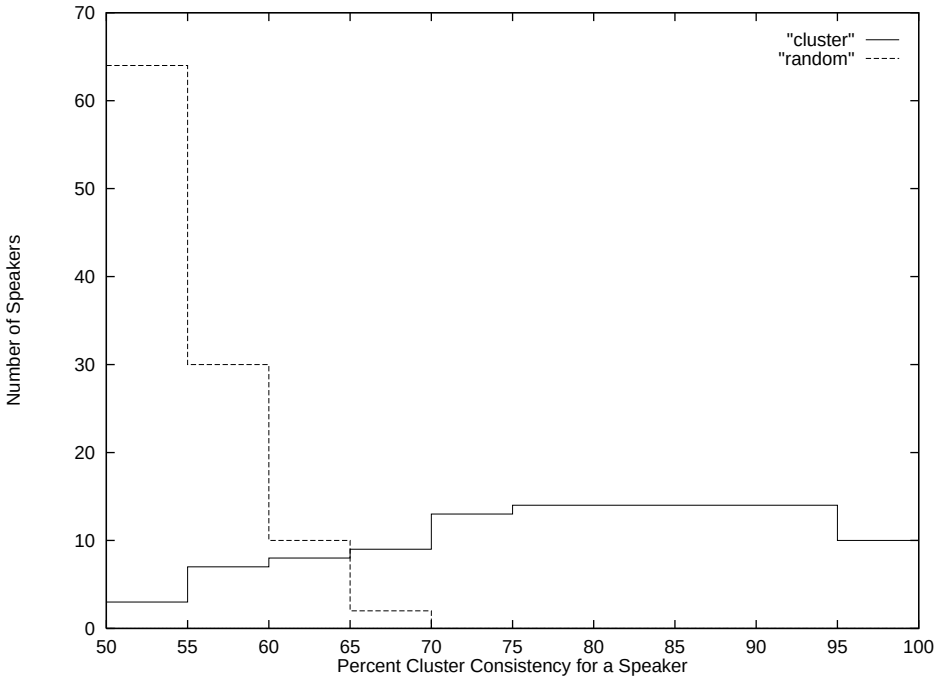

Figure 7.4: The frequency with which utterances from a single speaker were assigned to the same cluster. For example, about 15 speakers has their utterances clustered together with 85% consistency. On average, there are 68 utterances per speaker.

gender. To determine the word-characteristics associated with the two clusters, we examined the words that were very consistently assigned to a particular cluster. These are shown in Figure 7.6. One of the clusters is characterized by words beginning in liquid consonants, while the other is characterized by words ending in liquid consonants. The cluster with words beginning in liquid consonants also happens to be associated with female speakers; the cluster with terminal liquid consonants is associated with male speakers. Note, however, that since each word was spoken by approximately as many men as women, word-clustering comes at the expense of gender-clustering.

7.9 Discussion

7.9.1 Improvements

In every case that an auxiliary variable was used, there was a performance increase. Moreover, increasing the modeling power of the network by increasing the amount of context

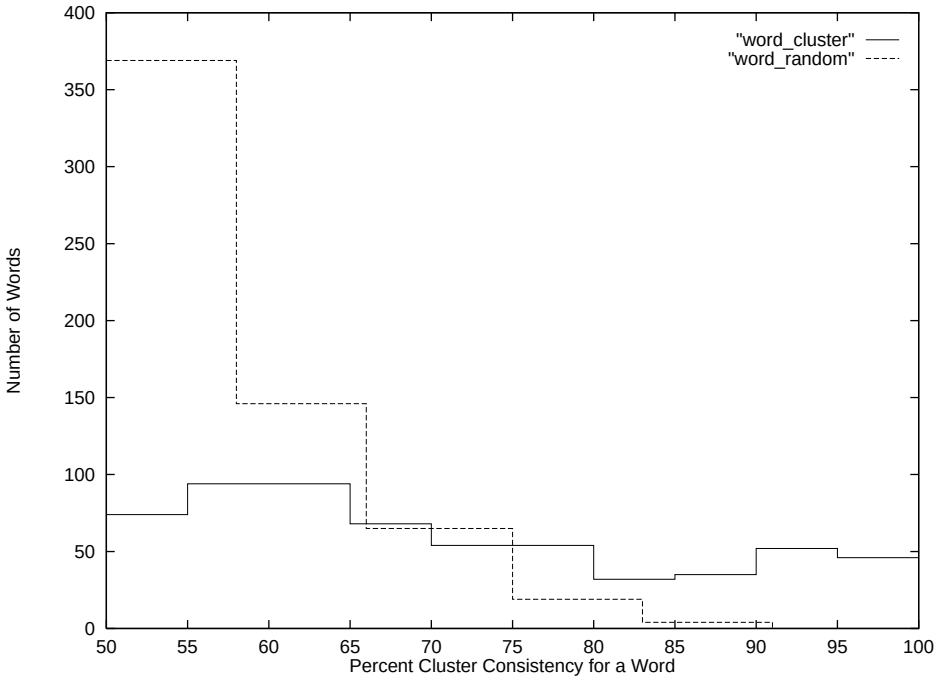

Figure 7.5: The frequency with which utterances of a single word were assigned to the same cluster. The area of the histogram representing a random distribution is greater than the area of the observed histogram because of of a binning artifact. On average, there are 12 occurrences of each word; the first bin represents 6 or 7 being classified together; the next 8, then 9, and so on. Due to the small number of bins, the widths are large.

Cluster 1			Cluster 2	
aboveboard	elsewhere	incapable	bathing	irving
mainville	melrose	oval	landberg	laundromat
store	unapproachable	ungovernable	lifeboat	livelihood
visual	whipples	arrivals	citizend	floodgates
gospels	salesroom	scarsdale	increasingly	motown
forced	starched	summerall	negligently	plaintiff
astronomical	bakeware	bridgeforth	redness	spacelink
fairchilds	geographical	gulps	implicitly	mcbee
mistrustful	pinwheel	quails	engagingly	heaves
torso	unforgivable	unusual	honda	nape
waffles	carlson	pathological	eighths	included
unborn	untraveled	westwall	lancelet	nat
strolls	totals	allies	peanut	lindsey
beagle	cashdrawer	dialed	cupcakes	woodlawn
reels	seldom	silverstone		
squirreled	tranquil	unethical		
isabell	spoilt	unquestionable		
foghorn	bale	unawares		
dimsdale	heartfelt	sparkled		
pebbles	seafowl	bulls		
baffled	dolphin	squabble		
westworld				

Table 7.6: The words that occurred in a particular cluster more than 90% of the time. About half the words in the first cluster end in liquid consonants (/l/ or /r/), even more if terminal /s/ is allowed. For example, "unapproachable" and "astronomical." None of the words in the second cluster end in liquid consonants. Instead, about a quarter of them *begin* with liquid consonants, e.g. "lifeboat" and "laundromat." Only one of the words in the first cluster, "reels," begins with a liquid consonant.

state produced further improvements. This indicates that context modeling with auxiliary state information is an effective way of decreasing speech-recognition error rates.

The context variable was able to capture several different phenomena, ranging from simple correlations between the observations within a single frame to gender and word-specific characteristics. In general, networks in which the context variables were linked across time did better than corresponding networks without temporal links. This is unsurprising because neither acoustic nor articulatory properties are expected to change rapidly over time; this is expected for the articulators because of physical inertia, and for acoustics both because they are generated by articulators, and because the acoustic features are generated from overlapping frames of speech.

A context-sensitive alphabet was an effective way of improving performance, but here too an auxiliary variable was beneficial. This makes sense because context-dependent alphabets encode prior knowledge about coarticulatory effects, but do not pay attention to the particulars of any specific utterance. An auxiliary variable has the ability to encode information on a case-by-case basis.

7.9.2 What Does it Mean?

In order to understand the meaning of the context variable, we examined its correlation with the different acoustic features, and found that it is highly correlated with the combined C_0 and delta-C_0 observation stream. This relation is graphed for four different network structures in Figure 7.6. Roughly speaking, C_0 is indicative of the overall energy in an acoustic frame. The maximum value in an utterance is subtracted, so the value is never greater than 0. Assuming that each frequency bin contributed an equal amount of energy, the C_0 range corresponds to an energy range of 50db.

This figure shows that despite similar word-error rates, the different network structures worked by learning different kinds of patterns in the data. [2]

A second important pattern that emerges is that in the networks with time-continuity arcs, the context variable is characterized by a high degree of continuity. To demonstrate that the networks will tend to learn continuity, an experiment was made in which the articulator network was initialized to reflect voicing in a very extreme way: the

[2] Figure revised 7/98.

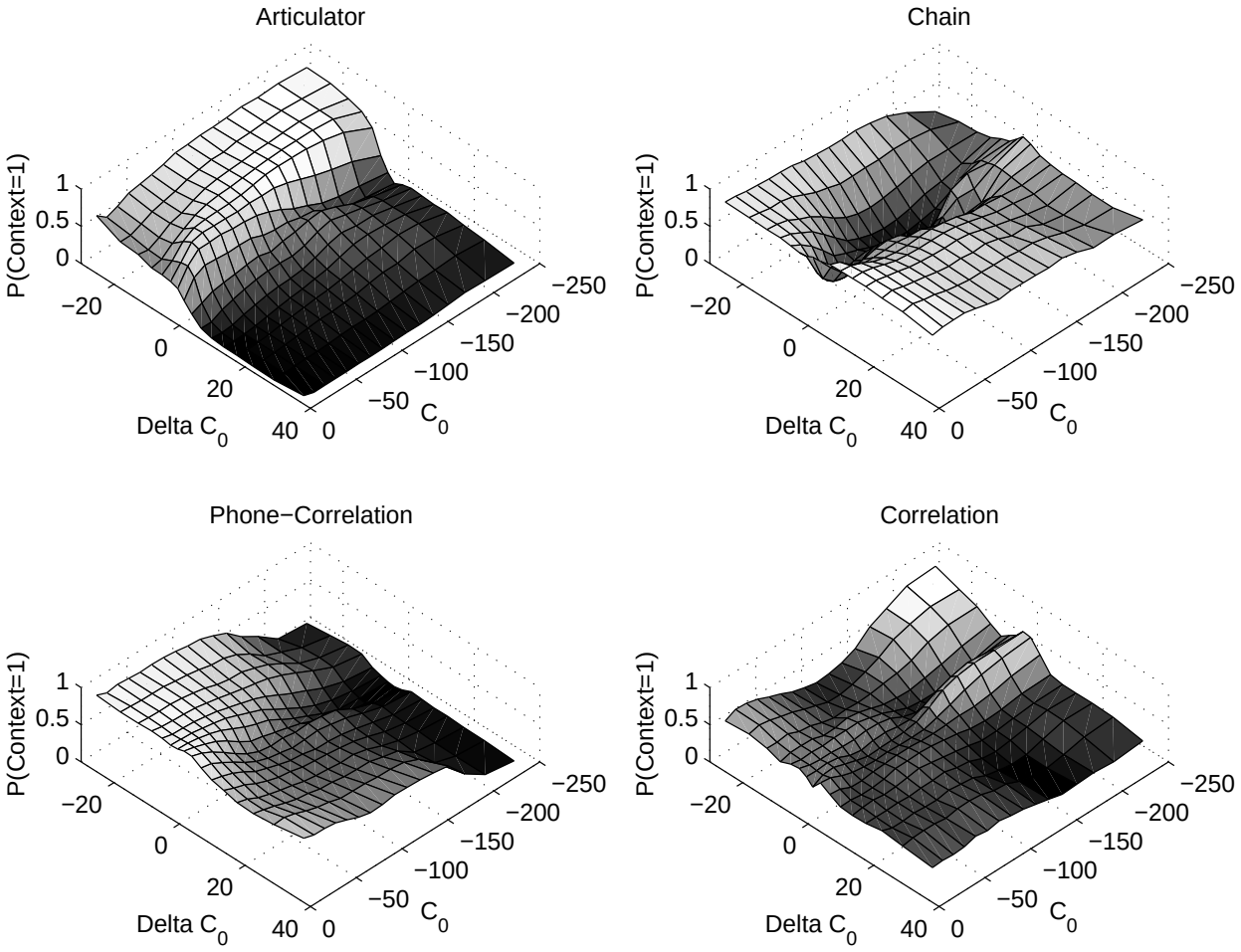

Figure 7.6: Probability that the context variable has the value 1 as a function of C_0 and delta-C_0.

probability of a context value of 1 was set almost to 1 (regardless of its previous value) for voiced phones, and almost to 0 for unvoiced phones. The EM procedure was then applied, and in the learned parameters, the striking characteristic is that the context value is almost always unlikely to change. This is consistent with a physical model of a slowly changing inertial process. The initial and learned parameters are shown in Figures 7.7 and 7.8. For the results presented in previous sections, the context variable was initialized with less extreme values; this decreased the number of training iterations, and gave slightly better performance.

Although Figures 7.7 and 7.8 demonstrate that the context variable displays the continuity which is expected from something that models a physical object, we have been unable to associate it with any specific articulator. In particular, the figures illustrate that the conditional probabilities after training show no clear correlation with the voiced/unvoiced distinction between phones. The EM training procedure has apparently taken advantage of all the degrees of freedom available to it to maximize the data-probability, resulting in parameters that reflect a combination of many underlying factors. Training with data in which the articulator positions are known will likely lead to simpler interpretations of the learned parameters.

From the previous discussion of the context variable's acoustic correlates, it is clear that the different networks learned different patterns in the data. One way of measuring this is to see how similar the errors made by the different networks are. This is shown for several of the networks in Table 7.7. This indicates that the Correlation and PD-Correlation networks are relatively similar, as are the Chain and Articulator networks. The biggest difference is between networks using a basic phoneme alphabet and those using a context-dependent alphabet.

7.9.3 Perspective

An important tradeoff which is often discussed in the speech recognition literature is between error-rates and the number of parameters used; this is shown in Figure 7.9 for the networks studied in this chapter. The correlation between the number of parameters used and the error rate is striking, and in this context, the use of Bayesian networks can be understood as a directed and intelligent way of increasing the number of model parameters.

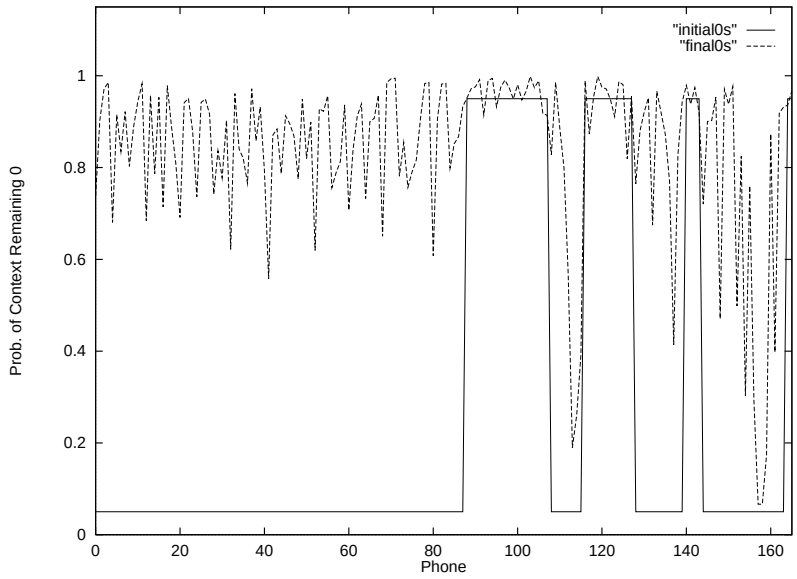

Figure 7.7: Learning continuity. The lines show $P(C_t = 0|C_{t-1} = 0, Q_t = p)$, i.e. the probability of the context value remaining 0 across two frames of speech, as a function of phone. The solid line is before training, and the dotted line is after training. The context variable represents voicing, so values close to 1.0 are for voiced phones. After training, the context value is unlikely to change, regardless of phone. This reflects temporal continuity.

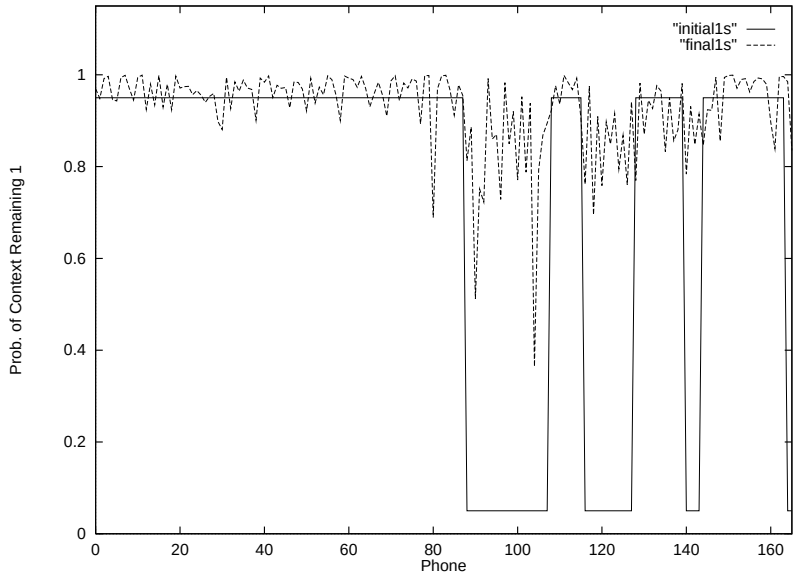

Figure 7.8: Learning continuity. This graph shows shows that a context value of 1 also shows continuity.

Network	Corr.	PD-Corr.	Chain	Art.	CDA-257	CDA-Art	CDA-510
Base	65	69	49	50	35	32	29
Corr.		68	55	55	37	34	32
PD-Corr.			51	50	38	34	32
Chain				56	44	38	32
Art.					41	38	33
CDA-257						53	32
CDA-Art							44

Table 7.7: Percent similarity in the errors made by pairs of recognizers. If A and B are the sets of words the systems respectively got wrong, similarity is defined as $100\frac{|A \cap B|}{|A \cup B|}$.

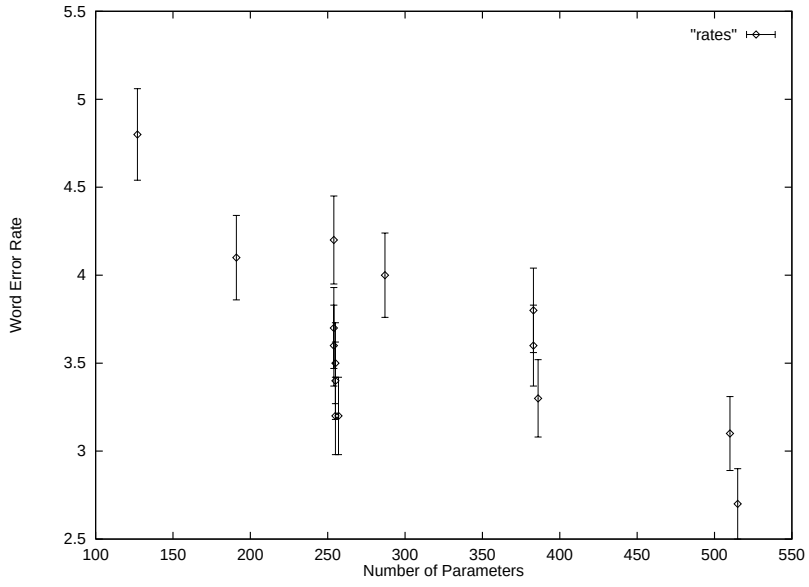

Figure 7.9: Error rate as a function of the number of network parameters. The errorbars represent one standard deviation in either direction.

Chapter 8

Conclusion and Future Work

8.1 A Roadmap for the Future

This thesis has demonstrated that Bayesian network based ASR systems can be built and expected to give good performance; the following sections point out some promising directions for future work. The discussion is divided into two parts: first a section on purely technological issues, and then a section on what the technology might be used for.

8.1.1 Technological Enhancements

Continuous Valued Observations

Most current ASR systems use real valued acoustic feature vectors, and they typically model the distribution over observations with a mixture of Gaussians. It is relatively easy to incorporate real-valued variables into a Bayesian network, provided that the variables are always observed and have only discrete parents. This is the case, for example, in a DBN set up to emulate an HMM: the discrete phonetic state variable is the parent of the observation variable.

In this case, it suffices to model the likelihood of an observation with a single Gaussian: the effect of mixtures can be obtained by adding an extra discrete "mixture" variable as one of the observation's parents. Depending on the value of this mixture variable, one of several Gaussians will be selected. Under these circumstances, the programming modi-

fications simply require using a Gaussian to compute the likelihood of an observation, and compiling sufficient statistics to reestimate the Gaussian mean and covariances. The cases where the real valued variables are hidden, or have real-valued parents, or are themselves the parents of discrete variables are more difficult; see (Murphy 1998) for a review of inference and learning with continuous variables.

Atemporal Variables

Several of the schemes we have discussed have used variables whose value does not change over time. For example, speaker accent and gender do not change during the course of a conversation. The structures proposed to model such phenomena deal with the problem by repeating a variable through every timeslice in the network, and then use deterministic constraints to ensure that it does not change its value. This is somewhat awkward, and could be better dealt with by introducing the concept of an atemporal variable into the representational framework.

Continuous Speech

This thesis has dealt only with isolated word recognition, and clearly continuous recognition is an important extension. As discussed in section 3.7, this is most easily done by using chain-structured clique trees, which are nevertheless more computationally efficient than a cross-product HMM. The next step in applying DBNs to continuous speech recognition would be to construct such a system. The frontier algorithm of (Zweig 1996) might form the basis for this work. Alternatively, the more complex backbone decoding procedure could be used.

8.1.2 Modeling Strategies

Many of the modeling techniques described in Section 6 have not yet been tried, and should be explored in the future. Rather than repeating them, however, this section will focus on a few areas that have either gone unstressed or undiscussed.

Use Articulatory Data

Clearly the most important step in constructing realistic articulatory models will be to train the models using the actual positions, as determined by X-rays, NMR, ultrasound, magnetic coils, radar, or some other imaging technique. This is not an easy step however, and even assuming that data is available, several important issues must be resolved:

- Should the data be discretized? If so, how? If not, it will be necessary to work with a hybrid Bayesian network, and the problems of mixing continuous and discrete variables must be resolved.
- What features are relevant? Positions? Velocities? Accelerations?
- Given a set of articulatory features or positions, how should the distribution over expected acoustics be modeled? This is especially difficult when the variables are continuous and the distribution must be represented functionally.
- What characteristics are invariant across a wide variety of speakers? How should the data be processed to generate speaker-independent features?

Clustering Phonetic Units

One way of creating context dependent units, say left-context biphones, is to create a new phone for each phoneme in the left-context of every possible preceding phoneme. Many current speech recognition systems refine this kind of technique by grouping together contextual phonemes with similar phonetic features (Young *et al.* 1997). For example, one might simply use a simple binary distinction based on whether the preceding phoneme is nasalized. While this kind of analysis requires the use of a-priori linguistic knowledge, it is also possible to derive context dependent units through a data-driven clustering process (Lee 1989; Young *et al.* 1997).

A similar procedure can be encoded in a Bayesian network. Suppose one maintains in each timeslice a variable representing not just the current phone, but also the preceding and following phones. (This information is readily available if linear word-models are used.) One can then link this triplet of phones to a single hidden variable that itself has relatively

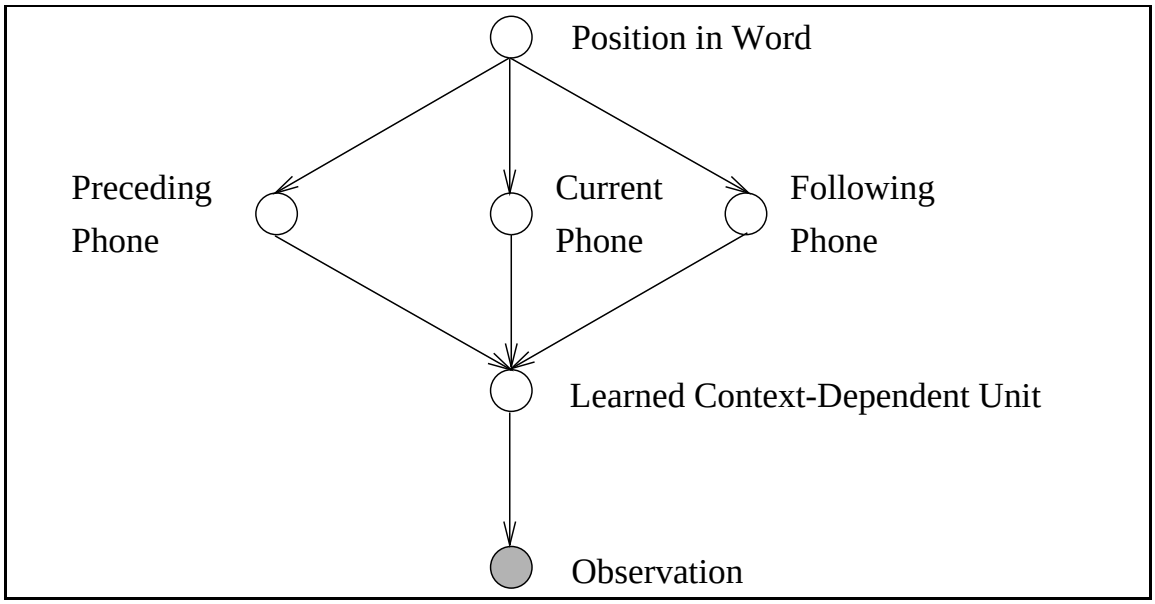

Figure 8.1: Network structure for automatic induction of context dependent units.

few values, and condition the observations on this hidden variable. In the process of EM learning, the network will be forced to learn a compact encoding of the triplet of available phones, thus automatically deriving a context-dependent alphabet. The network structure for this is shown in Figure 8.1.

Pronunciation Variants

As discussed in Section 6, the work of (Fosler-Lussier & Morgan 1998; Mirghafori *et al.* 1995) shows that pronunciations can vary according to speaking rate. By conditioning the phone variable both on position in the word model, and a rate-of-speech measure, it will be relatively straightforward to model phone-substitutions due to speaking-rate variability.

8.2 Closing

On the first page of his decade old classic, Kai-Fu Lee identified the "Lack of a sophisticated yet tractable model of speech" as a principle deficiency of speech recognition systems (Lee 1989). This thesis has argued that the Bayesian network framework is ideally suited to constructing just the kind of precise, expressive, and tractable models that are

needed.

At a low-level, the thesis has shown how to structure Bayesian networks to address the technical problems of model composition and efficient inference with deterministic variables. Solving these problems is crucial to encoding in a DBN the distributions over phonetic sequences that are typically represented in other systems with finite state automata. This has not been done before with Bayesian networks, and allows both pronunciation and acoustic modeling to be done in a unified framework.

At a higher level, the thesis shows how to model a large collection of phenomena in the Bayesian network framework. These phenomena include articulatory models, rate-of-speech variability, speaker characteristics, perceptual models, and noise modeling. The principle characteristic of the Bayesian network framework that gives it this flexibility is that the probabilistic models are expressed in terms of arbitrary sets of random variables. Thus, once the groundwork has been laid by writing a program for ASR with DBNs, a wide variety of model structures can be encoded and tested.

A system for doing isolated word recognition in the Bayesian network framework has been implemented, and experimental results indicate that a significant improvement is possible by augmenting the phonetic state information with one or more auxiliary state variables. This improvement is apparent both when a phonemic alphabet is used, and with a context-dependent alphabet. This suggests that the auxiliary variable is able to model utterance-specific contextual effects that a context-dependent alphabet is insensitive to.

Bibliography

Akmajian, A., R.A. Demers, A.K. Farmer, & R.M. Harnish. 1995. *Linguistics: An Introduction to Language*. Cambridge, Massachusetts: MIT Press, fourth edition.

Allen, James F. 1995. *Natural Language Understanding*. Redwood City, California: Benjamin/Cummings.

Anderson, Brian D. O., & John B. Moore. 1979. *Optimal Filtering*. Englewood Cliffs, New Jersey: Prentice-Hall.

A/S, Hugin Expert. 1995. *HUGIN API Reference Manual*. Hugin Expert A/S.

Bahl, L.R., P.F. Brown, P.V. deSouza, & L.R. Mercer. 1986. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *ICASSP-86*, 49–52.

Baker, J. 1975. The Dragon system—an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.24–29.

Baum, L. E., T. Petrie, G. Soules, & N. Weiss. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Annals of Mathematical Statistics* 41.164–171.

Bellman, Richard Ernest. 1957. *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

Bengio, Y., Y. LeCun, C. Nohl, & C. Burges. 1995. A nn/hmm hybrid for on-line handwriting recognition. *Neural Computation* 7.1289–1303.

Binder, John, Daphne Koller, Stuart Russell, & Keiji Kanazawa. 1997. Adaptive probabilistic networks with hidden variables. *Machine Learning* 29.213–244.

BISHOP, CHRISTOPHER M. 1995. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press.

BLANCHET, I., C. FRANKIGNOUL, & M.A. CANE. 1997. A comparison of adaptive kalman filters for a tropical pacific ocean model. *Monthly Weather Review* 125.40–58.

BOURLARD, HERVE, & NELSON MORGAN. 1994. *Connectionist Speech Recognition: A Hybrid Approach*. Dordrecht, The Netherlands: Kluwer.

BROWMAN, C.P., & L. GOLDSTEIN. 1992. Articulatory phonology: An overview. *Phonetica* 49.155–180.

BUNTINE, WRAY L. 1994. Operations for learning with graphical models. *Journal of Artificial Intelligence Research* 2.159–225.

CHAVEZ, R. M., & G. F. COOPER. 1990a. An empitrical evaluation of a randomized algorithm for probabilistic inference. In *Uncertainty in Artificial Intelligence 5*, ed. by M. Henrion, R. D. Shachter, L. N. Kanal, & J. F. Lemmer, 191–207. Elsevier Science Publishers.

——, & ——. 1990b. A randomized approximation algorithm for probabilkistic inference on bayesian belief networks. *Networks* 20.661–685.

COHEN, M., H. FRANCE, N. MORGAN, D. RUMELHART, & V. ABRASH. 1992. Hybrid neural network/hidden markov model continuous-speech recognition. In *ICSLP-92*, 915–918.

COOPER, G., & E. HERSKOVITS. 1992. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning* 9.309–347.

COOPER, GREGORY F. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42.393–405.

DAGUM, P., & M. LUBY. 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence* 60.141–153.

——, & ——. 1997. An optimal approximation algorithm for bayesian inference. *Artificial Intelligence* 93.1–27.

DATTELLIS, C.E., & E. CORTINA. 1990. Simultaneous estimation of neutron density and reactivity in a nuclear reactor using a bank of kalman filters. *Nuclear Science and Engineering* 105.297–299.

DAVIS, S, & P. MERMELSTEIN. 1980. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.357–366.

DEAN, THOMAS, & KEIJI KANAZAWA. 1988. Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, 524–528, St. Paul, Minnesota. American Association for Artificial Intelligence.

DELLER, J. R., J. G. PROAKIS, & J.H.L. HANSEN. 1993. *Discrete-Time Processing of Speech Signals*. London: Macmillan.

DEMPSTER, A., N. LAIRD, & D. RUBIN. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* 39 (Series B).1–38.

DENG, L. 1996. Speech recognition using autosegmental representation of phonological units with interface to the trended hmm. *Free Speech Journal* .

——, & K. ERLER. 1992. Structural design of hidden markov model speech recognizer using multivalued phonetic features: Comparison with segmental speech units. *Journal of the Acoustical Society of America* 92.3058–3067.

——, & D. SUN. 1994. A statistical approach to automatic speech recognition using the atomic speech units constructed from overlapping articulatory features. *Journal of the Acoustical Society of America* 95.2702–2719.

DIGALAKIS, V., J.R. ROHLICEK, & M. OSTENDORF. 1993. Ml estimation of a stochastic linear system with the em algorithm and its application to speech recognition. *IEEE Transactions on Speech and Audio Processing* 1.431–442.

DUPONT, S., H. BOURLARD, O. DEROO, & J.-M. FONTAINE, V.AND BOITE. 1997. Hybrid HMM/ANN systems for training independent tasks: Experiments on PhoneBook and related improvements. In *ICASSP-97: 1997 International Conference on Acoustics, Speech, and Signal Processing*, 1767–1770, Los Alamitos, CA. IEEE Computer Society Press.

EPHRAIM, Y., A. DEMBO, & L.R. RABINER. 1989. A minimum discrimination information approach for hidden markov modeling. *IEEE Transactions on Information Theory* 35.1001–1003.

ERLER, K., & L. DENG. 1993. Hidden markov model representation of quantized articulatory features for speech recognition. *Computer Speech and Language* 7.265–282.

——, & G. FREEMAN. 1996. An hmm-based speech recognizer using overlapping articulatory features. *Journal of the Acoustical Society of America* 100.2500–2513.

FAHLMAN, S.E., & C LEBIERE. 1990. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems II*, ed. by D.S. Touretzky, 524–532. San Mateo, California: Morgan Kaufmann.

FOSLER-LUSSIER, E., & N. MORGAN. 1998. Effects of speaking rate and word predictability on conversational pronunciations. In *Tutorial and Research Workshop on Modeling Pronunciation Variation for Automatic Speech Recognition*.

FREAN, M. 1990. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation* 2.198–209.

FRIEDMAN, NIR. 1997. Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning*, Nashville, Tennessee. Morgan Kaufmann.

FRITSCH, J. 1997. Acid/hnn: A framework for hierarchical connectionist acoustic modeling. In *ASRU-97*, 164–171.

GALES, M.J.F., & S. YOUNG. 1992. An improved approach to the hidden markov model decomposition of speech and noise. In *ICASSP-92*, I–233–I–236.

GHAHRAMANI, Z., & M. I. JORDAN. 1995. Factorial hidden Markov models. Technical Report 9502, MIT Computational Cognitive Science Report.

——, & ——. 1997. Factorial hidden Markov models. *Machine Learning* 29.

GHITZA, O. 1991. Auditory nerve representation as a basis for speech processing. In *Advances in Speech Processing*, ed. by S. Furui & M. Sondhi, 453–485. Marcel Dekker.

GOODWIN, GRAHAM C., & KWAI SANG SIN. 1984. *Adaptive Filtering and Control*. Englewood Cliffs, New Jersey: Prentice-Hall.

HAYKIN, SIMON. 1994. *Neural Networks: A Comprehensive Foundation*. London: Macmillan.

——. 1996. *Adaptive Filter Theory*. Englewood Cliffs, New Jersey: Prentice-Hall, third edition.

HECKERMAN, D. 1995. A tutorial on learning with Bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, Redmond, Washington. Revised June 1996.

HENNEBERT, J., C. RIS, H. BOURLARD, S. RENALS, & N. MORGAN. 1997. Estimation of global posteriors and forward-backward training of hybrid hmm/ann systems. In *Eurospeech 97*, volume 4, 1951–1954.

HENRION, MAX. 1988. Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence 2*, ed. by John F. Lemmer & Laveen N. Kanal, 149–163. Amsterdam, London, New York: Elsevier/North-Holland.

HERMANSKY, H., & N. MORGAN. 1994. Rasta processing of speech. *IEEE Transactions on Speech and Audio Processing* 2.578–589.

HERTZ, JOHN, ANDERS KROGH, & RICHARD G. PALMER. 1991. *Introduction to the Theory of Neural Computation*. Reading, Massachusetts: Addison-Wesley.

HOGDEN, J., A. LOFQVIST, V. GRACCO, I. ZLOKARNIK, P. RUBIN, & E. SALTZMAN. 1996. Accurate recovery of articulator positions from acoustics: New conclusions based on human data. *Journal of the Acoustical Society of America* 100.1819–1834.

HOLZRICHTER, J.F., W.A. LEA, L.C. NG, & G.C. BURNETT. 1996. Speech coding, recognition, and synthesis using radar and acoustic sensors. Technical Report UCRL-ID-123687, Lawrence Livermore National Laboratory.

HU, J., M.K. BROWN, & W. TURIN. 1996. Hmm based on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.1039–1045.

JACOBS, R.A, & M.I. JORDAN. 1991. A competitive modular connectionist architecture. In *Advances in Neural Information Processing Systems III*, ed. by R.P Lippmann, J.E. Moody, & Touretzky, 767–773. San Mateo, California: Morgan Kaufmann.

JACOBS, R.A., M.I. JORDAN, S.J. NOWLAN, & G.E. HINTON. 1991. Adaptive mixtures of local experts. *Neural Computation* 3.79–87.

JELINEK, F. 1976. Continuous speech recognition by statistical methods. *Proceedings of the IEEE* 64.532–556.

JELINEK, FREDERICK. 1997. *Statistical Methods for Speech Recognition*. Cambridge, Massachusetts: MIT Press.

JENSEN, F., & F. V. JENSEN. 1994. Optimal junction trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, 360–366, Seattle, Washington. Morgan Kaufmann.

——, ——, & S. L. DITTMER. 1994. From influence diagrams to junction trees. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, 367–373, Seattle, Washington. Morgan Kaufmann.

JENSEN, FINN V., STEFFEN L. LAURITZEN, & KRISTIAN G. OLESEN. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly* 5.269–282.

JORDAN, M.I. ANDS JACOBS. 1992. Hierarchies of adaptive experts. In *Advances in Neural Information Processing Systems IV*, ed. by J.E. Moody, S.J. Hanson, & R.P. Lippmann, 985–992. San Mateo, California: Morgan Kaufmann.

JUANG, B-H., & L.R. RABINER. 1985. Mixture autoregressive hidden markov models for speech signals. *IEEE Transactions on Acoustics, Speech and Signal Processing* 1404–1413.

KALMAN, R. E. 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* 35–46.

KANAZAWA, KEIJI, DAPHNE KOLLER, & STUART RUSSELL. 1995. Stochastic simulation algorithms for dynamic probabilistic networks. In *Uncertainty in Artificial Intelli-*

gence: Proceedings of the Eleventh Conference, 346–351, Montreal, Canada. Morgan Kaufmann.

KARPLUS, K., C. SJOLANDER, K. AMD BARRETT, & M. CLINE. 1997. Predicting protein structure using hidden markov models. *Proteins - Structure Function and Genetics* SUPP1.134–139.

KENNY, P., M. LENNIG, & P. MERMELSTEIN. 1990. A linear predictive hmm for vector-valued observations with applications to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38.220–225.

KJAERULFF, U. 1992. A computational scheme for reasoning in dynamic bayesian networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference*, 121–129, Stanford, California. Morgan Kaufmann.

KROGH, A., I. MIAN, & D. HAUSSLER. 1994. A hidden markov model that finds genes in e-coli dna. *Nucleic Acids Research* 22.4768–4778.

KULP, D., D. HAUSSLER, M.G. REESE, & F.H. EECKMAN. 1996. A generalized hidden markov model for the recognition of human genes in dna. In *4th Conference on Intelligent Systems in Molecular Biology*.

LADEFOGED, P. 1993. *A Course in Phonetics*. New York: Harcourt Brace Jovanovich, third edition.

LANG, K. J., & G. E. HINTON. 1988. The development of the time-delay neural network architecture for speech recognition. Technical Report CMU-CS-88-152, Carnegie Mellon University.

LAPEDES, A., & R. FARBER. 1988. How neural nets work. In *Neural Information Processing Systems*, ed. by D. Z. Anderson, 442–456. American Institute of Physics.

LAURITZEN, STEFFEN L. 1991. The EM algorithm for graphical association models with missing data. Technical Report TR-91-05, Department of Statistics, Aalborg University.

——, & DAVID J. SPIEGELHALTER. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society* B 50.157–224.

Le Cun, Y., B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, & L.D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1.541–551.

——, J.S. Denker, & S.A. Solla. 1990. Optimal brain damage. In *Advances in Neural Information Processing Systems II*, ed. by D.S. Touretzsky, 396–404.

Lee, Kai-Fu. 1989. *Automatic speech recognition: The development of the SPHINX system*. Dordrecht, The Netherlands: Kluwer.

Levinson, S.E. 1986. Continuously variable duration hidden markov models for automatic speech recognition. *Computer Speech and Language* 1.29–45.

——, L.R. Rabiner, & M.M. Sondhi. 1983. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *The Bell System Technical Journal* 62.1035–1074.

Linde, Y., A. Buzo, & Gray R. M. 1980. An algorithm for vector quantizer design. *IEEE Transactions on Communication* COM-28.84–95.

Makhoul, J. 1975. Linear prediction: A tutorial review. *Proceedings of the IEEE* 63.561–580.

——, & R. Schwartz. 1995. State of the art in continuous speech recognition. *Proceedings of the National Academy of Science* 92.9956–9963.

Mammone, R.J., X. Zhang, & R.P. Ramachandran. 1996. Robust speaker recognition. *IEEE Signal Processing Magazine* 58–71.

Manohar Rao, M.J. 1987. *Filtering and Control of Macroeconomic Systems: A Control System Incorporating the Kalman Filter for the Indian Economy*. Amsterdam, London, New York: Elsevier/North-Holland.

Mirghafori, N., E. Fosler, , & N. Morgan. 1995. Fast speakers in large vocabulary continuous speech recognition: Analysis and antidotes. In *Eurospeech-95*.

Moody, J.E., & C. J. Darken. 1989. Fast learning in networks of locally-tuned processing units. *Neural Computation* 1.281–294.

MORGAN, N., & H. BOURLARD. 1992. Factoring networks by a statistical method. *Neural Computation* 4.835–838.

——, ——, S. GREENBERG, & H HERMANSKY. 1994. Stochastic perceptual auditory-event-based models for speech recognition. In *Proceedings ICSLP-94*, 1943–1946.

——, & E. FOSLER-LUSSIER. 1998. Combining multiple estimators of speaking rate. In *ICASSP-98*.

MURPHY, K. 1998. Inference and learning in hybrid bayesian networks. Technical Report UCB/CSD-98-990, UC Berkeley.

NEY, H. 1984. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.263–271.

NOLL, M.A. 1964. Short-time spectrum and "cepstrum" techniques for vocal-pitch detection. *Journal of the Acoustical Society of America* 36.296–302.

PAPCUN, G., J. HOCHBERG, T.R. THOMAS, F. LAROCHE, J. ZACKS, & S. LEVY. 1992. Inferring articulation and recognizing gestures from acoustics with a neural network trained on x-ray microbeam data. *Journal of the Acoustical Society of America* 92.688–700.

PEARL, JUDEA. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann.

PEOT, MARK, & ROSS SHACHTER. 1991. Fusion and propagation with multiple observations. *Artificial Intelligence* 48.299–318.

PITRELLI, J., C. FONG, S. WONG, J. SPITZ, & H. LEUNG. 1995. Phonebook: A phonetically-rich isolated-word telephone-speech database. In *ICASSP-95: 1995 International Conference on Acoustics, Speech, and Signal Processing*, 101–104, Los Alamitos, CA. IEEE Computer Society Press.

PORITZ, A.B. 1982. Linear predictive hidden markov models and the speech signal. In *ICASSP-82*, 1291–1294.

PRICE, WILLIAM H. 1992. *Numerical Recipes in C*. Cambridge: Cambridge University Press.

RABINER, L. R., & B.-H. JUANG. 1986. An introduction to hidden markov models. *IEEE ASSP Magazine* 4–16.

——, & ——. 1993. *Fundamentals of Speech Recognition*. Prentice-Hall.

ROBINSON, A.J., & F. FALLSIDE. 1988. Static and dynamic error propagation networks with application to speech coding. In *Neural Information Processing Systems*, ed. by D.Z. Anderson, 632–641.

——, & ——. 1991. A recurrent error propagation speech recognition system. *Computer Speech and Language* 5.259–274.

ROSE, D. J. 1970. Triangulated graphs and the elimination process. *Journal of Mathematical Analysis and Applications* 597–616.

RUSSELL, M.J., & R.K. MOORE. 1985. Explicit modeling of state occupancy in hidden markov models for automatic speech recognition. In *ICASSP-85*, 5–8.

SCALETTAR, R., & A. ZEE. 1988. Emergence of grandmother memory in feed forward networks: Learning with noise and forgetfulness. In *Connectionist Models and their Implications: Readings from Cognitive Science*, ed. by D. Waltz & J.A. Feldman, 309–332.

SCHWARTZ, R., J. KLOVSTAD, J. MAKHOUL, & J. SORENSEN. 1980. A preliminary design of a phonetic vocoder based on a diphone model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 32–35.

SHACHTER, R. D., & M. A. PEOT. 1989. Simulation approaches to general probabilistic inference on belief networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*, Windsor, Ontario. Morgan Kaufmann.

SHACHTER, R. S., & C. R. KENLEY. 1989. Gaussian influence diagrams. *Management Science* 35.527–550.

SIEGLER, M. A., & R. M. STERN. 1995. On the effects of speech rate in large vocabulary speech recognition syst ems. In *ICASSP-95*.

SMYTH, P., D. HECKERMAN, & M. JORDAN. 1996. Probabilistic independence networks for hidden Markov probability models. Technical Report MSR-TR-96-03, Microsoft Research, Redmond, Washington.

STOKBRO, K., D. K. UMBERGER, & J. A. HERTZ. 1990. Exploiting neurons with localized receptive fields to learn chaos. *Complex Systems* 4.603–622.

TOWELL, G.G., & J.W. SHAVLIK. 1993. Extracting refined rules from knowledge-based neural networks. *Machine Learning* 13.71–101.

VARGA, A.P., & R.K. MOORE. 1990. Hidden markov model decomposition of speech and noise. In *ICASSP-90*, 845–848.

WAIBEL, A., T. HANAZAWA, G. HINTON, K. SHIKANO, & K. J. LANG. 1989. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing* ASSP-37.328–339.

WAN, E. A. 1990. Temporal backpropagation for fir neural networks. In *IEEE International Joint Conference on Neural Networks*, volume 1, 575–580.

WILLIAMS, R. J., & D. ZIPSER. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* 1.270–280.

WU, S., B. KINGSBURY, N. MORGAN, & S. GREENBERG. 1998. Incorporating information from syllable-length time scales into automatic speech recognition. In *ICASSP-98*.

——, M.L. SHIRE, S. GREENBERG, & N. MORGAN. 1997. Integrating syllable boundary information into speech recognition. In *ICASSP-97*.

YOUNG, S. 1996. A review of large-vocabulary continuous-speech recognition. *IEEE Signal Processing Magazine* 45–57.

——, J. ODELL, D. OLLASON, V. VALTCHEV, & P. WOODLAND. 1997. *The HTK Book*. Entropic Cambridge Research Laboratory, 2.1 edition.

ZWEIG, GEOFF, & STUART J. RUSSELL. 1997. Compositional modeling with dpns. Technical Report UCB/CSD-97-970, Computer Science Division, University of California at Berkeley.

ZWEIG, GEOFFREY. 1996. A forward-backward algorithm for inference in bayesian networks and an empirical comparison with hmms. MS report, Computer Science Division, UC Berkeley.

——, & STUART RUSSELL. 1998. Speech recognition with dynamic bayesian networks. In *AAAI-98*.