INFORMATION TO USERS

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

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

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

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

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

UMI®

Dynamic Bayesian Networks:
Representation, Inference and Learning

by

Kevin Patrick Murphy

B.A. Hon. (Cambridge University) 1992
M.S. (University of Pennsylvania) 1994

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 Russell, Chair
Professor Michael Jordan
Professor Peter Bickel
Professor Jeffrey Bilmes

Fall 2002

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Dynamic Bayesian Networks:

Representation, Inference and Learning

by

Kevin Patrick Murphy

ABSTRACT

Dynamic Bayesian Networks:

Representation, Inference and Learning

by

Kevin Patrick Murphy

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Stuart Russell, Chair

Modelling sequential data is important in many areas of science and engineering. Hidden Markov models (HMMs) and Kalman filter models (KFMs) are popular for this because they are simple and flexible. For example, HMMs have been used for speech recognition and bio-sequence analysis, and KFMs have been used for problems ranging from tracking planes and missiles to predicting the economy. However, HMMs and KFMs are limited in their "expressive power". Dynamic Bayesian Networks (DBNs) generalize HMMs by allowing the state space to be represented in factored form, instead of as a single discrete random variable. DBNs generalize KFMs by allowing arbitrary probability distributions, not just (unimodal) linear-Gaussian. In this thesis, I will discuss how to represent many different kinds of models as DBNs, how to perform exact and approximate inference in DBNs, and how to learn DBN models from sequential data.

In particular, the main novel technical contributions of this thesis are as follows: a way of representing Hierarchical HMMs as DBNs, which enables inference to be done in $O(T)$ time instead of $O(T^3)$, where T is the length of the sequence; an exact smoothing algorithm that takes $O(\log T)$ space instead of $O(T)$; a simple way of using the junction tree algorithm for online inference in DBNs; new complexity bounds on exact online inference in DBNs; a new deterministic approximate inference algorithm called factored frontier; an analysis of the relationship between the BK algorithm and loopy belief propagation; a way of applying Rao-Blackwellised particle filtering to DBNs in general, and the SLAM (simultaneous localization and mapping) problem in particular; a way of extending the structural EM algorithm to DBNs; and a variety of different applications of DBNs. However, perhaps the main value of the thesis is its catholic presentation of the field of sequential data modelling.

DEDICATION

To my parents

for letting me pursue my dream

for so long

so far away from home

&

To my wife

for giving me

new dreams to pursue

ACKNOWLEDGMENTS

I would like to thank my advisor, Stuart Russell, for supporting me over the years, and for giving me so much freedom to explore and discover new areas of probabilistic AI. My other committee members have also been very supportive. Michael Jordan has long been an inspiration to me. His classes and weekly meetings have proved to be one of my best learning experiences at Berkeley. Jeff Bilmes proved to be a most thorough reviewer, as I expected, and has kept me honest about all the details. Peter Bickel brought a useful outsider's perspective to the thesis, and encouraged me to make it more accessible to non computer scientists (although any failings in this regard are of course my fault).

I would like to thank my many friends and colleagues at Berkeley with whom I have had the pleasure of working over the years. These include Eyal Amir, David Andre, Serge Belongie, Jeff Bilmes, Nancy Chang, Nando de Freitas, Nir Friedman, Paul Horton, Srini Narayanan, Andrew Ng, Mark Paskin, Sekhar Tatikonda, Yair Weiss, Erix Xing, Geoff Zweig, and all the members of the RUGS and IR groups.

I would like to thank Jim Rehg for hiring me as an intern at DEC/Compaq/HP Cambridge Research Lab in 1997, where my Bayes Net Toolbox (BNT) was born. I would like to thank Gary Bradski for hiring me as an intern at Intel in 2000 to work on BNT, and for providing me with the opportunity to work with people spanning three countries formerly known as superpowers — USA, China and Russia. In particular, I would like to thank Wei Hu and Yimin Zhang, of ICRC, for their help with BNT. I would also like to thank the many people on the web who have contributed bug fixes to BNT. By chance, I was able to work with Sebastian Thrun during part of my time with Intel, for which I am very grateful.

I would like to thank my friends in Jennie Nation and beyond for providing a welcome distraction from school. Finally, I would like to thank my wife Margaret for putting up with my weekends in the office, for listening to my sagas from Soda land, and for giving me the motivation to finish this thesis.

Contents

Chapter 1

Introduction

1.1 State-space models

Sequential data arises in many areas of science and engineering. The data may either be a time series, generated by a dynamical system, or a sequence generated by a 1-dimensional spatial process, e.g., bio-sequences. One may be interested either in online analysis, where the data arrives in real-time, or in offline analysis, where all the data has already been collected.

In online analysis, one common task is to predict future observations, given all the observations up to the present time, which we will denote by $y_{1:t} = (y_1, \ldots, y_t)$. (In this thesis, we only consider discrete-time systems, hence t is always an integer.) Since we will generally be unsure about the future, we would like to compute a best guess. In addition, we might want to know how confident we are of this guess, so we can hedge our bets appropriately. Hence we will try to compute a probability distribution over the possible future observations; we denote this by $P(y_{t+h}|y_{1:t})$, where $h > 0$ is the horizon, i.e., how far into the future we want to predict.

Sometimes we have some control over the system we are monitoring. In this case, we would like to predict future outcomes as a function of our inputs. Let $u_{1:t}$ denote our past inputs, and $u_{t+1:t+h}$ denote our next h inputs. Now the task is to compute $P(y_{t+h}|u_{1:t+h}, y_{1:t})$.

"Classical" approaches to time-series prediction use linear models, such as ARIMA, ARMAX, etc. (see e.g., [Ham94]), or non-linear models, such as neural networks (either feedforward or recurrent) or decision trees [MCH02]. For discrete data, it is common to use n-gram models (see e.g., [Jel97]) or variable-length Markov models [RST96, McC95].

There are several problems with the classical approach. First, we must base our prediction of the future on only a finite window into the past, say $y_{t-\ell:t}$, where $\ell \geq 0$ is the lag, if we are to do constant work per time step. If we know that the system we are modelling is Markov with an order $\leq \ell$, we will suffer no loss of performance, but in general the order may be large and unknown. Recurrent neural nets try to overcome this problem by using internal state, but they are still not able to model long-distance dependencies

[BF95]. Second, it is difficult to incorporate prior knowledge into the classical approach: much of our knowledge cannot be expressed in terms of directly observable quantities, and black-box models, such as neural networks, are notoriously hard to interpret. Third, the classical approach has difficulties when we have multi-dimensional (multi-variate) inputs and/or outputs. For instance, consider the problem of predicting (and hence compressing) the next frame in a video stream using a neural network. Actual video compression schemes (such as MPEG) try to infer the underlying "cause" behind what they see, and use that to predict the next frame. This is the basic idea behind state-space models, which we discuss next.

In a state-space model, we assume that there is some underlying hidden state of the world that generates the observations, and that this hidden state evolves in time, possibly as a function of our inputs.[1] In an online setting, the goal is to infer the hidden state given the observations up to the current time. If we let X_t represent the hidden state at time t, then we can define our goal more precisely as computing $P(X_t|y_{1:t}, u_{1:t})$; this is called the belief state.

Astrom [Ast65] proved that the belief state is a sufficient statistic for prediction/control purposes, i.e., we do not need to keep around any of the past observations.[2] We can update the belief state recursively using Bayes rule, as we explain below. As in the case of prediction, we maintain a probability distribution over X_t, instead of just a best guess, in order to properly reflect our uncertainty about the "true" state of the world. This can be useful for information gathering; for instance, if we know we are lost, we may choose to ask for directions.

State-space models are better than classical time-series modelling approaches in many respects [Aok87, Har89, WH97, DK00, DK01]. In particular, they overcome all of the problems mentioned above: they do not suffer from finite-window effects, they can easily handle discrete and multi-variate inputs and outputs, and they can easily incorporate prior knowledge. For instance, often we know that there are variables that we cannot measure, but whose state we would like to estimate; such variables are called hidden or latent. Including these variables allows us to create models which may be much closer to the "true" causal structure of the domain we are modelling [Pea00].

Even if we are only interested in observable variables, introducing "fictitious" hidden variables often results in a much simpler model. For example, the apparent complexity of an observed signal may be more simply explained by imagining it is a result of two simple processes, the "true" underlying state, which may evolve deterministically, and our measurement of the state, which is often noisy. We can then "explain

[1] The term "state-space model" is often used to imply that the hidden state is a vector in $\mathbb{R}^K$, for some K; I use the term more generally to mean a dynamical model which uses any kind of hidden state, whether it is continuous, discrete or both. e.g., I consider HMMs an example of a state-space model. In contrast to most work on time series analysis, this thesis focuses on models with discrete and mixed discrete-continuous states. One reason for this is that DBNs have their biggest payoff in the discrete setting: combining multiple continuous variables together results in a polynomial increase in complexity (see Section 2.4.2), but combining multiple discrete variables results in an exponential increase in complexity, since the new "mega" state-space is the cross product of the individual variables' state-spaces; DBNs help ameliorate this combinatorial explosion, as we shall see in Chapter 2.

[2] This assumes that the hidden state space is sufficiently rich. We discuss some ways to learn the hidden state space in Chapter 6. However, learning hidden state representations is difficult, which has motivated alternative forms of sufficient statistics [LSS01].

away" unexpected outliers in the observations in terms of a faulty sensor, as opposed to strange fluctuations in "reality". The underlying state may be of much lower dimensionality than the observed signal, as in the video compression example mentioned above.

In the following subsections, we discuss, in general terms, how to represent state-space models, how to use them to update the belief state and perform other related inference problems, and how to learn such models from data. We then discuss the two most common kinds of state-space models, namely Hidden Markov Models (HMMs) and Kalman Filter Models (KFMs). In subsequent chapters of this thesis, we will discuss representation, inference and learning of more general state-space models, called Dynamic Bayesian Networks (DBNs). A summary of the notation and commonly used abbreviations can be found in Appendix D.

1.1.1 Representation

Any state-space model must define a prior, $P(X_1)$, a state-transition function, $P(X_t|X_{t-1})$, and an observation function, $P(Y_t|X_t)$. In the controlled case, these become $P(X_t|X_{t-1}, U_t)$ and $P(Y_t|X_t, U_t)$; we allow the observation to depend on the control so that we can model active perception. For most of this thesis, we will omit U_t from consideration, for notational simplicity.

We assume that the model is first-order Markov, i.e., $P(X_t|X_{1:t-1}) = P(X_t|X_{t-1})$; if not, we can always make it so by augmenting the state-space. For example, if the system is second-order Markov, we just define a new state-space, $\tilde{X}_t = (X_t, X_{t-1})$, and set

$$P(\tilde{X}_t = (x_t, x_{t-1})|\tilde{X}_{t-1} = (x'_{t-1}, x_{t-2})) = \delta(x_{t-1}, x'_{t-1})P(x_t|x_{t-1}, x_{t-2}).$$

Similarly, we can assume that the observations are conditionally first-order Markov: $P(Y_t|Y_{1:t-1}, X_t) = P(Y_t|X_t, Y_{t-1})$. This is usually further simplified by assuming $P(Y_t|Y_{t-1}, X_t) = P(Y_t|X_t)$. These conditional independence relationships will be explained more clearly in Chapter 2.

We assume that the transition and observation functions are the same for all time; the model is said to be time-invariant or homogeneous. (Without this assumption, we could not model infinitely long sequences.) If the parameters do change over time, we can just add them to the state space, and treat them as additional random variables, as we will see in Section 6.1.1.

There are many ways of representing state-space models, the most common being Hidden Markov Models (HMMs) and Kalman Filter Models (KFMs). HMMs assume X_t is a discrete random variable[3], $X_t \in \{1, \ldots, K\}$, but otherwise make essentially no restrictions on the transition or observation function; we will explain HMMs in more detail in Section 1.2. Kalman Filter Models (KFMs) assume X_t is a vector of continuous random variables, $X_t \in R^K$, and that $X_{1:T}$ and $Y_{1:T}$ are jointly Gaussian. We will explain

[3] In this thesis, all discrete random variables will be considered unordered (cardinal), as opposed to ordered (ordinal), unless otherwise stated. (For example, $X_t \in \{\text{male}, \text{female}\}$ is cardinal, but $X_t \in \{\text{low}, \text{medium}, \text{high}\}$ is ordinal.) Ordinal values are sometimes useful for qualitative probabilistic networks [Wel90].

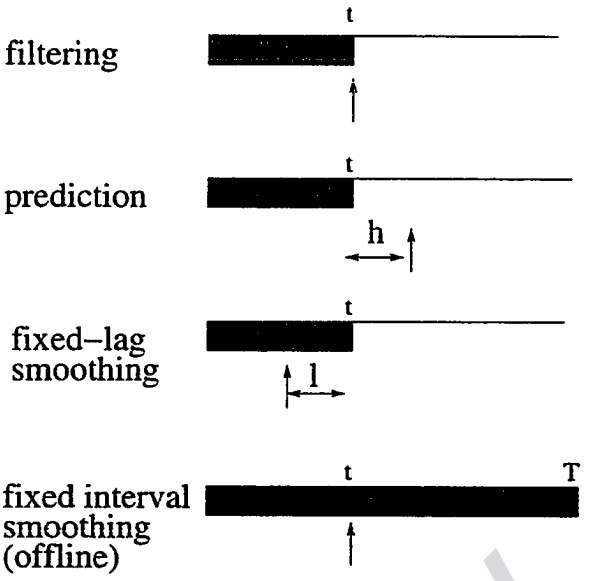

Figure 1.1: The main kinds of inference for state-space models. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, and T is the sequence length. See text for details.

KFMs in more detail in Section 1.3. Dynamic Bayesian Networks (DBNs) [DK89, DW91] provide a much more expressive language for representing state-space models; we will explain DBNs in Chapter 2.

A state-space model is a model of how X_t generates or "causes" Y_t and X_{t+1}. The goal of inference is to invert this mapping, i.e., to infer $X_{1:t}$ given $Y_{1:t}$. We discuss how to do this below.

1.1.2 Inference

We now discuss the main kinds of inference that we might want to perform using state-space models; see Figure 1.1 for a summary. The details of how to perform these computations depend on which model and which algorithm we use, and will be discussed later.

Filtering

The most common inference problem in online analysis is to recursively estimate the belief state using Bayes' rule:

$$\begin{aligned} P(X_t|y_{1:t}) &\propto P(y_t|X_t, y_{1:t-1})P(X_t|y_{1:t-1}) \\ &= P(y_t|X_t)\left[\sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|y_{1:t-1})\right] \end{aligned}$$

where the constant of proportionality is $1/c_t = 1/P(y_t|y_{1:t-1})$. We are licensed to replace $P(y_t|X_t, y_{1:t-1})$ by $P(y_t|X_t)$ because of the Markov assumption on Y_t. Similarly, the one-step-ahead prediction, $P(X_t|y_{1:t-1})$, can be computed from the prior belief state, $P(X_{t-1}|y_{1:t-1})$, because of the Markov

assumption on X_t.

We see that recursive estimation consists of two main steps: predict and update; predict means computing $P(X_t|y_{1:t-1})$, sometimes written as $\hat{X}_{t|t-1}$, and update means computing $P(X_t|y_{1:t})$, sometimes written as $\hat{X}_{t|t}$. Once we have computed the prediction, we can throw away the old belief state; this operation is sometimes called "rollup". Hence the overall procedure takes constant space and time (i.e., independent of t) per time step.

This task is traditionally called "filtering", because we are filtering out the noise from the observations (see Section 1.3.1 for an example). However, in some circumstances the term "monitoring" might be more appropriate. For example, X_t might represent the state of a factory (e.g., which pipes are malfunctioning), and we wish to monitor the factory state over time.

Smoothing

Sometimes we want to estimate the state of the past, given all the evidence up to the current time, i.e., compute $P(X_{t-\ell}|y_{1:t})$, where $\ell > 0$ is the lag, e.g., we might want to figure out whether a pipe broke L minutes ago given the current sensor readings. This is traditionally called "fixed-lag smoothing", although the term "hindsight" might be more appropriate. In the offline case, this is called (fixed-interval) smoothing; this corresponds to computing $P(X_t|y_{1:T})$ for all $1 \leq t \leq T$.

Smoothing is important for learning, as we discuss in Section 1.1.3.

Prediction

In addition to estimating the current or past state, we might want to predict the future, i.e., compute $P(X_{t+h}|y_{1:t})$, where $h > 0$ is how far we want to look-ahead. Once we have predicted the future hidden state, we can easily convert this into a prediction about the future observations by marginalizing out X_{t+h}:

$$P(Y_{t+h} = y|y_{1:t}) = \sum_x P(Y_{t+h} = y|X_{t+h} = x)P(X_{t+h} = x|y_{1:t})$$

If the model contains input variables U_t, we must specify $u_{t+1:t+h}$ in order to predict the effects of our actions h steps into the future, since this is a conditional likelihood model.

Control

In control theory, the goal is to learn a mapping from observations or belief states to actions (a policy) so as to maximize expected utility (or minimize expected cost). In the special case where our utility function rewards us for achieving a certain value for the output of the system (reaching a certain observed state), it is sometimes possible to pose the control problem as an inference problem [Zha98a, DDN01, BMI99]. Specifically, we set Y_{t+h} to the desired output value (and leave $Y_{t+1:t+h-1}$ hidden), and then infer the values (if any) for $U_{t+1}, \ldots, U_{t+h}$ which will achieve this, where h is our guess about how long it will take to achieve the goal.

(We can use dynamic programming to efficiently search over h.) The cost function gets converted into a prior on the control/input variable. For example, if the prior over U_t is Gaussian, $\mathcal{N}(U_t; 0, \Sigma)$, $U_t \sim \mathcal{N}(0, \Sigma)$, then the mode of the posterior $P(U_{t+1:t+h}|y_{1:t}, y_{t+h}, u_{1:t})$ will correspond to a sequence of minimal controls (minimal in the sense of having the smallest possible length, as measured by a Mahalanobis distance using Σ) which achieves the desired output sequence.

If U_t is discrete, inference amounts to enumerating all possible assignments to $U_{t+1:t+h}$, as in a decision tree; this is called receeding horizon control. We can (approximately) solve the infinite horizon control using similar methods so long as we discount future rewards at a suitably high rate [KMN99].

The general solution to control problems requires the use of influence diagrams (see e.g., [CDLS99, ch8], [LN01]). We will not discuss this topic further in this thesis.

Viterbi decoding

In Viterbi decoding (also called "abduction" or computing the "most probable explanation"), the goal is to compute the most likely sequence of hidden states given the data:

$$x^*_{1:t} = \arg\max_{x_{1:t}} P(x_{1:t}|y_{1:t})$$

(In the following subsection, we assume that the state space is discrete.)

By Bellman's principle of optimality, the most likely to path to reach state x_t consists of the most likely path to *some* state at time $t-1$, followed by a transition to x_t. Hence we can compute the overall most likely path as follows. In the forwards pass, we compute

$$\delta_t(j) = P(y_t|X_t = j) \max_i P(X_t = j|X_{t-1} = i)\delta_{t-1}(i)$$

where

$$\delta_t(j) \stackrel{\text{def}}{=} \max_{x_{1:t-1}} P(X_{1:t} = x_{1:t-1}, X_t = j|y_{1:t}).$$

This is the same as the forwards pass of filtering, except we replace sum with max (see Section B.2). In addition, we keep track of the identity of the most likely predecessor to each state:

$$\psi_t(j) = \arg\max_i P(X_t = j|X_{t-1} = i)\delta_{t-1}(i)$$

In the backwards pass, we can compute the identity of the most likely path recursively as follows:

$$x^*_t = \psi_{t+1}(x^*_{t+1})$$

Note that this is different than finding the most likely (marginal) state at time t.

One application of Viterbi is in speech recognition. Here, X_t typically represents a phoneme or syllable, and Y_t typically represents a feature vector derived from the acoustic signal [Jel97]. $x^*_{1:t}$ is the most likely

hypothesis about what was just said. We can compute the N best hypotheses in a similar manner [Nil98, NG01].

Another application of Viterbi is in biosequence analysis, where we are interested in offline analysis of a fixed-length sequence, $y_{1:T}$. Y_t usually represents the DNA base-pair or amino acid at location t in the string. X_t often represents whether Y_t was generated by substitution, insertion or deletion compared to some putative canonical family sequence. As in speech recognition, we might be interested in finding the most likely "parse" or interpretation of the data, so that we can align the observed sequence to the family model.

Classification

The likelihood of a model, M, is $P(y_{1:t}|M)$, and can be computed by multiplying together all the normalizing constants that arose in filtering:

$$P(y_{1:t}) = P(y_1)P(y_2|y_1)P(y_3|y_{1:2}) \dots P(y_T|y_{1:T-1}) = \prod_{t=1}^{T} c_t \tag{1.1}$$

which follows from the chain rule of probability. This can be used to classify a sequence as follows:

$$C^*(y_{1:T}) = \arg\max_C P(y_{1:T}|C)P(C)$$

where $P(y_{1:T}|C)$ is the likelihood according to the model for class C, and $P(C)$ is the prior for class C. This method has the advantage of being able to handle sequences of variable-length. By contrast, most classifiers work with fixed-sized feature vectors.[4]

Summary

We summarize the various inference problems in Figure 1.1. In Section 3.7, we will show how all of the above algorithms can be formulated in terms of a set of abstract operators which we will call forwards and backwards operators. There are many possible implementations of these operators which make different tradeoffs between accuracy, speed, generality, etc. In Sections 1.2 and 1.3, we will see how to implement these operators for HMMs and KFMs. In later chapters, we will see different implementations of these operators for general DBNs.

1.1.3 Learning

A state-space model usually has some free parameters θ which are used to define the transition model, $P(X_t|X_{t-1})$, and the observation model, $P(Y_t|X_t)$. Learning means estimating these parameters from data; this is often called system identification.

[4] One could pretend that successive observations in the sequence are iid, and then apply a naive Bayes classifier: $P(y_{1:T}|C) = \prod_{t=1}^{T} P(y_t|C)$. However, often it matters in what order the observations arrive, e.g., in classifying a string of letters as a word. There has been some work on applying support vector machines to variable length sequences [JH99], but this uses an HMM as a subroutine.

The usual criterion is maximum-likelihood (ML), which is suitable if we are doing off-line learning with large data sets. Suppose, as is typical in speech recognition and bio-sequence analysis, that we have N_{train} iid sequences, $Y = (y_{1:T}^1, \ldots y_{1:T}^{N_{\text{train}}})$, where we have assumed each sequence has the same length T for notational simplicity. Then the goal of learning is to compute

$$\theta_{ML}^* = \arg\max_\theta P(Y|\theta) = \arg\max_\theta \log P(Y|\theta)$$

where the log-likelihood of the training set is

$$\log P(Y|\theta) = \log \prod_{m=1}^{N_{\text{train}}} P(y_{1:T}^m|\theta) = \sum_{m=1}^{N_{\text{train}}} \log P(y_{1:T}^m|\theta)$$

A minor variation is to include a prior on the parameters and compute the MAP (maximum a posteriori) solution

$$\theta_{MAP}^* = \arg\max_\theta \log P(Y|\theta) + \log P(\theta)$$

This can be useful when the number of free parameters is much larger than the size of the dataset (the prior acting like a regularizer to prevent overfitting), and for online learning (where at timestep t the dataset only has size t).

What makes learning state-space models difficult is that some of the variables are hidden. This means that the likelihood surface is multi-modal, making it difficult to find the globally optimal parameter value.[5] Hence most learning methods just attempt to find a locally optimal solution.

The two standard techniques for ML/MAP parameter learning are gradient ascent[6], and EM (expectation maximization), both of which are explained in Appendix C. Note that both methods use inference as a subroutine, and hence efficient inference is a prerequisite for efficient learning. In particular, for offline learning, we need need to perform fixed-interval smoothing (i.e., computing $P(X_t|y_{1:T}, \theta)$ for all t): learning with filtering may fail to converge correctly. To see why, consider learning to solve murders: hindsight is always required to infer what happened at the murder scene.[7]

For online learning, we can use fixed-lag smoothing combined with online gradient ascent or online EM. Alternatively, we can adopt a Bayesian approach and treat the parameters as random variables, and just add them to the state-space. Then learning just amounts to filtering (i.e., sequential Bayesian updating) in the augmented model, $P(X_t, \theta_t|y_{1:t})$. Unfortunately, inference in such models is often very difficult, as we will see.

A much more ambitious task than parameter learning is to learn the structure (parametric form) of the model. We will discuss this in Chapter 6.

[5] One trivial source of multi-modality has to do with symmetries in the hidden state-space. Often we can permute the labels of the hidden states without affecting the likelihood.

[6] Ascent, rather than descent, since we are trying to maximize likelihood.

[7] This example is from [RN02].

1.2 Hidden Markov Models (HMMs)

We now give a brief introduction to HMMs.[8] The main purpose of this section is to introduce notation and concepts in a simple and (hopefully) familiar context; these will be generalized to the DBN case later.

1.2.1 Representation

An HMM is a stochastic finite automaton, where each state generates (emits) an observation. We will use X_t to denote the hidden state and Y_t to denote the observation. If there are K possible states, then $X_t \in \{1, \ldots, K\}$. Y_t might be a discrete symbol, $Y_t \in \{1, \ldots, L\}$, or a feature-vector, $Y_t \in \mathbb{R}^L$.

The parameters of the model are the initial state distribution, $\pi(i) = P(X_1 = i)$, the transition model, $A(i,j) = P(X_t = j | X_{t-1} = i)$, and the observation model $P(Y_t|X_t)$.

$\pi(\cdot)$ represents a multinomial distribution. The transition model is usually characterized by a conditional mulitnomial distribution: $A(i,j) = P(X_t = j|X_{t-1} = i)$, where A is a stochastic matrix (each row sums to one). The transition matrix A if often sparse; the structure of the matrix is often depicted graphically, as in Figure 1.2 which depicts a left-to-right transition matrix. (This means low numbered states can only make transitions to higher numbered states or to themselves.) Such graphs should not be confused with the graphical models we will introduce in Chapter 2.

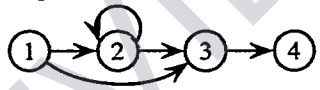

Figure 1.2: A left-to-right state transition diagram for a 4-state HMM. Nodes represent states, and arrows represent allowable transitions, i.e., transitions with non-zero probability. The self-loop on state 2 means $P(X_t = 2|X_{t-1} = 2) = A(2,2) > 0$.

If the observations are discrete symbols, we can represent the observation model as a matrix: $B(i,k) = P(Y_t = k|X_t = i)$. If the observations are vectors in $\mathbb{R}^L$, it is common to represent $P(Y_t|X_t)$ as a Gaussian:

$$P(Y_t = y|X_t = i) = \mathcal{N}(y; \mu_i, \Sigma_i)$$

where $\mathcal{N}(y; \mu, \Sigma)$ is the Gaussian density with mean μ and covariance Σ evaluated at y:

$$\mathcal{N}(y; \mu, \Sigma) = \frac{1}{(2\pi)^{L/2}|\Sigma|^{\frac{1}{2}}} \exp\left(-\tfrac{1}{2}(y-\mu)'\Sigma^{-1}(y-\mu)\right)$$

A more flexible representation is a mixture of M Gaussians:

$$P(Y_t = y|X_t = i) = \sum_{m=1}^{M} P(M_t = m|X_t = i)\mathcal{N}(y; \mu_{m,i}, \Sigma_{m,i})$$

where M_t is a hidden variable that specifies which mixture component to use, and $P(M_t = m|X_t = i) = C(i,m)$ is the conditional prior weight of each mixture component. For example, one mixture component

[8] See [Rab89] for an excellent tutorial, and [Ben99] for a review of more recent developments; [MZ97] provides a thorough mathematical treatise on HMMs. The book [DEKM98] provides an excellent introduction to the application of HMMs to biosequence analysis, and the book [Jel97] describes how HMMs are used in speech recognition.