Learning the Structure of Dynamic Probabilistic Networks

Nir Friedman **Kevin Murphy** **Stuart Russell**
Computer Science Division, U. of California, Berkeley, CA 94720
{nir,murphyk,russell}@cs.berkeley.edu

Abstract

Dynamic probabilistic networks are a compact representation of complex stochastic processes. In this paper we examine how to learn the *structure* of a DPN from data. We extend structure scoring rules for standard probabilistic networks to the dynamic case, and show how to search for structure when some of the variables are hidden. Finally, we examine two applications where such a technology might be useful: predicting and classifying dynamic behaviors, and learning causal orderings in biological processes. We provide empirical results that demonstrate the applicability of our methods in both domains.

1 Introduction

Probabilistic networks (PNs), also known as Bayesian networks or belief networks, are already well-established as representations of domains involving uncertain relations among several random variables. Somewhat less well-established, but perhaps of equal importance, are *dynamic* probabilistic networks (DPNs), which model the stochastic evolution of a set of random variables over time [4]. DPNs have significant advantages over competing representations such as Kalman filters, which handle only unimodal posterior distributions and linear models, and hidden Markov models (HMMs), whose parametrization grows exponentially with the number of state variables. For example, [25] show that DPNs can outperform HMMs on standard speech recognition tasks.

PNs and DPNs are defined by a graphical structure and a set of parameters, which together specify a joint distribution over the random variables. Algorithms for learning the parameters of PNs [16, 1] and DPNs [1, 10] are becoming widely used. These algorithms typically use either gradient methods or EM, and can handle hidden variables and missing values.

Algorithms for learning the graphical structure, on the other hand, have until recently been restricted to networks with *complete data*, i.e., where the values of all variables are specified in each training case [3, 11]. Friedman [9] has developed the *Structural EM* (SEM) algorithm for learning PN structure from data with hidden variables and missing values. SEM combines structural and parametric modification within a single EM process, and appears to be substantially more effective than previous approaches based on parametric EM operating within an outer-loop structural search. SEM can be shown to find local optima defined by a scoring function that combines the likelihood of the data with a structural penalty that discourages overly complex networks. This property holds for both the *Bayesian Information Criterion* (BIC) score [22], a variant of Minimum Description Length (MDL) scoring, and the BDe score [11], a Bayesian metric that uses an explicit prior over networks.

In this paper, we extend the BIC and BDe scores to handle the problem of learning DPN structure from complete data. (In the complete version of this paper, we also deal with the BDe score.) More importantly, we extend the SEM algorithm to learn DPNs from incomplete data with BIC scoring. Given partial observations of a set of random variables over time, the algorithm constructs a DPN (possibly including additional hidden variables) that fits the observations as well as possible. The addition of hidden variables in DPNs is particularly important as many processes—human decision making, speech generation, disease processes, for example—are only partially observable.

We begin with formal definitions of PNs and DPNs. Section 3 discusses the complete-data case, developing scores for DPNs so that existing algorithms for learning PN structures can be applied directly. In Section 4, we handle the case of incomplete data and show how to extend SEM to DPNs. Computation of the necessary sufficient statistics is highlighted as a principal bottleneck. Section 5 describes two applications: learning simple models of human driving behavior from videotapes, and learning models of biological processes from very sparse observations.

2 Preliminaries

We start with a short review of probabilistic networks and dynamic probabilistic networks.

We will be concerned with distributions over sets of discrete random variables, where each variable X_i may take on values from a finite set, denoted by $Val(X_i)$. We denote the size of $Val(X_i)$ by $||X_i||$. We use capital letters, such as X, Y, Z, for variable names and lowercase letters x, y, z to denote specific values taken by those variables. Sets of variables are denoted by boldface capital letters $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, with sets of values denoted by boldface lowercase letters $\mathbf{x}, \mathbf{y}, \mathbf{z}$. For a given network, we will use $\mathbf{X} = \{X_1, \ldots, X_n\}$ to denote the variables of the network in topological order, and P

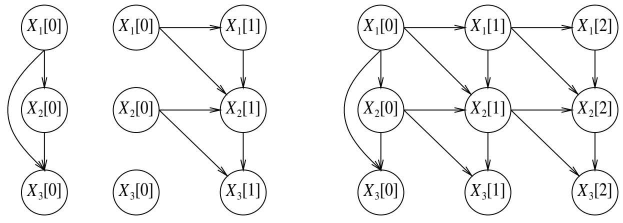

Figure 1: (a) A prior network and transition network defining a DPN for the attributes X_1, X_2, X_3. (b) The corresponding "unrolled" network.

to denote a joint probability distribution over the variables in $\mathbf{X}$

A *probabilistic network* (PN) is an annotated directed acyclic graph that encodes a joint probability distribution over $\mathbf{X}$. Formally, a PN for $\mathbf{X}$ is a pair $B = (G, \Theta)$. The first component, G, is a directed acyclic graph whose vertices correspond to the random variables $X_1, \ldots, X_n$ that encodes the following set of conditional independence assumptions: each variable X_i is independent of its non-descendants given its parents $\mathbf{Pa}(X_i)$ in G. The second component, Θ, represents the set of parameters that quantifies the network. In the simplest case, It contains a parameter $\theta_{x_i|\mathbf{pa}(X_i)} = P(x_i|\mathbf{pa}(X_i))$ for each possible value x_i of X_i and each possible set of values $\mathbf{pa}(X_i)$ of $\mathbf{Pa}(X_i)$. (Representations of the conditional probability distribution which require fewer parameters, such as noisy-ORs or trees, are also possible; indeed, we use them in the experimental results section.) Given G and Θ, a PN B defines a unique joint probability distribution over $\mathbf{X}$ given by:

$$P_B(x_1, \ldots, x_n) = \prod_{i=1}^{n} P_B(x_i|\mathbf{pa}(X_i))$$

A PN describes a probability distribution over a fixed set of variables. DPNs extend this representation to model temporal processes. For simplicity, we assume that changes occur between discrete time points that are indexed by the non-negative integers. We assume that $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a set of attributes that the process changes. $X_i[t]$ is the random variable that denotes the value of the attribute X_i at time t, and $\mathbf{X}[t]$ is the set of random variables $X_i[t]$.

To represent beliefs about the possible trajectories of the process, we need a probability distribution over the random variables $\mathbf{X}[0] \cup \mathbf{X}[1] \cup \mathbf{X}[2] \cup \ldots$. Of course, such a distribution can be extremely complex. In this paper, we assume the process is *Markovian* in $\mathbf{X}$, i.e., $P(\mathbf{X}[t+1] \mid \mathbf{X}[0], \ldots, \mathbf{X}[t]) = P(\mathbf{X}[t+1] \mid \mathbf{X}[t])$. We also assume that the process is *stationary*, i.e., that the *transition probability* $P(\mathbf{X}[t+1] \mid \mathbf{X}[t])$ is a constant independent of t.

Given these assumptions, a DPN that represents the joint distribution over all possible trajectories of a process consists of two parts:

- a prior network B_0 that specifies a distribution over initial states $\mathbf{X}[0]$; and
- a transition network $B_{\rightarrow}$ over the variables $\mathbf{X}[0] \cup \mathbf{X}[1]$ that is taken to specify the the transition probability $P(\mathbf{X}[t+1] \mid \mathbf{X}[t])$ for all t.

Figure 1(a) shows a simple example.[1] In the transition network (but not the prior network), the variables in $\mathbf{X}[0]$ have no parents. The transition probability implied by such a network is:

$$P_{B_{\rightarrow}}(\mathbf{x}[1] \mid \mathbf{x}[0]) = \prod_{i=1}^{n} P_{B_{\rightarrow}}(x_i[1]|\mathbf{pa}(X_i[1])).$$

A DPN defined by a pair $(B_0, B_{\rightarrow})$ corresponds to a semi-infinite network over the variables $\mathbf{X}[0], \ldots, \mathbf{X}[\infty]$. In practice, we reason only about a finite interval $0, \ldots, T$. To do this, we can notionally "unroll" the DPN structure into a PN over $\mathbf{X}[0], \ldots, \mathbf{X}[T]$. In slice 0, the parents of $X_i[0]$ are those specified in the prior network B_0; in slice $t+1$, the parents of $X_i[t+1]$ are those nodes in slice t corresponding to the parents of $X_i[1]$ in $B_{\rightarrow}$. We copy the conditional distributions for these variables in a similar manner. Figure 1(b) shows the result of unrolling the network in Figure 1(a) for 3 time slices. Given a DPN model, the joint distribution over $\mathbf{X}[0], \ldots, \mathbf{X}[T]$ is

$$P_B(\mathbf{x}[0], \ldots, \mathbf{x}[T]) = P_{B_0}(\mathbf{x}[0]) \prod_{t=0}^{T-1} P_{B_{\rightarrow}}(\mathbf{x}[t+1] \mid \mathbf{x}[t]) \quad (1)$$

where $P_{B_{\rightarrow}}(\mathbf{x}[t+1] \mid \mathbf{x}[t])$ is obtained in the obvious way from the transition model.

3 Learning from Complete Data

In this section we develop the theory for learning DPNs from *complete data*. We begin with a brief review of how one learns standard PNs. Then we derive the details of the BIC and BDe score for DPNs and finally we discuss how these are optimized in the search for good structures. The upshot of this section is that learning DPNs from complete data uses the same techniques as learning PNs from complete data. Learning DPNs is not quite the same as applying PN methods to the unrolled network, however, due to the constraint of repeated structure and repeated parameters.

3.1 Learning PNs

The problem of learning a probabilistic network is stated as follows. Given a *training set* D of instances of $\mathbf{X}$, find a network $B = (G, \Theta)$ that *best matches* D. The notion of "best match" is defined using a scoring function. Several different scoring functions have been proposed in the literature. The most frequently used are the *Bayesian Information Criterion* [22] and the BDe score [11]. Both of these scores combine the *likelihood* of the data according to the network, $L(B : D) = \log P(D \mid B)$, with some penalty

[1] Some authors define DPNs using just the transition network, assuming that all "slices," including slice 0, have the same structure. In general, however, the prior distribution on $\mathbf{X}[0]$ can have a quite different independence structure from other slices, since it may represent either the way in which the process is initialized or the dependencies induced among the variables in $\mathbf{X}[0]$ by the unobserved portion of the process before $t = 0$. For example, if $t = 0$ represents an arbitrary starting point taken from an infinite process, then it reasonable to use a dependency structure for $\mathbf{X}[0]$ that reflects the stationary distribution of the Markov chain defined by the transition network. One can combine the prior and transition networks into a single two-slice network, but, because the learning processes for the prior and transition models are distinct, we have chosen to represent them separately.

relating to the complexity of the network. When learning the structure of PNs, a complexity penalty is essential since the maximum-likelihood network is usually the completely connected network.

The BIC and BDe scores are derived from the posterior probability of the network structure. Let the random variable G range over the possible network structures that might in fact obtain in the real world. Then, using Bayes' rule, the posterior distribution over G is

$$\Pr(G \mid D) \propto \Pr(D \mid G) \Pr(G)$$

where the likelihood of the data given a network structure can be computed by conditioning on the associated network parameters:

$$\Pr(D \mid G) = \int \Pr(D \mid G, \Theta) \Pr(\Theta \mid G) d\Theta. \qquad (2)$$

Obviously, specifying the parameter priors and evaluating this integral can be difficult.

One approach to avoiding full computation of the integral in (2) is to examine the asymptotic behavior of this term. Given a large number of data points, the posterior probability is insensitive to the choice of prior (assuming that the prior does not give probability zero to any event). [22] derives the following asymptotic estimate for well-behaved priors:

$$\log \Pr(D \mid G) = \log \Pr(D \mid G, \hat{\Theta_G}) - \frac{\log N}{2} \#G + O(1), \qquad (3)$$

where $\hat{\Theta_G}$ are the parameter settings for G that maximize the likelihood of the data, N is the number of training instances, and $\#G$ is the *dimension* of G, which in the case of complete data is just the number of parameters. The BIC score uses Equation (3) to rank candidate network structures. Notice that it obviates the need for parameter priors, and the prior on structures is reduced to counting parameters.[2]

An alternative approach is to evaluate (2) in closed form given a restricted family of priors [2, 3, 11]. Roughly speaking, the prior on the parameters for a given structure G is assumed to factor into independent priors over the parameters for each conditional distribution $P(X_i \mid \mathbf{pa}(X_i))$. When the data is complete, this implies that the posterior factors in the same way. Thus, each conditional distribution can be updated and scored separately. The score can be computed in closed form if we assume in addition that the prior for each conditional distribution is from a conjugate family, such as the Dirichlet distribution. The details appear below.

Since we might examine a large number of possible network structures, we would like to avoid having to assign a prior distribution over parameters for each possible structure. [11] provide a set of assumptions that allow the parameter prior for all structures to be specified using a single network with Dirichlet priors, together with a single "virtual data count" that describes the confidence in that prior. This approach has the desirable property that the scores of two networks that are equivalent (i.e., describe the same set of independence assumptions) are the same. Finally, to compute the full BDe score, a simple description-length penalty corresponding to $\Pr(G)$ is added in.

[2]A similar formula arises from the *Minimum Description Length* (MDL) principle [15].

3.2 BIC Score for DPNs

We now describe the BIC score for DPNs. Unsurprisingly, the results here mirror the results for PNs.

Throughout this section, we assume that we are given a training set D consisting of N_{seq} complete observation sequences. The ℓth such sequence has length N_ℓ and specifies values for the variables $\mathbf{x}^\ell[0], \ldots, \mathbf{x}^\ell[N_i]$. Such a dataset give us N_{seq} instances of initial slices, from which we can train B_0, and $N = \sum_i N_i$ instances of transitions, from which we can train $B_\rightarrow$.

We start by examining properties of the likelihood function for the DPNs. To simplify notation, let us define

$$\theta^{(0)}_{i,j'_i,k'_i} = \Pr(X_i[0] = k'_i | \mathbf{Pa}(X_i[0]) = j'_i)$$

where j'_i indexes the set of possible values that $\mathbf{Pa}(X_i[0])$ can take on, and k'_i indexes the set of possible values that $X_i[0]$ can take on. Similarly, we define, for $t = 1, \ldots, T$,

$$\theta^{\rightarrow}_{i,j_i,k_i} = \Pr(X_t[0] = k_i | \mathbf{Pa}(X_i[t]) = j_i)$$

We will also need some notation for the sufficient statistics,

$$N^{(0)}_{i,j'_i,k'_i} = \sum_\ell I(X_i[0] = k'_i, \mathbf{Pa}(X_i[0]) = j'_i; \mathbf{x}^\ell)$$

and

$$N^{\rightarrow}_{i,j_i,k_i} = \sum_\ell \sum_t I(X_i[t] = k_i, \mathbf{Pa}(X_i[t]) = j_i; \mathbf{x}^\ell)$$

where $I(\cdot; \mathbf{x}^\ell)$ is an indicator function which takes on value 1 if the event $\cdot$ occurs in sequence $\mathbf{x}^\ell$, and 0 otherwise.

Using (1), and rearranging terms, we find that the likelihood function *decomposes* according to the structure of the DPN, just as with PNs:

$$\begin{aligned} \Pr(D|G, \Theta_G) &= \prod_i \prod_{j'_i} \prod_{k'_i} \theta^{(0)}_{i,j'_i,k'_i}{}^{N^{(0)}_{i,j'_i,k'_i}} \times \\ & \prod_i \prod_{j_i} \prod_{k_i} \theta^{\rightarrow}_{i,j_i,k_i}{}^{N^{\rightarrow}_{i,j_i,k_i}} \end{aligned} \qquad (4)$$

and hence the log-likelihood is given by

$$\begin{aligned} L(B : D) &= \sum_i \sum_{j'_i} \sum_{k'_i} N^{(0)}_{i,j'_i,k'_i} \log \theta^{(0)}_{i,j'_i,k'_i} + \\ & \sum_i \sum_{j_i} \sum_{k_i} N^{\rightarrow}_{i,j_i,k_i} \log \theta^{\rightarrow}_{i,j_i,k_i} \end{aligned}$$

This decomposition facilitates the computation of the BIC and BDe scores in several ways. First, note that the likelihood is expressed as a sum of terms, where each term depends only on the conditional probability of a variable given a particular assignment to its parents. Thus, if we want to find the maximum likelihood parameters, we can maximize within each family independently. Second, the decomposition implies that we can learn B_0 independently of $B_\rightarrow$. Finally, we can learn $B_\rightarrow$ in exactly the same manner as learning a PN for a set of samples of transitions.

We now make these arguments more precise. Using the standard maximum likelihood estimate for multinomial distributions, we immediately get the following expression for

$\hat{\Theta_G}$.

$$\theta^{(0)}_{i,j'_i,k'_i} = \frac{N^{(0)}_{i,j'_i,k'_i}}{\sum_{k'_i} N^{(0)}_{i,j'_i,k'_i}}$$

and

$$\theta^{\rightarrow}_{i,j_i,k_i} = \frac{N^{\rightarrow}_{i,j_i,k_i}}{\sum_{k_i} N^{\rightarrow}_{i,j_i,k_i}}$$

Thus, computing these counts gives the BIC score directly, in the following form:

$$\begin{aligned} BIC(G : D) &= \textstyle\sum_i BIC_0(X_i[0], \mathbf{Pa}(X_i[0]) : D) + \\ &\quad \textstyle\sum_i BIC_{\rightarrow}(X_i[1], \mathbf{Pa}(X_i[1]) : D), \quad (5) \end{aligned}$$

where

$$\begin{aligned} BIC_0(X, \mathbf{Y} : D) &= \textstyle\sum_{x,\mathbf{y}} N_0(x, \mathbf{y}) \log \frac{N_0(x,\mathbf{y})}{N_0(\mathbf{y})} - \\ &\quad \tfrac{\log N_{\text{seq}}}{2} ||Y||(||X|| - 1) \end{aligned}$$

and

$$\begin{aligned} BIC_{\rightarrow}(X, \mathbf{Y} : D) &= \textstyle\sum_{x,\mathbf{y}} N_{\rightarrow}(x, \mathbf{y}) \log \frac{N_{\rightarrow}(x,\mathbf{y})}{N_{\rightarrow}(\mathbf{y})} - \\ &\quad \tfrac{\log N}{2} ||Y||(||X|| - 1) \end{aligned}$$

Note that the penalty for families in the original time slice is a function of the number of sequences N_{seq}, since we only observe that many examples for this part of the model; whereas the penalty for the transition model is a function of N, the total number of transitions observed.

3.3 BDe Score for DPNs

Recall that to compute the BDe score, we need to evaluate the following integral.

$$Pr(D \mid G) = \int \Pr(D \mid G, \Theta_G) \Pr(\Theta_G \mid G) d\Theta_G$$

The first term inside the integral decomposes as in Equation 4. If we assume that the prior over each conditional probability is independent of the rest, then the prior term also decomposes as

$$\Pr(\Theta_G \mid G) = \prod_i \prod_{j'_i} \Pr(\theta^{(0)}_{i,j'_i,k'_i}) \times \prod_i \prod_{j_i} \Pr(\theta^{\rightarrow}_{i,j_i,k_i})$$

Inserting this into the preceding equation, we see that the entire expression consists of an integral over the product of independent terms. Hence $\Pr(D \mid G)$ can be written as a product of two integrals,

$$\prod_i \prod_{j'_i} \int \prod_{k'_i} \theta^{(0)}_{i,j'_i,k'_i}{}^{N^{(0)}_{i,j'_i,k'_i}} \times \Pr(\theta^{(0)}_{i,j'_i,k'_i}) \times d\theta^{(0)}_{i,j'_i,k'_i}$$

and

$$\prod_i \prod_{j_i} \int \prod_{k_i} \theta^{\rightarrow}_{i,j_i,k_i}{}^{N^{\rightarrow}_{i,j_i,k_i}} \times \Pr(\theta^{\rightarrow}_{i,j_i,k_i}) \times d\theta^{\rightarrow}_{i,j_i,k_i}$$

In order to obtain a closed-form solution, we assume *Dirichlet* priors [5]. A Dirichlet prior for a multinomial distribution of a variable X is specified by a set of *hyperparameters* $\{\hat{N}_x : x \in Val(X)\}$ as follows:

$$\Pr(\Theta_X) = Dirichlet(\{\hat{N}_x : x \in Val(X)\}) = \alpha \prod_x \theta_x^{\hat{N}_x}$$

where α is a normalization constant. Intuitively, the hyperparameters can be thought of as "pseudo counts", since they play a similar role to the actual counts we derive from the data. Under a Dirichlet prior, the probability of observing a sequence of values of X with counts $N(x)$ is

$$\int \prod_x \theta_x^{N(x)} \Pr(\Theta_X \mid G) d\Theta_X = \frac{\Gamma(\sum_x \hat{N}_x)}{\Gamma(\sum_x(\hat{N}_x + N(x)))} \times \prod_x \frac{\Gamma(\hat{N}_x + N(x))}{\Gamma(\hat{N}_x)},$$

where $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ is the *Gamma* function that satisfies the properties $\Gamma(1) = 1$ and $\Gamma(x + 1) = x\Gamma(x)$.

Returning to the BDe score, let us assume that for each structure G, we have chosen the hyperparameters $\hat{N}^{(0)}_{i,j'_i,k'_i}$ and $\hat{N}^{\rightarrow}_{i,j_i,k_i}$. Then we can rewrite $\Pr(D \mid G)$ as

$$\begin{aligned} &\textstyle\prod_i \prod_{j'_i} \frac{\Gamma(\sum_{k'_i} \hat{N}^{(0)}_{i,j'_i,k'_i})}{\Gamma(\sum_{k'_i} \hat{N}^{(0)}_{i,j'_i,k'_i} + \hat{N}^{\rightarrow}_{i,j_i,k_i})} \times \\ &\textstyle\prod_{k'_i} \frac{\Gamma(\hat{N}^{(0)}_{i,j'_i,k'_i} + N^{(0)}_{i,j'_i,k'_i})}{\Gamma(\hat{N}^{(0)}_{i,j'_i,k'_i})} \times \\ &\textstyle\prod_i \prod_{j_i} \frac{\Gamma(\sum_{k_i} \hat{N}^{\rightarrow}_{i,j_i,k_i})}{\Gamma(\sum_{k_i} \hat{N}^{\rightarrow}_{i,j_i,k_i} + N^{\rightarrow}_{i,j_i,k_i})} \times \\ &\textstyle\prod_{k_i} \frac{\Gamma(\hat{N}^{\rightarrow}_{i,j_i,k_i} + N^{\rightarrow}_{i,j_i,k_i})}{\Gamma(\hat{N}^{\rightarrow}_{i,j_i,k_i})} \end{aligned}$$

This still requires us to supply the Dirichlet hyperparameters for each candidate DPN structure. Since the number of possible DPN structures is large, these prior estimates might be hard to asses in practice. Following [11], we can assign all of these given a *prior* DPN $B' = (B'_0, B'_{\rightarrow})$ and two *equivalent sample sizes* $\hat{N}_0$ and $\hat{N}_{\rightarrow}$. Given these components, we assign the Dirichlet weights for $G = (G_0, G_{\rightarrow})$ as follows:

$$\begin{aligned} \hat{N}^{(0)}_{i,j'_i,k'_i} &= \hat{N}_0 * \theta^{(0)}_{i,j'_i,k'_i} \\ \hat{N}^{\rightarrow}_{i,j_i,k_i} &= \hat{N}_{\rightarrow} * \theta^{\rightarrow}_{i,j_i,k_i} \end{aligned}$$

(Note that the choice of parents here is based on G, and might differ from the structure of B'.) Intuitively, we can consider the belief in B' as equivalent to having previously experienced $\hat{N}_0$ sequences with $\hat{N}_{\rightarrow}$ transitions.

It is easy to verify that choosing priors in this manner preserves the main property we require: two DPN structures that imply identical independence assumptions will receive the same score. Thus, we claim that our definition is the natural extension of the BDe score to the case of dynamic probabilistic networks.

3.4 Learning in Practice

The BIC score has two important properties. First, the score of a DPN can be written as a sum of terms, where each term determines the score of a particular choice of parents for a particular variable. Thus, a local change to one family (e.g., arc addition or removal) affects only one of these terms. Second, the term that evaluates $X_i[t]$ given its parents is a function of the appropriate counts (either $N_0(\cdot)$ or $N_\rightarrow(\cdot)$) for that family. Thus, by caching these counts we can efficiently evaluate many families.

These two properties can be exploited by hill-climbing search procedures [2, 11] that gradually improve a candidate structure by applying the best arc addition, deletion, or reversal. In the case of DPNs, as opposed to static PNs, we have the additional constraint that the network structure must repeat over time. This reduces the number of options at each point in the search. Moreover, we can search for the best structure for B_0 independently of the search for the best structure for $B_\rightarrow$.

4 Learning from Incomplete Data

We now examine how to learn DPNs from incomplete data. Incomplete data is crucial since in most real life application we do not have complete observability of the process we want to model. This means that even if the process is a stationary Markov process, the partial observations we have are *not* Markovian. As an example, suppose we are tracking a car moving down the highway and we are observing the car's speed, lane, relative distance to other cars, and so on. A Markovian model might be reasonable, on the other hand, if we include as part of the state information the driver's goals—e.g., get to the left lane, exit at the next off ramp, etc. By learning hidden variables, we can capture state information about the process. This allows the model to "remember" additional information about the past and to make better predictions of the future.

The main difficulty with learning from partial observations is that we no longer have the score decomposition properties of Proposition **??**. This means that the optimal parameter choice in one part of the network depends on the parameter choices in other parts of the network. This problem can be understood better if we notice that once we have partial observability we can no longer talk about counts from the training data. For most events of interest, the counts are not defined, since we do not know the exact value of the variables in questions.

The most commonly used method to alleviate this problem is the *Expectation-Maximization* (EM) algorithm [6, 16]. The E-step of EM uses the currently estimated parameters to *complete* the data by computing the *expected* counts. The M-step then re-estimates the maximum-likelihood parameter values as if the expected counts were true observed counts The central theorem underlying EM's behavior is that each EM cycle is guaranteed to improve the likelihood of the data given the model until it reaches a local minimum.

EM has been traditionally viewed as method for adjusting the parameters of a fixed model structure. However, the underlying theorem can be generalized to apply to structural as well as parametric modifications. [9]'s *Structural EM* (SEM) algorithm has the same E-step as EM, completing the data by computing expected counts based on the current structure and parameters. In addition to re-estimating parameters, the M-step of SEM can use the expected counts according to the current structure to evaluate any other candidate structure—essentially performing a complete-data structural search in the inner loop. Friedman shows that for a large family of scorings rules, including the BIC score, the resulting network must have a higher score than the original. This is true even though the expected counts used in evaluating the new structure are computed using the old structure.

In theory, applying SEM to learning DPNs is relatively straightforward. The key step is to use an existing DPN $(B_0, B_\rightarrow)$ to complete the data. Thus, we run the same search procedure as with complete data, but whenever the search procedure requires $N_\rightarrow(\mathbf{x})$ to evaluate the score of a network, we use instead the *expected value* $E[N_\rightarrow(\mathbf{x}) \mid D, B_0, B_\rightarrow] = \sum_{D^+} \Pr(D^+ \mid D, B_0, B_\rightarrow) N_\rightarrow(\mathbf{x} : D^+)$ where D^+ represents a *completion* of the data. In other words, we consider all possible trajectories (i.e., joint assignments to all the variables in the training sequences) that are consistent with the partial information we have, and average the counts in each one of these, based on the probability that our current model assigns to that trajectory. In a similar manner we use the expected counts at the initial time slice.

We denote by $E[BIC((B'_0, B'_\rightarrow) : D^+) : D, (B_0, B_\rightarrow)]$ the BIC score of a DPN $(B'_0, B'_\rightarrow)$ based on the completion of $(B_0, B_\rightarrow)$. Using Theorem 3.1 of [9], we can prove the following:

Theorem 4.1:

$$\begin{aligned} \mathit{BIC}((B'_0, B'_\rightarrow) : D) - BIC((B_0, B_\rightarrow) : D) > \\ E[BIC((B'_0, B'_\rightarrow) : D^+) : D, (B_0, B_\rightarrow)] - \\ E[BIC((B_0, B_\rightarrow) : D^+) : D, (B_0, B_\rightarrow)] \end{aligned}$$

That is, if we choose a new DPN that has a higher score than the old DPN given the expected counts computed using the old DPN, then the *true* score of new DPN will also be higher than the *true* score of the old DPN. Moreover, the difference in the expected scores is a lower bound on the improvement in terms of the actual score we are trying to optimize.

Theorem 4.1 provides a general justification for procedure we outlined above. We now review the SEM procedure in more detail.

```
procedure DPN-SEM:
  Choose (B_0^0, B_→^0) (possibly randomly).
  Loop for n = 0, 1, ... until convergence
    Improve the parameters of (B_0^n, B_→^n) using EM
    Search over DPN structures
      (using expected counts computed with (B_0^n, B_→^n))
    Let the best scoring DPN seen be (B_0^{n+1}, B_→^{n+1})
    if (B_0^{n+1}, B_→^{n+1}) = (B_0^n, B_→^n)
      return (B_0^{n+1}, B_→^{n+1})
```

The computational bottleneck in the above procedure lies in computing the expected counts for each family in the DPN (a family is a node and its parents). Since, by assumption, a node's parents always occur in the same or previous slice, we can compute the statistics by summing up over pairs of consecutive slices. Formally, $N_\rightarrow(\mathbf{x}) = \sum_i \sum_{t=0}^{N_i-1} I(\mathbf{x}, i, t)$, where $I(\mathbf{x}, s_i, t)$ is an indicator variable that has the value 1 if the assignment in **x** holds in the time slices t and $t+1$ in the i'th sequence of the data. Using linearity of expectation, we have that

$$E[N_\rightarrow(\mathbf{y})] = \sum_i \sum_{t=0}^{N_i-1} E[I(\mathbf{y}, i, t)].$$

This shows that we do not need to sum over all possible trajectories to estimate expected counts. We only need to estimate the expected value of $I(\mathbf{y}, i, t)$, which is precisely the probability that $(\mathbf{x}^i[t], \mathbf{x}^i[t+1]) = \mathbf{y}$ given the evidence.

Thus, the key requirement is to compute the probability that a family of nodes takes on a certain set of values given the evidence, including evidence in the past and the future of the sequence. This can be done efficiently by converting the DPN to a join tree and using a two-pass dynamic programming algorithm [14], similar to the forwards-backwards algorithm used in HMMs [23].

For learning structure, we need to compute the expected counts for all the families that occur in all the models whose score we wish to evaluate. A simple approach, which is the one we currently use, is to connect together all the variables in a single timeslice into a single node (i.e., to convert the DPN to a Markov chain), compute the expected number of transitions between every pair of consecutive states in this chain, and then marginalize these counts to get the counts for each family. We also maintain a cache of the expected counts computed so far based on the current completion model, and use this cache to avoid recomputing expected counts. More efficient techniques are possible (e.g., [24]), but in general, computing exact expected sufficient statistics for all models of interest is likely to remain computationally challenging. In the discussion section we mention some approximation techniques that may allow us to learn larger models.

5 Applications

In this section we describe two preliminary investigations that attempt to evaluate the usefulness of the DPN technology we develop here for real-life applications.

5.1 Inferring driver behavior

In many tracking domains, we would like to learn a predictive model of the behavior of the object being tracked. The more accurate the model, the more robust the tracking system and the more useful its predictions will be in decision making. For objects with hidden state, it is our hope that DPNs can be learned that correctly reflect the unobserved process governing the behavior. The learned model may also provide insight into how the behavior is generated.

In this section, we describe some experiments we carried out using a simulated driving domain [8]. The data is an idealization of what cameras mounted on the side of the road can collect. In particular, at each time step of the simulation, we get a report on cars that are within the camera's range. The report for each car has the following attributes: position and velocity (relative to the camera's reference frame), relative speed and distance to the car in front, and whether there is a car immediately to the left or right. From this data, we want to learn models of typical classes of driving behavior. Such models can be useful for several tasks. A prime example arises in the BATmobile autonomous car project [7]. The BATmobile's autonomous controller attempts to predict the behavior of neighboring car. For example, it would be useful to know that someone who has just driven across two lanes might be attempting to leave the freeway, and consequently is likely to cut in front of you. Since tracking information from real cameras is readily available [18], it is reasonable to hope that realistic models of human drivers can be obtained. In addition to their use in autonomous vehicles, such models are of paramount importance in so-called "microscopic" traffic models used in freeway design and construction planning and also in safety studies.

	Number of hidden vars.		
Number of Seq.	0	1	2
200	-5.067	-4.937	-4.762
400	-2.569	-2.375	-2.352

Figure 2: Logloss (bits) per slice on test data for the networks learned in the driving domain.

In our experiments, we generate a variety of simulated traffic patterns consisting of populations of vehicles with a mixture of different driving tendencies (e.g., trucks, sports cars, Sunday drivers, etc.). We generated sets of 600 cars and tracked their behavior over sequences of 70 time steps. The observed data were discretized using fixed-sized bins. We then trained networks using a dataset consisting of first 200 and the first 400 sequences, and tested these networks on the last 200 sequences.

We learned networks with 0, 1, and 2 hidden variables. The initial network structure in each case had each hidden variable (if any) dependent on its value in the previous time slice, and each observable variable dependent only on the hidden variables in the same time slice. (Thus, we did not put in any persistence arcs in the initial model.) This structure connects each hidden variable to all the other variables, allowing that variable to carry forward information about previous time slices. The learning procedure applied random perturbation at the end of SEM runs to escape from local maxima.

Table 2 summarizes the logloss, per time slice, on the test data for the different networks we learned. We can roughly interpret the negative of the logloss as the number of bits needed to encode a time slice using each network.[3] These numbers indicate that the addition of hidden variables improves the predictive ability of the models. They also indicate that we might benefit from using many more training sequences; this is because many of the important events such as lane changes are quite rare in typical freeway traffic patterns. Some of the learned networks are shown in Figures 3. The networks seem to include at least some of the essential relationships that we expect to see. For example, in Figure 3(b), the single hidden node has parent "relative distance" and children "lane-position" and "longitudinal velocity" (but not "lateral velocity"). This suggests that it encodes "need to take avoiding action." These relationships correctly reflect the actual decision-making mechanism used by the simulated vehicles.

[3]For comparison, the `gzip` utility compresses the observations to approximately 5.6 bits per time slice.

(a) (b) (c)

Figure 3: Transition models learned from the driving domain. Shaded nodes correspond to hidden variables.

5.2 Learning Causal Pathways in Biological Processes

DPNs are a powerful representation language for describing causal models of stochastic processes. Such models are useful in many areas of science, including molecular biology, where one is often interested in inferring the structure of regulatory pathways. McAdams and Shapiro [20] model part of the genetic circuit of the lambda bacteriophage in terms of a sequential logic circuit; and attempts have even be made to automatically infer the form of such Boolean circuits from data [17]. However, it is well known that the abstraction of binary-valued signals together with deterministic switches often breaks down, and one must model the continuous nature and inherent noise in the underlying system to get accurate predictions [19]. Also, one must be able to deal with noisy and incomplete observations of the system.

We believe that DPNs provide a good tool for modelling such noisy, causal systems, and furthermore that SEM provides a good way to learn these models automatically from noisy, incomplete data. Here we describe some initial experiments using DPNs to learn small artificial examples typical of the causal processes involved in genetic regulation. We generate data from models of known structure, learn DPN models from the data in a variety of settings, and compare these with the original models. The main purpose of these experiments is to understand how well DPNs can represent such processes, how many observations are required, and what sorts of observations are most useful. We refrain from describing any *particular* biological process, since we do not yet have sufficient real data on the processes we are studying to learn a scientifically useful model.

Simple genetic systems are commonly described by a *pathway model*—a graph in which vertices represent genes (or larger chromosomal regions) and arcs represent causal pathways (Figure 4(a,b)). A vertex can either be "off/normal" (state 0) or "on/abnormal" (state 1). The system starts in a state which is all 0s, and vertices can "spontaneously" turn on (due to unmodelled external causes) with some probability per unit time. Once a vertex is turned on, it stays on, but may trigger other neighboring vertices to

Figure 4: (a) A simple pathway model with five vertices. Each vertex represents a site in the genome, and each arc is a possible triggering pathway. (b) A more complex model that includes a directed cycle. Even if vertices stay on once triggered, the link from 6 to 5 is meaningful because 6 can turn on spontaneously even when 5 is off. (c) A DPN model that is equivalent to the pathway model shown in (a).

turn on as well—again, with a certain probability per unit time. The arcs on the graph are usually annotated with the "half-life" parameter of the triggering process. Note that pathway models, unlike PNs, can contain directed cycles. For many important biological processes, the structure and parameters of this graph are completely unknown; their discovery would constitute a major advance in scientific understanding.

Pathway models have a very natural representation as DPNs: each vertex becomes a state variable, and the triggering arcs are represented as links in the transition network of the DPN. The tendency of a vertex to stay "on" once triggered is represented by persistence links in the DPN. Figure 4(c) shows a DPN representation of the five-vertex pathway model in Figure 4(a). The nature of the problem suggests that noisy-ORs (or noisy-ANDs) should provide a parsimonious representation of the conditional density function at each node. To specify a noisy-OR for a node with k parents, we use parameters $q_1, \ldots, q_k$, where q_i is the probability the child node will be in state 0 if the ith parent is in state 1. (Thus we need only k parameters instead of 2^k for a full CPT.) In the five-vertex DPN model that we used in the experiments reported below, all the q parameters (except for the persistence arcs) have value 0.2. For a strict persistence model (vertices stay on once triggered), q parameters for persistence arcs are fixed at 1. To learn such noisy-OR distributions, we follow the technique suggested in [21]; this entails introducing a new hidden node for each arc in the network, which is a noisy version of its parent, and replacing each noisy-OR gate with a deterministic-OR gate.

In all our experiments, we enforced the presence of the persistence arcs in the network structure. We used two alternative initial topologies: one that has only persistence arcs (so the system must learn to add arcs) and one that is fully interconnected (so the system must learn to delete arcs). Performance in the two cases was very similar.

We experimented with three observation regimes that correspond to realistic settings:

- The complete state of the system is observed at every time step.
- Entire time slices are hidden uniformly at random with probability h, corresponding to intermitten observation.
- Only two observations are made, one before the process begins and another at some unknown time t_{obs} after the process is initiated by some external or spontaneous event. This might be the case with some disease processes, where the DNA of a diseased cell can be observed but the elapsed time since the disease process began is not known. (The "initial" observation is of the DNA of some other, healthy cell from the same individual.)

This last case, which obtains in many realistic situations, raises a new challenge for machine learning. We resolve it as follows: we supply the network with the "diseased" observation at time slice T, where T is with high probability larger than the actual elapsed time t_{obs} since the process began.[4] We also augment the DPN model with a hidden "switch" variable S that is initially off, but can come on spontaneously. When the switch is off, the system evolves according to its normal transition model $P(\mathbf{X}[t+1] \mid \mathbf{X}[t], S=0)$, which is to be determined from the data. Once the switch turns on, however, the state of the system is frozen—that is, the conditional probability distribution $P(\mathbf{X}[t+1] \mid \mathbf{X}[t], S=1)$ is fixed so that $\mathbf{X}[t+1] = \mathbf{X}[t]$ with probability 1. The persistence parameter for S determines a probability distribution over t_{obs}; by fixing this parameter such that (a priori) $t_{\text{obs}} < T$ with high probability, we effectively fix a scale for time, which would otherwise be arbitrary. The learned network will, nonetheless, imply a more constrained distribution for t_{obs} given a pair of observations.

We consider two measures of performance. One is the number of different edges in the learned network compared to the generating network, i.e., the Hamming distance between their adjacency matrices. The second is the difference in the logloss of the learned model compared to the generating model, measured on an independent test set. Our results for the five-vertex model of Figure 4(a) are shown in Figures 5 and 5. We see that noisy-ORs perform much better than tabular CPTs when the amount of missing data is high. Even with 40% missing slices, the exact structure is learned from only 30 examples by the noisy-OR network.[5]

[4] With the q parameters set to 0.2 in the true network, the actual system state is all-1s with high probability after about $T = 20$, so this is the length used in training and testing.

[5] What is not clear from the graphs, however, is that noisy-ORs require many more EM iterations to converge, and each iteration is much slower; this is because of the extra hidden nodes we introduce.

These results are quite promising, in that some models of biological interest are not much more complex than the five-vertex process. One surprising finding was that the system seemed to be very insensitive to the starting parameters and initial structure: it always ended up learning a network with exactly the same score.

We have performed provisional experiments for the regime in which only initial and "final" observations are made. We have found that it is still possible to learn even under these adverse circumstances. Our initial results suggest that the amount of data required goes up by a factor of 10 for the five-vertex process with $T = 20$. This suggests that data availability may be a major obstacle in studies of such disease processes in humans. Obviously, the application of prior knowledge may be very important in reducing the data requirements.

6 Discussion

In this paper we addressed the question of learning the structure and parameters of DPNs from complete and incomplete data. To the best of our knowledge, we are the first to examine this problem. As our experiments show, we can learn non-trivial structures from synthetic data and "realistic" data from a nontrivial simulator.

The main bottleneck in the application of our procedures is inference. Unlike the case of PNs, a sparse DPN structure does not necessarily ensure fast inference—the minimum size of the posterior distribution for a slice is generally exponential in the number of variables that have parents in the previous slice. One proposed solution is approximation by stochastic simulation—for example, the ER/SOF algorithm [13]. Another possibility is variational approximation [12]. A third possibility is to learn DPNs represented using a multilevel abstraction hierarchy in order to reduce the overall complexity of inference.

Acknowledgments

This work was supported in part by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant number DAAH04-96-1-0341.

References

[1] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Mach. Learning*, 29:213–244, 1997.

[2] W. Buntine. Theory refinement on Bayesian networks. In *UAI-91*, pp. 52–60. 1991.

[3] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Mach. Learning*, 9:309–347, 1992.

[4] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *AAAI-88*, pp. 524–528, 1988.

[5] M. DeGroot. Optimal Statistical Decisions. 1970.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soc.*, B 39:1–39, 1977.

[7] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The BATmobile: Towards a Bayesian automated taxi. In *IJCAI-95*, 1995.

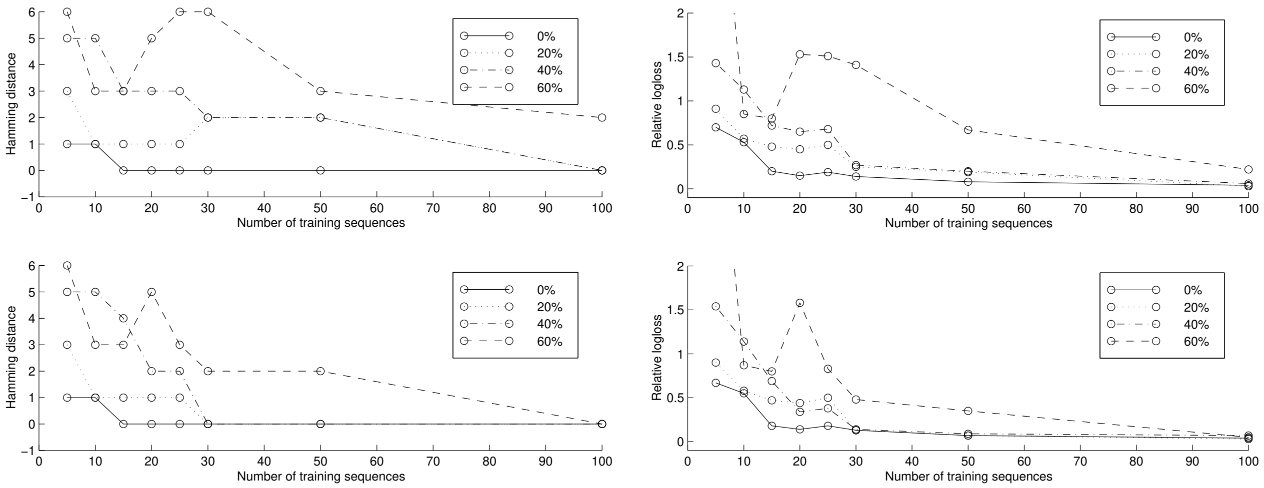

Figure 5: Hamming distance (left) of the learned models compared to the generating model for the five-vertex process, and relative log-loss (right) compared with the generating model on independent sample of 100 sequences. Tests are for regimes in which 0%, 20%, 40%, and 60% of the slices are hidden at random. Top: tabulated CPTs; bottom: noisy-OR network.

[8] J. Forbes, N. Oza, R. Parr, and S. Russell. Feasibility study of fully automated traffic using decision-theoretic control. Cal. PATH Research Report UCB-ITS-PRR-97-18, Inst. of Transportation Studies, U. C. Berkeley, 1997.

[9] N. Friedman. Learning belief networks in the presence of missing values and hidden variables. In *ICML-97*, 1997.

[10] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. *Mach. Learning*, 29:245–274, 1997.

[11] D. Heckerman, D. Geiger, and D. M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Mach. Learning*, 20:197–243, 1995.

[12] T.S. Jaakkola and M.I. Jordan. Recursive algorithms for approximating probabilities in graphical models. In *NIPS 9*, pp. 487–93, 1997.

[13] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI-95*, pp. 346–351, 1995.

[14] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *UAI-92*, 1992.

[15] W. Lam and F. Bacchus. Learning Bayesian belief networks: An approach based on the MDL principle. *Comp. Intel.*, 10:269–293, 1994.

[16] S. L. Lauritzen. The EM algorithm for graphical association models with missing data. *Comp. Stat. and Data Anal.*, 19:191–201, 1995.

[17] S. Liang, S. Fuhrman, and R. Somogyi. Reveal, a general reverse engineering algorithm for inference of genetic network architectures. In *Pacific Symp. on Biocomputing*, vol. 3, pp. 18–29, 1998.

[18] J. Malik and S. Russell. Traffic surveillance and detection technology development: New sensor technology final report. Research Report UCB-ITS-PRR-97-6, Cal. PATH Program, 1997.

[19] H. H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *Proc. of the Nat. Acad. of Sci.*, 94:814–819, 1997.

[20] H. H. McAdams and L. Shapiro. Circuit simulation of genetic networks. *Science*, 269:650–656, 1995.

[21] C. Meek and D. Heckerman. Structure and parameter learning for causal independence and causal interaction models. In *UAI-97*, pp. 366–375, 1997.

[22] G. Schwarz. Estimating the dimension of a model. *Ann. Stat.*, 6:461–464, 1978.

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

[24] H. Xu. Computing marginals for arbitrary subsets from marginal representations in Markov trees. *Art. Intel.*, 74:177–189, 1995.

[25] G. Zweig and S. Russell. Speech recognition with dynamic Bayesian networks. Submitted, 1998.