, , 1–34 ()

Adaptive Probabilistic Networks with Hidden Variables*

JOHN BINDER binder@cs.berkeley.edu
Computer Science Division, University of California, Berkeley, CA 94720-1776

DAPHNE KOLLER koller@cs.stanford.edu
Computer Science Department, Stanford University, Stanford, CA 94305-9010

STUART RUSSELL russell@cs.berkeley.edu
Computer Science Division, University of California, Berkeley, CA 94720-1776

KEIJI KANAZAWA keijik@microsoft.com
Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399

Received July 23, 1996; Revised March 24, 1997

Editor: Padhraic Smyth

Abstract. Probabilistic networks (also known as Bayesian belief networks) allow a compact description of complex stochastic relationships among several random variables. They are used widely for uncertain reasoning in artificial intelligence. In this paper, we investigate the problem of learning probabilistic networks with known structure and hidden variables. This is an important problem, because structure is much easier to elicit from experts than numbers, and the world is rarely fully observable. We present a gradient-based algorithm and show that the gradient can be computed locally, using information that is available as a byproduct of standard inference algorithms for probabilistic networks. Our experimental results demonstrate that using prior knowledge about the structure, even with hidden variables, can significantly improve the learning rate of probabilistic networks. We extend the method to networks in which the conditional probability tables are described using a small number of parameters. Examples include noisy-OR nodes and dynamic probabilistic networks. We show how this additional structure can be exploited by our algorithm to speed up the learning even further. We also outline an extension to *hybrid* networks, in which some of the nodes take on values in a continuous domain.

Keywords: Bayesian networks, gradient descent, prior knowledge, dynamic networks, hybrid networks

1. Introduction

Intelligent systems acting in the real world require the ability to make decisions under uncertainty using the available evidence. Over the past decade, probabilistic

* The second and third authors were supported by the Army Research Office through the Multidisciplinary University Research Initiative on Integrated Approaches to Intelligent Systems. The work of the second author was also supported by a University of California President's Postdoctoral Fellowship and an NSF Postdoctoral Associateship in Experimental Science, and more recently by ONR Grant N00014-96-1-0718. The work of the third author was also supported by NSF Grant IRI-9058427 and by a Miller Professorship of the University of California.

networks (also called *Bayesian belief networks*) have become the primary method for reasoning and acting under uncertainty. Probabilistic networks are based on sound probabilistic semantics, thereby allowing the application of well-understood techniques such as conditioning for incorporating new information and maximizing expected utility for decision making. They take advantage of the causal structure of the domain to allow a compact and natural representation of the joint probability distribution over the domain variables. Probabilistic networks have been shown to perform well in complex decision-making domains such as medical diagnosis, fault diagnosis, image analysis, natural language understanding, robot control, and real-time monitoring (see, e.g., Heckerman & Wellman, 1995).

While the compact and natural representation considerably facilitates knowledge acquisition, the process of eliciting a probabilistic network from experts is still a slow one, largely due to the need to obtain numerical parameters. Clearly, techniques for automatically learning probabilistic networks from data would be of significant value.

The advantages of learning probabilistic networks go beyond the standard ones obtained by any type of autonomous learning system: Probabilistic networks are known to be an effective representation for decision making and reasoning. This allows our learned model to be easily integrated as a component of a complex system. Furthermore, since probabilistic networks have a precise, local semantics, human experts can often provide provide prior knowledge in the form of strong constraints on the initial structure of the network. As we will show below, this can greatly reduce the amount of training data required. Finally, the output of the learning process is typically comprehensible to humans.

In this paper, we present a new learning algorithm for probabilistic networks. We focus on the problem of learning networks where some of the variables are *hidden*—that is, their values are not (necessarily) observable. This case is of particular interest since, in practice, we rarely observe all the variables. In a medical environment, for example, the actual disease is not always known even at the end of the treatment, and we rarely have results for all possible clinical tests. Furthermore, causal models often contain variables that are sometimes inferred but never observed directly, such as "syndromes" in medicine. As we will see, such variables can greatly reduce the complexity of the model.

We also restrict attention to the problem of learning the probabilistic parameters, assuming that the network structure is known. Although this is clearly only a partial solution to the problem of learning a probabilistic network, it is a particularly relevant one. It is often easy to elicit the causal structure from an expert, whereas eliciting exact probabilistic parameters can be very difficult. In a medical context, for example, the causal connections between diseases and their symptoms are often known. Furthermore, the probabilistic parameters are far likelier to change as the environment evolves, making it much more important to adapt them gradually over time to reflect actual events.

The network structure is used by the learning algorithm as a strong source of prior knowledge about the domain. As our results show, this significantly reduces the amount of data required to train the network. Clearly, this reduction results from the fact that a given network structure implies a set of conditional independence relationships that strongly constrain the joint probability distribution, thereby eliminating many parameters from the learning problem.

In very large networks, however, this reduction may not be enough. For example, the CPCS network (Pradhan, Provan, Middleton, & Henrion, 1994) would require 133,931,430 parameters if defined using explicit conditional probability tables. Instead, CPCS uses *parametric* descriptions of the conditional distributions in the network, such as *noisy-OR* and *noisy-MAX*, thereby reducing the network to only 8,254 parameters. An even more extreme illustration is provided by networks with continuous-valued variables, for which a general conditional distribution would require an infinite number of parameters.

In many practical problems, one must therefore use distributions characterized by an even smaller number of parameters. Hence, *provided the underlying domain can be approximated by a parameterized distribution*, we expect to realize a significant reduction in sample complexity by using parametric representations as our hypotheses. As we show, such hypotheses can be learned using a straightforward extension of our basic learning technique.

The paper begins with a basic introduction to probabilistic networks. We then present:

- A derivation of a gradient-based learning algorithm for probabilistic networks with hidden variables, where the gradient can be computed locally by each node using information that is available in the normal course of probabilistic network calculations.

- Experimental demonstrations of the algorithm on small and large networks, showing a dramatic improvement in learning rate resulting from inclusion of hidden variables.

- An extension of the algorithm to handle parameterized distributions.

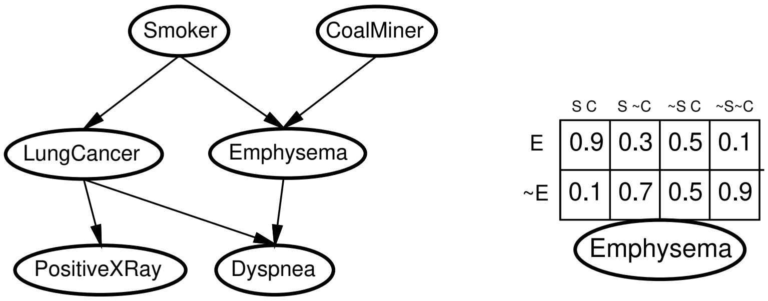
- Gradient derivations and experimental results for noisy-OR and dynamic probabilistic networks, along with discussion of continuous-variable networks.

We then describe related work, and we conclude with a discussion of future work and the prospects for adaptive probabilistic networks (APNs) as a general tool for learning complex models.

2. Probabilistic networks

Probability theory views the world as a set of random variables $X_1, \ldots, X_n$, each of which has a domain of possible values. For example, in describing cancer patients, the variables *LungCancer* and *Smoker* can each take on one of the values *True* and *False*. The key concept in probability theory is the *joint probability distribution*, which specifies a probability for each possible combination of values for all the random variables. Given this distribution, one can compute any desired posterior probability given any combination of evidence. For example, given observations and test results, one can compute the probability that the patient has lung cancer.

Unfortunately, an explicit description of the joint distribution requires a number of parameters that is exponential in n, the number of variables. Probabilistic networks (Pearl, 1988) derive their power from the ability to represent conditional independences among variables, which allows them to take advantage of the "locality" of causal influences. Intuitively, a variable is independent of its indirect causal influences given its direct causal influences. In Figure 1, for example, the outcome of the X-ray does not depend on whether the patient is a smoker given that we know that the patient has lung cancer. If each variable has at most k other variables that directly influence it, then the total number of required parameters is linear in n and exponential in k. This enables the representation of fairly large problems.



(a) **(b)**

Figure 1. (a) A simple probabilistic network showing a proposed causal model. (b) A node with associated conditional probability table. The table gives the conditional probability of each possible value of the variable *Emphysema*, given each possible combination of values of the parent nodes *Smoker* and *CoalMiner*.

Formally, a probabilistic network is defined by a directed acyclic graph together with a conditional probability table (CPT) associated with each node (see Figure 1). Each node represents a random variable. The CPT associated with variable X encodes $\mathbf{P}(X \mid Parents(X))$—the conditional distribution of the node given the different possible assignments of values to its parents. While the CPT is often represented as a simple table, as in Figure 1(b), it can also be represented implicitly as a parameterized function from values for X and $Parents(X)$ to probability values. With implicitly represented CPTs, a network can include continuous as well as discrete variables.

The arcs in the network encode a set of probabilistic independence assumptions: each variable must be conditionally independent of its non-descendants in the graph, given its parents. This constraint implies that the network provides a complete representation of the joint distribution through the equation

$$P(x_1 \ldots x_n) = \prod_{i=1}^{n} P(x_i \mid Parents(X_i)) \tag{1}$$

where $P(x_1 \ldots x_n)$ is the probability of a particular combination of values for $X_1, \ldots, X_n$.

Since a probabilistic network encodes a complete representation of the joint distribution, it implicitly contains the information to compute the probability of any set of query variables given any set of observations. That is, there is no restriction on what variables can be observed and what variables can be queries.

The general inference problem in probabilistic networks is NP-hard, so that all inference algorithms are of exponential complexity in the worst case. However, a number of algorithms have been developed that take advantage of network structure to perform the inference process effectively, often allowing the solution of large networks in practice. The most widely used algorithms for exact inference are *clustering algorithhms* (Lauritzen & Spiegelhalter, 1988), which modify the network topology, transforming it into a Markov random field.[1] Stochastic simulation algorithms have also been developed (e.g., Shachter & Peot, 1989; Fung & Chang, 1989), allowing for an anytime approximation of the solution. Massive parallelism can easily be applied, particularly with simulation algorithms.

3. Learning probabilistic networks

How can probabilistic networks be learned from data? There are several variants of this question. The structure of the network can be *known* or *unknown*, and the variables in the network can be *observable* or *hidden* in all or some of the data points. (The latter distinction is also described by the terms "complete data" and "incomplete data.") All of these tasks are described in the excellent tutorial by Heckerman (1995).

The case of known structure and fully observable variables is the easiest. In this case, we need only learn the CPT entries. Since every variable is observable, each data case can be pigeonholed into the CPT entries corresponding to the values of the parent variables at each node. Simple Bayesian updating then computes posterior values for the conditional probabilities based on Dirichlet priors (Olesen, Lauritzen, & Jensen, 1992; Spiegelhalter, Dawid, Lauritzen, & Cowell, 1993).

The case of unknown structure and fully observable variables has received a significant amount of attention. In this case, the problem is to reconstruct the topology of the network—a discrete optimization problem usually solved by a greedy search in the space of structures (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994). For any given proposed structure, the CPTs can be reconstructed as described above. The resulting algorithms are capable of recovering fairly large networks from large data sets with a high degree of accuracy.

Unfortunately, as we pointed out in the introduction, it is difficult to find real-life data sets where all of the variables are always observed. The existence of hidden variables significantly complicates the learning problem, so that the problem of learning networks with hidden variables has received much less attention than the fully-observable case. We discuss the previous work on this topic in Section 7.

The case on which we focus in this paper is that of known structure[2] with hidden variables. This is a particularly important case, since it seems to capture a significant range of interesting practical problems. In many applications, such as insurance, finance, and medicine, domain experts have a well-developed understanding of the qualitative properties of the domain. Furthermore, as we discuss later on, an algorithm for learning the numerical parameters is necessarily a component in any algorithm for learning networks with hidden variables.

One might ask why the hidden-variable problem cannot be reduced to the fully observable case by ignoring the hidden variables, or (in the case of known structure) by eliminating them using marginalization ("averaging out"). There are several reasons for this. First, it is not necessarily the case that any particular variable is hidden in all the observed cases (although we do not rule this out). Second, the hidden variable might be one that we are interested in querying, as in learning mixture models (Titterington, Smith, & Makov, 1985). Finally, networks with hidden variables can be more *compact* than the corresponding fully observable network (see Figure 2). In general, if the underlying domain has significant local structure, then with hidden variables it is possible to take advantage of that structure to find a more concise representation for the joint distribution on the observable variables. This, in turn, makes it possible to learn from fewer examples.

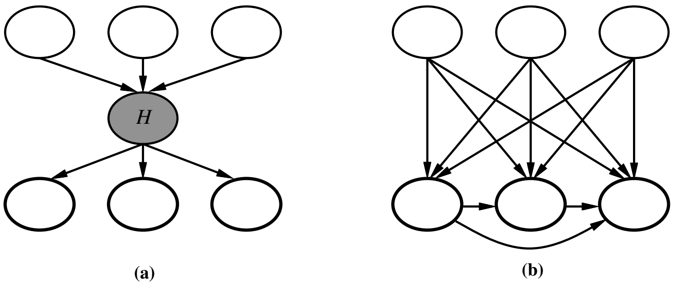


Figure 2. (a) A probabilistic network with a hidden variable, labelled H, where H is two-valued and the other variables are three-valued. The network requires 45 independent parameters. (b) The corresponding fully observable network, following arc reversal and node removal. The network now requires 708 parameters.

Before describing the details of our solution, we will explain the task in more detail. The algorithm is provided with a network structure and initial (possibly randomly generated) values for the CPTs. It is presented with a set $\mathbf{D}$ of data cases $D_1, \ldots, D_m$. We assume that the cases are generated independently from some underlying distribution. In each data case, values are given for some subset of the variables; this subset may differ from case to case. The object is to find the CPT parameters $\mathbf{w}$ that best model the data, where w_{ijk} denotes a specific CPT entry, the probability that variable X_i takes on its jth possible value assignment given that its parents $\mathbf{U}_i$ take on their kth possible value assignment:

$$w_{ijk} \equiv P(X_i = x_{ij} \mid \mathbf{U}_i = \mathbf{u}_{ik}). \tag{2}$$

To operationalize the phrase "best model the data," we assume for the purposes of this paper that each possible setting of $\mathbf{w}$ is equally likely *a priori*, so that the *maximum likelihood* (ML) model is appropriate. This means that the aim is to maximize $P_{\mathbf{w}}(\mathbf{D})$, the probability assigned by the network to the observed data when the CPT parameters are set to $\mathbf{w}$.

One can draw a straightforward analogy between this task and the task of learning in feedforward neural networks. In the latter task, the weights on the links are adjusted to minimize an error function $E_{\mathbf{w}}(\mathbf{D})$ that describes the degree of fit between the data and the network's outputs. A number of different error functions can be used. Minimizing the *cross-entropy* error function (see below) can be shown to result in maximum likelihood estimation for discrete data (Baum & Wilczek, 1988; Bishop, 1995).

In some situations, maximizing likelihood is not appropriate and can result in overfitting. A maximum *a posteriori* (MAP) analysis assumes a prior distribution $P(\mathbf{w})$ on the network parameters and adjusts $\mathbf{w}$ to maximize $P_{\mathbf{w}}(\mathbf{D})P(\mathbf{w})$. It is straightforward to extend the methods described below to solve this problem, as shown by Thiesson (1995a). This extension can also be performed for feedforward neural networks— the so-called *regularization* method (e.g., Poggio & Girosi, 1990). Finally, it is possible to perform a full Bayesian analysis, which uses the posterior distribution over the weights, $P(\mathbf{w} \mid \mathbf{D})$, to make predictions for new cases by integrating over the predictions made for each possible weight setting. Bishop (1995, Chapter 10) gives an excellent discussion of Bayesian methods applied to neural networks, including the seminal work of MacKay (1992). Some preliminary work on the application of the full Bayesian method to probabilistic networks has been done—see Heckerman (1995, Section 6), Friedman, Geiger, and Goldszmidt (1997), and the references therein. However, even approximate methods are typically computationally intractable. Furthermore, as Heckerman points out, as the size of our training set increases, the MAP and even the ML solutions become increasingly more accurate approximations to the full Bayesian solution.

4. Gradient-based algorithms

Our approach is based on viewing the probability $P_{\mathbf{w}}(\mathbf{D})$ as a function of the CPT entries $\mathbf{w}$. This reduces the problem to one of finding the maximum of a multivariate nonlinear function.[3] Algorithms for solving this problem typically follow a path on a surface whose "coordinates" are the parameters and whose "height" is the value of the function, trying to get to the "highest" point on the surface. In fact, it is easier to maximize the log-likelihood function $\ln P_{\mathbf{w}}(\mathbf{D})$. Since the two functions are monotonically related, maximizing one is equivalent to maximizing the other.

The simplest variant of this approach is *gradient ascent* (also known as "hill climbing"). At each point $\mathbf{w}$, it computes $\nabla_{\mathbf{w}}$, the *gradient* vector of partial derivatives with respect to the CPT entries. The algorithm then takes a small step in the direction of the gradient to the point $\mathbf{w} + \alpha\nabla_{\mathbf{w}}$, where α is the step-size parameter. This simple algorithm will converge to a local optimum for small enough α.

In the problem at hand, this approach needs to be modified to take into account the constraint that $\mathbf{w}$ consists of conditional probability values, so that $w_{ijk} \in [0, 1]$. Furthermore, in any CPT, the entries corresponding to a particular conditioning case (an assignment of values to the parents) must sum to 1, i.e., $\sum_j w_{ijk} = 1$.

One can incorporate these constraints by projecting $\nabla_{\mathbf{w}}$ onto the constraint surface. In general, this type of projection is accomplished by taking the vector that is normal to the constraint surface, projecting the gradient vector onto that, and subtracting it from the original gradient vector. In this case, the constraint surface requires we have $\sum_j w_{ijk} = 1$ for every i and k. If, for a given value of i and k we have J different values for j, the vector which is orthonormal to the constraint surface has $1/\sqrt{J}$ in every one of the relevant components. Therefore, the projection of the gradient onto that vector will have, in the position corresponding to ijk, the average of the gradient components corresponding to $w_{i1k}, \ldots, w_{iJk}$. Subtracting this vector from the original gradient vector results in the renormalized vector.

Note that, in the renormalized gradient vector, the sum of the components corresponding to w_{ijk} for fixed i, k is zero. Therefore, if we take a small step along this vector, the sum $\sum_j w_{ijk}$ remains unchanged. Hence, if we started out on the constraint surface, we necessarily remain on the constraint surface, as desired. The algorithm terminates when a local maximum is reached, that is, when the projected gradient is zero.

An alternative method, commonly used in statistics, is to reparameterize the problem so that the new parameters automatically respect the constraints on w_{ijk} no matter what their values. In this case, we can define parameters β_{ijk} such that

$$w_{ijk} = \frac{\beta_{ijk}^2}{\sum_{j'} \beta_{ij'k}^2}.$$

This can easily be seen to enforce the constraints given above.[4] Furthermore, a local maximum with respect to β_{ijk} is also a local maximum with respect to w_{ijk}, and vice versa.

We can optimize for the β_{ijk}s using a similar process of gradient ascent. The gradient can be found by computing the (unnormalized) gradient with respect to the w_{ijk}, and then deriving the gradient with respect to the β_{ijk}s using the chain rule, as described in Section 6.

A variety of techniques can be used to speed up the convergence of the algorithm. Gradient-based methods are the standard approach in a variety of applications, including the task of training the parameters (weights) of a neural network. This has resulted in a huge literature on optimizations for gradient-based methods, much of which can be applied here.

5. Local computation of the gradient with explicit CPTs

The usefulness of gradient-based methods depends on the ability to compute the gradient efficiently. This is one of the main keys to the success of gradient descent in neural networks. There, *back-propagation* is used to compute the gradient of an error function with respect to the network parameters (i.e., the weights on the links). The existence of a local training algorithm that uses the results of inference simplifies the design and implementation of neural network systems. Furthermore, the fact that real biological learning processes are presumably local in the same sense lends a certain plausibility to the entire neural-network paradigm.

We now show that a similar situation occurs in probabilistic networks. In fact, for probabilistic networks, the inference algorithm already incorporates all of the necessary computations, so that no additional back-propagation is needed. The gradient can be computed locally by each node using information that is available in the normal course of probabilistic network calculations.

5.1. Derivation of the gradient formula

First, we show that we can compute the contribution of each case to the gradient separately, and sum the results.

$$\begin{aligned}\frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial w_{ijk}} &= \frac{\partial \ln \prod_{l=1}^{m} P_{\mathbf{w}}(D_l)}{\partial w_{ijk}} \qquad \text{(independent cases)} \\ &= \sum_{l=1}^{m} \frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial w_{ijk}} \\ &= \sum_{l=1}^{m} \frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)}. \qquad (3)\end{aligned}$$

Now the aim is to find a simple local algorithm for computing each of the expressions $\frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)}$. In order to get an expression in terms of information local to the parameter w_{ijk}, we introduce X_i and $\mathbf{U}_i$ by averaging over their possible values:

$$\begin{aligned}&\frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)} \\ &= \frac{\frac{\partial}{\partial w_{ijk}}\left(\sum_{j',k'} P_{\mathbf{w}}(D_l \mid x_{ij'}, \mathbf{u}_{ik'}) P_{\mathbf{w}}(x_{ij'}, \mathbf{u}_{ik'})\right)}{P_{\mathbf{w}}(D_l)} \\ &= \frac{\frac{\partial}{\partial w_{ijk}}\left(\sum_{j',k'} P_{\mathbf{w}}(D_l \mid x_{ij'}, \mathbf{u}_{ik'}) P_{\mathbf{w}}(x_{ij'} \mid \mathbf{u}_{ik'}) P_{\mathbf{w}}(\mathbf{u}_{ik'})\right)}{P_{\mathbf{w}}(D_l)}.\end{aligned}$$

For our purposes, the important property of this expression is that w_{ijk} appears only in linear form. In fact, w_{ijk} appears only in one term in the summation: the term for $j' = j$, $k' = k$. For this term, $P_{\mathbf{w}}(x_{ij'} \mid \mathbf{u}_{ik'})$ is just w_{ijk}. Hence

$$\begin{aligned}
\frac{\partial P_{\mathbf{w}}(D_l)/\partial w_{ijk}}{P_{\mathbf{w}}(D_l)} &= \frac{P_{\mathbf{w}}(D_l \mid x_{ij}, \mathbf{u}_{ik}) P_{\mathbf{w}}(\mathbf{u}_{ik})}{P_{\mathbf{w}}(D_l)} \\
&= \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik} \mid D_l) P_{\mathbf{w}}(D_l) P_{\mathbf{w}}(\mathbf{u}_{ik})}{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik}) P_{\mathbf{w}}(D_l)} \\
&= \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik} \mid D_l)}{P_{\mathbf{w}}(x_{ij} \mid \mathbf{u}_{ik})} \\
&= \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik} \mid D_l)}{w_{ijk}}. \qquad (4)
\end{aligned}$$

This last equation lets us "piggyback" the computation of the gradient on the calculations of posterior probabilities done in the normal course of probabilistic network operation. Essentially any standard probabilistic network algorithm, when executed with the evidence D_l, will compute the term $P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik} \mid D_l)$ as a by-product. We are therefore able to use standard probabilistic network packages as a substrate for our learning system.

For example, clustering algorithms (Lauritzen & Spiegelhalter, 1988) compute a posterior for each clique in the Markov network corresponding to the original probabilistic network. Because a node and its parents always appear together in at least one clique, the required probabilities can be found by summing out the other variables in that clique.

Approximate inference algorithms can also be used as the subroutine for our learning method. Such algorithms, particularly stochastic simulation techniques such as likelihood weighting (Shachter & Peot, 1989; Fung & Chang, 1989), provide "anytime" estimates of the required probabilities. This suits our needs very well: Early in the gradient descent process, we need only very rough estimates of the gradient; the use of an anytime algorithm allows these to be generated very quickly. As the learning process converges to a maximum, better estimates are called for, so the inference algorithm can be run for longer periods of time.

5.2. The basic algorithm

We can now summarize the above discussion in the form of a basic algorithm, which we call the APN (adaptive probabilistic network) algorithm, for learning probabilistic networks from data with hidden variables (see Table 1). For the sake of clarity, we show the simplest possible version. As discussed above, the algorithm can (and, in many cases, should) be extended to incorporate nonuniform priors over parameters, and to take advantage of more sophisticated methods for function optimization.

In the experiments described below, we have adapted the conjugate gradient method (Price, 1992) to keep the probabilistic variables in the legal [0,1] range

Table 1. A simplified skeleton of the algorithm for adaptive probabilistic networks.

function BASIC-APN(N, $\mathbf{D}$) **returns** a modified probabilistic network
  inputs: N, a probabilistic network with CPT entries $\mathbf{w}$
    $\mathbf{D}$, a set of data cases

  repeat until $\Delta\mathbf{w} \approx \mathbf{0}$
    $\Delta\mathbf{w} \leftarrow \mathbf{0}$
    for each $D_l \in \mathbf{D}$
      set the evidence in N from D_l
      for each variable i, value j, conditioning case k
        $\Delta w_{ijk} \leftarrow \Delta w_{ijk} + \frac{P_{\mathbf{w}}(x_{ij}, \mathbf{u}_{ik} \mid D_l)}{w_{ijk}}$
    $\Delta\mathbf{w} \leftarrow$ the projection of $\Delta\mathbf{w}$ onto constraint surface
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\, \Delta\mathbf{w}$
  return N

as described above. This uses repeated line minimizations (with direction chosen by the Polak–Ribière method) and a heuristic termination condition to signal a maximum. We use the HUGIN package (Andersen, Olesen, Jensen, & Jensen, 1989), which uses a clustering algorithm, for most of the inference computations.

5.3. Experimental evaluation

In this section, we report on two experiments. The first shows the importance of prestructuring the probabilistic network using hidden variables. The second shows the effectiveness of the algorithm on a large network with many hidden variables. In our experiments, we are primarily interested in the generalization ability of the algorithm as a function of the number of training cases (X–axis). Other experimental work aimed at reducing the amount of computation required—through approximate inference and improved optimization methods—will be discussed in other papers.

5.3.1. Performance metric

Generalization ability is typically measured by evaluating the predictions made by a learning algorithm against the true state of affairs for a set of test cases. Several issues are involved in deciding how this evaluation should be done.

The first question is to decide what sort of predictions will be evaluated. The APN algorithm described above is really designed for *density estimation*—that is, estimating the joint distribution over all variables. Most machine learning techniques, on the other hand, are designed for *classification*—that is, predicting the values of specified output variables given evidence on specified input variables. For the sake of comparison, we will treat the APN algorithm as a classification technique, recognizing that this may place it at a disadvantage given that it is estimating many more parameters than necessary for classification.[5]

The next question is how to measure the quality of a prediction. To do this, we use an error function given by the average negative log likelihood of the observed values of the specified output variables according to the learned model, where the average is taken over the m' cases in the test set $\mathbf{D}'$. Let $\mathbf{Y}$ denote the output variables and $\mathbf{E}$ denote the input variables, with $\mathbf{y}_l$ and $\mathbf{e}_l$ denoting the actual values in test case l. Then the error function for learned model $P_{\mathbf{w}}$ is given by

$$\hat{H}(\mathbf{D}', P_{\mathbf{w}}) = -\frac{1}{m'} \sum_{l=1}^{m'} \ln P_{\mathbf{w}}(\mathbf{y}_l \,|\, \mathbf{e}_l).$$

Notice that this is an empirical approximation of the *expected conditional cross-entropy* on the output variables between the learned distribution $P_{\mathbf{w}}$ and the true distribution P_* from which the data is generated. More precisely,

$$H(P_*, P_{\mathbf{w}}) = -\sum_{\mathbf{e},\mathbf{y}} P_*(\mathbf{e}, \mathbf{y}) \ln P_{\mathbf{w}}(\mathbf{y} \,|\, \mathbf{e}).$$

To compute the joint probability of the output variables using a standard probabilistic network inference algorithm, we can decompose it into marginals using the chain rule (Pearl, 1988, p.226):

$$P_{\mathbf{w}}(\mathbf{y} \,|\, \mathbf{e}) = \prod_i P_{\mathbf{w}}(y_i \,|\, \mathbf{e}, y_1, \ldots, y_{i-1}).$$

5.3.2. Methodology and comparison algorithm

Data were generated randomly from a target probabilistic network and then partitioned into training and test sets. The training data was used to train probabilistic networks that were initialized with random parameter values. Training set sizes ranged from 0 to 5000 cases. For each training set size, 10 training sets were selected randomly with replacement from the training data. For each training set, five different initial random parameter settings were tried. The learned model with the highest quality predictions on the training set was selected and evaluated on the test set. Reported results were then averaged over the ten training sets at each training set size.

Prediction performance for the APN algorithm was compared against that of a feedforward neural network. The neural network was trained using the update rule for cross-entropy minimization (Baum & Wilczek, 1988), which allows the output unit values to be interpreted as probabilities. To ensure that the distribution over the output units for each output variable summed to 1, the output layer used the softmax parameterization (Bridle, 1990). Because a feedforward neural network does not represent correlations among output nodes, the joint probability on the output variables was approximated by the product of the marginals from the output units. For the purposes of comparison, both APN and neural network algorithms incorporated a simple version of the early stopping rule: cross-entropy was measured on 10% of the available training data, and parameter adjustment was halted as soon as the cross-entropy began increasing. This provides a form of regularization, or MAP training. For each probabilistic network used to generate data, a comparison neural network was constructed with local coding for input and output variables—that is, a 0–1 unit for each value of the variable. Results are reported for a variety of hidden layer sizes. (We also tried cross-validation to optimize the hidden layer size, but this required too much computation given our experimental methodology.)

5.3.3. Results

The first experiment used data generated from the "3–1–3" network in Figure 2(a). We used the APN algorithm to train both a 3–1–3 network and the "3–3" network shown in Figure 2(b). We also trained a 9–N–9 feedforward neural network, as described above, for N = 1, 3, 5, 10. The results are shown in Figure 3.

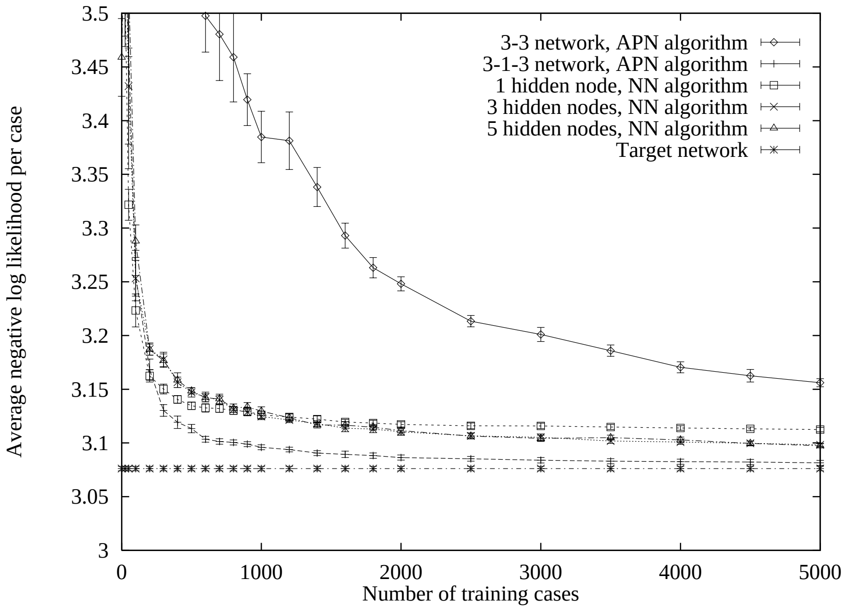


Figure 3. The output prediction error $\hat{H}(\mathbf{D}', P_{\mathbf{w}})$ as a function of the number of cases observed, for data generated from the network shown in Figure 2(a). The plot shows learning curves for the APN algorithm using the network structure in Figure 2(b), for the APN algorithm using the correct network structure, and for feedforward neural networks with 1, 3, and 5 hidden units.

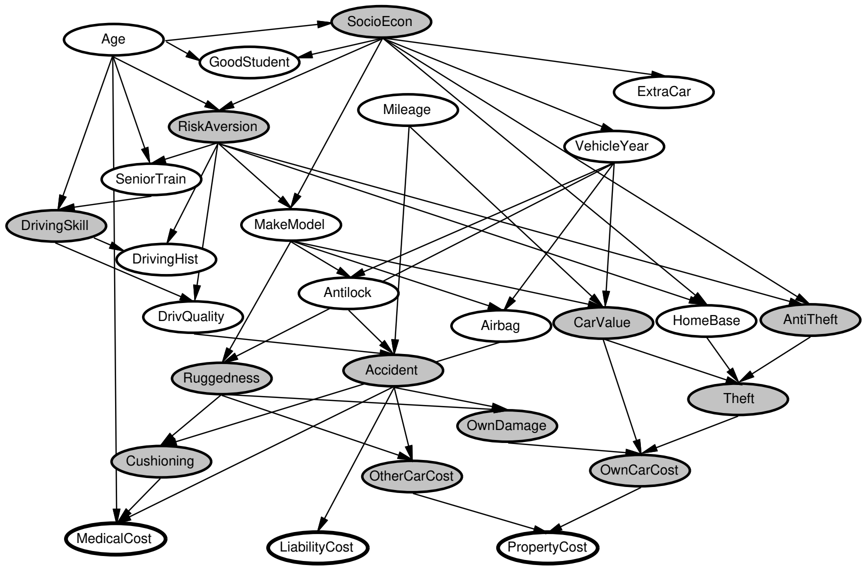


Figure 4. A network for estimating the expected claim costs for a car insurance policyholder. Hidden nodes are shaded and output nodes are shown with heavy lines.

The second experiment used data generated from a network for car insurance risk estimation (Figure 4). The network has 27 nodes, of which only 15 are observable, and over 1400 parameters. Three of the observable nodes are designated as "outputs." We ran an APN with the correct structure, an APN with a "12–3" structure analogous to the 3–3 network in Figure 2(b), and feedforward neural networks as before. The results are shown in Figure 5.

5.3.4. Discussion

The results for the 3–1–3 network in Figure 3 clearly demonstrate the advantage of using the network structure that includes the hidden node. The advantage derives, of course, from the reduction in the number of parameters from 708 to 45. The neural network shows fast convergence to a large error for the 1-unit hidden layer, with slower convergence for larger hidden layers. The APN algorithm with the correct network structure shows somewhat better performance than the neural networks, presumably because of the APN's ability to represent correlations among output variables.

For the insurance network, the correctly structured APN reaches a cross-entropy value of about 2.0 from about 500 cases, whereas the 12–3 APN requires approximately 200,000 cases to reach the same level. The neural network, on the other hand, shows rapid convergence to a cross-entropy error of about 2.2 for all hidden

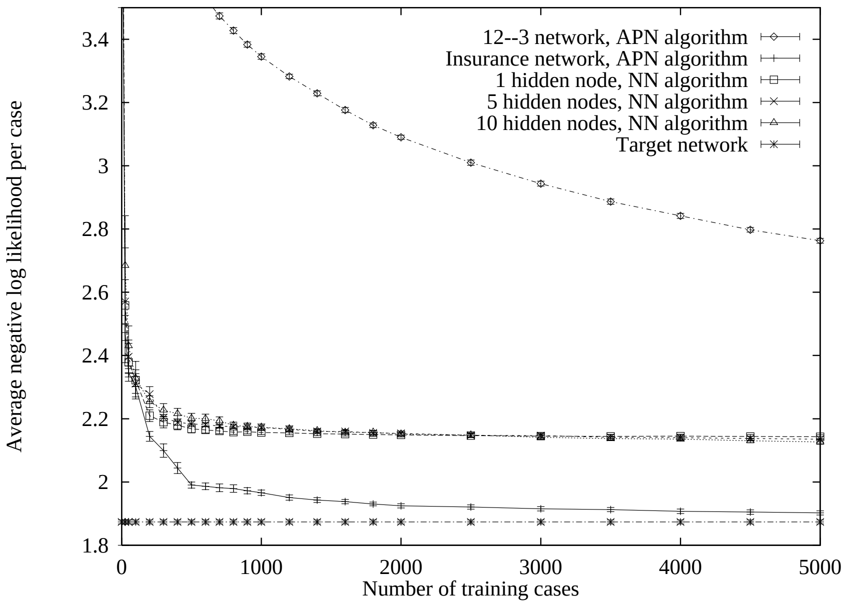


Figure 5. The output prediction error $\hat{H}(\mathbf{D}', P_{\mathbf{w}})$ as a function of the number of cases observed, for data generated from the network shown in Figure 4. The plot shows learning curves for the APN algorithm using a "12–3" network structure, for the APN algorithm using the correct network structure, and for feedforward neural networks with hidden layers of size 1, 5, and 10.

layer sizes, followed by a very slow reduction thereafter. Very similar behavior was found for decision trees with pruning and for k-nearest-neighbor learning. Thus, these three methods seem to be able to learn the most obvious regularities in the domain—principally, that most people never claim on their insurance—but are unable to learn the more complex patterns. In contrast, the correctly structured APN is able to detect the underlying patterns in the data and converges quickly to a lower error rate as a result.

Of course, this comparison between APN and neural network is not "fair" because the APN is given prior knowledge in the form of the network structure. On the other hand, the ability to use prior knowledge is a crucial element in machine learning. One might ask whether such prior knowledge can be provided to a neural network system, as suggested by Towell and Shavlik (1994). Perhaps a neural network structured similarly to the structured APN would be able to overcome this problem. Unfortunately, this approach seems unlikely to work, at least in such a simple form, because the structure in a neural network represents *deterministic* functional dependencies rather than the probabilistic dependencies represented by the probabilistic network structure. Also, training sparse neural networks with more than a few layers seems to be very difficult because of exponential attenuation of the error signal. Finally, standard feed-forward neural networks only allow inputs at the root nodes; restructuring the network to make the input nodes into root nodes may result in an exponential increase in the number of network parameters.

6. Extensions for generalized parameters

Our analysis above applies only to networks where there is no relation between the different parameters (CPT entries) in the network. Clearly, this is not always the case. If we do a particular clinical test twice, for example, the parameters associated with these two nodes in the network should probably be the same—even though the results of the two tests can differ.

In many situations, the causal influences on a given node are related in such a way that it becomes possible to represent the conditional distribution of the node given its parents using a more compact representation than an explicit table. Viewing a CPT as a function from the parent values $\mathbf{u}_{ik}$ and the child value x_{ij} to the number $P(X_i = x_{ij} \mid \mathbf{U}_i = \mathbf{u}_{ik})$, it is often reasonable to describe this function *parametrically*.

One common approach is to use a general-purpose function approximator such as a neural network. In other contexts, we might have more information about the structure of this function. A *noisy-OR model*, for example, encodes our belief that a number of diseases all have an independent chance of causing a certain symptom. We then have a parameter q_{ip} describing the probability that disease p in isolation fails to cause the symptom i. The probability of the symptom appearing given a combination of diseases is fully determined by these parameters. If the symptom node has k parents, the CPT for the node can be described using k rather than 2^k parameters (assuming that all nodes are binary-valued). As we show below, our algorithm can be extended to learn the noisy-OR parameters directly.

The ability to learn parametrically represented probabilistic networks confers many advantages. First, as we mentioned in the introduction, certain networks are simply impractical unless we reduce the size of their representation in this way. For example, the CPCS network mentioned above has only 8,254 parameters, but would require 133,931,430 parameters if the CPTs were defined by explicit tables rather than by noisy-OR and noisy-MAX nodes. This reduction in the number of parameters is even more important when learning such networks, because estimating each CPT entry separately would almost certainly require an unreasonable amount of training data. The ability of our algorithm to learn parametrized representations is another instance where prior knowledge can be utilized in the right way to speed up the learning process.

Even more importantly, this enables our learning algorithm to handle networks that otherwise would not fit into this framework. For example, as we mentioned above, probabilistic networks can also contain continuous-valued nodes. The "CPT" for such nodes must be parametrically defined, for example as a Gaussian distribution with parameters for the mean and the variance (Lauritzen & Wermuth, 1989). *Dynamic probabilistic networks*, which are potentially infinite networks that represent temporal processes, also require a parametrized representation. This is because the parameters that encode the stochastic model of state evolution appear many times in the network. Equation 5 (below) is the fundamental tool needed for learning both hybrid networks and dynamic probabilistic networks.

The rest of this section shows how the APN learning algorithm can be extended to deal with parametrized representations, and then demonstrates how this technique can be applied to networks with noisy-OR nodes, dynamic probabilistic networks, and hybrid networks.

6.1. The chain rule

Given that we want our network to be defined using parameters that are different from the CPT entries themselves, we would like to learn these parameters from the data. Our basic algorithm remains unchanged; rather than doing gradient ascent over the surface whose coordinates are the CPT entries, it does gradient ascent over the surface whose coordinates are these new parameters. The only issue we must address is the computation of the gradient with respect to these parameters. As we now show, our previous analysis can easily be extended to this more general case using a simple application of the *chain rule* for derivatives.

Technically, assume that the network is defined using a vector of adjustable parameters $\boldsymbol{\lambda}$. Each CPT entry w_{ijk} can be viewed as a function $w_{ijk}(\boldsymbol{\lambda})$. Assuming these functions are differentiable, we obtain

$$\frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial \lambda_p} = \sum_{i,j,k} \frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial w_{ijk}} \times \frac{\partial w_{ijk}}{\partial \lambda_p} \ . \tag{5}$$

Our earlier analysis showed how the first term in each product can be easily computed as a by-product of any standard probabilistic network algorithm. In fact, the computation in this case is somewhat simpler, since we do not have to normalize the gradient to account for the dependencies between the different w_{ijk}'s. (This is automatically dealt with by the second gradient term.) Assuming that the derivative $\frac{\partial w_{ijk}}{\partial \lambda_m}$ is a known function, the second term requires only a simple function application.

The approach using the chain rule may, in the worst case, involve summing over all combinations of i, j, and k in the network (i.e., all possible combinations of values for each node and its parents). In the cases we will consider, each parameter λ_p affects only a subset of the w_{ijk}s, so most of the terms $\partial w_{ijk}/\partial \lambda_p$ will be zero. Another approach is to express $P_{\mathbf{w}}(\mathbf{D})$ directly in terms of $\boldsymbol{\lambda}$ and then differentiate, thereby avoiding the summation over ijk. In some cases this may be more elegant, but the chain rule method has the advantage of requiring less creativity to solve each new parametrization. The end results are of course the same.

Table 2. Conditional probability table for $P(\mathit{Fever} \mid \mathit{Cold}, \mathit{Flu}, \mathit{Malaria})$, as calculated from the noisy-OR model.

Cold	*Flu*	*Malaria*	$P(\mathit{Fever})$	$P(\neg \mathit{Fever})$
F	F	F	0.0	1.0
F	F	T	0.9	0.1
F	T	F	0.8	0.2
F	T	T	0.98	$0.02 = 0.2 \times 0.1$
T	F	F	0.4	0.6
T	F	T	0.94	$0.06 = 0.6 \times 0.1$
T	T	F	0.88	$0.12 = 0.6 \times 0.2$
T	T	T	0.988	$0.012 = 0.6 \times 0.2 \times 0.1$

6.2. The noisy-OR model

Noisy-OR relationships generalize the logical notion of disjunctive causes. In propositional logic, we might say *Fever* is true if and only if at least one of *Cold*, *Flu*, or *Malaria* is true. The noisy-OR model adds some uncertainty to this strict logical description. The model makes three assumptions. First, it assumes that each of the listed causes (*Cold*, *Flu*, and *Malaria*) causes the effect (*Fever*), unless it is inhibited from doing so by some other cause. Second, it assumes that whatever inhibits, say, *Cold* from causing a fever is independent of whatever inhibits *Flu* from causing a fever. These inhibitors cause the uncertainty in the model, giving rise to "noise parameters." If $P(\mathit{Fever} \mid \mathit{Cold}, \neg\mathit{Flu}, \neg\mathit{Malaria}) = 0.4$, $P(\mathit{Fever} \mid \neg\mathit{Cold}, \mathit{Flu}, \neg\mathit{Malaria}) = 0.8$, and $P(\mathit{Fever} \mid \neg\mathit{Cold}, \neg\mathit{Flu}, \mathit{Malaria}) = 0.9$, then the noise parameters are 0.6, 0.2, and 0.1, respectively. Finally, it assumes that all the possible causes are listed; that is, if none of the causes hold, the effect does not happen. (This is not as strict as it seems, because we can always add a so-called *leak node* that covers "miscellaneous causes.")

These assumptions lead to a model where the causes—*Cold*, *Flu*, and *Malaria*—are the parent nodes of the effect—*Fever*. The conditional probabilities are such that if no parent node is true, then the effect node is false with 100% certainty. If exactly one parent is true, then the effect is false with probability equal to the noise parameter for that node. In general, the probability that the effect node is false is just the product of the noise parameters for all the input nodes that are true. For the *Fever* variable in our example, we have the CPT shown in Table 2.

The conditional probabilities for a noisy-OR node can therefore be described using a number of parameters linear in the number of parents rather than exponential in the number of parents. Let q_{ip} be the noise parameter for the pth parent node of X_i—that is, q_{ip} is the probability that X_i is false given that only its pth parent is true, and let T_{ik} denote the set of parent nodes of X_i that are set to T in the kth assignment of values for the parent nodes $\mathbf{U}_i$. Then

$$w_{ijk} = \begin{cases} \prod_{p \in T_{ik}} q_{ip} \text{ if } x_{ij} = F \\ 1 - \prod_{p \in T_{ik}} q_{ip} \text{ if } x_{ij} = T \end{cases} . \tag{6}$$

Thus, the parameterization $\boldsymbol{\lambda}$ consists of all the noise parameters q_{ip} in the network. Because w_{ijk} is independent of the values of parameters other than those associated with node i, we can mix noisy-OR and other types of nodes in the same network without difficulty. The gradient term for each noise parameter can be computed as follows:

$$\frac{\partial w_{ijk}}{\partial q_{ip}} = \begin{cases} 0 \text{ if } p \notin T_{ik} \\ \prod_{p' \in (T_{ik}-\{p\})} q_{ip'} \text{ if } x_{ij} = F \text{ and } p \in T_{ik} \\ -\prod_{p' \in (T_{ik}-\{p\})} q_{ip'} \text{ if } x_{ij} = T \text{ and } p \in T_{ik}. \end{cases} \tag{7}$$

The gradient of the log-likelihood with respect to each noise parameter is then computed by plugging Equations 4 and 7 into Equation 5.

We illustrate the application of the resulting equations using the noisy-OR network shown in Figure 6. Using the experimental methodology described in Section 5.3, we obtained the learning curve shown in Figure 7. As expected, the noisy-OR model results in faster learning than the corresponding model with explicitly represented CPTs.

6.3. Dynamic probabilistic networks

One of the most important applications of Equation 5 is in learning the behavior of stochastic temporal processes. Such processes are typically represented using *dynamic probabilistic networks (DPNs)* (Dean & Kanazawa, 1988). A DPN is structured as a sequence of *time slices*, where the nodes at each slice encode the state at the corresponding time point. Figure 8 shows the coarse structure of a generic DPN. The CPTs for a DPN encode both a *state evolution model*, which describes the transition probabilities between states, and a *sensor model*, which describes the observations that can result from a given state. Typically, one assumes that the CPTs in each slice do not vary over time. The same parameters therefore are duplicated in every time slice in the network.

As in a standard probabilistic network, we typically wish to describe our environment in terms of an assignment of values to multiple random variables. Thus, in an actual DPN, we often have many state and sensor variables in each time slice. A simple example DPN is shown in Figure 9.

For simplicity, we will use indices i and t to denote a single node in the DPN—the ith node in time slice t. Let w_{ijkt} be a CPT entry for time slice t, with ijk interpreted as above, and let λ_{ijk} be the generalized parameter such that

$$w_{ijkt} = \lambda_{ijk} \text{ for all } t.$$

Since $\partial w_{ijkt}/\partial \lambda_{i'j'k'} = 1$ if $ijk = i'j'k'$ and 0 otherwise, we obtain

$$\frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial \lambda_{ijk}} = \sum_t \frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial w_{ijkt}}. \tag{8}$$

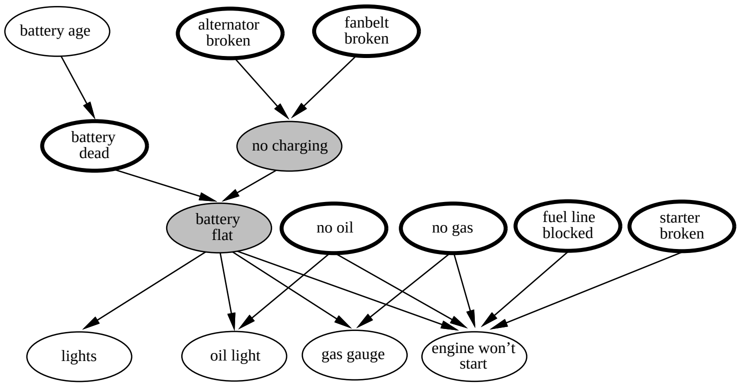


Figure 6. A network with noisy-OR nodes. Each node with multiple parents has a CPT described by the noisy-OR model (Equation 6). The network has 22 parameters, compared to 59 for a network that uses explicitly tabulated conditional probabilities.

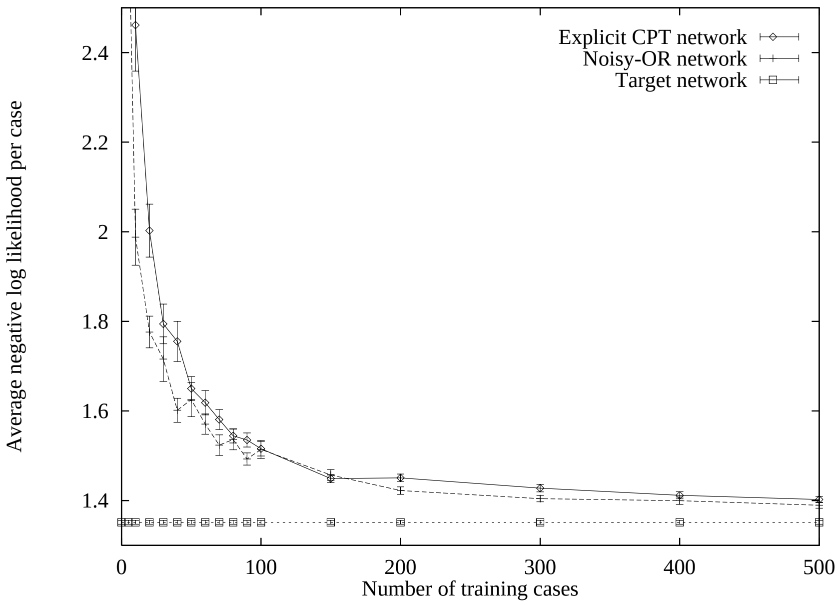


Figure 7. The output prediction error $\hat{H}(\mathbf{D}', P_{\mathbf{w}})$ as a function of the number of cases observed, for data generated from the network shown in Figure 6. The plot shows learning curves for the APN algorithm using a noisy-OR representation and using explicit CPTs.

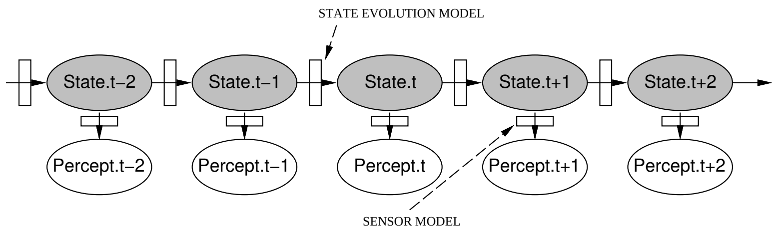


Figure 8. Generic structure of a dynamic probabilistic network.

In other words, we simply sum the gradients corresponding to the different instances of the parameter.[6]

We illustrate the application of the preceding equation using the DPN shown in Figure 9. As a strawman comparison, we show results for the "hidden Markov model" (HMM) version of this network. The HMM version has exactly the structure shown in Figure 8, with 32 states for the hidden variable and 8 values for the percept variable. Using the experimental methodology described in Section 5.3, We obtain the learning curves shown in Figure 10.

As a way of modelling a partially observable process, DPNs are directly comparable to HMMs. In an HMM, the hidden state of the process at a given time point is represented as the value of a single state variable. Similarly, at each time point, the observation is the value of a single sensor variable. Clearly, one can therefore view an HMM as a degenerate version of a DPN, essentially as in Figure 8. Smyth, Heckerman, and Jordan (1997) give a thorough analysis of the mapping netween the two representations.

In comparing DPNs and HMMs as tools for learning, the primary differences arise from the way in which state information and the state transition model are represented. An HMM represents each state as a node in a graph with state transition probabilities on the links, whereas a DPN decomposes the state into a set of state variables with the state transition model decomposed into the conditional distributions on each state variable. HMMs provide a natural sparse encoding for transition models in which each state can transition only to a small number of other states—for example, the motion of a stochastic robot on a grid. Such models are not necessarily naturally encoded by DPNs, although in the worst case the DPN can use a single state variable and a sparse transition matrix to obtain the same representation size as the HMM. On the other hand, sparse DPNs, in which each state variable influences only a small number of variables in the next slice, can have very large encodings as HMMs. For example, a completely *disconnected* DPN in which each state variable has no connections to other state variables translates into a completely *connected* HMM, provided there are no zeroes in the conditional distributions for each state variable. It was exactly this observation that led Ghahramani and Jordan (1997) to develop *factorial* HMMs, which are essentially DPNs.

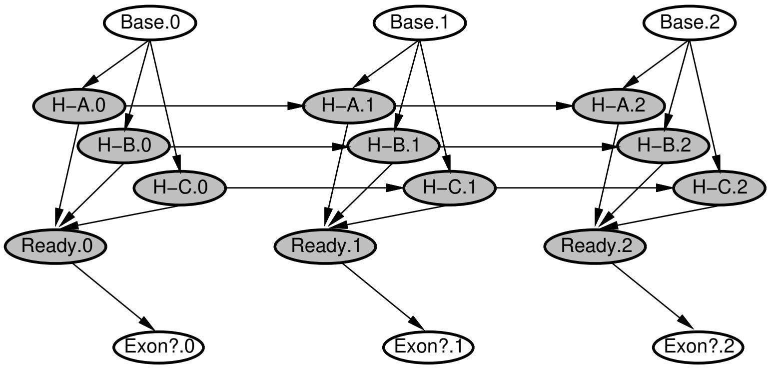


Figure 9. Three slices of a simple dynamic probabilistic network modelling an abstract version of the DNA splicing process. Hidden nodes are shaded.

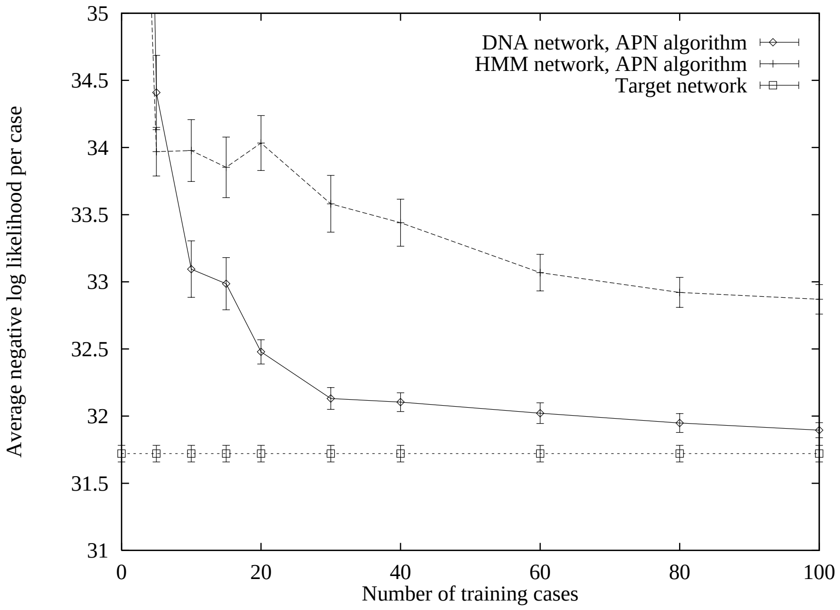


Figure 10. The output prediction error $\hat{H}(\mathbf{D}', P_{\mathbf{w}})$ as a function of the number of cases observed, for data generated from the network shown in Figure 9. The plot shows learning curves for the APN algorithm using the correct structure and for the APN algorithm using the HMM structure.

We expect that, for complex structured environments, the state variable decomposition afforded by DPNs will usually allow for a more parsimonious representation. Essentially, if one wants to carry n bits of state information, an HMM requires $O(2^n)$ states with $O(2^n)$ parameters for a sparse model with each state connected to a constant number of other states, or $O(2^{2n})$ parameters for a dense model. On the other hand, a DPN requires n state variables with $O(n)$ parameters for a locally structured model with a constant number of parents for each variable, or $O(2^{2n})$ parameters for a dense model. Experiments by Zweig (1996) show that this difference leads to a significant improvement in learning rates for DPNs when the underlying domain is locally structured.

6.4. Continuous variables

Many real-world problems involve continuous variables such as heights, masses, temperatures, and amounts of money; in fact, much of statistics deals with random variables whose domains are continuous. Given this fact, it is perhaps surprising that standard probabilistic network systems usually handle only discrete variables, forcing the user to discretize variables that are more naturally considered as continuous-valued. Discretization is sometimes an adequate solution, but often results in considerable loss of accuracy and very large CPTs.

Whenever a continuous variable appears in a probabilistic network, the CPT for that node becomes a conditional density function, since it must specify probabilities for the infinite domain of values of the variable. On the other hand, whenever a node has a continuous parent, then we must define an infinite set of distributions for the child—one for every possible value that the parent can take. In general, maintaining the analogy to CPTs, we have one 'CPT entry' for each possible value of the node and each possible value of the parents (although there can be infinitely many of each). Hence, for each node i, we will have a function $w_i(x, \mathbf{u})$ that denotes the conditional density (or probability) of $X_i = x$ given $\mathbf{U}_i = \mathbf{u}$.

6.4.1. The chain rule for continuous-variable parametrizations

It turns out that the application of the chain rule, given by Equation 5, holds unchanged if we simply substitute the probabilities by densities and the integrals by sums. To see this, consider some particular parameter λ_p, and assume (for simplicity) that λ_p only affects the parameter associated with the family consisting of the node X and its parents $\mathbf{U}$.

As before, we can separate the contribution of the different data cases:

$$\frac{\partial \ln P_{\mathbf{w}}(\mathbf{D})}{\partial \lambda_p} = \sum_{l=1}^{m} \frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial \lambda_p} = \sum_{l=1}^{m} \frac{\frac{\partial P_{\mathbf{w}}(D_l)}{\partial \lambda_p}}{P_{\mathbf{w}}(D_l)}. \tag{9}$$

Now, letting $p_{\mathbf{w}}$ be the density function for this probabilistic network, and by using Leibniz's rule and then the chain rule, we obtain

$$\begin{aligned}\frac{\partial P_{\mathbf{w}}(D_l)}{\partial \lambda_p} &= \frac{\partial}{\partial \lambda_p}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(p_{\mathbf{w}}(D_l \mid x,\mathbf{u})p_{\mathbf{w}}(x \mid \mathbf{u})p_{\mathbf{w}}(\mathbf{u}))dxd\mathbf{u} \\ &= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\frac{\partial}{\partial \lambda_p}(p_{\mathbf{w}}(D_l \mid x,\mathbf{u})p_{\mathbf{w}}(x \mid \mathbf{u})p_{\mathbf{w}}(\mathbf{u}))dxd\mathbf{u} \\ &= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\frac{\partial}{\partial p_{\mathbf{w}}(x \mid \mathbf{u})}(p_{\mathbf{w}}(D_l \mid x,\mathbf{u})p_{\mathbf{w}}(x \mid \mathbf{u})p_{\mathbf{w}}(\mathbf{u})) \times \frac{\partial p_{\mathbf{w}}(x \mid \mathbf{u})}{\partial \lambda_p}dxd\mathbf{u} \\ &= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(p_{\mathbf{w}}(D_l \mid x,\mathbf{u})p_{\mathbf{w}}(\mathbf{u})) \times \frac{\partial p_{\mathbf{w}}(x \mid \mathbf{u})}{\partial \lambda_p}dxd\mathbf{u}\ .\end{aligned}$$

Reintroducing the denominator of $P_{\mathbf{w}}(D_l)$, we can now repeat the steps used in the derivation of Equation 4, yielding

$$\frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial \lambda_p} = \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\frac{p_{\mathbf{w}}(x,\mathbf{u} \mid D_l)}{w_\lambda(x,\mathbf{u})} \times \frac{\partial w_\lambda(x,\mathbf{u})}{\partial \lambda_p}dxd\mathbf{u}\ . \tag{10}$$

6.4.2. Conditional Gaussian networks

Up to now, most continuous and hybrid networks have used a Gaussian distribution for the density function at a continuous node. In this case, the conditional density function can be described by two parameters—the mean μ and the variance σ^2. For the case of a continuous child with a continuous parent, it is common to use a linear function to describe the relationship between the continuous parent value and the mean of the child's Gaussian, while leaving the variance fixed (Pearl, 1988; Shachter & Kenley, 1989). The conditional density is then defined by the parameters of the linear function and the variance.[7]

In a *conditional Gaussian* (CG) distribution (Lauritzen & Wermuth, 1989), which describes the case where a continuous node has both discrete and continuous parents, one set of parameters is provided for each possible instantiation of the discrete parents. The full parameterized description is rather complicated, so we begin by illustrating the idea with a simple example.

Consider the *Price* node in Figure 11, which denotes the price of a particular type of fruit. The price depends on *Support?* (whether a government price support mechanism is operating) and *Crop* (the amount of fruit harvested that year). The CG model describes the conditional density distribution for *Price* as

$$\begin{aligned}p(Price = x \mid Support? = T \wedge Crop = h) &= \frac{1}{\sigma_t\sqrt{2\pi}}exp\left(-\frac{1}{2}\left(\frac{x-(a_th+b_t)}{\sigma_t}\right)^2\right) \\ p(Price = x \mid Support? = F \wedge Crop = h) &= \frac{1}{\sigma_f\sqrt{2\pi}}exp\left(-\frac{1}{2}\left(\frac{x-(a_fh+b_f)}{\sigma_f}\right)^2\right)\end{aligned}$$

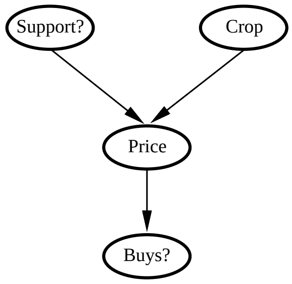


Figure 11. A simple network with discrete variables (*Support?* and *Buys*) and continuous variables (*Crop* and *Price*).

These equations describe the value of $w(x, S, h)$, where S is either T or F. The parameters $\boldsymbol{\lambda}$ on which w depends are a_t, a_f, b_t, b_f, σ_t, and σ_f.

In general, a continuous node X in a CG network may have both continuous parents $\mathbf{U}$ and discrete parents $\mathbf{Z}$. The conditional density of X is defined via a set of parameters as follows: For each possible assignment of values $\mathbf{z}$ to the discrete parents $\mathbf{Z}$, we have a real vector $\mathbf{a_z}$ whose length is the number of continuous parents $\mathbf{U}$ of X, and two parameters $b_{\mathbf{z}}$ and $\sigma_{\mathbf{z}}$. The conditional density $w_\lambda(x, \mathbf{u})$ then becomes

$$p(x \mid \mathbf{u}, \mathbf{z}) = \frac{1}{\sigma_{\mathbf{z}}\sqrt{2\pi}} exp\left(-\frac{1}{2}\left(\frac{x - (\mathbf{a_z} \cdot \mathbf{u} + b_{\mathbf{z}})}{\sigma_{\mathbf{z}}}\right)^2\right) \quad .$$

Differentiating with respect to $b_{\mathbf{z}}$ and with respect to component $a_{\mathbf{z}m}$ of $\mathbf{a_z}$, we obtain

$$\begin{aligned} \frac{\partial}{\partial b_{\mathbf{z}}} p(x \mid \mathbf{u}, \mathbf{z}) &= p(x \mid \mathbf{u}, \mathbf{z}) \times \frac{x - (\mathbf{a_z} \cdot \mathbf{u} + b_{\mathbf{z}})}{\sigma_{\mathbf{z}}^2} \\ \frac{\partial}{\partial a_{\mathbf{z}m}} p(x \mid \mathbf{u}, \mathbf{z}) &= p(x \mid \mathbf{u}, \mathbf{z}) \times u_m \frac{x - (\mathbf{a_z} \cdot \mathbf{u} + b_{\mathbf{z}})}{\sigma_{\mathbf{z}}^2} \quad , \end{aligned}$$

where u_m is the mth component of $\mathbf{u}$. This yields a particularly simple form for Equation 10 in this case:

$$\begin{aligned} \frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial b_{\mathbf{z}}} &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} p_{\mathbf{w}}(x, \mathbf{u} \mid D_l) \times \frac{x - (\mathbf{a_z} \cdot \mathbf{u} + b_{\mathbf{z}})}{\sigma_{\mathbf{z}}^2} dx d\mathbf{u} \\ \frac{\partial \ln P_{\mathbf{w}}(D_l)}{\partial a_{\mathbf{z}m}} &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} p_{\mathbf{w}}(x, \mathbf{u} \mid D_l) \times u_m \frac{x - (\mathbf{a_z} \cdot \mathbf{u} + b_{\mathbf{z}})}{\sigma_{\mathbf{z}}^2} dx d\mathbf{u} \quad . \end{aligned}$$

6.4.3. Discrete children of continuous parents

CG networks are a useful class because they allow for exact solution algorithms using an extension of standard clustering methods. However, this comes at a cost of restricting to a somewhat narrow class of hybrid networks. In particular, CG networks preclude discrete variables with continuous parents. Such variables are common in many domains. For example, whether or not a consumer buys a given product (a Boolean variable) depends on its price (a continuous variable). One simple formulation of this relationship would be to say that the consumer will buy the product if and only if the price is below some sharp threshold a (such as the price of some other substitutable fruit). A somewhat softer version of this case may have the probability of the purchase depending on whether the price is above or below a. In this case, we can describe the probability of *Buys?* as follows:

$$P(\mathit{Buys?} = T \mid \mathit{Price} = x) = \begin{cases} q_1 & \text{if } x \leq a \\ q_2 & \text{if } x > a \end{cases} .$$

The weight function $w(B, x)$ (for $B \in \{T, F\}$) can be viewed as depending on the parameters q_1 and q_2, and possibly on a as well.

It is instructive to consider the gradient $\frac{\partial w(B,x)}{\partial \lambda_p}$ for different parameters λ_p. In spite of the discontinuous nature of a threshold function, this derivative is well-defined for both q_1 and q_2, since the change in $w(B, x)$ is continuous in both these parameters. However, the gradient with respect to a is in general *not* continuous. Hence, we cannot use the method described above to adjust the threshold parameter.

The same problem was encountered in the context of neural networks, where the usual solution is to use the sigmoid function as an approximation to a threshold. A similar idea can be used in this context, where it has the natural interpretation that the probability of purchase varies from 0 to 1 continuously as the price drops. Neal (1992b) has investigated the learning of sigmoid models in belief networks.

In statistics, models for discrete choices dependent on continuous variables include both the sigmoid model (often called the *logit* model) and the *probit* model, which is the cumulative distribution of the normal density function (Finney, 1947). The use of the probit model in our example can be justified by positing a random, normally distributed noise process that interferes with the consumer's pure price comparison by imposing extra costs or benefits on the purchasing decision. Thus we have the weight function

$$w(F, x) = p(\mathit{Buys?} = F \mid \mathit{Price} = x) = \int_{-\infty}^{x} N_{\mu,\sigma}(x')dx'.$$

Here μ and σ parameterize the conditional distribution. Again, gradients with respect to μ and σ can be computed easily. For example, we have

$$\frac{\partial w(F, x)}{\partial \mu} = -N_{\mu,\sigma}(x) .$$

The probit model can be extended to the multinomial case, where there are more than two discrete choices (Daganzo, 1979). We can also include multiple parents. There is a large body of work on constructing and fitting complex probit models for a variety of applications. Most of this work can be imported into the context of probabilistic networks, which provide a natural platform for combining any of a huge variety of probabilistic models into composite representations for complex processes.

6.4.4. *Computation of the gradient*

As these examples illustrate, it is typically straightforward to compute the gradient $\partial w(x, \mathbf{u})/\partial \lambda_p$. But how do we compute the value of the entire expression in Equation 10? In the discrete case, the conditional probability of $x, \mathbf{u}$ given the data was easily computed as a byproduct of a number of inference algorithms. In the case of hybrid networks, however, the applicability of this approach is more limited. For one thing, exact inference algorithms are currently known only for CG networks. For another, even if we could somehow compute both elements in the product, how would we compute the integral? After all, it is very unlikely that, in general, the joint distributions can be expressed in closed form.

Stochastic simulation algorithms may solve both problems at once. First, stochastic simulation algorithms can be used to approximate the posterior distribution for general networks. Second, we can use the same sampling process to approximate the integral in a straightforward way. Each sample that we generate in the stochastic simulation process defines a value for each node in the network. The conditional density of each such point (given the data) is defined by the network. For each parameter λ_p at a given node, we simply maintain a running sum of this conditional density divided by the appropriate $w(x, \mathbf{u})$ and multiplied by $\frac{\partial w(x,\mathbf{u})}{\partial \lambda_p}$. This is essentially a numerical integration process for the integral in Equation 10, where the points at which we take the integral are the samples generated by the stochastic simulation process. Note that this process does not sample uniformly throughout the entire space of possible values for x and $\mathbf{u}$. Rather, it tends to focus on those points where the density $w(x, \mathbf{u})$ is highest. But these are points where the value of the function we are integrating also tends to be higher, so that their effect on the integral is also higher. We believe that this phenomenon will tend to improve the quality of the numerical integration process, but this has yet to be verified.

Whether this is true or not, it appears that stochastic simulation allows us to obtain an anytime approximation for the gradient, as in the discrete case. In this way, arbitrary combinations of discrete and continuous variables and arbitrary (finitely describable) distributions can be handled without resorting to complicated mathematics for each new model.

7. Related work

In this section, we discuss related work on the problem of learning parameters for probabilistic networks with hidden variables. For a survey on the full spectrum of topics in learning complex probabilistic models, see Buntine (1996).

Most of the work on learning networks with hidden variables has been restricted to the case of a fixed structure. This problem has been studied by several researchers, often explicitly noting the connection to neural network learning. The earliest work of which we are aware is that of Laskey (1990), who pointed out that both Boltzmann Machines and probabilistic networks are special cases of Markov networks. She used this fact to apply the Boltzmann Machine learning algorithm to probabilistic networks. Apolloni and de Falco (1991) devised a learning algorithm for what they called *asymmetric parallel* Boltzmann machines, which, as shown by Neal (1992a), are essentially sigmoid probabilistic networks. Neal (1992b) independently derived expressions for the likelihood gradient in sigmoid and noisy-OR probabilistic networks. Golmard and Mallet (1991) described a gradient-based algorithm for learning in tree-structured networks, and Kwoh and Gillies (1996) derived a version for polytree (singly connected) networks; both papers use an L_2 error function.

Our results, which apply to any probabilistic network, are significantly more general than those of Neal, Golmard and Mallet, or Kwoh and Gillies. As is clear from our derivation, the simple structure of the gradient expression follows from the fact that the network represents the joint probability as a sum of products of the local parameters, which is true both for multiply connected networks and for arbitrary CPTs. Buntine (1994), in the course of a general mathematical analysis of structured learning problems, also suggests that one could use generalized network differentiation for learning probabilistic networks with hidden variables, observing that the method should work for any distribution in the exponential family (of which probabilistic networks are a particular case, provided the conditional distributions at the nodes are themselves in the exponential family). Thiesson (1995b) analyzes this class of models, which he calls *recursive* exponential models (REMs). DPN models can be viewed as a special case of REMs.

For the specific case of maximizing likelihood by altering parameters of a probability distribution, the EM (Expectation Maximization) algorithm (Dempster, Laird, & Rubin, 1977) can be used. This observation was made by Lauritzen (1991, 1995), who discussed its application to general probabilistic networks (see also Spiegelhalter et al., 1993; Olesen et al., 1992; Spiegelhalter & Cowell, 1992; Heckerman, 1995). Like gradient descent, EM can be used to find local maxima on the likelihood surface defined by the network parameters. EM can also be used in the context of maximum a posteriori (MAP) learning, where we have a prior over the set of parameters (Heckerman, 1995). Thiesson (1995a) shows that a similar analysis can be used to do MAP learning with gradient-based methods.

EM can be shown to converge faster than simple gradient ascent, but the comparison between EM and conjugate gradient remains unclear. Lauritzen notes some difficulties with the use of EM for this problem, and suggests gradient-based methods as a possible alternative. Thiesson (1995a) combines EM with conjugate gradient methods, using the latter when close to a maximum in order to speed up convergence.

When learning a network with explicitly enumerated CPT entries, the EM algorithm reduces to a particularly simple form in which each E-step requires a computation of the posterior for a node and its parents, exactly as in Equation 4, and each M-step is trivial, requiring only that we instantiate the CPT entries to the average, over the different data cases, of the posterior probability of a node given its parents. However, for more complex problems with generalized parameters, the M-step may itself require an iterative optimization algorithm. Dempster et al. discuss generalized EM (GEM) algorithms that execute only a partial M-step; as long as this step increases the likelihood, the algorithm will converge (Neal & Hinton, 1993). Thus, a gradient-based approach is closely related to a GEM algorithm.

There has also been some work on the more complex problem of learning with hidden variables when the structural properties of the network are not all known. As in the fully observable case, dealing with an unknown structure necessarily involves a search over the space of possible structures. This, however, may be further complicated in the case of *latent* variables: hidden variables of whose existence we may be unaware. If we allow the introduction of new variables, the space of possible network structures becomes infinite, greatly complicating the problem. There is some work on this problem (Spirtes, Glymour, & Scheines, 1993; Geiger, Heckerman, & Meek, 1996), but much remains to be done.

8. Conclusions

We have demonstrated a gradient-descent learning algorithm for probabilistic networks with hidden variables that uses localized gradient computations piggybacked on the standard network inference calculations. Although a detailed comparison between neural and probabilistic networks requires more extensive analysis than is possible in this paper, one is struck by the fact that some of the primary motivations for the widespread adoption of neural networks as cognitive and neural models—localized learning, massive parallelism, and robust handling of noisy data—are also satisfied by probabilistic networks. Therefore, probabilistic networks seem to have the potential to provide a bridge between the symbolic and neural paradigms, as suggested by Laskey (1990). They also provide a way of composing a variety of local probabilistic models in order to represent a complex process; a huge amount of work in statistics can be imported for the purpose of fitting these local models to observations.

In addition to the basic algorithm, we have extended our analysis to the case of parametrically represented conditional distributions. This extension allows networks to be learned by estimating far fewer parameters, and improves the sample complexity accordingly. This speed-up has been demonstrated for noisy-OR and dynamic networks, and similar extensions for continuous variables have been outlined. We are currently investigating the application of APNs to modelling and prediction in financial areas such as insurance and credit approval, and in scientific areas such as intron/exon prediction in molecular biology. Results of the type obtained in this paper are crucial in extending APNs to these complex tasks.

The precise, local semantics of probabilistic networks allows humans or other systems to provide prior knowledge to constrain the learning process. We have demonstrated the improvements that can be achieved by pre-structuring the network, especially using hidden variables. Theoretical analysis of the sample complexity of learning probabilistic networks is an obvious next step, and first results have been obtained by Friedman and Yakhini (1996) and Dasgupta (1997). These results are currently being extended to cover dynamic probabilistic networks and networks with continuous variables.

Another possible improvement over the basic APN model is to allow the user or domain expert to prespecify constraints on the conditional distributions. One possibility is to specify a qualitative probabilistic network (Wellman, 1990) that provides appropriate monotonicity constraints on the conditional distributions. For example, the higher one's driving skill, the less likely it is that one has had an accident. In the extreme case, the user may be able to provide exact conditional distributions for some of the nodes in the network, particularly when those nodes depend deterministically on their parents. Since all such constraints eliminate some hypotheses from the hypothesis space, they will inevitably reduce the number of cases required for learning.

We are also looking at the problem of learning probabilistic parameters in cases where the network is implicitly represented as a set of first-order probabilistic rules, as described by Haddawy (1994) and by Ngo, Haddawy, and Helwig (1995). Just as for other parameterized representations, the resulting reduction in the number of parameters should allow for significantly faster learning. Koller and Pfeffer (1996) give some preliminary results along these lines. Such an approach transfers into the probabilistic domain the idea of *declarative bias* (Russell & Grosof, 1987). Results by Tadepalli (1993) show that learning within a structured model derived from a declarative bias can be made much more efficient using membership queries—that is, generating specific experiments on the domain rather than sampling it randomly. It would be interesting to see if the same ideas can be made to work in the context of APNs.

The extensions to parameterized representations and additional constraints make it possible to learn in situations where we have more prior knowledge about the domain than just the network topology. We are also considering the problem of learning in contexts where we have *less* prior knowledge. For example, as discussed in Section 7, we may have only partial knowledge about the network structure, about the set of values taken by a hidden variable, and even about the existence of hidden variables. These problems all require a discrete search over the space of possibilities. However, one important component in every such search algorithm is the instantiation of network parameters for the different structural options under consideration. The APN algorithm can clearly be used in that role. Its ability to handle parameterized representation may turn out to be particularly valuable in this case. As demonstrated in the recent results of Friedman and Goldszmidt (1996), incorporating parametric representations of CPTs into the structure search algorithm significantly changes the bias of the learning algorithm, greatly reducing the penalty for introducing additional arcs. In many cases, this results in greater prediction accuracy that is achievable with explicit CPTs.

Acknowledgments

We gratefully acknowledge useful comments and contributions from Wray Buntine, David Heckerman, Kathy Laskey, Steffen Lauritzen, Radford Neal, Judea Pearl, Avi Pfeffer, Terry Sejnowski, Padhraic Smyth, Bo Thiesson, Geoff Zweig, and the anonymous reviewers.

Notes

1. The term "clustering" here refers to the fact that the algorithms cluster nodes together to form larger entities; it should not be confused with the use of the term to refer to a form of unsupervised learning.
2. In this case, knowledge of structure is taken to include knowledge of the number of values of each variable. In other settings, particularly in learning mixture models (Titterington et al., 1985), finding the number of values of a class variable may be part of the learning task.
3. Since the function being maximized is a likelihood, the EM algorithm can also be used, as discussed in Section 7.
4. One can also use the *softmax* reparameterization (Bridle, 1990) which uses $\exp(\beta)$ instead of β^2.
5. It is possible to compute the gradient for the likelihood of specified output variables, rather than for the likelihood of all the variables. This form of optimization is discussed by Spiegelhalter and Cowell (1992).
6. Werbos (1990) surveys work on *back-propagation through time*, which uses a similar analysis based on the chain rule to derive the same conclusion for "unrolled" neural networks that represent temporal processes.
7. The model family obtained in this way is identical to that studied in the widely used technique of *factor analysis*.

References

Andersen, S. K., Olesen, K. G., Jensen, F. V., & Jensen, F. (1989). HUGIN—a shell for building Bayesian belief universes for expert systems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Vol. 2, pp. 1080–1085. Detroit, Michigan: Morgan Kaufmann.

Apolloni, B. & de Falco, D. (1991). Learning by asymmetric parallel Boltzmann machines. *Neural Computation*, *3*(3), 402–408.

Baum, E. B. & Wilczek, F. (1988). Supervised learning of probability distributions by neural networks. In Anderson, D. Z. (Ed.), *Neural Information Processing Systems*, pp. 52–61. American Institute of Physics, New York.

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

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Fogelman Soulié, F. & Hérault, J. (Eds.), *Neurocomputing: Algorithms, Architectures and Applications.* Springer-Verlag, Berlin.

Buntine, W. L. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, *2*, 159–225.

Buntine, W. L. (1996). A guide to the literature on learning probabilistic networks from data. *IEEE Transactions on Knowledge and Data Engineering*, *8*, 195–210.

Cooper, G. & Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

Daganzo, C. (1979). *Multinomial probit: The theory and its application to demand forecasting.* Academic Press, New York.

Dasgupta, S. (1997). The sample complexity of learning Bayesian nets. *Machine Learning, this issue.*

Dean, T. & Kanazawa, K. (1988). Probabilistic temporal reasoning. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pp. 524–528. St. Paul, Minnesota: American Association for Artificial Intelligence.

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

Finney, D. J. (1947). *Probit analysis; a statistical treatment of the sigmoid response curve.* Cambridge University Press, Cambridge.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning, this issue.*

Friedman, N. & Goldszmidt, M. (1996). Learning Bayesian networks with local structure. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 252–262. Portland, Oregon: Morgan Kaufmann.

Friedman, N. & Yakhini, M. (1996). On the sample complexity of learning Bayesian networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 274–282. Portland, Oregon: Morgan Kaufmann.

Fung, R. & Chang, K. C. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence (UAI-89)*. Windsor, Ontario: Morgan Kaufmann.

Geiger, D., Heckerman, D., & Meek, C. (1996). Asymptotic model selection for directed networks with hidden variables. In *Uncertainty in Artificial Intelligence: Proceedings of the Twelfth Conference*, pp. 283–290. Portland, Oregon: Morgan Kaufmann.

Ghahramani, Z. & Jordan, M. I. (1997). Factorial hidden Markov models. *Machine Learning, this issue.*

Golmard, J.-L. & Mallet, A. (1991). Learning probabilities in causal trees from incomplete databases. *Revue d'Intelligence Artificielle, 5*, 93–106.

Haddawy, P. (1994). Generating Bayesian networks from probability logic knowledge bases. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, pp. 262–269. Seattle, Washington: Morgan Kaufmann.

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

Heckerman, D., Geiger, D., & Chickering, M. (1994). Learning Bayesian networks: The combination of knowledge and statistical data. Technical report MSR-TR-94-09, Microsoft Research, Redmond, Washington.

Heckerman, D. & Wellman, M. (1995). Bayesian networks. *Communications of the Association for Computing Machinery, 38*(3), 27–30.

Koller, D. & Pfeffer, A. (1996). Learning the parameters of first order probabilistic rules. In *Working Notes of the AAAI Fall Symposium on Learning Complex Behaviors in Adaptive Intelligent Systems* Cambridge, Massachusetts.

Kwoh, C.-K. & Gillies, D. F. (1996). Using hidden nodes in Bayesian networks. *Artificial Intelligence, 88*(1–2), 1–38.

Laskey, K. B. (1990). Adapting connectionist learning to Bayes networks. *International Journal of Approximate Reasoning, 4*, 261–282.

Lauritzen, S. L. (1995). The EM algorithm for graphical association models with missing data. *Computational Statistics and Data Analysis, 19*, 191–201.

Lauritzen, S. L. & Wermuth, N. (1989). Graphical models for associations between variables, some of which are qualitative and some quantitative. *Annals of Statistics, 17*, 31–57.

Lauritzen, S. L. (1991). The EM algorithm for graphical association models with missing data. Technical report TR-91-05, Department of Statistics, Aalborg University.

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

MacKay, D. J. C. (1992). A practical Bayesian framework for back-propagation networks. *Neural Computation*, *4*(3), 448–472.

Neal, R. M. (1992a). Asymmetric parallel Boltzmann machines are belief networks. *Neural Computation*, *4*(6), 832–834.

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

Neal, R. M. & Hinton, G. E. (1993). A new view of the EM algorithm that justifies incremental and other variants. Unpublished manuscript.

Ngo, L., Haddawy, P., & Helwig, J. (1995). A theoretical framework for context-sensitive temporal probability model construction with application to plan projection. In *Uncertainty in Artificial Intelligence: Proceedings of the Eleventh Conference*, pp. 419–426. Montreal, Canada: Morgan Kaufmann.

Olesen, K. G., Lauritzen, S. L., & Jensen, F. V. (1992). aHUGIN: A system for creating adaptive causal probabilistic networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Eighth Conference*. Stanford, California: Morgan Kaufmann.

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

Poggio, T. & Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, *247*, 978–982.

Pradhan, M., Provan, G. M., Middleton, B., & Henrion, M. (1994). Knowledge engineering for large belief networks. In *Uncertainty in Artificial Intelligence: Proceedings of the Tenth Conference*, pp. 484–490. Seattle, Washington: Morgan Kaufmann.

Price, W. H. (1992). *Numerical Recipes in C*. Cambridge University Press, Cambridge.

Russell, S. J. & Grosof, B. (1987). A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*. Seattle, Washington: Morgan Kaufmann.

Shachter, R. D. & Peot, M. A. (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. & Kenley, C. R. (1989). Gaussian influence diagrams. *Management Science*, *35*(5), 527–550.

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

Spiegelhalter, D., Dawid, P., Lauritzen, S., & Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science*, *8*, 219–282.

Spiegelhalter, D. J. & Cowell, R. G. (1992). Learning in probabilistic expert systems. In Bernardo, J. M., Berger, J. O., Dawid, A. P., & Smith, A. F. M. (Eds.), *Bayesian Statistics 4* Oxford. Oxford University Press.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, prediction, and search*. Springer-Verlag, Berlin.

Tadepalli, P. (1993). Learning from queries and examples with tree-structured bias. In *Proceedings of the Tenth International Conference on Machine Learning*. Amherst, Massachusetts: Morgan Kaufmann.

Thiesson, B. (1995a). Accelerated quantification of Bayesian networks with incomplete data. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pp. 306–311. Montreal, Canada: AAAI Press.

Thiesson, B. (1995b). Score and information for recursive exponential models with incomplete data. Technical report R-95-2020, Institute for Electronic Systems, Aalborg University, Denmark.

Titterington, D., Smith, A., & Makov, U. (1985). *Statistical analysis of finite mixture distributions*. Wiley, New York.

Towell, G. & Shavlik, J. (1994). Knowledge-based artificial neural networks.. *Artificial Intelligence*, *70*, 119–165.

Wellman, M. P. (1990). Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, *44*(3), 257–303.

Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.

Zweig, G. (1996). Methods for learning dynamic probabilistic networks and a comparison with hidden Markov models.. MS report, Computer Science Division, UC Berkeley.

Received Date
Accepted Date
Final Manuscript Date