Adaptive probabilistic networks

Stuart Russell, John Binder, Daphne Koller
Computer Science Division
University of California
Berkeley, CA 94720*

July 25, 1994

Technical report UCB//CSD-94-824

Abstract

Belief networks (or *probabilistic networks*) and *neural networks* are two forms of network representations that have been used in the development of intelligent systems in the field of artificial intelligence. Belief networks provide a concise representation of general probability distributions over a set of random variables, and facilitate exact calculation of the impact of evidence on propositions of interest. Neural networks, which represent parameterized algebraic combinations of nonlinear activation functions, have found widespread use as models of real neural systems and as function approximators because of their amenability to simple training algorithms. Furthermore, the simple, local nature of most neural network training algorithms provides a certain biological plausibility and allows for a massively parallel implementation. In this paper, we show that similar local learning algorithms can be derived for belief networks, and that these learning algorithms can operate using only information that is directly available from the normal, inferential processes of the networks. This removes the main obstacle preventing belief networks from competing with neural networks on the above-mentioned tasks. The precise, local, probabilistic interpretation of belief networks also allows them to be partially or wholly constructed by humans; allows the results of learning to be easily understood; and allows them to contribute to rational decision-making in a well-defined way.

*This research was supported by NSF grant IRI-9058427 (PYI), and by a University of California Presidential Postdoctoral Fellowship to Daphne Koller.

1 Introduction

This paper presents a simple, local learning algorithm for belief networks with fixed structure and hidden variables. The algorithm operates by hillclimbing in the space of network parameters, resulting in a local or global maximum for the likelihood of the observed data. The algorithm is derived in a manner analogous to the derivation of backpropagation for neural networks, and yields an even simpler expression for the gradient in parameter space. We begin with a brief introduction to belief networks, Bayesian learning, and the various learning problems associated with belief networks. We then present the derivation of our algorithm and some comments on its implementation, performance, and applicability.

1.1 Belief networks

Over the last decade, probabilistic representations have come to dominate the field of *reasoning under uncertainty*, which underlies the operation of most expert systems, and of any agent that must make decisions with incomplete information. *Belief networks* (also called *causal networks* and *Bayesian networks*) are currently the principal tool for representing probabilistic knowledge [12]. They provide a concise representation of general probability distributions over a set of propositional (or multi-valued) random variables. The basic task of a belief network is to calculate the probability distribution for the unknown variables, given observed values for the remaining variables. Belief networks containing several thousand nodes and links have been used successfully to represent medical knowledge and to achieve high levels of diagnostic accuracy [6], among other tasks.

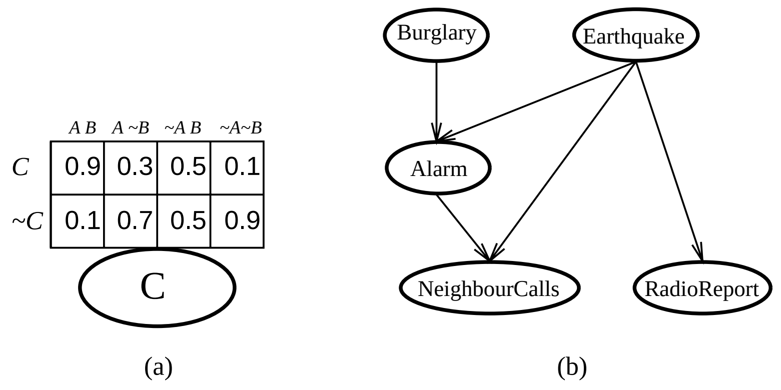

Figure 1: (a) A belief network node with associated conditional probability table. The table gives the conditional probability of each possible value of the variable, given each possible combination of values of the parent nodes. (b) A simple belief network.

The basic unit of a belief network is the *node*, which corresponds to a single random variable.

With each node is associated a *conditional probability table* (or CPT), which gives the conditional probability of each possible value of the variable, given each possible combination of values of the parent nodes. Figure 1(a) shows a node C with two Boolean parents A and B. Figure 1(b) shows an example network. Intuitively, the topology of the network reflects the notion of *direct causal influences*: the occurrence of an earthquake and/or burglary directly influences whether or not a burglar alarm goes off, which in turn influences whether or not your neighbour calls you at work to tell you about it. Formally speaking, the topology indicates that a node is conditionally independent of its ancestors given its parents; for example, given that the alarm has gone off, the probability that the neighbour calls is independent of whether or not a burglary has taken place.

A general network is a directed acyclic graph with nodes corresponding to random variables $X_1, \ldots, X_n$. Each CPT gives the values of $P(x_i|Parents(X_i))$ for each value x_i of the variable X_i. Variables can be either discrete or continuous. In this report, we restrict our attention to discrete variables. We will also assume that CPTs are represented explicitly (as tables of values). Our results will be extended to continuous variables and implicit functional representations of CPTs in a subsequent report.

A belief network provides a complete description of the domain. Every entry in the joint probability distribution for $X_1, \ldots, X_n$ can be calculated from the information in the network. A generic entry in the joint is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. We use the notation $P(x_1 \ldots x_n)$ as an abbreviation for this. The value of this entry is given by the following formula:

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

Thus each entry in the joint is represented by the product of the appropriate elements of the conditional probability tables in the belief network. The CPTs therefore provide a decomposed representation of the joint.

Belief network systems provide an inference service that calculates the posterior probability distribution $\mathbf{P}(X|E)$ for a variable X given evidence E (which consists of known values for some subset of the network variables). It is important to note that nodes in the network do not have to be designated as "input" and "output" nodes; for any given situation, whatever information is available can be provided to the network by setting the appropriate variables. A variety of inference algorithms have been developed, and will not be discussed in detail here. There are, however, some points worth making. For the subclass of *singly connected* (or *polytree*) networks, in which any two nodes are connected by at most one undirected path, the posterior distribution can be calculated in time linear in the size of the network. In general networks, the inference task is NP-hard [2], as is the corresponding approximation problem [4]. For general networks, a common technique [10] is to *cluster* variables in the network to form a *join tree* that is singly connected. Join tree algorithms calculate a posterior *potential* for clusters of nodes, where the entries in the potential table are proportional to the posterior joint probability distribution for the variables in the cluster.

1.2 Bayesian learning

Bayesian learning attempts to make predictions based on observed data, using *hypotheses* as intermediary representations. First, a set of *possible* hypotheses is identified, without reference to the observations, and prior probabilities are assigned. Based on the observations, the posterior probabilities of the hypotheses are calculated using Bayes' rule. Predictions are then made from the hypotheses, using the posterior probabilities of the hypotheses to weight the predictions.

Suppose that we have data D consisting of a set of cases, each described by assignments of values to variables in the domain. Assume that the hypotheses $H_1, H_2 \ldots$ are the possible hypotheses, and that we are interested in making a prediction concerning an unknown quantity X. Then we have

$$\mathbf{P}(X|D) = \sum_i \mathbf{P}(X|D, H_i)\mathbf{P}(H_i|D) = \mathbf{P}(X|H_i)\mathbf{P}(H_i|D)$$

where the simplification holds provided that each H_i determines a probability distribution for X. Since our hypotheses will be completely-specified belief networks, the simplification is valid for our purposes.

The above equation describes full Bayesian learning, and may require a calculation of $\mathbf{P}(H_i|D)$ for all H_i. The most common approximation is to use a *most probable* hypothesis, that is, an H_i that maximizes $\mathbf{P}(H_i|D)$. This often called a MAP (maximum a posteriori) hypothesis H_{MAP}:

$$\mathbf{P}(X|D) \approx \mathbf{P}(X|H_{\mathrm{MAP}})\mathbf{P}(H_{\mathrm{MAP}}|D)$$

The problem is now to find H_{MAP}.

By applying Bayes's rule, we can rewrite $\mathbf{P}(H_i|D)$ as follows:

$$\mathbf{P}(H_i|D) = \frac{\mathbf{P}(D|H_i)\mathbf{P}(H_i)}{\mathbf{P}(D)}$$

Notice that in comparing hypotheses, $P(D)$ remains fixed. Hence to find H_{MAP}, we need only maximize the numerator of the fraction.

The first term, $\mathbf{P}(D|H_i)$, represents the probability that this particular data set would have been observed, given H_i as the underlying model of the world. The second term represents the prior probability assigned to the given model. Arguments over the nature and significance of this prior probability distribution, and its relation to preference for simpler hypotheses (Ockham's razor), have raged unchecked in the statistics and learning communities for decades. The choice of an appropriate prior certainly depends on the complete space of hypotheses under consideration, and in some cases (for example, when arbitrary Turing machines are considered) it must vary inversely with the encoding length of the hypothesis. In our case, a *uniform* prior over belief networks seems to be appropriate. With a uniform prior, we need only choose an H_i that maximizes $\mathbf{P}(D|H_i)$. This is called a Maximumum Likelihood (ML) hypothesis, H_{ML}. The algorithm described below calculates an ML network for a set of observations.

1.3 Belief network learning problems

The learning problem for belief networks comes in several varieties. The structure of the network can be *known* or *unknown*, and the variables in the network can be *observable* or *hidden*.

- **Known structure, fully observable**: In this case, the only learnable part is the set of CPTs. These can be estimated directly using the statistics of the set of examples. Some belief network systems incorporate automatic updating of CPT entries to reflect the cases seen. In this case, the prior distribution over CPT values is crucial.
- **Unknown structure, fully observable**: In this case the problem is to reconstruct the topology of the network. An MAP analysis of the most likely network structure given the data has been carried out by Cooper and Herskovitz [3], and by Heckerman *et al.* [7]. The resulting algorithms are capable of recovering fairly large networks from large data sets with a high degree of accuracy. However, because of the intractability of the problem of finding the best topology, they usually adopt a greedy approach to choosing the set of parents for a given node.
- **Known structure, hidden variables**: This case is handled by the algorithm given below.
- **Unknown structure, hidden variables**: When some variables are sometimes or always unobservable, the techniques mentioned above for recovering structure become difficult to apply, since they essentially require averaging over all possible combinations of values of the unknown variables. At present, no good, general algorithms are known for this problem.

1.4 Related work

For a thorough introduction to belief networks, see [12]. The general problem of recovering distributions from data with missing values and hidden variables is addressed by the EM algorithm [5]. Our algorithm can be seen as as a variant of EM in which the "maximize" phase is carried out by a gradient-following method. Lauritzen [9] also considers the application of EM to belief networks. Spiegelhalter, Dawid, Lauritzen and Cowell [13] provide a thorough analysis of the statistical basis of belief network modification using Dirichlet priors, although they provide only a heuristic approximation for the hidden-variable case. Heckerman, Geiger and Chickering [7] describe an elegant and effective heuristic algorithm for recovering the structure of general networks in the fully observable case, building on the work of Cooper and Herskovits [3].

Gradient-following algorithms have been proposed by Neal [11], who derives an expression for the likelihood gradient in sigmoid networks using stochastic simulation, and uses it to show that the Boltzmann Machine (a variety of neural network) is a special case of a belief network. Neal's results show that an extremely close connection exists between neural and belief networks. Our results can be seen as a generalization of Neal's, and yield a more efficient algorithm. Burger and Connolly [1] also apply neural network techniques to learning belief networks, using a somewhat *ad hoc* error function to derive an error gradient for polytree networks.

2 Learning networks with fixed structure

Experience in constructing belief networks for applications has shown that finding the topology of the network is often straightforward. Humans find it easy to say what causes what, but hard to put exact numbers on the links. This is particularly true when some of the variables cannot usually be observed directly in actual cases. The "known structure, hidden variable" learning problem is therefore of great importance.

One might ask why the problem cannot be reduced to the fully observable case by eliminating the hidden variables using marginalization ("averaging out"). There are two 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, networks with hidden variables can be more *compact* than the corresponding fully observable network. Figure 2 shows an example. 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.

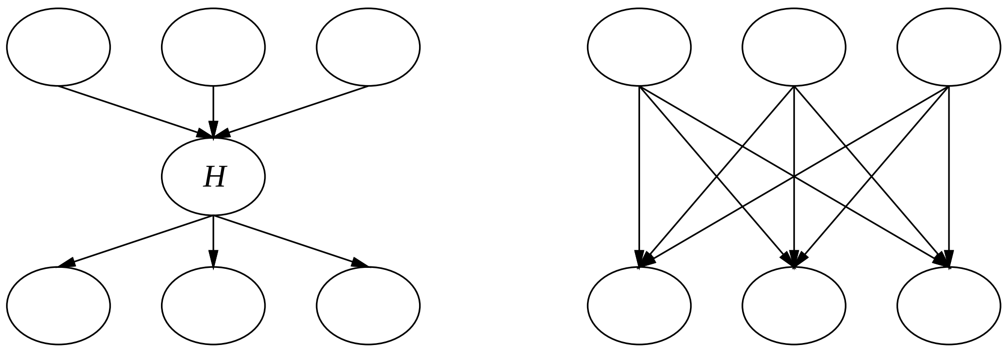

Figure 2: A network with a hidden variable (labelled H), and the corresponding fully observable network. If the variables are Boolean, then the hidden-variable network requires 17 independent conditional probability values, whereas the fully observable network requires 27.

If we are to approach this problem in Bayesian terms, then the "hypotheses" H_i are the different possible complete assignments to all the CPT entries. We will assume that all possible assignments are equally likely *a priori*, which means that we are looking for a maximum likelihood hypothesis. That is, we wish to find the set of CPT entries that maximizes the probability of the data, $P(D|H_i)$.

The method we will use to do this is quite similar to the gradient descent method for neural networks. We will write the probability of the data as a function of the CPT entries, and then calculate a gradient. As with neural networks, we will find that the gradient can be calculated locally by each node using information that is available in the normal course of belief network calculations.

Suppose we have a training set $\mathbf{D} = \{D_1, \ldots, D_m\}$, where each case D_j consists of an assignment of values to some subset of the variables in the network. We assume that each case is drawn independently from some underlying distribution that we are trying to model. The problem is to adjust the conditional probabilities in the network in order to maximize the likelihood of the data. We will write this likelihood as $P(\mathbf{D})$. The reader should bear in mind that here $P(\cdot)$ is really $P_{\mathbf{w}}(\cdot)$, that is, the probability according to a joint distribution specified by $\mathbf{w}$, the set of all of the conditional probability values in the network. In order to construct a hill-climbing algorithm to maximize the likelihood, we need to calculate the derivative of the likelihood with respect to each of the conditional probability values w_i in $\mathbf{w}$.

It turns out to be easiest to compute the derivative of the logarithm of the likelihood. Since the log-likelihood is monotonically related to the likelihood itself, the gradient vector on the log-likelihood surface is parallel to the gradient vector on the likelihood surface. We calculate the gradient using partial derivatives, varying a single value w_i while keeping the others constant:[1]

$$\begin{aligned}\frac{\partial \ln P(\mathbf{D})}{\partial w_i} &= \frac{\partial \ln \prod_j P(D_j)}{\partial w_i} \\ &= \sum_j \frac{\partial \ln P(D_j)}{\partial w_i} \\ &= \sum_j \frac{\partial P(D_j)/\partial w_i}{P(D_j)}\end{aligned}$$

Hence we can calculate separately the contribution of each case to the gradient, and sum the results.

Now the aim is to find an expression for the gradient contribution from a single case, such that the contribution can be calculated using only information local to node with which w_i is associated. Let w_i be a specific entry in the conditional probability table for a node X given its parent variables $\mathbf{U}$. We'll assume that it is the entry for the case $X = x_i$ given $\mathbf{U} = \mathbf{u}_i$:

$$w_i = P(X = x_i \mid \mathbf{U} = \mathbf{u}_i) = P(x_i \mid \mathbf{u}_i)$$

In order to get an expression in terms of local information, we introduce X and $\mathbf{U}$ by averaging over their possible values:

$$\begin{aligned}\frac{\partial P(D_j)/\partial w_i}{P(D_j)} &= \frac{\frac{\partial}{\partial w_i}\left(\sum_{x,\mathbf{u}} P(D_j \mid x, \mathbf{u}) P(x, \mathbf{u})\right)}{P(D_j)} \\ &= \frac{\frac{\partial}{\partial w_i}\left(\sum_{x,\mathbf{u}} P(D_j \mid x, \mathbf{u}) P(x \mid \mathbf{u}) P(\mathbf{u})\right)}{P(D_j)}\end{aligned}$$

For our purposes, the important property of this expression is that w_i appears only in linear form. In fact, w_i appears only in one term in the summation, namely the term for x_i and $\mathbf{u}_i$. For this term, $P(x \mid \mathbf{u})$ is just w_i, hence

$$\frac{\partial P(D_j)/\partial w_i}{P(D_j)} = \frac{P(D_j \mid x_i, \mathbf{u}_i) P(\mathbf{u}_i)}{P(D_j)}$$

Further manipulation reveals that the gradient calculation can "piggyback" on the calculations of posterior probabilities done in the normal course of belief network operation — that is, calculations of the probabilities of network variables given the observed data. To do this, we apply Bayes' theorem to the above equation, yielding

$$\frac{\partial P(D_j)/\partial w_i}{P(D_j)} = \frac{P(x_i, \mathbf{u}_i \mid D_j) P(D_j) P(\mathbf{u}_i)}{P(x_i, \mathbf{u}_i) P(D_j)} = \frac{P(x_i, \mathbf{u}_i \mid D_j)}{P(x_i \mid \mathbf{u}_i)} = \frac{P(x_i, \mathbf{u}_i \mid D_j)}{w_i}$$

[1] Strictly speaking, we need to include the constraint that the conditional probability values for any given conditioning case must remain normalized. A formal analysis shows that renormalizing after changing the probability values has the same effect.

In most implementations of belief network inference, the term $P(x_i, \mathbf{u}_i \mid D_j)$ is either computed directly or is easily obtained by summing a small number of table entries. (In joint tree algorithms, for example, it is the case that a node and its parents always appear together in a single cluster, and their posterior joint can be calculated by marginalizing out the other members of the cluster.) The complete gradient vector is obtained by summing the above expression over the data cases to give the gradient component with respect to each w_i for the likelihood of the entire training set.

3 Implementation considerations

Finding maxima of multivalued functions is a well studied problem, and dozens of algorithms have been proposed. At present we see no reason why the search problem should differ significantly from that encountered when using backpropagation in neural network learning. The majority of algorithms rely on taking small steps in the parameter space, using a gradient function as derived above. The simplest is hill-climbing (or gradient descent), which follows the gradient vector, recalculating it after each step, until a maximum is reached. A variety of techniques can be used to speed up the process, such as Polak-Ribiere conjugate gradient methods.

It can be shown that the likelihood function has local as well as global maxima, although to date our algorithms have not encountered any on the simple networks that we have used. Algorithms capable of finding global maxima include random-restart hillclimbing, in which a new starting point is chosen randomly after each hillclimbing phase reaches a maximum; and simulated annealing, in which some "downhill" steps are allowed stochastically, with a frequency depending on a decreasing "temperature" schedule [8].

In addition to the constraint that each CPT column must sum to 1, the search algorithm must also respect the constraint that CPT entries must fall in the range $[0, 1]$. It often occurs that the gradient-following algorithm reaches the edge of the "constraint surface," such that the next step would result in probability values less than 0 or greater than 1. In such cases, our algorithm truncates the step, and subsequently moves along the edge of the constraint surface. A maximum can lie on the edge of such a surface (where one of the probability values is 0 or 1) or at a "corner" where two or more values are extremal.

In order to integrate the gradient calculation with the standard probability calculations carried out in belief networks, we must take into account the fact that join tree algorithms work with a network representation that is derived from the original network by an expensive clustering algorithm—"compiling" the network. If we used gradient-following to modify the CPT entries in the original network, this might involve recompilation after every step, a prohibitively expensive prospect. It turns out that it is possible to apply incremental modifications directly to the join tree representations, since each entry in a clique table is proportional to the product of a set of CPT entries from the original network. If j is a clique table entry, with

$$j = \alpha \prod_i w_i$$

then if each w_i is modified by δw_i we have

$$\begin{aligned} \delta j &= \alpha \left(\prod_i (w_i + \delta w_i) - \prod_i w_i \right) \\ &= \sum_i \frac{j \delta w_i}{w_i} \end{aligned}$$

to first order in δw_i. Thus, by updating both the original network and the join tree network in parallel, recompilation can be avoided.

Using such results, the gradient-following method can be implemented for *any* belief network system, including systems based on stochastic simulation algorithms, and can provide online updating of conditional probability values whenever a new observation is made and posterior probabilities are calculated by the network.

4 Conclusions

We have presented the basis for a new algorithm for finding maximum likelihood belief networks with hidden variables and known structure. The algorithm promises to be useful in a wide variety of applications, and may allow probabilistic networks to take over many of the roles now filled by neural networks. Because it is relatively easy for humans to specify prior knowledge in the network toplogy and CPTs, and to interpret the results of learning, adaptive probabilistic networks may yield better results and have wider applicability than neural networks.

Separate reports will cover the following topics:

- Continuous variables.
- Parameterized functional representations of conditional probability tables, including "sigmoid" and "noisy-OR" networks.
- Learning in *dynamic* networks, which represent stochastic processes over time.
- Inclusion of quantitative and qualitative prior knowledge, such as "qualitative influences" [14], in the initial network to facilitate learning.
- Detailed discussion of implementations and applications.

References

[1] Burger, J., and Connolly, D. (1994). Probabilistic resolution of anaphoric inference.

[2] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

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

[4] Dagum, P., and Luby, M. (1993). Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, **60**, 141-53.

[5] Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm (with discussion). *J. Roy. Statist. Soc. B* **39**, 1-38.

[6] David Heckerman. *Probabilistic Similarity Networks.* MIT Press, Cambridge, Massachusetts, 1990.

[7] Heckerman, D., Geiger, D., and Chickering, M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data.* Technical report no. MSR-TR-94-09, Microsoft Research, Redmond, WA.

[8] Kirkpatrick, S., Gelatt, J., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, **220**, 671–680.

[9] Lauritzen, S. (1991). *The EM algorithm for graphical association models with missing data.* Technical report no. TR-91-05, Department of Statistics, Aalborg Univ.

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

[11] Neal, R. M. (1991). Connectionist learning of belief networks. Ărtificial Intelligence, 56, 71–113.

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

[13] Spiegelhalter, D., Dawid, P., Lauritzen, S., and Cowell, R. (1993). Bayesian analysis in expert systems. *Statistical Science*, **8**, 219-282.

[14] M.P. Wellman. Fundamental concepts of qualitative probabilistic networks. *Artificial Intelligence*, 44:257–303, 1990.