Proceedings of SOUTHCON'94 (Orlando, March 1994), 97–102.

Using Bayesian networks for incorporating probabilistic a priori knowledge into Boltzmann machines

Petri Myllymäki
University of Helsinki, Department of Computer Science
P.O.Box 26, FIN-00014 University of Helsinki, Finland

***Abstract*— We present a method for automatically determining the structure and the connection weights of a Boltzmann machine corresponding to a given Bayesian network representation of a probability distribution on a set of discrete variables. The resulting Boltzmann machine structure can be implemented efficiently on massively parallel hardware, since the structure can be divided into two separate clusters where all the nodes in one cluster can be updated simultaneously. The updating process of the Boltzmann machine approximates a Gibbs sampling process of the original Bayesian network in the sense that the Boltzmann machine converges to the same final state as the Gibbs sampler does. The mapping from a Bayesian network to a Boltzmann machine can be seen as a method for incorporating probabilistic a priori information into a neural network architecture, which can then be trained further with existing learning algorithms.**

I. Introduction

Neural network models have been suggested as a massively parallel platform for solving (NP-)hard problems approximatively [10, 2]. Especially suitable for this task seems to be the *Boltzmann machine* neural network architecture [8, 9], as this model is based on a stochastic energy minimization process similar to the famous Gibbs sampling method [14]. So far, the Boltzmann machine network structure corresponding to a problem domain has been either designed completely manually, which requires much work, or created by learning algorithms using a sample of data from the problem environment as the input. However, the learning algorithms start usually with a random initial state on an arbitrarily chosen network structure, being unable to use any a priori knowledge of the problem environment, although in many cases this kind of information would be available. This results in a very slow and unreliable learning process, and moreover, usually it is not possible to explain the behavior of the model after the learning. To overcome these difficulties, we present a method for automatically determining the structure and the connection weights of a two-layer stochastic neural network corresponding to a given Bayesian network representation of a probability distribution on a set of discrete variables. The resulting neural network is structurally close to the neural network model presented earlier in [19], but unlike the earlier model with directed connecting arcs, the model presented in this paper conforms directly to an existing neural network architecture, the Boltzmann machine. Compared to the Harmony network model in [15, 16], the model presented in this paper is very similar but more homogeneous as all the nodes are identical, whereas in the Harmony network model there are two types of nodes. The two-layer Boltzmann machine structure is optimal in the sense that all the nodes in one layer can be updated simultaneously. This is in contrast to the basic Boltzmann machine model where only one node of the network can be updated at a time, if the convergence of the corresponding Markov chain is to be guaranteed [1].

This research was supported by the Technology Development Center (TEKES).

II. Bayesian networks

Let $\boldsymbol{U}$ denote a set of N discrete random variables, $\boldsymbol{U} = \{U_1, \ldots, U_N\}$. We call this set our *variable base*. In the sequel, we use capital letters for denoting the actual variables, and small letters $u_1, \ldots, u_N$ for denoting their values. For clarity, we shall restrict ourselves in this study to binary variables, $u_i \in \{0, 1\}$. We say that variable U_i is *true*, if it has value 1, otherwise it is said to be *false*.

The values of all the variables in the variable base form a *state vector* $\vec{u} = (u_1, \ldots, u_N)$, and all the 2^N possible state vectors form our *state space* $\boldsymbol{\Omega}$. Let $\tilde{\mathcal{P}}$ denote a probability measure defined on $\boldsymbol{\Omega}$. A *Bayesian belief network* [20] is a directed acyclic network which can be used to represent any probability distribution by showing all the direct (causal) dependencies between the random variables of the problem domain. In a Bayesian network, each node represents one of the random variables of the problem domain, and two nodes are connected to each other if there is an asymmetric binary dependency relation between the corresponding variables (for an exact formulation of this relation, see e.g. [20] or [18]).

Given a variable X in a Bayesian network $\mathcal{G}$, let $\boldsymbol{F}_X$ denote the set of predecessors of X. It can be shown [20] that the joint probability distribution $\tilde{\mathcal{P}}$ corresponding to a Bayesian network can be represented as a product of conditional probabilities, one for each variable U_i:

$$\tilde{\mathcal{P}}\{\vec{u}\} = \prod_{i=1}^{N} \tilde{\mathcal{P}}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{U_i}} U_j = u_j\}, \quad (1)$$

where the conditional probability for variable U_i is defined as the apriori probability $\tilde{\mathcal{P}}\{U_i = u_i\}$, if $\boldsymbol{F}_{U_i} = \emptyset$.

The importance of Bayesian network representations lies in the way such structures can be used as compact representations for many naturally occurring distributions, where the dependencies between variables arise from a relatively sparse network of connections. In particular, given a Bayesian network representation of a probability distribution, and for each random variable, a set of conditional probabilities $\mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{U_i}} U_j = u_j\}$, it is possible to construct algorithms for different probabilistic reasoning tasks. In this work we are concerned with a task where the goal is to find the maximum probability state for the random variables, given a set of instantiated variables fixed to some values in advance. More explicitly, let us set the values of variables in a subset $\boldsymbol{E} \subset \boldsymbol{U}$, and let $\boldsymbol{\Omega}_E$ denote the set of states that are consistent with the given value assignment. We now define a new probability measure $\mathcal{P}$ as

$$\mathcal{P}\{\vec{u}_i\} = \begin{cases} \tilde{\mathcal{P}}\{\vec{u}_i\} & \text{, if } \vec{u}_i \in \boldsymbol{\Omega}_E \\ 0 & \text{, otherwise.} \end{cases} \quad (2)$$

Among all the states $\vec{u}_i \in \boldsymbol{\Omega}$, our goal is to find a state $\vec{u}^*$ with the property

$$\mathcal{P}\{\vec{u}^*\} \geq \mathcal{P}\{\vec{u}_i\}, i = 1, \ldots, 2^N.$$

We call this state a *maximum likelihood estimate (MLE)* state. For a general network structure, finding a MLE state can be shown to be NP-hard [4], which means that very probably it is not possible for any algorithm to solve this task exactly in polynomial time with respect to the size of the network. In the next section, we describe a stochastic method for solving this problem approximatively.

III. Gibbs sampling

Let C_i denote a set consisting of a variable U_i and all its predecessors in a Bayesian network $\mathcal{G}$, $C_i = \{U_i\} + \boldsymbol{F}_{U_i}$. We call these N sets the *cliques* of $\mathcal{G}$. For each clique C_i, we define a *potential function* V_i which maps a given state vector to a real number,

$$V_i(\vec{u}) = \ln \mathcal{P}\{U_i = u_i \mid \bigwedge_{U_j \in \boldsymbol{F}_{U_i}} U_j = u_j\}. \quad (3)$$

The value of a potential function V_i depends only on the values of the variables in set C_i. As has been noted in several occasions [11, 12, 13, 21, 16, 15], these cliques can be used to construct an undirected graphical *Markov random field* model of the probability distribution $\mathcal{P}$, which means that the joint probability distribution (2) for a Bayesian network representation can also be expressed as a *Gibbs distribution*

$$\mathcal{P}\{\vec{u}\} = \frac{1}{Z} e^{-\sum_i -V_i(\vec{u})} = \frac{1}{Z} e^{\sum_i V_i(\vec{u})},$$

where the clique potential functions V_i are given in (3), and $Z = 1$.

Let us define the potential V of a state $\vec{u}$ as

$$V(\vec{u}) = \sum_i V_i(\vec{u}).$$

We can now find a MLE state by maximizing the potential function V. In Gibbs sampling this task is solved by producing a random process $\{\vec{u}(t) \mid t = 0, 1, \ldots\}$, which converges to the MLE solution. The random process used is a Markov chain, which means that, at time t, the next state $\vec{u}(t+1)$ depends only on the current state $\vec{u}(t)$.

Let us assume that the state $\vec{u}(t)$ is $\vec{u}_i$. In Gibbs sampling and other stochastic simulation methods, the *candidate* for the next state $\vec{u}(t+1)$ is $\vec{u}_j$ with a *generation probability* G_{ij}. The generated candidate is accepted with an *acceptance probability* A_{ij}. Hence the whole *transition probability* P_{ij} from state $\vec{u}_i$ to state $\vec{u}_j$ at time t is

$$P_{ij}(t) = \begin{cases} G_{ij} A_{ij}(t) & \text{, if } i \neq j, \\ 1 - \sum_{j \neq i} G_{ij} A_{ij}(t) & \text{, if } i = j. \end{cases}$$

In this work, we are only concerned with *Barker's method* [3], where the acceptance probabilities are of the form

$$A_{ij}(t) = \frac{1}{1 + \frac{\mathcal{P}_t\{\vec{u}_i\}}{\mathcal{P}_t\{\vec{u}_j\}}} = \frac{1}{1 + \exp(\frac{-(V(\vec{u}_j) - V(\vec{u}_i))}{T(t)})}, \quad (4)$$

where $T(t)$ is a monotonely decreasing function converging to zero as t approaches infinity. Provided that the generation probabilities G_{ij} fulfill certain conditions, it can be shown that the corresponding stochastic process will converge to a MLE solution with probability p, where p approaches one as the number of iterations increases [1]. Moreover, it can be shown that if the temperature $T(t)$ decreases towards zero slowly enough, the convergence is almost certain even with a finite time process [6]. Unfortunately, for a theoretically guaranteed convergence, a computationally infeasible exponential number of iterations is needed. Although in practice good results are sometimes obtained with a relatively small number of iterations, the method can be excruciatingly slow. In the following, we show how to create a massively parallel implementation of Gibbs sampling using the Boltzmann machine architecture.

IV. Gibbs sampling by Boltzmann machines

A *Boltzmann machine (BM)* is a neural network consisting of a set of binary nodes $\{S_1, \ldots, S_n\}$, where the state s_i of a node S_i is either 1 ("on"), or 0 ("off"). Each arc from a unit S_i to unit S_j is provided with a real-valued parameter w_{ji}, the *weight* of the arc. The arcs are symmetric, so $w_{ij} = w_{ji}$. Each unit S_i is also provided with a real-valued parameter θ_i, the *bias* of the unit.

Let $\vec{s} = (s_1, \ldots, s_n) \in \{0,1\}^n$ denote a global state vector of the nodes in the network. We define the *consensus* of a state $\vec{s}$ as

$$C(\vec{s}) = \sum_{j=1}^{n} \sum_{i=j}^{n} w_{ij} s_i s_j, \tag{5}$$

where w_{ii} denotes the bias θ_i of node S_i, and the weight w_{ij} is defined to be 0 if S_i and S_j are not connected. Disregarding the sign reversal, the consensus function is equal to the normal definition of the energy of a BM model.

The nodes of a BM network are updated stochastically according to the following probabilistic rule:

$$P(s_i = 1) = \frac{1}{1 + \exp(\frac{-\sum_j w_{ij} s_j}{T(t)})}, \tag{6}$$

where $T(t)$ is a real-valued parameter called temperature, which decreases with increasing time t towards zero. Consequently, each node is able to update its state locally, using the information arriving from the connecting neighbors as the input for a sigmoid function, thus offering a possibility for a massively parallel implementation of this algorithm.

Let us first consider changing the state of one randomly chosen unit at a time. Let us denote the current state vector by $\vec{s}^t$, and let S_k be the node to be updated. The new state vector $\vec{s}^{t+1}$ is obtained by flipping the state s_k of node S_k while the other nodes remain unchanged:

$$s_i^{t+1} = \begin{cases} s_i^t & , \text{if } i \neq k, \\ 1 - s_i^t & , \text{if } i = k. \end{cases}$$

It is easy to see that the difference between consensus of states $\vec{s}^{t+1}$ and $\vec{s}^t$ is

$$C(\vec{s}^{t+1}) - C(\vec{s}^t) = (1 - 2s_k^t) \sum_{j=1}^{n} w_{kj} s_j^t.$$

Let us first assume that $s_k^t = 0$ (and hence $s_k^{t+1} = 1$). According to Barker's method (4), to maximize the consensus, the probability of accepting the new state $\vec{s}^{t+1}$ should be

$$\begin{aligned} P(\vec{s}^{t+1}) &= \frac{1}{1 + \exp(-(C(\vec{s}^{t+1}) - C(\vec{s}^t))/T(t))} \\ &= \frac{1}{1 + \exp(-((1 - 2s_k^t)\sum_{j=1}^n w_{kj} s_j^t)/T(t))} \\ &= \frac{1}{1 + \exp(-\sum_{j=1}^n w_{kj} s_j^t / T(t))}. \end{aligned}$$

On the other hand, if $s_k^t = 1$ (and hence $s_k^{t+1} = 0$), using Barker's method we get

$$\begin{aligned} P(\vec{s}^t) &= 1 - P(\vec{s}^{t+1}) \\ &= 1 - \frac{1}{1 + \exp(-(C(\vec{s}^{t+1}) - C(\vec{s}^t))/T(t))} \\ &= \frac{1}{1 + \exp((C(\vec{s}^{t+1}) - C(\vec{s}^t))/T(t))} \\ &= \frac{1}{1 + \exp(((1 - 2s_k^t)\sum_{j=1}^n w_{kj} s_j^t)/T(t))} \\ &= \frac{1}{1 + \exp(-\sum_{j=1}^n w_{kj} s_j^t / T(t))}. \end{aligned}$$

Hence in both cases the resulting acceptance probability is identical to the value of the sigmoid function in (6). Consequently, the BM updating process can be seen as a Gibbs sampling process with acceptance probabilities identical to those proposed by Barker (4), with respect to a Gibbs distribution

$$P\{\vec{s}^t\} = e^{C(\vec{s}^t)/T(t)}.$$

It follows that in principle the BM model can be used as a massively parallel tool for finding the maximum of the consensus function, provided that the generating probability matrix used fulfills the general requirements in [1, Th. 3.3]. However, the acceptance probability (6) sets here implicitly some additional requirements to the generation probabilities, since the difference in energy is calculated by keeping all nodes except one constant. For this reason, if two or more adjacent nodes of the BM network are to be updated at the same time, the corresponding transition probability matrix is no more stochastic, and hence convergence of the algorithm can not be guaranteed. On the other hand, if we allow only one node to be updated at a time, convergence can be guaranteed, but then the parallel nature of the algorithm is lost. To solve this dilemma, it has been suggested [7] that the nodes of a BM network should be divided into clusters, where no two nodes inside of a cluster are connected to each other. Using this kind of *clustered BM* models we can maintain some parallelism and update all the nodes in one cluster at the same time, while a convergence theorem can be proved [1, p. 139]. Obviously, the degree of parallelism depends on the number of clusters in the network. Unfortunately, the problem of finding a minimal set of clusters in a given network is NP-complete [1, p. 141]. In the next section we deal with a special class of BM architectures which has by definition only two clusters, being in this sense an optimal BM architecture.

V. Mapping Bayesian networks to Boltzmann machines

To use the BM model for finding a maximum likelihood state of a given Bayesian network, we need to find a way to map the potential function V of a given Bayesian network to the consensus function of a BM, and construct the corresponding BM structure. If the maximal clique size of a given Bayesian network is 2, then this mapping is relatively straightforward, as the consensus function is defined as a sum of factors depending of the values of two variables. The corresponding BM has N units, one for each random variable in the Bayesian network [8, 5]. However, in this case the parallelism of the resulting BM is lost, as no two adjacent units (variables) can be updated at the same time. Moreover, this type of a Bayesian network is a tree, which means that there is no need for approximate stochastic methods since polynomial time exact algorithms for solving the MLE task exist [20].

For a general Bayesian network structure, a similar N-unit construction is possible, if one uses a noisy-OR approximation (see [20, p. 184]) of the probability distribution instead of the accurate form (1)[17]. For an accurate representation of the general case, it seems at a first glance that binary connections are not sufficient, but higher-order connections (arcs connecting three or more nodes to each other) are necessary [5]. We avoid the need to generalize the BM model by introducing *hidden units*, i.e. units that do not directly correspond to any of the random variables U_i, but to combinations of these variables.

Our BM network consists of two layers of nodes: a set of N visible *feature nodes* corresponding to the variables $U_1, \ldots, U_N$, and a set of M hidden *pattern nodes* $\{A_1, \ldots, A_M\}$ corresponding to all possible values of the potential function V. As $V = \sum_k V_k$, and each clique potential V_k depends only on the values of the variables in clique C_k, we get

$$M = \sum_k 2^{|C_k|},$$

where $|C_k|$ is the number of variables in clique C_k. We can now rewrite the consensus function (5) as

$$C(\vec{s}) = \sum_{j=1}^{M} s_j \sum_{i=1}^{N} w_{ji} s_i = \sum_{j=1}^{M} s_j I_j, \qquad (7)$$

where $I_j = \sum_i w_{ji}$ is the *net input* to pattern node A_j.

Let us denote the potential value corresponding to a pattern node A_j by v_j, and let C_k be the clique containing the units determining the value v_j. The clique value assignment corresponding to unit A_j is denoted by $\vec{\omega}_j = (\omega_{j1}, \ldots, \omega_{jm})$, where m is the number of variables in clique C_k. We connect the pattern node A_j to all the feature nodes U_i belonging to clique C_k (see Figure 1). The connection weight $w_{ji} = w_{ij}$ on the arc connecting unit U_i and unit A_j is

$$w_{ji} = \begin{cases} v_j + c & \text{, if } \omega_{ji} = 1, \\ -(v_j + c) & \text{, if } \omega_{ji} = 0, \end{cases}$$

where $c > 0$ is some constant. Let m_1 be the number of variables having value one in the clique assignment $\vec{\omega}_j$, $m_1 = \sum_i^m \omega_{ji}$, and let $m_0 = m - m_1$. The bias of a unit A_j is

$$w_{jj} = (m_1 - 1)(v_j + c) - c.$$

Let us now consider the net input to unit A_j when the feature unit state vector is $\vec{u}$. We denote by $m_0^1 \le m_0$ the number of feature units U_i in clique C_k that are falsely on when ω_{ji} is zero, and by $m_1^0 \le m_1$ the number of feature units in $\vec{u}$ which are falsely off when ω_{ji} is one, so the number of inconsistent elements between $\vec{\omega}_j$ and $\vec{u}$ is $m_0^1 + m_1^0$. Now the net input I_j to unit A_j is

$$\begin{aligned} I_j &= (m_1 - m_1^0)(v_j + c) - m_0^1(v_j + c) \\ &\quad -(m_1 - 1)(v_j + c) - c \\ &= m_1 v_j + pc - m_1^0 v_j - m_1^0 c - m_0^1(v_j + c) \\ &\quad -m_1 v_j - pc + v_j + c - c \\ &= v_j - m_1^0(v_j + c) - m_0^1(v_j + c). \end{aligned}$$

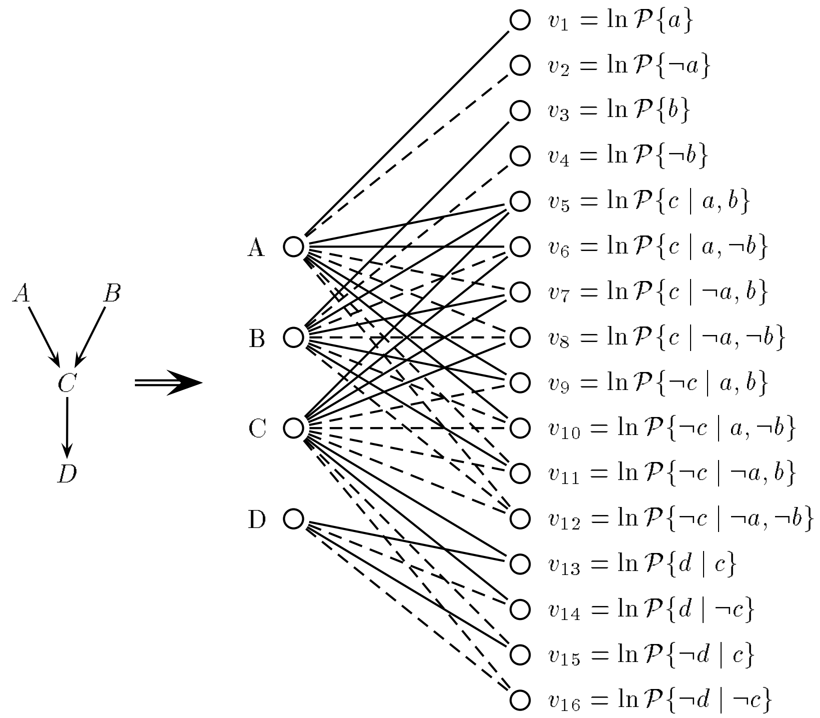

Figure 1: A simple Bayesian network and the corresponding BM architecture.

Assuming that v_j is positive, the net input is positive (i.e. v_j) only when $\vec{\omega}_j$ and $\vec{u}$ are consistent ($m_0^1 = m_1^0 = 0$), and negative otherwise. It follows that the net input to a pattern unit A_j is positive only for one pattern unit per each clique at a time. Hence in the final zero temperature state, only one pattern unit for each clique is on, and all the other pattern units are off. In this case, the number of pattern units to be on is N, and the consensus (7) of our BM model can be written as

$$C(\vec{s}) = \sum_{j=1}^{M} s_j I_j = \sum_{j=1}^{M} s_j v_j = \sum_{k=1}^{N} V_k(\vec{u}) = V(\vec{u}). \tag{8}$$

Consequently, as the temperature approaches zero, our BM updating process converges (with high probability) to the MLE solution. As the nodes on one layer are not connected to each other, and therefore do not affect the net input of each other, they can be updated at the same time.

In the construction above, it was assumed that all the clique potentials v_j are positive. In our case, however, all the clique potentials are initially non-positive (see formula (3)). For this reason, the clique potentials have to be scaled to positive numbers. Let v be some constant with property

$$v > \max_k \max_{\vec{u}} \mid V_k(\vec{u}) \mid,$$

and let us scale all the clique potentials to positive numbers by setting

$$V_k(\vec{u}) = V_k(\vec{u}) + v.$$

Assuming that only one pattern unit for each clique is on, we get

$$C(\vec{s}) = \sum_{k=1}^{N} (V_k(\vec{u}) + v) = V(\vec{u}) + Nv.$$

This new consensus function has the same maximum points as the original function (8), so the scaling does not affect the convergence of the BM updating process.

VI. Conclusion

We have showed how to map a given Bayesian network to a two-layer Boltzmann machine architecture, where all the nodes on one layer can be updated at the same time. The updating process of the BM model approximates the Gibbs sampling process of the given Bayesian network, in the sense that it provably converges to the same final state as the Gibbs sampler does. The number of nodes in the BM structure depends heavily on the structure of the Bayesian network: for sparse networks, where the maximum number of incoming arcs to a node is small, the corresponding BM model is relatively small. With dense Bayesian networks, the corresponding BM can grow quite big, as the number of hidden BM pattern nodes created for each node in the Bayesian network is exponential with the number of incoming arcs.

The Boltzmann machine model presented here can be used for massively parallel Bayesian reasoning directly, or "fine-tuned" further with existing gradient descent learning algorithms. What is more, the fine-tuned network can also be transformed back to a symbolic Bayesian network representation after the learning, to be examined by problem domain experts.

References

[1] Aarts, E., and Korst, J., Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley & Sons, Chichester, 1989.

[2] Baum, E.B., Towards practical 'neural' computation for combinatorial optimization problems. Pp. 53–58 in *Proceedings of the AIP Conference 151: Neural Networks for Computing* (Snowbird, UT, 1986), edited by J.Denker. American Institute of Physics, New York, NY, 1986.

[3] Barker, A.A., Monte Carlo calculations of the radial distribution functions for a proton-electron plasma. *Aust. J. Phys.* 18 (1965), 119–133.

[4] Cooper, G.F., The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42 (1990) 2–3 (March), 393–405.

[5] Geffner, H., and Pearl, J., On the probabilistic semantics of connectionist networks. Technical report R-84, UCLA Computer Science Department. Los Angeles, CA, 1987.

[6] Geman, S., and Geman, D., Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1984), 721–741.

[7] de Gloria, A., Faraboschi, P., and Olivieri, M., Clustered Boltzmann Machines: Massively parallel architectures for constrained optimization problems. *Parallel Computing* 19 (1993), 163–175.

[8] Hinton, G.E., and Sejnowski, T.J., Optimal perceptual inference. Pp. 448–453 in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Washington DC, June 1983). IEEE, New York, NY, 1983.

[9] Hinton, G.E., and Sejnowski, T.J., Learning and relearning in Boltzmann machines. Pp. 282-317 in: Rumelhart, D.E., and McClelland, J.L. (eds.), *Parallel Distributed Processing*, Vol. 1. The MIT Press, Cambridge, MA, 1986.

[10] Hopfield, J.J., and Tank, D.W., Neural computation of decisions in optimization problems. *Biological Cybernetics* 52 (1985), 141–152.

[11] Hrycej, T., Gibbs sampling in Bayesian networks. *Artificial Intelligence* 46 (1990), 351–363.

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

[13] Lauritzen, S.L. and Spiegelhalter, D.J., Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Stat. Soc., Ser. B* 50 (1988) 2, 157–224. Reprinted as pp. 415–448 in *Readings in Uncertain Reasoning*, edited by G.Shafer and J.Pearl. Morgan Kaufmann, San Mateo, CA, 1990.

[14] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, M.N. and Teller, E., Equations of state calculations by fast computing machines. *Journal of Chem. Phys.* 21 (1953), 1087–1092.

[15] Myllymäki, P., *Bayesian Reasoning by Stochastic Neural Networks*. Ph.Lic. Thesis, Department of Computer Science, University of Helsinki. Report C-1993-67, University of Helsinki, December 1993.

[16] Myllymäki, P., and Orponen, P., Programming the Harmonium. Pp. 671-677 in *Proceedings of the International Joint Conference on Neural Networks* (Singapore, November 1991), Vol. 1. IEEE, New York, NY, 1991.

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

[18] Neapolitan, R.E., *Probabilistic Reasoning in Expert Systems*. John Wiley & Sons, New York, NY, 1990.

[19] Orponen, P., Floréen, P., Myllymäki, P. and Tirri, H., A neural implementation of conceptual hierarchies with Bayesian reasoning. Pp. 297–303 in *Proceedings of the International Joint Conf. on Neural Networks* (San Diego, CA, June 1990), Vol. I. IEEE, New York, NY, 1990.

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

[21] Spiegelhalter, D.J., Probabilistic reasoning in predictive expert systems. Pp. 47–67 in *Uncertainty in Artificial Intelligence*, edited by L.N.Kanal and J.F.Lemmer. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 1986.