Bericht 96-29 des Fachbereichs Informatik der Technischen Universität Berlin

Decimatable Boltzmann Machines vs. Gibbs Sampling

Stefan Rüger[*], Anton Weinberger, and Sebastian Wittchen
Technische Universität Berlin

July 1996

Abstract

Exact Boltzmann learning can be done in certain restricted networks by the technique of decimation. We have enlarged the set of decimatable Boltzmann machines by introducing a new and more general decimation rule. We have compared solutions of a probability density estimation problem with decimatable Boltzmann machines to the results obtained by Gibbs sampling in unrestricted (non-decimatable) Boltzmann machines.

1 Introduction

Boltzmann machines [3] have been the first stochastical neural networks for which a learning algorithm [1] has been defined. This learning rule is based upon local interactions of nodes. Generally, their calculation is intractable in the number of hidden nodes. However, Gibbs sampling with simulated annealing can estimate them sufficiently precise. Many efforts have been made since then to overcome the — even for modern computers — comparatively slow Gibbs sampling of a Boltzmann machine. One especially interesting

[*]Sekr. FR 5-9, Franklinstr. 28-29, 10 587 Berlin, async@cs.tu-berlin.de

technique presented by Saul and Jordan [10] is decimation: The interaction of two nodes can be calculated exactly by reducing the rest of the network such that the interaction is not disturbed. This cannot be done in arbitrarily complex networks, but works, e. g., in tree-like structures (Boltzmann trees and chains), where the calculation time is polynomial in the number of nodes.

The idea of decimation can be applied to an even larger class of *decimatable Boltzmann machines* by a new decimation rule that is introduced in Section 2. The rest of the paper is organized as follows. In Section 3 we construct a pattern multi-set from a particular belief network that was introduced by Lauritzen and Spiegelhalter [5] as a didactical example. This example is not only rich enough to contain most of the interesting aspects of Boltzmann machines but also small enough to calculate all interesting properties exactly. In Section 4 we compare and assess several Boltzmann machines — decimatable and non-decimatable — by their ability to approximate the probability density given by the belief network. Finally, we discuss several possible extensions of our work.

2 Decimation in Boltzmann Machines

2.1 Boltzmann Machines

A Boltzmann machine consists of

- an undirected graph (N, E) without self-interaction, i. e., a set N of nodes and a non-empty symmetric set $E \subset N \times N$ of edges with $(a, b) \in E \Rightarrow (b, a) \in E$, and $a \in N \Rightarrow (a, a) \notin E$,
- a partition of N into input I, output O, and hidden nodes U,
- symmetric weights $w\colon E \to \mathbb{R}$ ($w_{ab} = w_{ba}$ for all $(a, b) \in E$),
- and stochastic bipolar activations $s\colon N \to \{-1, 1\}$

$$s_a = \begin{cases} +1 & \text{with probability } \sigma(\frac{2}{T} \sum_{b \in \{n \,|\, (a,n) \in E\}} w_{ab} s_b) \\ -1 & \text{else} \end{cases}$$

  for $a \neq 0$, where the special bias node 0 always has the activation 1. σ denotes the Fermi function $x \mapsto 1/(1 + \exp(-x))$. Here, the parameter $T \in \mathbb{R}^+$ acts as a temperature, i. e., controls the randomness of s.

The function $H\colon \mathbb{R}^E \times \{-1,1\}^N \to \mathbb{R}$

$$H(w,s) := -\frac{1}{2} \sum_{(a,b)\in E} w_{ab} s_a s_b \tag{1}$$

assigns an energy to every state s; the stochastic behavior of the network favors states of lower energy. When the network is in equilibrium, s follows a Boltzmann-Gibbs distribution [8]

$$P_w := s \mapsto \frac{\exp(-H(w,s)/T)}{Z_w}, \tag{2}$$

where Z_w, also known as the partition sum, is a normalization factor:

$$Z_w = \sum_{s\in\{-1,1\}^N} \exp(-H(w,s)/T) \tag{3}$$

The problem to train the Boltzmann machine is one of supervised learning. After clamping an input pattern to the input nodes, one wants to find the output nodes in a correlated target state during the equilibrium. Let $\gamma \in \{-1,1\}^O$ be a state of the output nodes, $\beta \in \{-1,1\}^U$ a state of the hidden nodes, and $\alpha \in \{-1,1\}^I$ a state of the input nodes. Then the state s of the Boltzmann machine may be denoted by $s := \alpha\beta\gamma \in \{-1,1\}^{I\cup U\cup O}$. The probability distribution of the visible nodes in equilibrium is given by the marginal distribution of the Boltzmann-Gibbs distribution with

$$p_w(\alpha\gamma) = \sum_\beta P_w(\alpha\beta\gamma).$$

Let r be the desired probability distribution. Then, with $q = \alpha \mapsto \sum_\gamma r(\alpha\gamma)$, a suitable cost function is the so-called information gain

$$E_w := \sum_\alpha q(\alpha) \sum_\gamma r(\gamma|\alpha) \log \frac{r(\gamma|\alpha)}{p_w(\gamma|\alpha)}. \tag{4}$$

Gradient descent $\Delta w_{ab} = -\eta \nabla_w E_w$ yields the Boltzmann learning rule [1, 8]

$$\Delta w_{ab} = \frac{\eta}{2T} \left(\overline{\langle s_a s_b \rangle_{\alpha\gamma \text{ clamped}}}^{r} - \overline{\langle s_a s_b \rangle_{\alpha \text{ clamped}}}^{q} \right). \tag{5}$$

$\overline{X}^r$ denotes the average of X with respect to r, which in turn is given empirically by a multi-set of input-output patterns $\alpha\gamma$. The terms in angle brackets represent expectation values with respect to the Boltzmann-Gibbs distribution (2); in the first term, both input and output nodes are clamped, whereas in the second term only the input nodes are clamped and the output nodes are allowed to equilibrate. Especially in larger networks it is impossible to compute the expectation values directly from the Boltzmann-Gibbs distribution. Instead, they can be estimated through Gibbs sampling. However, this method turns out to be very computation-intensive. In the next section the technique of decimation [2, 4, 10] will be presented that allows the exact and efficient calculation of expectation values in certain Boltzmann machines.

2.2 Decimation in Boltzmann Trees

We will describe the decimation rules proposed by [10] in a way that they are most easily extensible. The basic idea behind decimation is to transform networks into reduced ones without changing their properties. A useful reduction for Boltzmann machines is to eliminate a node, with the simpler network retaining the same Boltzmann distribution for the remaining nodes. This might be possible if all nodes which had an effective interaction mediated by the node to be decimated are connected by an additional edge. This ansatz is visualized in Figure 1, where node 1 is decimated by inserting additional edges between nodes that are mediated by the node to be decimated. It will turn out that this works if node 1 is connected to $n-1 \leq 3$ other nodes. Throughout this article we will consider effective weights $v_{ij} := w_{ij}/T$. The

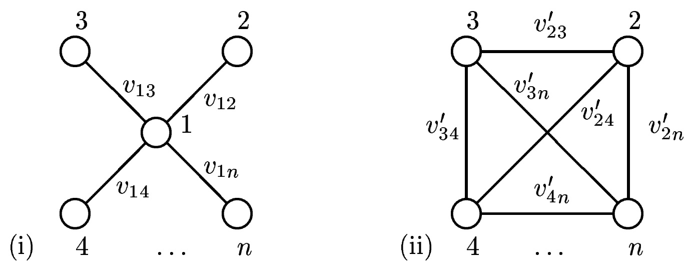


Figure 1: Ansatz for decimating a node

effective weights that are inserted by decimation can be computed from the constraint

$$\sum_{s_1} P_w^{(\mathrm{i})}(s_1, s_2, s_3, \dots) = P_{w'}^{(\mathrm{ii})}(s_2, s_3, \dots), \tag{6}$$

which simply requires that the distribution of the remaining nodes be a marginal distribution of all former nodes. Consider the case $n = 3$: three

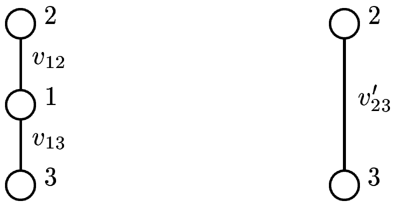


Figure 2: Combining weights in series

nodes 1, 2, and 3 are connected as shown on the left hand side of Figure 2. This network can be transformed to the simpler one on the right hand side of Figure 2. Applying Eq. (6), we get

$$\frac{\exp(v_{12}s_2 + v_{13}s_3)}{Z_w^{(\mathrm{i})}} + \frac{\exp(-v_{12}s_2 - v_{13}s_3)}{Z_w^{(\mathrm{i})}}. = \frac{\exp(v'_{23}s_2s_3)}{Z_{w'}^{(\mathrm{ii})}}$$

Note that nodes 2 and 3 might be embedded in a larger network without changing the result. The partition sum $Z_{w'}^{(\mathrm{ii})}$ depends on the effective weight v'_{23} in a subtle way. Considering relative probabilities, Eq. (6) changes into

$$\frac{\sum_{s_1} P_w^{(\mathrm{i})}(s_1, s_2, s_3, \dots)}{\sum_{s_1} P_w^{(\mathrm{i})}(s_1, 1, 1, \dots)} = \frac{P_{w'}^{(\mathrm{ii})}(s_2, s_3, \dots)}{P_{w'}^{(\mathrm{ii})}(1, 1, \dots)},$$

from which immediately follows:

$$\exp(v'_{23}(s_2s_3 - 1)) = \frac{\cosh(v_{12}s_2 + v_{13}s_3)}{\cosh(v_{12} + v_{13})} \tag{7}$$

Instantiating all possible values of $s_2, s_2 \in \{-1, 1\}$ and observing the symmetry of cosh, (7) collapses into a system of two equations for v'_{23}, one being trivially fulfilled. Hence, we have

$$v'_{23} = \log\left(\frac{\cosh(v_{12} + v_{13})}{\cosh(v_{12} - v_{13})}\right)/2, \tag{8}$$

which is equivalent to the result of [10]: $\tanh(v'_{23}) = \tanh(v_{12})\tanh(v_{13})$

Thus decimation can be used to combine weights in series. The much simpler case of combining parallel weights is illustrated in Figure 3: the effective weight is the sum of v_1 and v_2, which is a direct consequence of the linearity of the energy function in the weights. These two rules suffice

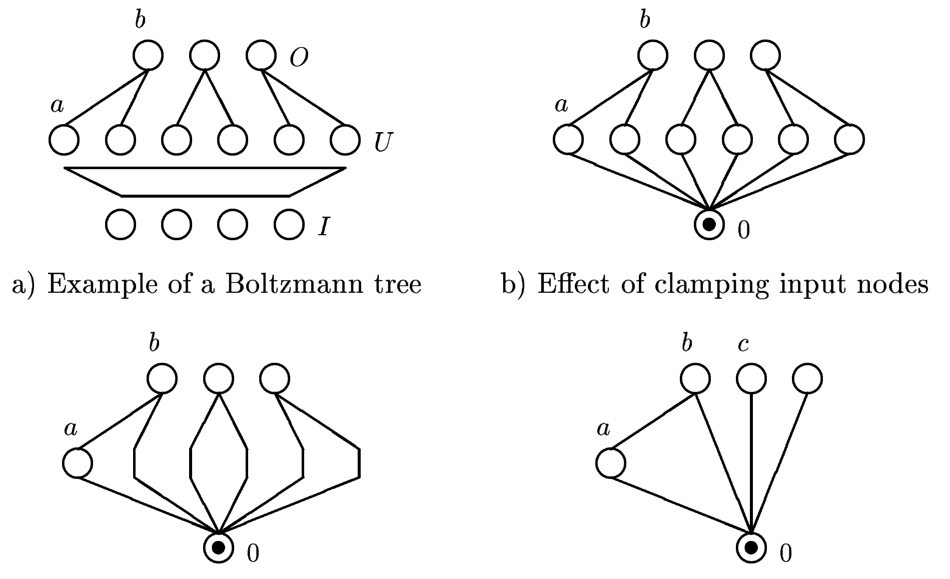
Figure 3: Combining parallel weights

to compute the expectation values for the weight updates in any tree-like network. Tree-like means that the output and hidden nodes are arranged hierarchically in one or more isolated trees. Typically (but not necessarily), each output node is the root of an isolated tree. An example with three output nodes is shown in Figure 4a.

There are no restrictions to the input nodes; they may be connected to all other nodes. We have chosen to fully connect the input nodes I with the hidden nodes U, which is indicated by the trapezium symbol in Figure 4a.

Now consider the weight between the nodes a and b in Figure 4a. According to Eq. (5), two different expectation values are needed for the weight update. Therefore, the expectation values are computed in two phases. In the first phase only the input nodes of a given pattern from the training multi-set are clamped.

Clamping a node a means subsuming its weights w_{ab} under the respective bias weights w_{ob}. The effect of clamping can be seen in Figure 4b. After clamping all input nodes, the network can be reduced by iterating the rules until only nodes a, b, and the bias 0 are left (Figure 5a). The intermediate stages are shown in Figure 4c and Figure 4d. Solitary nodes linked to the bias 0 may be simply dropped, as is done with node c in the transition from Figure 4d to Figure 5a: Imagine an edge with a vanishing weight between c and a; from Eq. (8) it follows that the weight v'_{oa} also vanishes after the decimation of node c.



a) Example of a Boltzmann tree b) Effect of clamping input nodes

c) After combining serial weights d) After combining parallel weights

Figure 4: Decimation of a Boltzmann tree

The Boltzmann-Gibbs distribution of the equivalent resulting network in Figure 5a is then tractable: a short calculation using Eqs. (2), (3), and (1) yields

$$\begin{aligned}\langle s_a s_b \rangle &= \sum_{s_a, s_b} P_w^{\mathrm{5a}}(s_a, s_b) s_a s_b \\ &= \frac{\exp(v_{ab})\cosh(v_{oa}+v_{ob}) - \exp(-v_{ab})\cosh(v_{oa}-v_{ob})}{\exp(v_{ab})\cosh(v_{oa}+v_{ob}) + \exp(-v_{ab})\cosh(v_{oa}-v_{ob})}.\end{aligned}$$

In the second phase both the input and the output nodes are clamped to the full network. Since s_b of our previous example is now a constant value, the computation of $\langle s_a s_b \rangle$ reduces to $\langle s_a \rangle s_b$. Again, $\langle s_a \rangle$ may be calculated by decimation of the original network to the equivalent network of Figure 5b. Here, we have

$$\langle s_a \rangle = \sum_{s_a} P_w^{\mathrm{5b}}(s_a) s_a = \tanh(v_{oa}).$$

Obviously, these rules are suitable for hierarchical networks with tree-like architectures (termed *Boltzmann trees* by Saul and Jordan). To allow

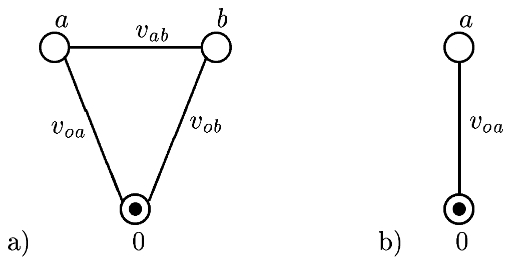


Figure 5: Reduced networks for the computation of $\langle s_a s_b \rangle$ and $\langle s_a \rangle$

for more complex, graph-like structures we have established a third rule for nodes connected to three other nodes.

2.3 Enhanced Decimation Rule

The general idea of decimation may be applied to $n = 4$ nodes; this situation is depicted in Figure 6. In analogy to the case of a node connected to two other nodes, Eq. (7), we get the following system of equations:

$$\sum_{1<a<b<5} v'_{ab}(1 - s_a s_b) = \log \left(\frac{\cosh \left(\sum_{a=2}^{4} v_{1a} \right)}{\cosh \left(\sum_{a=2}^{4} v_{1a} s_a \right)} \right)$$

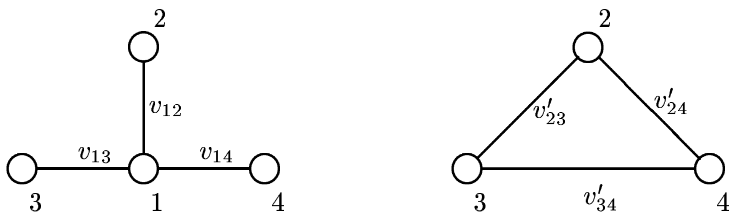


Figure 6: Decimating a node connected to three other nodes

The eight possible combinations of $s_2, s_3, s_4 \in \{-1, 1\}$ are reduced to four combinations by the symmetry $(s_2, s_3, s_4) \leftrightarrow (-s_2, -s_3, -s_4)$. The equation for $(s_2, s_3, s_4) = (1, 1, 1)$ is trivially fulfilled; three linear equations are left for v'_{23}, v'_{24}, and v'_{34}. From these equations the effective weights of the new edges can easily be determined:

$$\begin{aligned}
v'_{23} &= \log\left(\frac{\cosh(v_{12}+v_{13}-v_{14})\cosh(v_{12}+v_{13}+v_{14})}{\cosh(v_{12}-v_{13}+v_{14})\cosh(v_{12}-v_{13}-v_{14})}\right)/4 \\
v'_{24} &= \log\left(\frac{\cosh(v_{12}-v_{13}+v_{14})\cosh(v_{12}+v_{13}+v_{14})}{\cosh(v_{12}+v_{13}-v_{14})\cosh(v_{12}-v_{13}-v_{14})}\right)/4 \qquad (9)\\
v'_{34} &= \log\left(\frac{\cosh(v_{12}+v_{13}+v_{14})\cosh(v_{12}-v_{13}-v_{14})}{\cosh(v_{12}+v_{13}-v_{14})\cosh(v_{12}-v_{13}+v_{14})}\right)/4
\end{aligned}$$

The careful reader will note that Eqs. (9) collapse into the solution (8) for $n = 3$ nodes, if one of the effective weights $v_{12}, \dots, v_{14}$ vanishes. Boltzmann machines that (for every edge $(i, j) \in E$) can be reduced to a system with three nodes i, j, and 0 by the decimation rule (9) are termed *decimatable.*

This is the most general case, observing that the ansatz described in Figure 1 cannot be solved for $n > 4$ nodes. Clearly, Boltzmann trees are decimatable, since the Saul-Jordan decimation rule (8) is a special case of Eqs. (9). With the help of the enhanced decimation rule it becomes possible to decimate more complex graphs than Boltzmann trees like the architectures $D_1, \dots, D_6$ of Figure 8.

Decimatability is easily checked by subsuming clamped input nodes under bias weights and successively reducing the Boltzmann machine by rule (9) whenever a node has two or three neighbor nodes (including bias node 0) until nothing can be changed. The exact calculation of $\langle s_i s_j \rangle$ is then feasible by calculation of the Boltzmann-Gibbs distribution of an equivalent system with three nodes.

2.4 Calculating the Information Gain

Pure gradient descent is certainly not the fastest method of learning. Virtually any other method outperforms gradient descent, but needs access to not only the gradient of the cost function but also the cost function itself.

Decimation may help in calculating the information gain (4) in certain cases. We assume that the desired distribution r is given by m examples (α^1, γ^1), ..., (α^m, γ^m). Sorting these examples lexicographically, it is easy to determine $q(\alpha)$ as the number of examples whose first component is α

divided by m. $r(\gamma|\alpha)$ may then be set to the frequency of examples that coincide with (α, γ) divided by $q(\alpha)$. The evaluation of (4) is then reduced to the summation over αs and γs that occur in the training data. This leaves $p_w(\gamma|\alpha)$ to be determined.

Clamping α results in a change of the bias weights. The set $U \cup O \cup \{0\}$ of nodes remains. *If* this set can be completely reduced by decimation with an intermediate equivalent Boltzmann machine $\tilde{N} = O \cup \{0\}$, then $p_w(\gamma|\alpha)$ can be efficiently calculated as described below. Note that not all decimatable Boltzmann machines can be decimated by decimating the hidden nodes first, e. g., D_1, D_2, D_4, and D_6 of Figure 8 as opposed to D_3, D_5, and D_7. Let $\tilde{P}_{\tilde{w}}(\gamma) = \exp(-\tilde{H}(\tilde{w}, \gamma)/T)/\tilde{Z}_{\tilde{w}} = \sum_\beta P_w(\beta\gamma|\alpha)$ denote the Boltzmann-Gibbs distribution of $\tilde{N}$. Then we have

$$\log(p_w(\gamma|\alpha)) = \log(\tilde{P}_{\tilde{w}}(\gamma)) = -\tilde{H}(w, \gamma)/T - \log(\tilde{Z}_{\tilde{w}}).$$

The quantity $\tilde{Z}_{\tilde{w}}$ may now be calculated by decimation, again. Although decimation was made to leave the interactions between the remaining nodes invariant, the partition sum changes in every step of decimation. Eq. (6), for $n = 4$ written as

$$\sum_{s_1} \exp\left(s_1 \sum_{a=2}^{4} v_{1a} s_a\right) = \frac{Z_w^{(\mathrm{i})}}{Z_{w'}^{(\mathrm{ii})}} \exp\left(\sum_{1<a<b<5} v'_{ab} s_a s_b\right),$$

allows to determine the factor $Z_w^{(\mathrm{i})}/Z_{w'}^{(\mathrm{ii})}$ using the solution (9):

$$\frac{Z_w^{(\mathrm{i})}}{Z_{w'}^{(\mathrm{ii})}} = \begin{cases} 2\left(\prod_{s_3,s_4} \cosh(v_{12} + v_{13}s_3 + v_{14}s_4)\right)^{1/4} & \text{if } n = 4 \\ 2\left(\cosh(v_{12} + v_{13})\cosh(v_{12} - v_{13})\right)^{1/2} & \text{if } n = 3 \\ 2\cosh(v_{12}) & \text{if } n = 2 \\ 2 & \text{if } n = 1 \end{cases}$$

Adding all terms $\log(Z^{(\mathrm{i})}/Z^{(\mathrm{ii})})$ during decimation of the Boltzmann machine $\tilde{N}$ until the trivial Boltzmann machine $\{0\}$ with partition sum 1 is reached finally yields the quantity $\log(\tilde{Z}_{\tilde{w}})$.

Thus the cost function E_w may be determined in a tractable way. This allows to exploit many interesting modifications of gradient descent strategies like conjugate gradient, quasi-Newton methods [7] or stable dynamic parameter adaptation [9] in order to increase the speed of convergence.

3 Belief Networks and a Diagnosis Problem

A variable X in a probability space is a set containing mutually exclusive and exhaustive states. For the sake of clarity we restrict ourselves to two possible states, true and false. $P(A)$ denotes the probability that the event A is true. $P(\neg A) = 1 - P(A)$ denotes the probability that A is false.

As another convention we use a comma instead of the intersection symbol $\cap$. For example $P(\neg A|B, \neg C)$ is the probability that A is not true when we know that B is true and C is false. Our variables will be nodes in a directed acyclic graph and we are able to define a belief network:

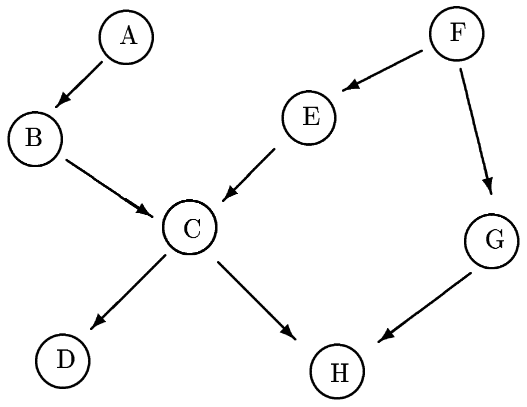
Definition (belief network) *Let X_i denote the event that node i represents a certain value x_i in a directed acyclic graph (DAG). For each X_i let* $\mathrm{par}(X_i)$ *be the set of events assigned to the parent nodes of i. The graph is a belief network, if the joint probability $P(X)$ is the product of the probabilities for every event given its parents:*

$$P(X_1, \ldots, X_n) = \prod_i^n P(X_i|\mathrm{par}(X_i))$$

That means, if one knows the conditional probability distribution of each variable given its parents, one can compute the joint probability distribution of all the variables in the network. This can enormously reduce the complexity of determining the distribution.

Now we introduce a fictitious medical example based on a belief network from [5]: Tuberculosis and lung cancer may be indicated by a positive chest X-ray. Both can cause shortness of breath (dyspnœa). The same is true for bronchitis. A recent visit to Asia increases the probability of tuberculosis, while smoking is a possible cause of both lung cancer and bronchitis. This situation is depicted in the DAG of Figure 7. We can create a belief network from this graph by assigning probabilities. For those who are interested in the details, we have used the following set of conditional probabilities: $P(a) = 0.01$, $P(b|a) = 0.05$, $P(b|\neg a) = 0.01$, $P(c|b, e) = 1$, $P(c|b, \neg e) = 1$, $P(c|\neg b, e) = 1$, $P(c|\neg b, \neg e) = 0$, $P(d|c) = 0.98$, $P(d|\neg c) = 0.05$, $P(e|f) = 0.1$, $P(e|\neg f) = 0.01$, $P(f) = 0.5$, $P(g|f) = 0.6$, $P(g|\neg f) = 0.3$, $P(h|c, g) = 0.9$, $P(h|c, \neg g) = 0.7$, $P(h|\neg c, g) = 0.8$, $P(h|\neg c, \neg g) = 0.1$

Because each variable has two alternatives, only one alternative has been listed. For example $P(\neg a)$ is 0.99, $P(\neg c|b, e)$ is 0 and so on. We can retrieve



a = person visited Asia
b = tuberculosis present
c = lung cancer or tuberculosis
d = positive X-ray
e = lung cancer present
f = person is a smoker
g = bronchitis present
h = dyspnea present

Figure 7: Dyspnœa example

all $2^8 = 256$ joint probabilities from the 18 numbers above by calculating

$$\begin{aligned} &P(A,B,C,D,E,F,G,H) = \\ &\qquad P(A)\,P(B|A)\,P(C|B,E)\,P(D|C)\,P(E|F)\,P(F)\,P(G|F)\,P(H|C,G). \end{aligned}$$

In medical practice, it is often difficult to decide whether a variable is the cause or the symptom of a disease. For an easier construction of a pattern multi-set for Boltzmann learning, we have divided the variables into input and output variables. We define A, D, F and H as input variables because they can be observed easily; moreover, they might be causes for a disease. B, E and G are our output patterns — the possible effects. C is neither a cause nor an effect; we model this variable by hidden nodes in the neural network. As opposed to many other pattern sets, we can not determine the output values exactly for given input values. For example, when we know that D, F, H appears and A is not present, we, e. g., know that the patient will have E and not both B and G with a probability of around 0.0134.

We will construct a pattern multi-set in the following way: The symptoms and causes are modeled by nodes with the state 1 when present. When they are not present, they are modeled by nodes with the state -1. The frequency of the patterns is determined by joint probabilities. For example the pattern $(a, d, f, h; b, e, g) = (-1, 1, 1, 1; -1, 1, -1)$ should appear with a frequency of around 0.0134 when we randomly draw a pattern from our pattern multi-set.

4 Simulations

Usually, a belief network models very precisely a joint probability density. However, in many real world applications, a good model of the probability distribution does not exist. Instead, we have a more or less large multi-set of examples. Imitating this, we have generated an artificial pattern multi-set from a precise model as described in the previous section. Our goal was to use Boltzmann machines to approximate the probability density given by the pattern multi-set.

We have used different types of architectures — decimatable as well as non-decimatable — in order to minimize the information gain (4). Depending on the architecture, the gradient of the information gain was either calculated exactly or estimated by Gibbs sampling with simulated annealing (the annealing schedule being 22 geometrical steps from $T = 10$ to the working temperature of $T = 1$).

It is possible to calculate the information gain E_w exactly in some networks in a tractable way as described in Section 2.4. However, in networks of small size, the information gain E_w can be calculated with brute force, too. Hence, in order to assess different architectures and in order to improve the overall speed of learning, we have used conjugate gradient utilizing the exact E_w. Our investigations have shown that this saves much learning time as opposed to pure gradient descent.

In all experiments the maximal number of iterations was 40. Most experiments settled in a local minimum of the information gain in considerably less than 40 epochs. In order to obtain significant statistics, we have repeated each type of experiment with different (random) starting weight vectors.

4.1 Training Results with Decimation

Table 1 shows some statistics of the training results, i. e., the information gain E_w, for nine different decimatable architectures (see Figure 8; D_8 and D_9 have the same architecture as D_5 and D_7 except that the inputs are also connected to the outputs). A small information gain indicates a good approximation, 0 being a perfect fit.

Many architectures exhibit a large variance of the information gain as can be seen, e. g., by a relative root mean square error of around 100%. This indicates a deficiency of the learning algorithm in connection with this architecture. As we can expect by the high empirical standard deviation, the median seems to be a more robust statistical measure than the mean.

A second hidden layer as in architecture D_2 and architecture D_6 reduces the number of outliers but worsens the information gain. The best results were obtained from architecture D_5 and architecture D_7, D_8, and D_9 with six decimatable hidden nodes in one layer (to keep these architectures decimatable, each hidden node is connected to no more than two nodes of the output layer). Not only the information gain achieved was minimal but also its variance was very small indicating that these architectures yield a consistent learning result. Surprisingly, additional edges between input and output nodes did not improve the information gain. We conjecture that six hidden nodes are enough to model the correlations between the visible nodes.

Table 1: Statistics of E_w obtained by exact learning (decimation)

Architecture	Median	Mean ± RMSE
D_1	0.034	0.078±0.089
D_2	0.168	0.127±0.089
D_3	0.046	0.102±0.097
D_4	0.033	0.044±0.047
D_5	0.015	0.015±0.004
D_6	0.074	0.132±0.116
D_7	0.017	0.017±0.001
D_8	0.016	0.016±0.002
D_9	0.018	0.018±0.001

4.2 Training Results with Gibbs Sampling

Table 2 shows twelve different architectures that have been trained with Gibbs sampling. Among those are architectures with 2, 3, 4, 5, and 6 hidden nodes in one layer: $D_1 = A_2$, A_3, A_4, A_5, and A_6; as indicated in Figure 8, neither the output nodes nor the hidden nodes have been connected with each other. H_3 is very similar to A_3 except that the hidden nodes are mutually connected. Finally, the architectures F_i ($i = 0, \ldots, 5$) represent the most typical kind of Boltzmann machine, where the set of four input, i hidden,

Table 2: Statistics of E_w obtained by Gibbs sampling

Architecture	Median	Mean $\pm$ RMSE
$D_1 = A_2$	0.040	0.074$\pm$0.082
A_3	0.037	0.052$\pm$0.056
A_4	0.024	0.029$\pm$0.018
A_5	0.021	0.023$\pm$0.007
A_6	0.022	0.023$\pm$0.006
H_3	0.067	0.148$\pm$0.146
F_o	0.118	0.115$\pm$0.051
F_1	0.081	0.101$\pm$0.053
F_2	0.099	0.106$\pm$0.050
F_3	0.077	0.081$\pm$0.033
F_4	0.086	0.112$\pm$0.066
F_5	0.102	0.103$\pm$0.047

and three output nodes are fully connected (except irrelevant edges between the input nodes).

The only decimatable architecture of Table 2 is $D_1 = A_2$ that also appears in Table 1. As expected, the results for this architecture obtained by decimation and Gibbs sampling agree within statistical significance. This is a strong indication that Gibbs sampling estimates the gradient of the information gain sufficiently precise.

The three architectures $A_3 \subset H_3 \subset F_3$ are contained in each other; one striking result is that the most special architecture A_3 yields the best result, although — in principle — the same result could have been obtained by the largest architecture F_3 by setting some weights to zero. Probably, additional weights create more local non-equivalent minima of the information gain.

The A-series of architectures yields its best results with five or six hidden nodes, while the F-series shows optimal behavior with three hidden nodes. Unfortunately, using a fully connected network with many hidden nodes does not seem to help in finding a good architecture. Weights that are not useful are not automatically set to zero. This is in accordance with the observation that, in general, the task of finding an optimal architecture is NP-hard [6].

5 Conclusions

Technically, decimation is much simpler to handle than Gibbs sampling: the interactions of nodes are simply calculated with an algorithm that is designed to yield exact results. With Gibbs sampling, parameters for the annealing schedule and for the sampling itself have to be determined. In practice, it can be a lengthy process to verify that a certain parameter set will help to estimate the interactions of nodes sufficiently precise. From our own experience, we conjecture that even the time needed for the actual process of Gibbs sampling with adequate parameters does not scale very well with the problem size.

The problem we have studied can be solved with several decimatable Boltzmann machines (D_5, D_7, D_8, and D_9), probably owing to special properties of the desired joint probability given by the underlying belief network. Any *missing* edge implies a certain stochastic conditional independence of the resulting Boltzmann-Gibbs distribution. One open question is, whether one can always find additional sparsely connected hidden nodes such that the Boltzmann machine remains decimatable and such that they help in approximating the desired probability distribution (as is the case when changing D_3 to D_5). Although the task of finding an *optimal* architecture may be intractable, as suggested by [6], one would like to have an algorithm that finds a *sufficiently good* decimatable Boltzmann architecture. These issues are left for further studies.

References

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] T. P. Eggarter. Cayley trees, the Ising problem, and the thermodynamic limit. *Physical Review B*, 9(7):2989–2992, 1974.

[3] Geoffrey E. Hinton and Terrence J. Sejnowski. Optimal perceptual inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 448–453. IEEE, 1983.

[4] C. Itzykson and J. Drouffe. *Statistical Field Theory.* Cambridge University Press, 1991.

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

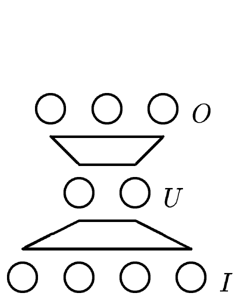
[6] Jyh-Han Lin and Jeffrey S. Vitter. Complexity issues in learning by neural nets. In R. Rivest, D. Haussler, and M. Warmuth, editors, *COLT89*, pages 118–132. Morgan Kaufmann Publishers, 1989.

[7] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1988.

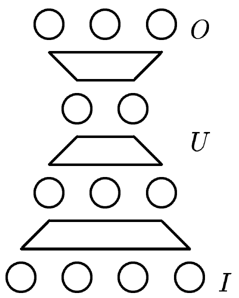
[8] Stefan M. Rüger. *Ausgewählte Kapitel der Theorie neuronaler Netze.* Lecture notes, Technische Universität Berlin, 1996.

[9] Stefan M. Rüger. Stable dynamic parameter adaptation. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 225–231. MIT Press, 1996.

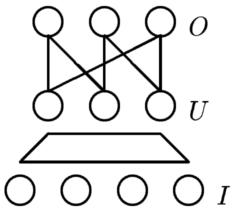
[10] Lawrence Saul and Michael I. Jordan. Learning in Boltzmann trees. *Neural Computation*, 6(6):1174–1184, 1994.



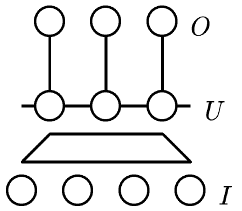
a) $D_1 = A_2$



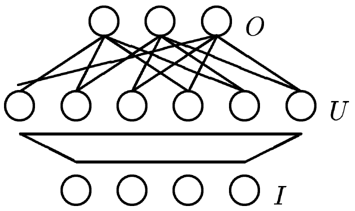
b) D_2



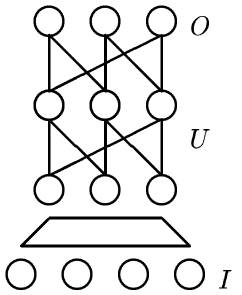
c) D_3



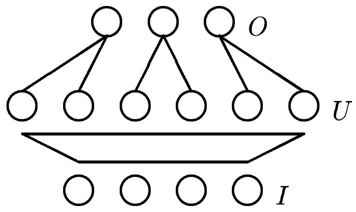
d) D_4



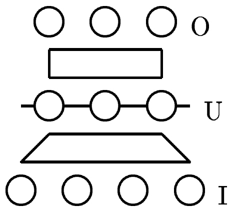
e) D_5



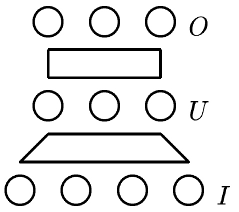
f) D_6



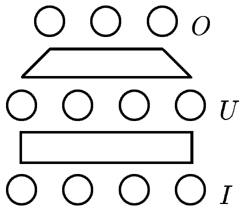
g) D_7



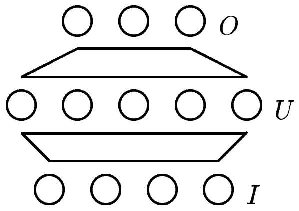
h) H_3



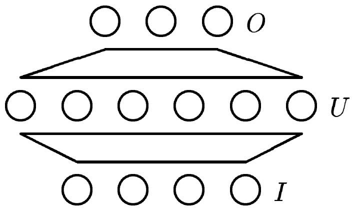
i) A_3



j) A_4



k) A_5



l) A_6

Figure 8: Some architectures ($D_1, \ldots, D_7$ are decimatable)