Learning in Boltzmann Trees

Lawrence Saul and Michael Jordan
Center for Biological and Computational Learning
Massachusetts Institute of Technology
79 Amherst Street, E10-243
Cambridge, MA 02139

January 31, 1995

Abstract

We introduce a large family of Boltzmann machines that can be trained using standard gradient descent. The networks can have one or more layers of hidden units, with tree-like connectivity. We show how to implement the supervised learning algorithm for these Boltzmann machines exactly, without resort to simulated or mean-field annealing. The stochastic averages that yield the gradients in weight space are computed by the technique of decimation. We present results on the problems of N-bit parity and the detection of hidden symmetries.

1 Introduction

Boltzmann machines (Ackley, Hinton, & Sejnowski, 1985) have several compelling virtues. Unlike simple perceptrons, they can solve problems that are not linearly separable. The learning rule, simple and locally based, lends itself to massive parallelism. The theory of Boltzmann learning, moreover, has a solid foundation in statistical mechanics. Unfortunately, Boltzmann machines—as originally conceived—also have some serious drawbacks. In practice, they are relatively slow. Simulated annealing (Kirkpatrick, Gellat, & Vecchi, 1983), though effective, entails a great deal of computation. Finally, compared to back-propagation networks (Rumelhart, Hinton, & Williams, 1986), where weight updates are computed by the chain rule, Boltzmann machines lack a certain degree of exactitude. Monte Carlo estimates of stochastic averages (Binder & Heerman, 1988) are not sufficiently accurate to permit further refinements to the learning rule, such as quasi-Newton or conjugate-gradient techniques (Press, Flannery, Teukolsky, & Vetterling 1986).

There have been efforts to overcome these difficulties. Peterson and Anderson (1987) introduced a mean-field version of the original Boltzmann learning

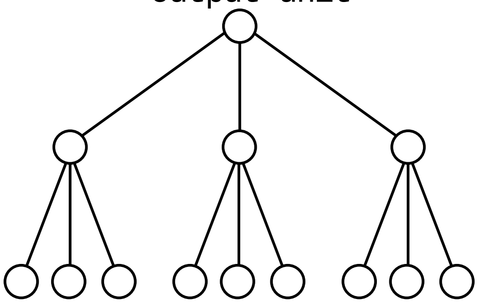

Figure 1: Boltzmann tree with two layers of hidden units. The input units (not shown) are fully connected to all the units in the tree.

rule. For many problems, this approximation works surprisingly well (Hinton, 1989), so that mean-field Boltzmann machines learn much more quickly than their stochastic counterparts. Under certain circumstances, however, the approximation breaks down, and the mean-field learning rule works badly if at all (Galland, 1993). Another approach (Hopfield, 1987) is to focus on Boltzmann machines with architectures simple enough to permit exact computations. Learning then proceeds by straightforward gradient descent on the cost function (Yair & Gersho, 1988), without the need for simulated or mean-field annealing. Hopfield (1987) wrote down the complete set of learning equations for a Boltzmann machine with one layer of non-interconnected hidden units. Freund and Haussler (1992) derived the analogous equations for the problem of unsupervised learning.

In this paper, we pursue this strategy further, concentrating on the case of supervised learning. We exhibit a large family of architectures for which it is possible to implement the Boltzmann learning rule in an exact way. The networks in this family have a hierarchical structure with tree-like connectivity. In general, they can have one or more layers of hidden units. We call them Boltzmann trees; an example is shown in Figure 1. We use a decimation technique from statistical physics to compute the averages in the Boltzmann learning rule. After describing the method, we give results on the problems of N-bit parity and the detection of hidden symmetries (Sejnowksi, Kienker, & Hinton, 1986). We also compare the performance of deterministic and true Boltzmann learning. Finally, we discuss a number of possible extensions to our work.

2 Boltzmann Machines

We briefly review the learning algorithm for the Boltzmann machine (Hertz, Krogh, and Palmer, 1991). The Boltzmann machine is a recurrent network with binary units $S_i = \pm 1$ and symmetric weights $w_{ij} = w_{ji}$. Each configuration of units in the network represents a state of energy

$$\mathcal{H} = -\sum_{ij} w_{ij} S_i S_j . \tag{1}$$

The network operates in a stochastic environment in which states of lower energy are favored. The units in the network change states with probability

$$P(S_i \to -S_i) = \frac{1}{1 + e^{\Delta \mathcal{H}/T}} . \tag{2}$$

Once the network has equilibrated, the probability of finding it in a particular state obeys the Boltzmann distribution from statistical mechanics:

$$P = \frac{1}{Z} e^{-\mathcal{H}/T} . \tag{3}$$

The partition function $Z = \sum e^{-\mathcal{H}/T}$ is the weighted sum over states needed to normalize the Boltzmann distribution. The temperature T determines the amount of noise in the network; as the temperature is decreased, the network is restricted to states of lower energy.

We consider a network with I input units, H hidden units, and O output units. The problem to be solved is one of supervised learning. Input patterns are selected from a training set with probability $P^*(I_\mu)$. Likewise, target outputs are drawn from a probability distribution $P^*(O_\nu \mid I_\mu)$. The goal is to teach the network the desired associations. Both the input and output patterns are binary. A particular example is said to be learned if after clamping the input units to the selected input pattern and waiting for the network to equilibrate, the output units are in the desired target states.

A suitable cost function for this supervised learning problem is

$$E = \sum_\mu P^*(I_\mu) \sum_\nu P^*(O_\nu \,|\, I_\mu) \ln \left[\frac{P^*(O_\nu \,|\, I_\mu)}{P(O_\nu \,|\, I_\mu)} \right] \tag{4}$$

where $P^*(O_\nu \mid I_\mu)$ and $P(O_\nu \mid I_\mu)$ are the desired and observed probabilities that the output units have pattern O_ν when the input units are clamped to pattern I_μ. The Boltzmann learning algorithm attempts to minimize this cost function by gradient descent. The calculation of the gradients in weight space is straightforward. The final result is the Boltzmann learning rule

$$\Delta w_{ij} = \frac{\eta}{T} \sum_\mu P^*(I_\mu) \sum_\nu P^*(O_\nu | I_\mu) \left[\langle S_i S_j \rangle^\mu_{I,O} - \langle S_i S_j \rangle^\mu_I \right] , \tag{5}$$

where brackets $\langle \cdots \rangle$ indicate expectation values over the Boltzmann distribution. The gradients in weight space depend on two sets of correlations—one in which the output units are clamped to their desired targets O_ν, the other in which they are allowed to equilibrate. In both cases, the input units are clamped to the pattern I_μ being learned. The differences in these correlations, averaged over the examples in the training set, yield the weight updates Δw_{ij}. An on-line version of Boltzmann learning is obtained by foregoing the average over input patterns and updating the weights after each example. Finally, the parameter η sets the learning rate.

The main drawback of Boltzmann learning is that, in most networks, it is not possible to compute the gradients in weight space directly. Instead, one must resort to estimating the correlations $\langle S_i S_j \rangle$ by Monte Carlo simulation (Binder *et al*, 1988). The method of simulated annealing (Kirkpatrick *et al*, 1983) leads to accurate estimates but has the disadvantage of being very computation-intensive. A mean-field version of the algorithm (Peterson & Anderson, 1987) was proposed to speed up learning. It makes the approximation $\langle S_i S_j \rangle \approx \langle S_i \rangle \langle S_j \rangle$ in the learning rule and estimates the magnetizations $\langle S_i \rangle$ by solving a set of nonlinear equations. This is done by iteration, combined when necessary with an annealing process. So-called mean-field annealing can yield an order-of-magnitude improvement in convergence. Clearly, however, the ideal algorithm would be one that computes expectation values exactly and does not involve the added complication of annealing. In the next section, we investigate a large family of networks amenable to exact computations of this sort.

3 Boltzmann Trees

A Boltzmann tree is a Boltzmann machine whose hidden and output units have a special hierarchical organization. There are no restrictions on the input units, and in general, we will assume them to be fully connected to the rest of the network. For convenience, we will focus on the case of one output unit; an example of such a Boltzmann tree is shown in Figure 1. Modifications to this basic architecture and the generalization to many output units will be discussed later.

The key technique to compute partition functions and expectation values in these trees is known as decimation (Eggarter, 1974; Itzykson & Drouffe, 1991). The idea behind decimation is the following. Consider three units connected in series, as shown in Figure 2a. Though not directly connected, the end units S_1 and S_2 have an effective interation that is mediated by the middle one S. Define the temperature-rescaled weights $J_{ij} \equiv w_{ij}/T$. We claim that the combination of the two weights J_1 and J_2 in series has the same effect as a single weight J. Replacing the weights in this way, we have integrated out, or "decimated", the degree of freedom represented by the intermediate unit. To derive an expression for J, we require that the units S_1 and S_2 in both systems obey the same Boltzmann distribution. This will be true if

$$\sum_{S=\pm 1} e^{J_1 S_1 S + J_2 S S_2} = \sqrt{C} e^{J S_1 S_2}, \tag{6}$$

where C is a constant prefactor, independent of S_1 and S_2. Enforcing equality for the possible values of $S_1 = \pm 1$ and $S_2 = \pm 1$, we obtain the constraints

$$\sqrt{C} e^{\pm J} = 2\cosh(J_1 \pm J_2).$$

It is straightforward to eliminate C and solve for the effective weight J. Omitting the algebra, we find

$$\tanh J = \tanh J_1 \tanh J_2. \tag{7}$$

Choosing J in this way, we ensure that all expectation values involving S_1 and/or S_2 will be the same in both systems.

Decimation is a technique for combining weights "in series". The much simpler case of combining weights "in parallel" is illustrated in Figure 2b. In this case, the effective weight is simply the additive sum of J_1 and J_2, as can be seen by appealing to the energy function of the network, eq. (1). Note that the rules for combining weights in series and in parallel are valid if either of the end units S_1 or S_2 happen to be clamped. They also hold locally for weight combinations that are embedded in larger networks. The rules have simple analogs in other types of networks (e.g. the law for combining resistors in electric circuits). Indeed, the strategy for exploiting these rules is a familiar one. Starting with a complicated network, we iterate the rules for combining weights until we have a simple network whose properties are easily computed. Clearly, the rules do not make all networks tractable; networks with full connectivity between hidden units, for example, cannot be systematically reduced. Hierarchical networks with tree-like connectivity, however, lend themselves naturally to these types of operations.

Let us see how we can use these rules to implement the Boltzmann learning rule in an exact way. Consider the two-layer Boltzmann tree in Figure 1. The effect of clamping the input units to a selected pattern is to add a bias to each of the units in the tree, as in Figure 3a. Note that these biases depend not only on the input weights, but also on the pattern distributed over the input units. Having clamped the input units, we must now compute expectation values. For concreteness, we consider the case where the output unit is allowed to equilibrate. Correlations between adjacent units are computed by decimating over the other units in the tree; the procedure is illustrated in Figure 3b for the lower leftmost hidden units. The final, reduced network consists of the two adjacent units with weight J and effective biases (h_1, h_2). A short calculation gives

$$\langle S_1 S_2 \rangle = \frac{e^J \cosh(h_1 + h_2) - e^{-J}\cosh(h_1 - h_2)}{e^J \cosh(h_1 + h_2) + e^{-J}\cosh(h_1 - h_2)}. \tag{8}$$

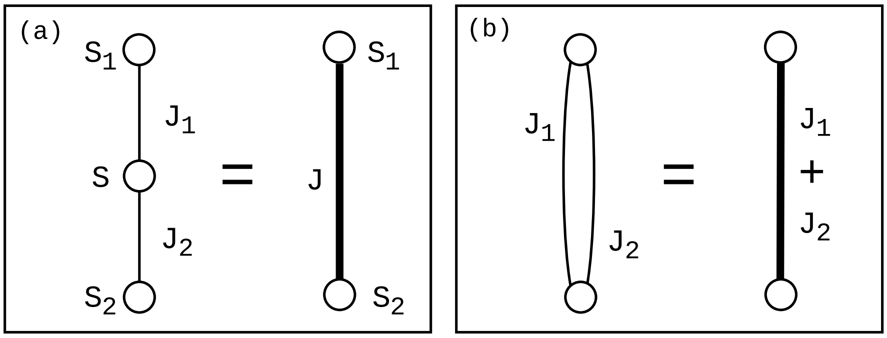

Figure 2: (a) Combining weights in series: the effective interaction between units S_1 and S_2 is the same as if they were directly connected by weight J, where $\tanh J = \tanh J_1 \tanh J_2$. (b) Combining weights in parallel: the effective weight is simply the additive sum. The same rules hold if either of the end units is clamped.

The magnetization of a tree unit can be computed in much the same way. We combine weights in series and parallel until only the unit of interest remains, as in Figure 3c. In terms of the effective bias h, we then have the standard result

$$\langle S_1 \rangle = \tanh h. \tag{9}$$

The rules for combining weights thus enable us to compute expectation values without enumerating the $2^{13} = 8192$ possible configurations of units in the tree. To compute the correlation $\langle S_1 S_2 \rangle$ for two adjacent units in the tree, one successively removes all "outside" fluctuating units until only units S_1 and S_2 remain. To compute the magnetization $\langle S_1 \rangle$, one removes unit S_2 as well.

Implementing these operations on a computer is relatively straightforward, due to the hierarchical organization of the output and hidden units. The entire set of correlations and magnetizations can be computed by making two recursive sweeps through the tree, storing effective weights as necessary to maximize the efficiency of the algorithm. Having to clamp the output unit to the desired target does not introduce any difficulties. In this case, the output unit merely contributes (along with the input units) to the bias on its derivative units. Again, we use recursive decimation to compute the relevant stochastic averages. We are thus able to implement the Boltzmann learning rule in an exact way.

4 Results

We tested Boltzmann trees on two familiar problems: N-bit parity and the detection of hidden symmetries (Sejnowski *et al*, 1986). We hope our results

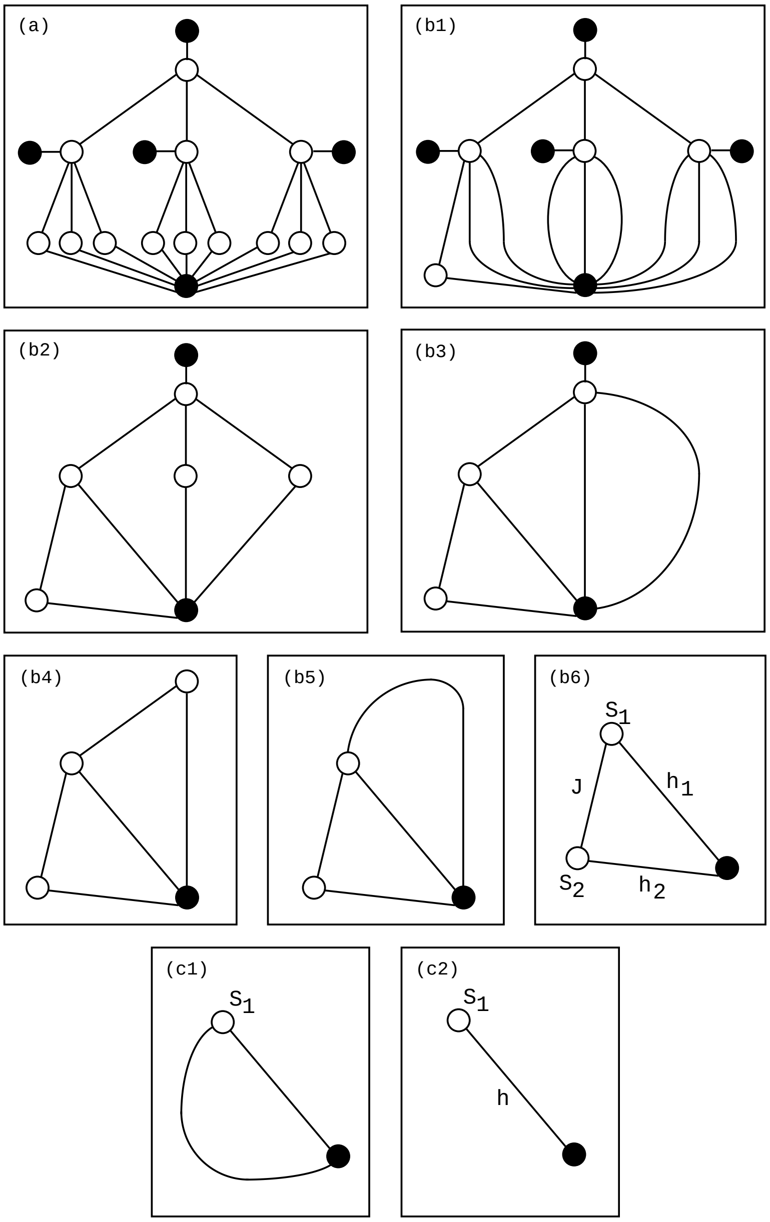

Figure 3: Reducing Boltzmann trees by combining weights in series and parallel. Solid circles represent clamped units. (a) Effect of clamping the input units to a selected pattern. (b) Computing the correlation between adjacent units. (c) Computing the magnetization of a single unit.

N	hidden units	e_{max}	success %	e_{avg}
2	1	50	97.2 (89.3)	25.8
3	1	250	96.1 (88.5)	42.1
4	3	1000	95.1 (69.2)	281.1
5	4	1000	92.9 (84.2)	150.0

Table 1: Boltzmann tree performance on N-bit parity. The results in parentheses are for mean-field learning.

demonstrate not only the feasibility of the algorithm, but also the potential of exact Boltzmann learning. Table 1 shows our results on the N-bit parity problem, using Boltzmann trees with one layer of hidden units. In each case, we ran the algorithm 1000 times. All 2^N possible input patterns were included in the training set. A success indicates that the tree learned the parity function in less than e_{max} epochs. We also report the average number of epochs e_{avg} per successful trial; in these cases, training was stopped when $P(O^*|I_\mu) \geq 0.9$ for each of the 2^N inputs, with $O^* = \text{parity}(I_\mu)$. The results show Boltzmann trees to be competitive with standard back-propagation networks (Møller, 1993).

We also tested Boltzmann trees on the problem of detecting hidden symmetries. In the simplest version of this problem, the input patterns are square pixel arrays which have mirror symmetry about a fixed horizontal or vertical axis (but not both). We used a two-layer tree with the architecture shown in Figure 1 to detect these symmetries in 10×10 square arrays. The network learned to differentiate the two types of patterns from a training set of 2000 examples. After each epoch, we tested the network on a set of 200 unknown examples. The performance on these patterns measures the network's ability to generalize to unfamiliar inputs. The results, averaged over 100 separate trials, are shown in Figure 4. After 100 epochs, average performance was over 95% on the training set and over 85% on the test set.

Finally, we investigated the use of the deterministic, or mean-field, learning rule (Peterson & Anderson, 1987) in Boltzmann trees. We repeated our experiments, substituting $\langle S_i\rangle\langle S_j\rangle$ for $\langle S_i S_j\rangle$ in the update rule. Note that we computed the magnetizations $\langle S_i\rangle$ exactly using decimation. In fact, in most deterministic Boltzmann machines, one does not compute the magnetizations exactly, but estimates them within the mean-field approximation. Such networks therefore make two approximations—first, that $\langle S_i S_j\rangle \approx \langle S_i\rangle\langle S_j\rangle$ and second, that $\langle S_i\rangle \approx \tanh(\sum J_{ij}\langle S_j\rangle + h_i)$. Our results speak to the first of these approximations. At this level alone, we find that exact Boltzmann learning is perceptibly faster than mean-field learning. On one problem in particular, that of $N = 4$ parity (see Table I), the difference between the two learning schemes was quite pronounced.

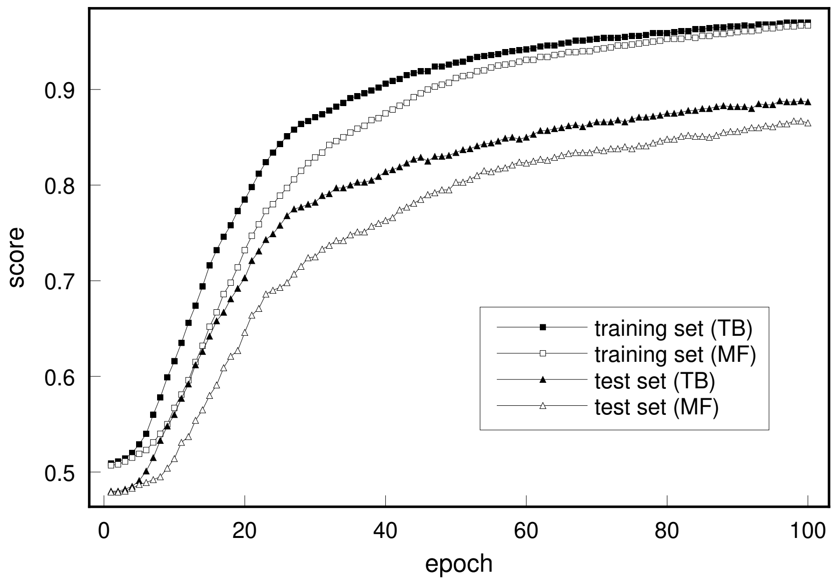

Figure 4: Results on the problem of detecting hidden symmetries for true Boltzmann (TB) and mean-field (MF) learning.

5 Extensions

In conclusion, we mention several possible extensions to the work in this paper. Clearly, a number of techniques used in back-propagation networks, such as conjugate-gradient and quasi-Newton methods (Press *et al*, 1986), could also be used to accelerate learning in Boltzmann trees. In this paper, we have considered the basic architecture in which a single output unit sits atop a tree of one or more hidden layers. Depending on the problem, a variation on this architecture may be more appropriate. The network must have a hierarchical organization to remain tractable; within this framework, however, the algorithm permits countless arrangements of hidden and output units. In particular, a tree can have one or more output units, and these output units can be distributed in an arbitrary way throughout the tree. One can incorporate certain intralayer connections into the tree at the expense of introducing a slightly more complicated decimation rule, valid when the unit to be decimated is biased by a connection to an additional clamped unit. There are also decimation rules for q-state (Potts) units, with $q > 2$ (Itzykson & Drouffe, 1991).

The algorithm for Boltzmann trees raises a number of interesting questions. Some of these involve familiar issues in neural network design—for instance, how to choose the number of hidden layers and units. We would also like to characterize the types of learning problems best-suited to Boltzmann trees. A recent study by Galland (1993) suggests that mean-field learning has trouble in networks with several layers of hidden units and/or large numbers of output units. Boltzmann trees with exact Boltzmann learning may present a viable option for problems in which the basic assumption behind mean-field learning—that the units in the network can be treated independently—does not hold. We know of constructive algorithms (Frean, 1990) for feed-forward nets that yield tree-like solutions; an analogous construction for Boltzmann machines has obvious appeal, in view of the potential for exact computations. Finally, the tractability of Boltzmann trees is reminiscent of the tractability of tree-like belief networks, proposed by Pearl (1986, 1988); more sophisticated rules for computing probabilities in belief networks (Lauritzen & Spiegelhalter, 1988) may have useful counterparts in Boltzmann machines. These issues and others are left for further study.

Acknowledgements

The authors thank Mehran Kardar for useful discussions. This research was supported by the Office of Naval Research and by the MIT Center for Materials Science and Engineering through NSF Grant DMR-90-22933.

References

Ackley, D.H., Hinton, G.E., and Sejnowski, T.J. (1985), A Learning Algorithm for Boltzmann Machines, *Cognitive Science* **9**, 147-169

Binder, K., and Heerman, D.W. (1988), *Monte Carlo Simulation in Statistical Mechanics*, Berlin: Springer-Verlag.

Eggarter, T.P. (1974), Cayley Trees, the Ising Problem, and the Thermodynamic Limit, *Physical Review B* **9**, 2989-2992.

Frean, M. (1990), The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks, *Neural Computation* **2**, 198-209.

Freund, Y. and Haussler, D. (1992), Unsupervised Learning of Distributions on Binary Vectors using Two Layer Networks, In *Advances in Neural Information Processing Systems IV* (Denver 1992), ed. J. E. Moody, S. J. Hanson, and R. P. Lippman, 912-919. San Mateo: Morgan Kaufman.

Galland, C. C. (1993), The Limitations of Deterministic Boltzmann Machine Learning, *Network: Computation in Neural Systems* **4**, 355-379.

Hertz, J., Krogh, A., and Palmer, R.G. (1991), *Introduction to the Theory of Neural Computation*, Redwood City: Addison-Wesley.

Hinton, G.E. (1989), Deterministic Boltzmann Learning Performs Steepest Descent in Weight Space, *Neural Computation* **1**, 143-150.

Hopfield, J.J. (1987), Learning Algorithms and Probability Distributions in Feed-Forward and Feed-Back Networks, *Proceedings of the National Academy of Sciences, USA* **84**, 8429-8433.

Itzykson, C. and Drouffe, J. (1991), *Statistical Field Theory*, Cambridge: Cambridge University Press.

Kirkpatrick, S., Gellatt Jr, C.D. and Vecchi, M.P. (1983), Optimization by Simulated Annealing, *Science* **220**, 671-680.

Lauritzen, S. L. and 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**, 157-224.

Møller, M. F. (1993), A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, *Neural Networks* **6**, 525-533.

Pearl, J. (1986), Fusion, Propagation, and Structuring in Belief Networks, *Artificial Intelligence* **19**, 241-288.

Pearl, J. (1988), *Probabilistic Reasoning in Intelligent Systems*, San Mateo: Morgan Kauffman.

Peterson, C. and Anderson, J.R. (1987), A Mean Field Theory Learning Algorithm for Neural Networks, *Complex Systems* **1**, 995-1019.

Press, W.H., Flannery, B.P., Teukolsky, S.A, and Vetterling, W.T. (1986), *Numerical Recipes*, Cambridge: Cambridge University Press.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986), Learning Representations by Back-Propagating Errors, *Nature* **323**, 533-536.

Sejnowski, T.J. Kienker, P.K., and Hinton, G.E. (1986), Learning Symmetry Groups with Hidden Units, *Physica* **22D**, 260-275.

Yair, E. and Gersho, A. (1988), The Boltzmann Perceptron Network: A Multi-Layered Feed-Forward Network Equivalent to the Boltzmann Machine, In *Advances in Neural Information Processing Systems I* (Denver 1988), ed. D.S. Touretzky, 116-123. San Mateo: Morgan Kaufman.