Mean Field Theory Neural Networks for Feature Recognition, Content Addressable Memory and Optimization

CARSTEN PETERSON

Various applications of the mean field theory (MFT) technique for obtaining solutions close to optimal minima in feedback networks are reviewed. Using this method in the context of the Boltzmann machine gives rise to a fast deterministic learning algorithm with a performance comparable with that of the backpropagation algorithm (BP) in feature recognition applications. Since MFT learning is bidirectional its use can be extended from purely functional mappings to a content addressable memory. The storage capacity of such a network grows like O *(10–20)*n_H *with the number of hidden units. The MFT learning algorithm is local and thus it has an advantage over BP with respect to VLSI implementations. It is also demonstrated how MFT and BP are related in situations where the number of input units is much larger than the number of output units. In the context of finding good solutions to difficult optimization problems the MFT technique again turns out to be extremely powerful. The quality of the solutions for large travelling salesman and graph partition problems are in parity with those obtained by optimally tuned simulated annealing methods. The algorithm employed here is based on multistate* K*-valued (*$K > 2$*) neurons rather than binary (*$K = 2$*) neurons. This method is also advantageous for more nested decision problems like scheduling. The MFT equations are isomorfic to resistance–capacitance equations and hence naturally map onto custom-made hardware. With the diversity of successful application areas the MFT approach thus constitutes a convenient platform for hardware development.*

1. Introduction

Present neural network model architectures for feature recognition applications fall into two broad categories, feed-forward and feedback. The notion of final state is very different for the two approaches. In feed-forward or perceptron-like models information is processed from the input layer through the hidden units to the output layer where the information is compared with some target value. This stage represents the final state. Backpropagation (BP) [1] is the most widely used learning algorithm for this kind of architecture. In contrast, a feedback algorithm is

Carsten Peterson, Department of Theoretical Physics, University of Lund, Sölvegatan 14A, S-22362 Lund, Sweden. thepcap@seldc52 (bitnet); carsten@thep.lu.se (internet).

characterized by a dynamics where all units settle to a fixed point[1] in a relaxation process common to that of physical processes with many degrees of freedom. The Boltzmann machine (BZ) [2] and mean field theory (MFT) learning [3,4] are the most promising learning algorithms for this kind of architecture. Being a stochastic algorithm BZ is very time consuming on the simulation level and has therefore not been fully explored for larger problem sizes. The stochastic process in the BZ can be emulated by a set of deterministic equations of motions in the so-called MFT approximation, which give rise to a fast learning algorithm for feedback networks. In this paper we review this promising alternative to BP both with respect to derivations and to numerical explorations of two feature recognition benchmarks [4]. The performance of MFT is in parity with or slightly better than BP.

Being a feedback supervised learning algorithm, MFT turns out to be very powerful in content addressable memory applications. In ref. [4] it was demonstrated that a fully connected network of n_V visible and n_H hidden units could store around $6n_H$ random patterns. Very recently, $O(10\text{–}20)n_H$ or possibly a superlinear capacity in terms of the number of hidden nodes was reached with MFT learning using a more restricted architecture [5, 6]. These capacities should be compared with $0.14N$ for the Hopfield model [7].

MFT has the advantage that a VLSI implementation is natural; not only are the MFT equations isomorphic to resistance–capacitance (RC) equations for a circuitry of analog amplifiers but also the learning (or updating of weights) is local. This is in contrast to BP where the learning is non-local since the error has to be propagated. However, for architectures where the number of input nodes is much larger than the number of output nodes, BP with the entropy error is more or less equal to MFT and can thus in those situations easily be mapped onto VLSI. Also, the equations of motions used in MFT are identical to those obtained when mapping optimization problems onto neural networks (see below). Efforts in building hardware might therefore yield multiple payoffs.

The neural network paradigm has also shown great promise for finding approximate solutions to difficult optimization problems [8]. The approach here is based on feedback network architecture together with the MFT approximation. Since the appearance of the pioneering work by Tank and Hopfield [8], which contained only small-sized testbeds, it has been questioned whether the promise of this approach will survive more realistic problem sizes. In a recent work [9] it was demonstrated that large problem sizes can be successfully handled with no parameter fine tuning provided an alternative encoding scheme is employed, which is based on extending the neurons from being binary ($K=2$) to K-valued ($K>2$). Correspondingly the non-linear gain functions are generalized from being logistic functions to probabilistic functions. A recipe for estimating the crucial phase transition temperature parameter in advance also turns out to be important for a reliable performance of the algorithm. The K-valued approach turns out to be extremely convenient and efficient for solving scheduling problems [10].

This paper is organized as follows. In Section 2 we give a brief and simple description of the MFT approximation in general terms. Section 3 contains a derivation of the MFT learning algorithm and in Section 4 two benchmarks against BP are presented. In Section 4 we also discuss the relation between BP and MFT. The MFT approach to content addressable memories (CAMs) can be found in Section 5. In Section 6 we briefly review the recent progress with using the novel encoding for optimization and decision problems. Finally, in Section 7, we give a brief summary.

2. The MFT Approximation

Here we give a simple description of the MFT approximation. For a more detailed treatment, refer to refs [4] and [11] and to the original work in the context of magnetic materials [12].

Consider the Hopfield model [7] as defined by the energy function

$$E(\boldsymbol{S}) = -\frac{1}{2}\sum_i \sum_j T_{ij} S_i S_j \tag{1}$$

where the neurons are binary, $S_i = \pm 1$. With the 'local fields' U_i given by

$$U_i = -\frac{\partial E}{\partial S_i} = \sum_j T_{ij} S_j \tag{2}$$

the corresponding updating equations read

$$S_i = \mathrm{sgn}(U_i) \tag{3}$$

Equation (3) is based upon gradient descent on the energy of equation (1) and is hence suitable for content addressable memory applications; given an initial configuration it takes us to the closest local minimum. Other neural network applications like optimization problems and feature recognition tasks in the BZ framework require that the global minimum of equation (1) is reached. In those instances one wants to avoid local minima. There exist different stochastic procedures for performing the hill-climbing necessary for avoiding getting stuck in such local minima. A frequently used scheme is *simulated annealing* [13], where fluctuations of the energy of equation (1) are allowed according to the Boltzmann distribution

$$P(\boldsymbol{S}) = \frac{1}{Z} \mathrm{e}^{-E(\boldsymbol{S})/T} \tag{4}$$

$$Z = \sum_{\boldsymbol{S}} \mathrm{e}^{-E(\boldsymbol{S})/T} \tag{5}$$

In equations (4) and (5) the 'temperature' T has been introduced, which sets the magnitude of the fluctuations. Thus the probability for a particular neuron i to be 'on' and 'off' is given by

$$P(S_i = \pm 1) = \frac{\mathrm{e}^{\pm U_i/T}}{\mathrm{e}^{+U_i/T} + \mathrm{e}^{-U_i/T}} \tag{6}$$

Several algorithms exist (Heat bath, Metropolis, Langevin, etc.) for generating configurations obeying equation (4). Needless to say these are all very time consuming. The MFT approximation aims at replacing this stochastic process by a set of deterministic equations that capture the notion of noise. The key idea in this approximation is to approximate the local field U_i by its thermal average

$$U_i^{\mathrm{MFT}} = \sum_i T_{ij} V_j \tag{7}$$

where

$$V_i = \langle S_j \rangle_T \tag{8}$$

is the average of S_i at temperature T. In this approximation one then obtains

$$V_i = P(S_i = +1) - P(S_i = -1) = \tanh(U_i^{\mathrm{MFT}}/T) \tag{9}$$

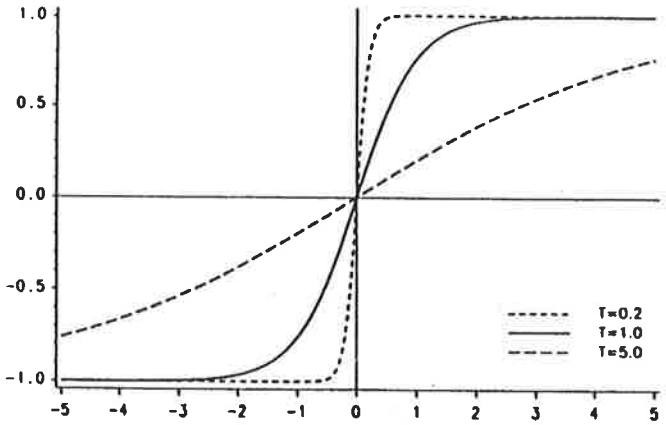


Figure 1. Sigmoid gain functions of equation (10) for different temperatures T. The step function updating rule of equation (3) corresponds to $T=0$.

or

$$V_i=\tanh\left(\sum_j T_{ij}V_j/T\right) \tag{10}$$

In other words the stochastic process is being emulated by a set of deterministic equations with sigmoid gain functions (see Figure 1). At $T=0$ the Hopfield step function updating rule of equation (3) is restored (see Figure 1). The validity of this approximation depends on the degree of connectivity. In magnetic systems it works quite well for infinite-range systems whereas for locally connected systems it is quite poor [14]. In neural network applications the approximation works amazingly well even in situations where the connectivity is limited [3].

The MFT equations (equation (10)) have a very important property from a VLSI implementation point of view. They are identical to the static limit of the RC charging equations of a circuit of analogue amplifiers [15, 8]. Consider a circuit of non-linear amplifiers, which convert an input voltage U_i to an output voltage $V_i=\tanh(U_i/T)$ and which are connected with resistors T_{ij}^{-1} between amplifiers i and j (see Figure 2).[2] The steady state circuit equations are then given by

$$\frac{U_i}{R_i}=\sum_j T_{ij}(V_j-U_i)=0 \tag{11}$$

where R_i are input resistors for the amplifiers leading to reference ground. With the replacement

$$T_{ij}\to\left(\frac{1}{R_i}-\sum_k T_{ik}\right)^{-1} T_{ij} \tag{12}$$

and substituting $T\tanh^{-1}(V_i)$ for U_i equation (11) is identical to equation (10). This close mapping between feedback neural models and VLSI circuitry is indeed very appealing.

Equation (10) is solved by iteration on the simulation level. It is well known that such a procedure might give rise to oscillatory behaviour, in particular for synchronous updating. In Section 6 we briefly discuss how to consistently avoid oscillatory behaviour by employing a systematic scheme for choosing parameters.

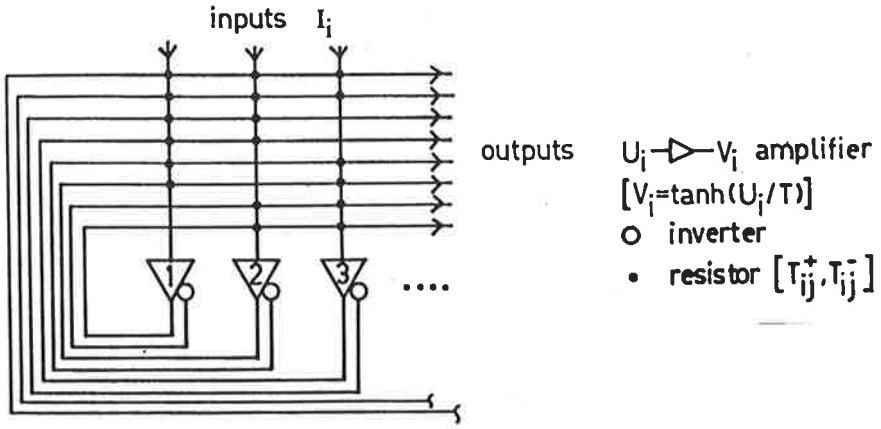


Figure 2. A VLSI implementation of a fully connected network.

3. MFT Learning

3.1. The Boltzmann Machine

The Boltzmann machine [2] is a learning algorithm originally designed for architectures containing hidden units.[3] The dynamics is based on the Hopfield energy function (equation (1)). The model learns by making an internal representation of its environment. The learning procedure changes weights so as to minimize the distance between two probability distributions, as measured by the G function or the so-called Kullback measure [16]

$$G=\sum_{\alpha} P_\alpha \log\left(\frac{P_\alpha}{P'_\alpha}\right) \tag{13}$$

where P_α is the probability that the visible units are collectively in state α when their states are determined by the environment. P_α represents the *desired* probabilities for these states. The corresponding probabilities when the network runs freely are denoted P'_α. G is zero if and only if the distributions are identical; otherwise it is positive. The word 'free' either means that all visible (input and output) units are free or that only the output units are free. The formalism is the same. In our feature recognition applications to be discussed below we use the latter alternative. In content addressable memory applications more elaborate definitions of 'free' are needed (see Section 5.2).

The Boltzmann machine prescription for changing T_{ij} such that G is minimized is as follows:

(1) *Clamping phase.* The values of the input and output units of the network are clamped to a training pattern, and for a sequence of decreasing temperatures $T_n, T_{n-1}, \ldots, T_0$, the network of equation (1) is allowed to relax according to the Boltzmann distribution (equation 4). At $T=T_0$ statistics are collected for the correlations

$$\rho_{ij}=\langle S_i S_j\rangle \tag{14}$$

Relaxing at each temperature is performed by updating unclamped units according to the heatbath algorithm [18]

$$P(S_i \to 1) = \left[1 + \exp\left(\sum_j T_{ij} S_j / T\right)\right]^{-1} \tag{15}$$

(2) *Free running phase.* The same procedure as in Step 1, but this time the network runs freely or with only the input units clamped. Correlations

$$\rho'_{ij} = \langle S_i S_j \rangle \tag{16}$$

are again measured at $T = T_0$.

(3) *Updating.* After each pattern has been processed through Steps 1 and 2, the weights are updated according to

$$\Delta T_{ij} = \eta(\rho_{ij} - \rho'_{ij}) \tag{17}$$

where η is the learning rate parameter. Equation (17) corresponds to gradient descent in G [2]. Steps 1, 2 and 3 are repeated until no more changes in T_{ij} take place.

3.2. MFT Learning

With the mean field theory approximation in our hands, the Boltzmann machine procedure takes the following form [3, 4] (see also ref. [17]).

(1) *Clamping phase.* The stochastic unit updating of equation (15) is replaced by solving equation (10) for a sequence of decreasing temperatures $T_n, T_{n-1}, \ldots, T_0$, and the correlations ρ_{ij} are now given by

$$\rho_{ij} = V_i V_j \tag{18}$$

The factorization of the MFT variables in equation (18) follows from the MFT approximation [12].[4]

(2) *Free running phase.* Similarly, in the free phase the correlations ρ'_{ij} are given by

$$\rho'_{ij} = V_i V_j \tag{19}$$

(3) *Updating.* As in the Boltzmann machine (equation (17)) above.

To use the MFT algorithm it is necessary to specify a few parameters: initital T_{ij} values, annealing schedule, learning rate and weight updating frequency. Also the option of discrete updating is sometimes useful. A detailed discussion of these issues can be found in ref. [4].

4. MFT versus BP on Feature Recognition: Two Benchmarks

Let us now compare the performance of MFT and BP with respect to learning and generalization for two pattern recognition problems. The term 'generalization' refers to the response of a network, after some amount of training, to novel (unlearned) inputs. There are at least two different ways to test generalization.

(1) *Continuous learning.* The training set covers the entire input space. Each time the network is presented with a training pattern it is first tested for generalization on that pattern. In this mode of operation, the distinction between learning and generalization is blurred.

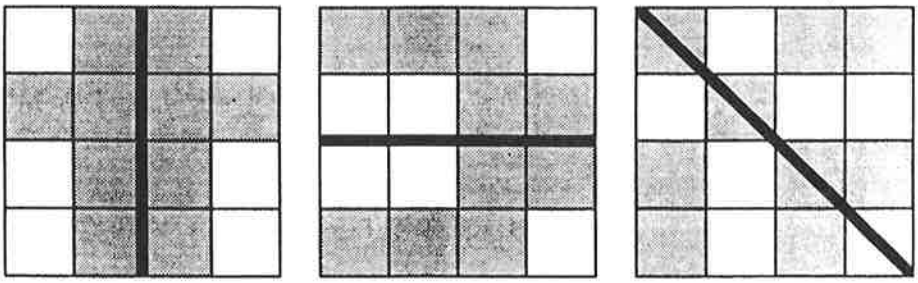


Figure 3. The two-dimensional mirror symmetry problem.

(2) *Fixed training set.* After learning a training set consisting of a fixed subset of the total input space, the network is tested for generalization on patterns it has not seen before.

We have investigated the generalization properties of MFT and BP using the two-dimensional mirror symmetry problem [19] and a statistical pattern classification task consisting of two multi-dimensional heavily overlapping Gaussians [20]. The mirror symmetry problem requires detecting which one of three possible axes of symmetry is present in a $N \times N$ pixel (binary) input (see Figure 3). The overlapping Gaussians problem consists of correctly assigning input patterns to one of two overlapping classes (see Figure 4). The statistical nature of this problem makes it particularly challenging as it necessarily involves inconsistent training. These synthetic problems are different and difficult enough to represent suitable benchmarks. The mirror symmetry problem is characterized by a second-order predicate [21] and a very large number of possible input patterns. The overlapping Gaussians problem is an artificial abstraction of the statistical nature of many natural signal (e.g. speech) processing tasks.

4.1. The Mirror Symmetry Problem

For this problem we used an architecture consisting of $N \times N$ input units, one layer of 12 hidden units and three output units (one for each axis of symmetry). Our experiments were performed with two problem sizes: 4×4 and 10×10. Only input patterns with exactly one of the three possible axes of symmetry were included. There are about 1.5×10^3 such patterns in the 4×4 case and about 3.7×10^{16} in the 10×10 case.

When using MFT and BP in this kind of application, two subtle but important issues arise: $[-1, 1]$ versus $[0, 1]$ representation and endpoint versus midpoint success criterion for the testing phase [4].

(1) *[−1, 1] versus [0, 1].* In Section 2 we used $[-1, 1]$ representation for the neurons. With a linear transformation, the whole formalism could trivially be redone for $[0, 1]$ representation with one important difference: in the $[-1, 1]$ case, both 'on–on' and 'off–off' correlations are counted as positive correlations in the learning rule of equation (17), but in the $[0, 1]$ case, only 'on–on' correlations are counted. For this reason we expect faster learning for both BZ and MFT when using the $[-1, 1]$ alternative. For BP, one also expects the $[-1, 1]$ representation to allow faster learning since, like BZ and MFT,

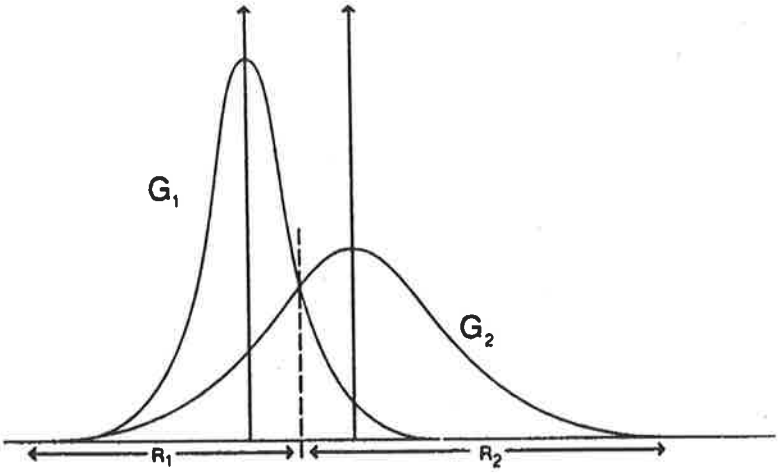
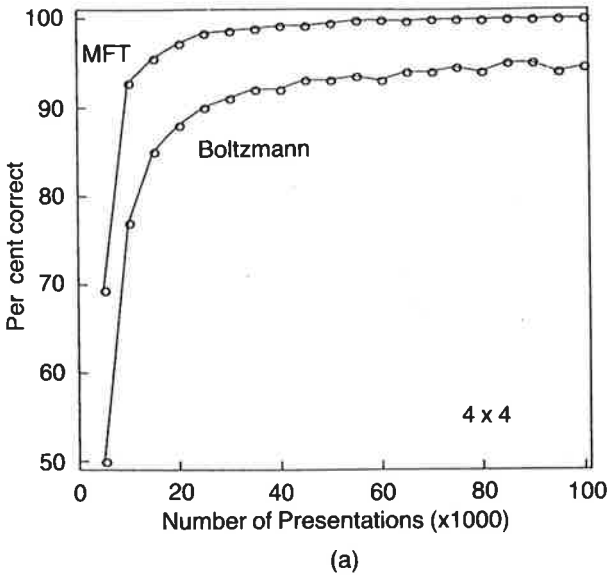


Figure 4. A one-dimensional example of two overlapping Gaussian distribution. The non-statistical limit of the problem consists of two delta-functions located at the centre of the Gaussians. Areas of misclassification are indicated.

this algorithm is unable to modify weights on connections from input units that are set to zero. With respect to generalization power, the situation could very well be the opposite. In cases where two neurons are undecided, i.e. have values near 0.0 and 0.5, respectively, no learning takes place in the $[-1, 1]$ case, whereas this 'uncertainty' is emphasized with a positive correlation in the $[0, 1]$ case. In other words, one expects less 'stiff' learning in the latter case and hence better generalization. The issue of representation choice is also important in CAM Hopfield-type networks, but for somewhat different reasons (see ref. [22]).

(2) *Endpoint versus midpoint success criterion.* As a success criterion in the learning process, a value fairly close to the target is typically demanded of both BP and MFT (e.g. $|V_i| > 0.8$ in $[-1, 1]$ representation); we call this an endpoint criterion. When testing for generalization, the question arises whether this same endpoint criterion should be used or just a midpoint criterion: V_i on the correct side of 0. It turns out that the performance of BP is sensitive to this choice while MFT is insensitive to it. When trained with endpoints as targets, MFT output units tend to take on values near the endpoints during generalization testing,[5] while BP outputs often take on intermediate values. This difference between the algorithms is very likely due to the feed-forward versus feedback dynamics.

4.1.1. Continuous Learning. The results are shown in Figures 5 and 6, from which we make the following observations. First, as discussed above, the relative performance of BP improves significantly with the midpoint criterion, whereas MFT is virtually unaffected. Secondly, even with use of the midpoint criterion, BP (using $[0, 1]$) lags behind MFT (using $[-1, 1]$). Also note that any difference in performance between the algorithms decreases as learning progresses.



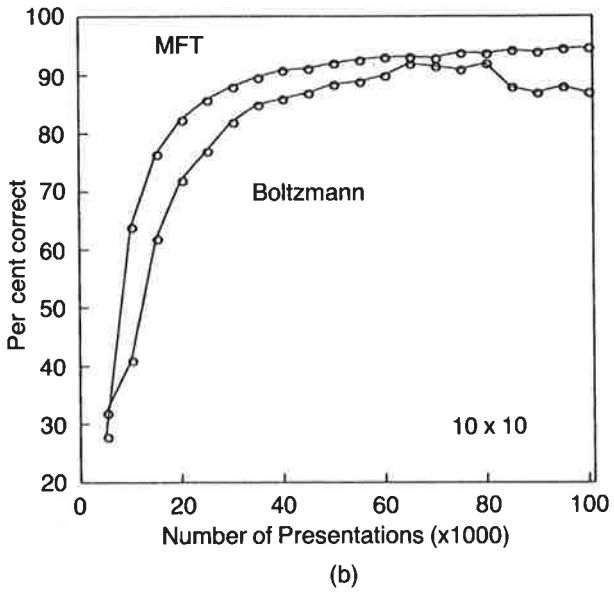


Figure 5. Learning curves using the endpoint criterion for the 4×4 and 10×10 mirror symmetry problems. For MFT, $[-1, 1]$ representation was used and for BP, $[0, 1]$ representation.

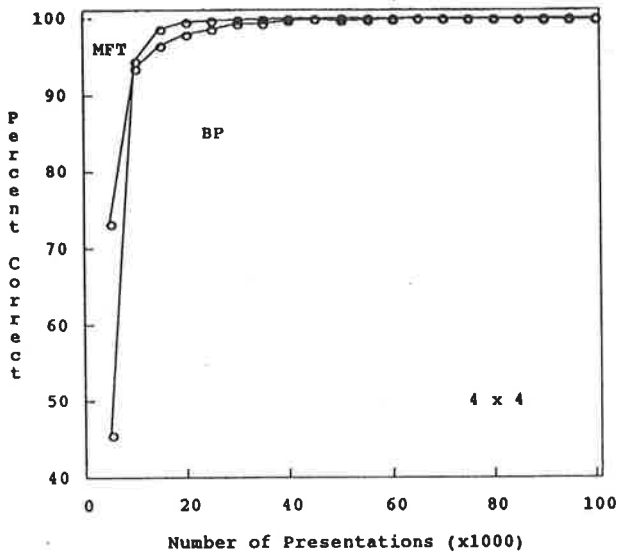


Figure 6. Learning curve using the midpoint criterion for the 4×4 mirror symmetry problem. For MFT, $[-1, 1]$ representation was used and for BP, $[0, 1]$ representation.

4.1.2. Fixed training set learning. Next MFT and BP networks for the 4×4 problem were trained on different fixed sets of 100 patterns. For details on how these sets were chosen we refer the reader to ref. [4]. From these experiments the following observations and conclusions were drawn.

(1) MFT learns faster than BP, as expected from the discussion above. The two algorithms do equally well on generalization on $[0, 1]$, and MFT does somewhat better on $[-1, 1]$.[6]
(2) For both algorithms, learning is faster with $[-1, 1]$ than $[0, 1]$, also as expected from the discussion above.
(3) For both algorithms generalization is more powerful with $[0, 1]$ than $[-1, 1]$, consistent with the same discussion. An opposite result is reported in [24] for the 'majority' problem.
(4) MFT appears less sensitive to the representation choice than BP.

4.2. A Statistical Pattern Recognition Problem

Most natural problems entail noisy and inconsistent training. The success of the neural network technology will therefore be judged largely according to its ability to deal with statistical problems. The potential difficulty in this problem lies in the presence of inconsistent training: for inputs where the two Gaussians overlap, the training examples map the same inputs to both of two different outputs. This is in contrast to the mirror symmetry problem discussed above and the delta function limit in Figure 4 where the same training output is consistently associated with a given input. Reference [20] benchmarked the BP, BZ and learning vector quantization algorithms for testbeds consisting of heavily overlapping Gaussian distributions with dimensionality ranging from two to eight.

In ref. [20], the neural network algorithms generally produced good results, with the BZ performing very close to the theoretical Bayesian limit. BP and MFT were compared with the theoretical limit in ref. [4] using three of the testbeds of ref. [20]. The first case (A) consists of two overlapping Gaussians (G_1, G_2) in eight dimensions, centered around $(0,0,0,\ldots,0)$ and $(2.32,0,0,\ldots,0)$ with standard deviations σ equal to 1 and 2 respectively. At the theoretical minimum (Bayesian limit), one has [23]

$$P_{\text{error}} = \int_{R_1} G_2 + \int_{R_2} G_1 \tag{20}$$

where R_1 and R_2 are chosen such that P_{error} is minimized. In Figure 4, the optimal choice of R_1 and R_2 is illustrated in two dimensions. For the above Gaussians, $P_{\text{error}} = 0.062$, i.e. a 93.8% success rate is the maximum achievable. In the second case (B), the difficulty of the problem is increased by using identical means for the two classes: G_2 is shifted to $(0,0,0,\ldots,0)$. The maximum success rate for this case is 91.0%. In the third (C) and most difficult case, there are only two input dimensions instead of eight, and again the two classes have identical means of $(0,0)$). The maximum success rate for this case is only 73.6%.

The details of our MFT and BP simulations can be found in ref. [4]. In the experiments, both algorithms used $[0,1]$ units, the midpoint was used as the correctness criterion, and in all cases there were eight hidden units and one output unit.[7] A few technicalities are important when aiming for peak performance and when comparing the results of ref. [4] with those of ref. [20].

(1) *Architecture.* A fully connected architecture was used, where all connections except input–input connections were present.[8]
(2) *Encoding of input values.* The continuous input was subdivided into 20 subranges and a local representation was used: an input pattern consisted of exactly one unit 'on' in each of the D sets of 20 units.
(3) *'Manhattan' updating.* In equation (17) the weights are changed according to gradient descent, i.e. steps in weight space are taken along the gradient vector—each gradient component (weight change) will be of different size. If one instead updates with a fixed step size, the step is taken in a slightly different direction along a vector whose components are all of equal size. Everything about the gradient is thrown away except the knowledge of which 'quadrant' it lies in; learning proceeds on a lattice. In situations where it is advisable to present many examples before taking a step in weight space, we have found this 'Manhattan' updating procedure to be beneficial.

In Table I we show the peak performance and number of training patterns required. Both MFT and BP essentially reach the theoretical limit, which is very impressive given that this problem is so non-trivial.

4.3. MFT and BP—Different Algorithms?

One may wonder why MFT and BP behave so similarly despite their different foundations. The answer lies in the choice of architectures used in the two examples; one hidden layer and $n_I \gg n_O$. In this limit one can show that MFT and BP are

Table I. Peak performance and number of patterns required for the statistical pattern classification problems (see text). The results for BZ and testbed C for BP were taken from ref. [20]. Percentages from the present study are with respect to the preceeding 10 000 patterns.

	MFT		BP		BZ	
Testbed	%	patts	%	patts	%	patts
A	93.2	140 k	93.2	120 k	93.3	400 k
B	90.0	170 k	90.7	160 k	90.6	400 k
C	73.3	60 k	73.7	?	73.5	400 k

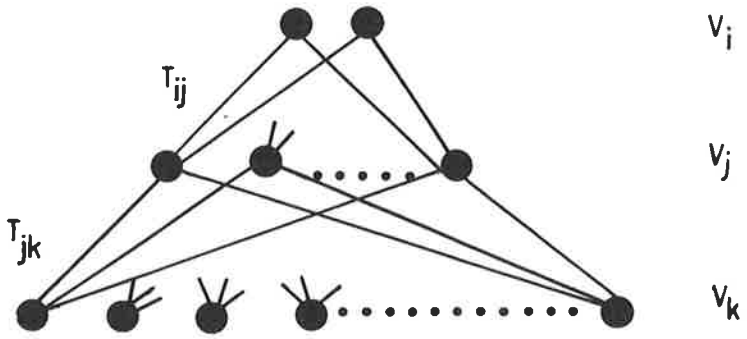


Figure 7. A one hidden layer architecture with $n_I \gg n_O$.

approximately equal. To this end consider the updating of the hidden neurons V_j in Figure 7. (Our way of reasoning is closely related to the one of ref. [26].)

$$V_j = g\left[\left(\sum_k T_{jk} V_k^c + \sum_i T_{ij} V_i\right)\Big/ T\right] \tag{21}$$

where the notation is defined in Figure 7 and $g(x) = \tanh(x)$. With $n_I \gg n_O$ and bounded values on T_{ij} and T_{kj} the output units will have minor influence on V_j and we can expand the latter around the part of the input sum equation (21) containing clamped input units V_k^c to obtain

$$V_j \approx g\left(\sum_k T_{jk} V_k^c / T\right) + g'\left(\sum_k T_{jk} V_k^c / T\right) \sum_i T_{ij} V_i / T \tag{22}$$

For the weight changes between the hidden and output layers one then obtains for MFT (keeping the first term in equation (22) only)

$$\Delta T_{ij} = V_i^c V_j^c - V_i V_j \approx (V_i^c - V_i) V_j^c \tag{23}$$

This is identical to what one gets from BP using the entropy error measure [26, 27], which for each pattern reads

$$E^{BP} = \sum_i [V_i^c \log(1 \text{ } V_i) - V_i \log(1 - V_i^c)] \tag{24}$$

where we have identified the clamped value V_i^c with the target value t_i and the unclamped value V_i with the output value o_i using standard BP notation. The corresponding updating rule for the weights is

$$\Delta T_{ij}=(V_i^c-V_i)V_j^c \tag{25}$$

Updating the input-hidden layer in MFT using the approximation of equation (22) gives

$$\Delta T_{jk}=V_j^cV_k^c-V_jV_k^c=V_k^cg'\left(\sum_k T_{jk}V_k^c/T\right)\sum_i T_{ij}(V_i^c-V_i)/T \tag{26}$$

which is identical to the propagated error in BP.

So the fact that MFT and BP came out so close in the comparisons is not surprising. The question now is under what circumstances do we expect the algorithm to really differ. Two such situations immediately emerge:

(1) Architectures with more than one hidden layer.
(2) $O(n_I)\approx O(n_O)$. Such a situation is natural in time-prediction applications where a number of variables at $t=\tau$ are the inputs and the corresponding variables for $t=\tau+\Delta$ are the outputs. Also, content addressable memories belong to a domain of applications where the output nodes are not limited to a few features (see below).

4.3.1. Time consumption. We end this section by comparing the time consumption of the two algorithms. As stated above the real power of MFT is its straightforward mapping onto analogue VLSI circuitry. For pattern recognition applications that are intended to be implemented in MFT analogue hardware it is of course important to test the algorithm on the simulation level in advance. It has been stated now and then that MFT is 'far too slow'. Let us pin down what 'far too slow' means in terms of CPU cycles in layered architectures with n_I inputs, n_O output and n_H hidden units. For both MFT and BP the number of operations per pass is approximately given by the number of connections n_C.

MFT

$n_C=n_H(n_O+n_I)$ clamped phase

$n_C=n_H(2n_O+n_I)$ free phase

BP

$n_C=n_H(n_O+n_I)$ both forward and backward pass

In the case of MFT one also has to take into account the number of temperatures n_T used in the free phase for the annealing. For each epoch one thus has the ratio between time consumptions

$$\frac{\tau_{MFT}}{\tau_{BP}}=\frac{n_T(2n_O+n_I)+(n_O+n_I)}{2(n_O+n_I)} \tag{27}$$

Equations 27 holds for one hidden layer architectures with no hidden–hidden connections where no feedback in the MFT clamped phase is present. The annealing in the clamped phase is therefore unnecessary. For the entire learning process one obtains the ratio

$$R=\frac{t_{MFT}}{t_{BP}}=\frac{N_E^{MFT}}{N_E^{BP}}\frac{\tau_{MFT}}{\tau_{BP}} \tag{28}$$

Table II. Comparison of serial CPU time consumption for BP and MFT (for notations see text)

Problem	N_E^{MFT}	N_E^{BP}	n_T	n_I	n_O	R
Mirror symmetry $(-1,1)$	38k	123k	11	16	3	2.0
Mirror symmetry $(0,1)$	49k	264k	11	16	3	1.2
Gaussian overlap	140k	120k	11	160	1	7.0

where N_E^{MFT} and N_E^{BP} are the number of epochs. Based on empirical information we have computed R for the 4×4 mirror symmetry and the Gaussian overlap applications. The results are shown in Table II. R in the 2–5 range seems to be typical. (The Gaussian overlap problem is particularly favourable for BP since 'Manhattan' updating was used and hence learning is not slowed down by the BP $g'(\ )$-factor.

5. A Content Addressable Memory with MFT Learning

In *feature recognition* applications a functional mapping from input to output units takes place. Both feedback (e.g. MFT) and feed-forward (e.g. BP) algorithms are well suited for this. A feature recognizer can be viewed as a special case of *content addressable memory* (CAM). Here, the visible units are not partitioned into input units and output units. Hence, while bidirectional (e.g. MFT) networks are appropriate for CAM, BP is not.[9]

We define CAM as follows. The network is initialized (but not permanently clamped) to a noisy version of a pattern, i.e. some bits are changed at random, unknown positions. This initial state evolves to the stored pattern (the network moves to the closest attractor), correcting the random errors. This is formally an error-correction CAM.[10] The computational task of a CAM can be recognized as a restatement of the task of decoding codewords received from a noisy communication channel. The set of legal codewords corresponds to memory items and decoding a noisy codeword means error correction—correcting the errors in the pattern by identifying the closest legal codeword. Conventional algorithms for error correction take time that increases with the size and number of the legal codes [29].

5.1. *The Hopfield Model*

The architecture of the Hopfield model consists of N fully connected *visible* units. The storage is Hebbian [30]

$$T_{ij}=\sum_{p=1}^{M} S_i^p S_j^p \tag{29}$$

where M is the number of N-bit patterns ($\boldsymbol{S}^p$) to be stored. The dynamics is governed by equation (3). With the storage prescription of equation (29) the system has the stored patterns $\boldsymbol{S}^p$ as attractors. For uncorrelated (random) patterns, the storage capacity is given by $M_{max}\approx0.14N$ [31] (For correlated patterns this number is even smaller.) Subsequent work has demonstrated that 0.64 patterns per synapse can be stored in partially connected networks [32].

Various modifications of equation (29) have been suggested to improve the storage capacity. They fall into two classes: local [33, 34] and non-local [35]. We briefly mention two modifications which are local: *REM sleep* [33] and the *bidirectional perceptron learning algorithm* [34, 36]; both are related to MFT learning.

It has been suggested that during REM sleep, mammals 'dream in order to forget' as a means for removing spurious undesired memories [37]. It was demonstrated in ref. [33] that if the rule of equation (29) is supplemented by 'unlearning' of the spurious states the CAM performance improves. Subsequent work has verified the power of this method with storage capacities in the range 30–40% as a result [38]. This 'unlearning' procedure is closely related to the Boltzmann machine learning prescription of equation (17); positive learning (equation (29)) corresponds to the clamped phase whereas 'unlearning' corresponds to the free phase [19]. Since MFT relies on the same learning rule (equation (17)) it effects the same 'pruning' of state space or 'sculpting' of the energy landscape.

The *bidirectional perceptron learning algorithm* [34, 36] uses the same unit updating rule as the Hopfield model, allows visible units only and is a direct extension of the perceptron algorithm [21] to bidirectional networks. The learning process goes as follows. For each pattern p and unit i, the error ε_i^p is recorded such that

$$\varepsilon_i^p = \frac{1}{2}\left[1 - \operatorname{sgn}\left(S_i^p \sum_j T_{ij} S_j^p\right)\right] \tag{30}$$

The weights are then updated according to

$$\Delta T_{ij} = \frac{1}{N} \sum_p (\varepsilon_i^p + \varepsilon_j^p) S_i^p S_j^p \tag{31}$$

This process is repeated until all errors are corrected. This algorithm is a special case of MFT with $T = 0$ and no hidden units [4].

5.2. *A Content Addressable Memory with MFT Learning*

5.2.1. The algorithm. We now depart from the Hopfield CAM in the sense that hidden units will be used to build up internal representations of the stored states. (The Hopfield model has only visible units.) For an N-bit memory, we choose an architecture consisting of N visible units and N hidden units. We return to the question of degree of connectivity below.

Since no distinction between input and output units exists in a CAM modified *retrieval* and *clamping* procedures are called for.

Retrieval. In an error-correction CAM the goal is to correct any errors in the state of the visible units. Thus each visible unit functions both as an input and as an output unit: the input to the network is the initial state of the visible units, and the output is their final state. With no hidden units, error-correcting retrieval or course proceeds as with the Hopfield model: initialize (not clamp) the network to a noisy version of a stored pattern, turn on the dynamics, and let the network evolve to a stable final state. If hidden units are present one wants to evoke the internal representation of the stored pattern learned by the hidden units, enlisting their influence to help correct the errors in the visible units. To accomplish this effectively in a fully connected network requires annealing the hidden units while clamping the visible units. But if the visible units are clamped, their errors cannot be corrected. Several retrieval methods can solve this dilemma. In refs [4–6] the following procedure was used.

(1) Clamp the visible units to the initial state.
(2) Anneal down to an intermediate temperature; the state of the hidden units begins to approximate the learned internal representation of the stored pattern.
(3) Release the visible units. Visible units representing input bits known in advance to be correct should remain clamped.
(4) Complete annealing. The network now settles as a whole, with the evoked representation in the hidden units helping the visible units settle into the stored pattern, correcting any errors.

Clamping. In this phase clamping of course proceeds as in the feature recognition case. For the free phase the situation is different again due to the fact that there are no input–output distinctions. In refs [4–6] one-half of the visible units were chosen at random and then clamped for each pattern presentation.

Convergence criterion. In feature recognition applications of MFT a natural stopping criteria can be defined with the error of the output units as in BP. Again, since no output units are distinguished in CAM applications an alternative is needed. A possibility would be to monitor G in equation (13) as learning progresses. To compute G in the BZ is of course intractable given the vast summations underlying P_α and P'_α. However, with the MFT approximation at our hands, G should be easily computable in the saddlepoint approximation for P_α and P'_α. However, it turns out that G does not progress smoothly. The reason is that different units are clamped in each free phase (or each pattern). It is therefore necessary to rely on a more pragmatic criteria [5], which turns out to be very effective. After every 100 epochs, learning is interrupted and retrieval is performed on all training patterns (without noise). In this way, the number of successfully stored patterns can be determined. When this number does not improve within 400 epochs, learning is terminated.

Connectivity. Two different architectures have been subject to experimentation, *fully connected* [4, 5] and *no hidden–hidden* connectivity [5] (see Figure 8). It turned out in refs [5, 6] that with the fully connected architecture variant the performance in terms of number of stored patterns as a function of n_H deteriorates with M. This phenomenon is not too surprising. With the hidden–hidden connections present, if the number of hidden units is large, the majority of connections to a hidden unit are from other hidden units, while very few are from the visible units. In this case, during training or retrieval, when the visibles are clamped and the hiddens are settling, the 'noise' from the other hidden units swamps the 'signal' from the visible units. In what follows we will therefore stick to architectures with no hidden–hidden connections (Figure 8 (b)).

Figure 8. (a) A fully connected MFT CAM. (b) An MFT CAM with no hidden–hidden connections.

5.2.2. Numerical performance. Throughout this section, the number of visible and hidden units in a network will be denoted as n_V and n_H. All patterns that are being learned are random. For details on parameter values, etc., we refer the reader to refs [5, 6]. Figures 9 and 10 show plots for networks of size $n_V = 24$ and $n_V = 32$, without hidden–hidden connections, of the maximum number of storable patterns as a function of the number of hidden units.

The data points of Figures 9 and 10 were determined as follows. Networks of each size (n_V, n_H) were trained with training sets of increasing size. For a given size, the number of patterns stored consistently reached a maximum and eventually decreased. With capacity of a network we then mean the largest number of patterns stored before the decline starts (this always occurred such that the maximum number stored was at least 90% of the size of its training set). Fitting the data of Figures 9 and 10 to a linear behaviour gives $M \approx O(10\text{–}20)n_H$, which is very impressive. There are even indications of superlinear behaviour [5, 6]. If these latter results survive the MFT learning method might be a candidate for tapping the exponential storage capability of recurrent networks [39].

5.3. Comments

We close the CAM section with a few comments.

(1) *Precision.* How many bits of precision are necessary for each weight and how does the required number of bits for the weights compare with the number of pattern bits stored? Remarkably, in the larger networks, provided the network is allowed to make just a few errors in retrieval, many more pattern bits can be stored and retrieved than are needed to represent the weights in the network [5, 6]. Since a stored pattern explicitly exists only when the network adopts that stable state, one can regard the pattern bits as implicitly stored, as compared to the bits in the weight registers, which are explicitly stored. In this view, the number of implicitly stored bits can greatly exceed the number of explicitly stored bits.

(2) *Basins of attraction.* Number of stored patterns is not the only measure of the performance of a CAM. The basin size is also very important. Encouraging results for this quantity were reported in refs [5, 6]. The average basin size for any network is a decreasing function of the number of patterns stored (loading). For representative architectures these functions were measured and plotted in ref. [6]. The general quality of the plots improves with increasing $n_H = n_V$ networks, but worsens (in normalized terms) with increasing $n_H > n_V$. For instance, the 'critical point' at which the average basin size decreases to zero is approximately, in fractions of capacity, 0.75 for an $n_H = n_V = 24$ network, 0.90 for an $n_H = n_V = 32$ network and 0.40 for an $n_V = 24$, $n_H = n_V = 36$ network. This relative worsening with increasing $n_H > n_V$ results from a growing number of spurious states brought about by an interesting identity map effect (see ref. [6]).

(3) *No hidden units.* In this case the MFT CAM in the $T \to 0$ reduces [3] to the bidirectional perceptron [34, 36] and the storage performance is indeed similar [5]. The basins of attraction are, however, distinctly better for the MFT approach [5, 6]. One should mention, though, that subsequent work on the bidirectional perceptron gives good evidence that the basins of attraction may widen when trained on noisy patterns [40, 41]. This could give a consistent picture since the MFT CAM learning procedure also includes noisy versions of the patterns.[11]

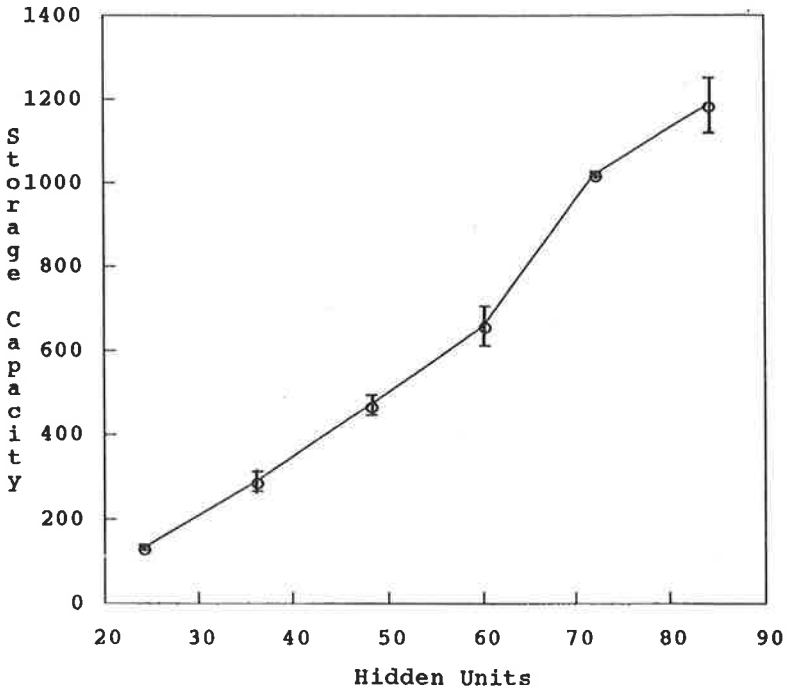


Figure 9. Maximum number of storable patterns as a function of the number of hidden units for $n_V = 24$ using the architecture of Figure 8 (b).

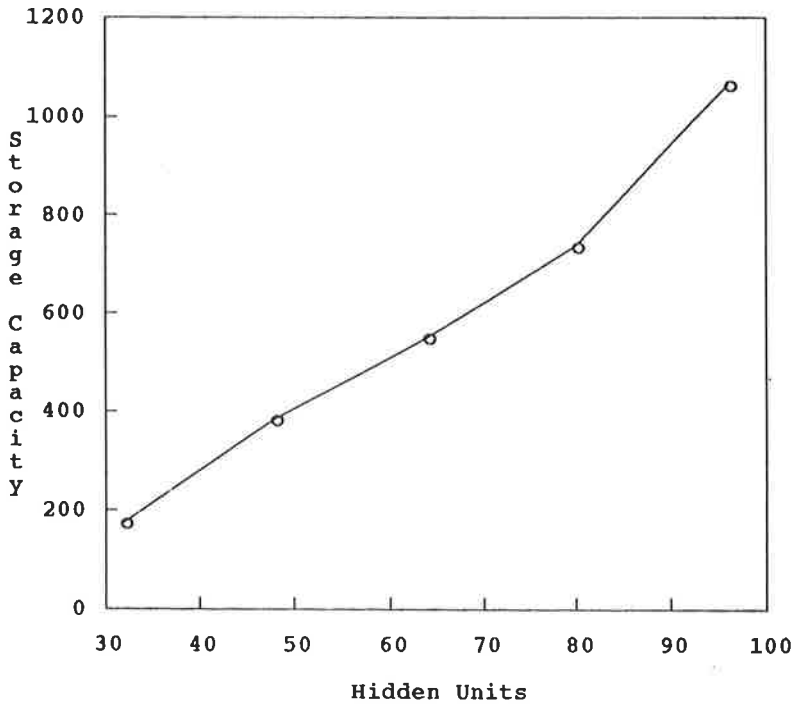


Figure 10. Maximum number of storable patterns as a function of the number of hidden units for $n_V = 32$ using the architecture of Figure 8 (b).

6. Solving Optimization Problems with MFT Networks

Neural networks have shown great promise for finding approximate solutions to difficult optimization problems [8, 42–44]. In their pioneering work Hopfield & Tank [8] formulated the travelling salesman problem (TSP) on a highly interconnected neural network and made exploratory numerical studies on modest-sized samples. The main ingredient of the approach is to map the problem onto a neural network such that a neuron being 'on' corresponds to a certain decision and then to relax the system with mean field techniques in order to avoid local minima.

In the original paper [8], 10- and 30-city problems were studied with good results for the $N=10$ case. For $N=30$ the authors report on difficulties in finding optimal parameters. In ref. [45] further studies of the Tank–Hopfield approach were made with respect to refinements and extension to larger problem sizes. The authors of ref. [45] find the results discouraging. The origin of the observed problems is twofold. Many of the solutions are not 'legal' in the sense that a city is visited not exactly once, which can easily be remedied with a greedy heuristics [3]. Once this greedy heuristics has been applied the core problem stands out; that of redundancy in the imbedding of the problem. In ref. [9] a more compact encoding was suggested by using multi-state neurons rather than the neuron multiplexing of ref. [8]. Technically, this corresponds to a Potts glass [46] rather than an Ising spin glass model.

In order to lay out the general principles and potential problems when mapping optimization problems onto neural networks we start with a fairly detailed description of the graph bisection problem.

6.1. Graph bisection

This problem is defined as follows (see Figure 11 (a)): partition a set of N nodes with given connectivity into two halves such that the net connectivity (cutsize) between the two halves is minimized. The problem is mapped onto the Hopfield energy function (*cf.* equation 1) by the following representation. For each node, assign a neuron $S_i=1$ and for each pair of vertices S_iS_j, $i=j$, we assign a value $T_{ij}=1$ if they are connected and $T_{ij}=0$ if they are not connected. In terms of Figure 11 (a), we let $S_i=\pm 1$ represent whether node i is in the left or in the right position. With this notation, the product $T_{ij}S_iS_j$ is zero whenever nodes i and j are not connected at all, positive whenever connected nodes i and j are in the same partition, and negative when they are in separate partitions. With this representation, minimization of equation (1) energy function will maximize the connections within a partition while minimizing the connections between partitions. However, the net result will be that all nodes are forced into one partition. Hence we must add a 'constraint term' to the right-hand side of equation (1) that penalizes situations where the nodes are not equally partitioned. We note that $\Sigma S_i=0$ when the partitions are balanced. A term proportional to $(\Sigma S_i)^2$ will increase the energy whenever the partition is unbalanced. Our neural network energy function for graph bisection then takes the form

$$E=-\frac{1}{2}\sum_{ij}T_{ij}S_iS_j+\frac{\alpha}{2}\left(\sum_i S_i\right)^2 \tag{32}$$

where the imbalance parameter α sets the relative strength between the cutsize and the balancing term. This balancing term represents a *global constraint*. The generic form of equation (32) is

$$E=\text{'cost'}+\text{'global constraints'} \tag{33}$$

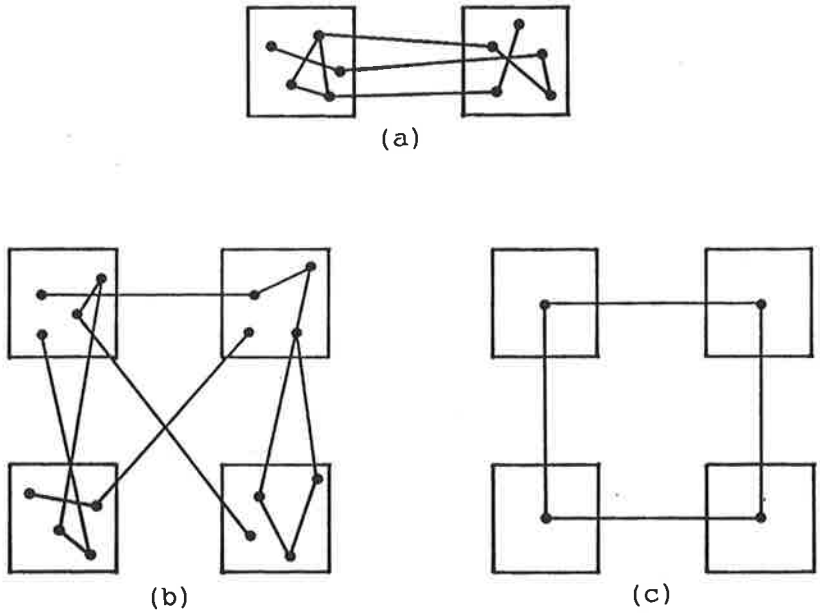


Figure 11. (a) A graph bisection problem. (b) A $K=4$ graph partition problem. (c) An $N=4$ TSP problem.

which is typical when casting 'difficult' optimization problems onto neural networks. The origin of the 'difficulty' is very transparent here; the problem is frustrated in the sense that the two constraints ('cost' and 'global constraint') are competing with each other with the appearance of many local minima. These local minima can to a large extent be avoided by applying the MFT technique to equation (32) yielding

$$V_i = \tanh\left[\sum_j (T_{ij}-\alpha)V_j/T\right] \tag{34}$$

where $V_i = \langle S_i \rangle_T$ (equation (8)). The generic form of the energy function in equation (32) is very different from a more standard heuristic treatment of the optimization problem. For example, in the case of graph bisection one typically starts in a configuration where the nodes are equally partitioned and then proceeds by swapping pairs subject to some acceptance criteria. In other words the constraint of equal partition is respected throughout the updating process. This is in sharp contrast to neural network techniques (equation (32)), where the constraints are implemented in a 'soft' manner by a Lagrange multiplier. The final MFT solutions are therefore sometimes plagued with a minor imbalance, which is easily remedied by applying a *greedy heuristic* to the solutions [11].

In addition to the Lagrange parameter α, the MFT equation (34) contains the temperature T as a free parameter. It is desirable to be able to estimate the values for T in advance so that a 'trial-and-error' process can be avoided when applying the algorithm to different problems; the algorithm should be a 'black box' from a users perspective. The spin systems onto which we are mapping the optimization problems typically have two phases; at large enough temperatures the system relaxes into the trivial fixed point $V_i^{(0)}=0$. As the temperature is lowered a phase transition is passed at $T=T_c$ and as $T\to 0$ fixed points $V_{ia}^{(*)}$ emerge representing a specific decision

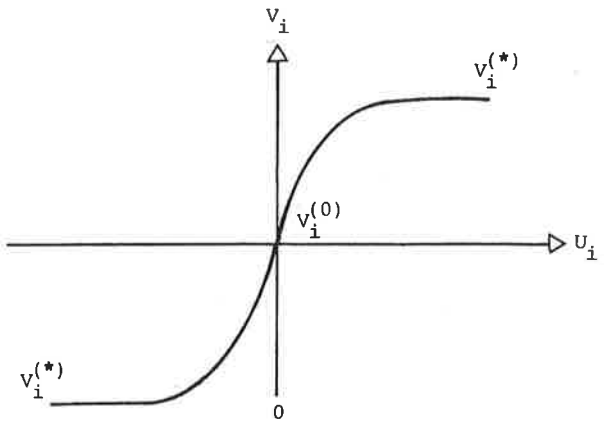


Figure 12. Fixed points in $\tanh(U_i)$.

made as to the solution to the optimization problems in question. The position of T_c, which depends on T_{ij} and α, can be estimated by expanding the sigmoid function (tanh) around $V_i^{(0)}=0$ (see Figure 12). The fluctuations around $V_i^{(0)}$

$$V_i = V_i^{(0)} + \varepsilon_i \tag{35}$$

satisfy

$$\varepsilon_i = \frac{1}{T}\sum_j M_{ij}\varepsilon_j \tag{36}$$

where

$$M_{ij} = T_{ij} - \alpha \tag{37}$$

For synchronous updating it is clear that if one of the eigenvalues to M_{ij}/T in equation (36) is greater than unity in absolute value the solutions will wander away into the non-linear region. Hence T_c will be determined by the eigenvalue distribution of M_{ij}. In the case of serial updating the philosophy is the same but the analysis slightly more complicated. We refer the reader to ref. [9] for a more detailed discussion. The position of T_c is in general not the same for synchronous and serial updating. From equation (37) it is clear that the strengths of diagonal or 'feedback' terms (α) are important for determining T_c. Finding the largest eigenvalue to M_{ij} could be computationally explosive by itself. Is there an approximate way of doing it? Yes, given $t=\langle T_{ij}\rangle$ and the corresponding standard deviation σ_t turns out to be sufficient for obtaining estimates within 10% of T_c. Also, this analysis is important for avoiding oscillatory behaviour [49], which appears for eigenvalues less than -1.

6.2. *Graph Partition*

When generalizing to *graph partition* (GP), the N nodes are to be partitioned into K sets, each with N/K nodes, again with minimal cutsize (see Figure 11 (b)). Let us first review the formalism for the 1-of-K or neuron multiplexing method for this problem [9], which was also used in the context of the TSP in ref. [8]. Then we reduce this 1-of-K encoding to the corresponding multistate neuron encoding approach which has turned out to be the winner [9].

*6.2.1. 1-of-*K *encoding: neuron multiplexing.* In order to map the GP problem onto a neural network we first introduce a second index for the neurons (neuron multiplexing)

$$S_{ia}=0,1 \tag{38}$$

where the index i denotes the node ($i=1,\ldots,N$) and a the set ($a=1,\ldots,K$). S_{ia} takes the value 1 or 0 depending on whether node i belongs to set a or not. We use $[0,1]$ notation (rather than $[-1,+1]$) in order to get a more convenient form of the energy function, which reads

$$E=\frac{1}{2}\sum_{ij}\sum_{a\neq b}T_{ij}S_{ia}S_{jb}+\frac{\beta}{2}\sum_{i}\sum_{a\neq b}S_{ia}S_{ib}+\frac{\alpha}{2}\sum_{a}\left(\sum_{i}S_{ia}-\frac{N}{K}\right)^2 \tag{39}$$

The first term corresponds to the cutsize to be minimized. The third term in equation (39) represents the *global constraint* of equipartition; it is zero only, if each of the K sets contains N/K nodes. The second term, which has the form 'winner-takes-all', is new; it ensures that the 1-of-K encoding is satisfied. In its structure this equation differs from that of equations (32) and (33) by the presence of his second term enforcing the 'syntax' constraint.

Again we define mean field variables, $V_{ia}=\langle S_{ia}\rangle_T$ and the corresponding MFT equations are given by[12]

$$V_{ia}=\frac{1}{2}\left[\left[1+\tanh\left\{\left[-\sum_{j}\sum_{b\neq a}T_{ij}V_{jb}-\beta\sum_{b\neq a}V_{ib}-\alpha\left(\sum_{j}V_{ja}-\frac{N}{K}\right)\right]\Big/T\right\}\right]\right] \tag{40}$$

The solution space of these MFT equations consists of the interior of the direct product of N K-dimensional hypercubes. In Figure 13 we show the cube corresponding to $K=3$. Using equation (40) to solve graph partition problems gives poor and very parameter sensitive results [9]. Let us next turn to an alternative encoding which compact the solution space by one dimension.

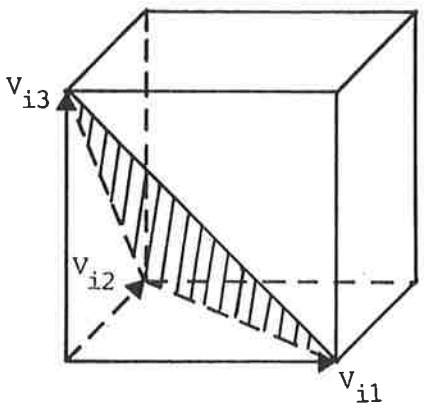


Figure 13. The volume of solutions corresponding to the 1-of-K encoding for $K=3$. The shaded plane corresponds to the solution space of the reduced 1-of-K encoding for $K=3$.

*6.3. Reduced 1-of-*K *Encoding: Multi-state Neurons*

In this section we restrict the allowed states for the neurons, such that exactly one neuron at every site is on, and derive the corresponding K-state Potts glass MFT equations.[13]

The restriction corresponding to the second term in equation (39) can be compactly written as

$$\sum_a S_{ia} = 1 \tag{41}$$

Thus for every i, S_{ia} is one for only one value of a, and zero for the remaining values of a. So, the allowed values of the vector $\boldsymbol{S}_i = (S_{i1}, S_{i2}, \ldots, S_{iK})$ are the principal unit vectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_K$ in an obvious vector notation. The number of states available at every node is thereby reduced from 2^K to K, and technically we have a K-state Potts model at our hands. In Figure 13 we show the space of states at one node for the case $K=3$.

The energy function of equation (39) can now be rewritten, using the constraint of equation (41), as

$$E = \frac{1}{2}\sum_{ij}\sum_a T_{ij} S_{ia} S_{ja} + \frac{\alpha}{2}\sum_{ij}\sum_a S_{ia} S_{ja} \tag{42}$$

or, in vector notation,

$$E = -\frac{1}{2}\sum_{ij} T_{ij} \boldsymbol{S}_i \boldsymbol{S}_j + \frac{\alpha}{2}\left(\sum_i \boldsymbol{S}_i\right)^2 \tag{43}$$

This expression has exactly the same structure as the energy (equation (32)) for the graph bisection problem. Indeed, for $K=2$, they are completely equivalent (apart from a trivial factor 2).

6.3.1. MFT equations for multi-state neurons. For the multi-state neurons $\boldsymbol{S}_i$ it is straightforward to work out the corresponding MFT equations [9]. They read

$$\boldsymbol{V}_i = \boldsymbol{F}_K\left(-\frac{\partial E}{\partial \boldsymbol{V}_i}\frac{1}{T}\right) \tag{44}$$

where we have again introduced the MFT variables $\boldsymbol{V}_i = \langle \boldsymbol{S}_i \rangle$ and the K generalized sigmoid functions are given by

$$F_K^a(\boldsymbol{U}) = \frac{e^{U_a}}{\Sigma_b e^{U_b}} \tag{45}$$

which for $K=2$ reduces to a tanh function as expected. Note that this expression automatically satisfies the constraint

$$\sum_a F_K^a(\boldsymbol{U}) = 1 \tag{46}$$

Thus, when iterating equation (44), the mean field variables $\boldsymbol{V}_i$ will be forced to exist in this $(K-1)$-dimensional subspace of the original K-dimensional unit hypercube, as shown in Figure 13 for the case $K=3$.

For the GP problem we now have the MFT equations

$$\boldsymbol{V}_i = \boldsymbol{F}_K(\boldsymbol{U}_i) \tag{47}$$

$$\boldsymbol{U}_i = -\frac{\partial E}{\partial \boldsymbol{V}_i}\frac{1}{T} = \left[\sum_j (T_{ij}-\alpha)V_j + \beta \boldsymbol{V}_i\right]\frac{1}{T} \tag{48}$$

We note that with equation (44) $\boldsymbol{V}_i$ is guaranteed to lie in the subspace $\Sigma_a V_{ia} = 1$. The interpretation of V_{ia} as probabilities is obvious. In equation (48) we have kept the β term of equation (39), which is redundant in terms of encoding the problem, but potentially important for the MFT dynamics (*cf.* the discussion on diagonal terms in Section 6.1).

6.4. *The Travelling Salesman Problem*

The travelling salesman problem (TSP) is related to GP in the case of $K = N$ with the modification that the set connections constitute a closed loop (see Figure 11 (c)) and an extension in the sense that analogue values (city distance) are used for the node connection matrix.

Now we apply the reduction trick to above coding of the TSP.[14] Again we consider only states of the neurons satisfying the constraint of equation (41), which reduces the number of states from $2^{N\times N}$ to N^N. The energy can then be written (up to an uninteresting constant) as

$$E = \sum_{ij} D_{ij} \sum_a S_{ia} S_{j(a+1)} - \frac{\beta}{2}\sum_i \sum_a S_{ia}^2 + \frac{\alpha}{2}\sum_a \left(\sum_i S_{ia}\right)^2 \tag{49}$$

where D_{ij} is the distance between cities i and j, and $a+1$ is defined modulo N.[15].

The corresponding MFT equations for V_{ia} read

$$\boldsymbol{V}_i = \boldsymbol{F}_N(\boldsymbol{U}_i) \tag{50}$$

with

$$U_{ia} = -\frac{\partial E}{\partial V_{ia}}\frac{1}{T} = \left\{-\sum_j [D_{ij}(V_{j(a+1)} + V_{j(a-1)})] - \alpha\sum_j V_{ja} + \beta V_{ia}\right\}\Big/ T \tag{51}$$

6.5 *Scheduling*

Consider the following scheduling problem: N_p teachers are supposed to have N_q classes in N_x class rooms at N_t time slots. The problem is to find a solution where all N_p teachers give a lecture to each of the N_q classes, using the available space–time slots with no conflicts in space (classrooms) or time. These are the *hard constraints* that have to be satisfied. In addition, one could imagine having a set of *soft constraints* like preferences in time slots, continuity in classrooms, etc. This is of course just one example of a scheduling problem but we fell it is general and difficult enough to illustrate the neural network approach.

The basic entities of this problem can be represented by four sets consisting of N_p, N_q, N_x and N_t elements, respectively. One could imagine formulating the problem in terms of neurons connecting two or more elements in different sets. However, there is a more transparent way to describe the problem that naturally lends itself to the Potts neural encoding [10]. Consider *events*, defined by teacher–class pairs (p, q), to be mapped onto *space–time slots* (x, t).

(1) An event (p, q) should occupy precisely one space–time slot (x, t).
(2) Different events (p_1, q_1) and (p_2, q_2) should not occupy the same space–time slot (x, t).

(3) A teacher p should have at most one class at a time.
(4) A class q should have at most one teacher at a time.

The first constraint can be imbedded in a neural network in terms of multi-state neurons $\boldsymbol{S}_{pq}$ with components

$$S_{pq;xt}=0,1 \tag{52}$$

by demanding

$$\sum_{x,t} S_{pq;xt}=1 \tag{53}$$

for each event (p, q). In other words we have N_pN_q neurons, each of which has N_xN_t possible states. The other three constraints are implemented using energy penalty terms as follows

$$E_{xt}=\frac{1}{2}\sum_{x,t}\sum_{p_1,q_1}\sum_{p_2,q_2} S_{p_1q_1;xt}S_{p_2q_2;xt}=\frac{1}{2}\sum_{x,t}\left[\sum_{pq} S_{pq;xt}\right]^2 \tag{54}$$

$$E_{pt}=\frac{1}{2}\sum_{p,t}\sum_{q_1,x_1}\sum_{q_2,x_2} S_{pq_1;x_1t}S_{pq_2;x_2t}=\frac{1}{2}\sum_{pt}\left[\sum_{qz} S_{pq;xt}\right]^2 \tag{55}$$

$$E_{qt}=\frac{1}{2}\sum_{q,t}\sum_{p_1,x_1}\sum_{p_2,x_2} S_{p_1q;x_1t}S_{p_2q;x_2t}=\frac{1}{2}\sum_{qt}\left[\sum_{px} S_{pq;xt}\right]^2 \tag{56}$$

From equations (52) and (53) it is obvious that any linear combination X of different components of a neuron also must be zero or unity, so that

$$X^2=X \tag{57}$$

For the particular combinations $S_{pq;xt}$, $\Sigma_x S_{pq;xt}$ and $\Sigma_t S_{pq;xt}$, this property can be used to add trivial-valued auxiliary terms to the energy

$$E_{\text{aux}}=-\frac{\beta}{2}\sum_{p,q}\sum_{x,t} S_{pq;xt}^2-\frac{\beta_x}{2}\sum_{p,q}\sum_{x,t_1,t_2} S_{pq;xt_1}S_{pq;xt_2}-\frac{\beta}{2}\sum_{p,q}\sum_{x_1,x_2,t} S_{pq;x_1t}S_{pq;x_2t} \tag{58}$$

These are the only non-trivial terms of this kind, which also respect the obvious permutation symmetries of the problem. The effect of these extra terms on the energy value is merely that of adding a fixed constant, but they turn out important for the mean field dynamics [10].

The problem is now that of finding a configuration that minimizes the total energy

$$E=E_{xt}+E_{pt}+E_{qt}+E_{\text{aux}}+\frac{1}{2}N_pN_q(-3+\beta+\beta_x+\beta_t) \tag{59}$$

where we have added a constant in order for the minimal energy value to be zero.

Most problems also have additional soft constraints. An example of a soft constraint that is easy to implement, yet not trivial, is to require that a class should remain in the same room from one time-slot to the next. This constraint might be encoded by including an additional energy term

$$E_\alpha=-\frac{\alpha}{2}\sum_{p'\neq p}\sum_{q}\sum_{xt} S_{pq;xt}(S_{p'q;x(t+1)}+S_{p'q;x(t-1)}) \tag{60}$$

where $t\pm1$ is defined modulo N_t.

Again mean field variables $V_{pq:xt} = \langle S_{pq:xt} \rangle_T$ are introduced and the corresponding MFT equations read

$$U_{pq;xt} = -\frac{1}{T}\frac{\partial E}{\partial V_{pq;xt}} \tag{61}$$

and

$$V_{pq;xt} = \frac{e^{U_{pq;xt}}}{\sum_{xt} e^{U_{pq;xt}}} \tag{62}$$

As before the eigenvalue analysis described in Section 6.1 is used to estimate T_c.

6.6. *Numerical Explorations*

In ref. [9] exhaustive simulation studies were performed for GP and TSP with the multi-state encoding with impressive results. Not only were the solutions consistently of high quality but the algorithm was implemented in a 'black-box' manner with no free parameters. We refer the reader to ref. [9] for details.

In Figures 14 and 15 results for 10×100 GP and $N = 200$ TSP are shown. The results are impressive! Our neural network algorithm performs as well (in some cases even better) as the simulated annealing method with excessive annealing and sweep schedule. This is accomplished with a very modest number of sweeps, $O(50\text{–}100)$. The imbalance prior to applying the greedy heuristics is negligible; on average 0.1% of $K \times N$ and N^2 for GP and TSP respectively.

Numerical studies of the neural network approach to the scheduling problem also give very good results (see ref. [10]).

7. Summarizing Remarks

The purpose of this review has been to discuss three very different application areas of MFT network models. The common denominator is feedback architecture with 'tanh' gain functions. In learning applications a substantial advantage in terms of hardware implementations exists; learning is *local*. Also the MFT equations are *isomorphic* to the corresponding circuit equations. Furthermore, given that all three applications are mapped onto the same class of network investments in hardware implementations could have a tremendous payoff. Highest performance quality of MFT models is of course a necessary condition for taking advantage of the above-mentioned benefits. We will next summarize the quality situation with respect to the different application areas.

7.1. *Feature Recognition*

We have benchmarked MFT against BP on two different testbeds: the two-dimensional mirror symmetry problem and a statistical pattern classification problem consisting of two multi-dimensional overlapping Gaussians. Our main results can be summarized as follows.

(1) The generalization capabilities of the two algorithms are similar. When the algorithms are used in their standard forms, MFT learns in a substantially smaller number of training epochs than BP; when 'Manhattan' learning is used the learning speeds are similar.

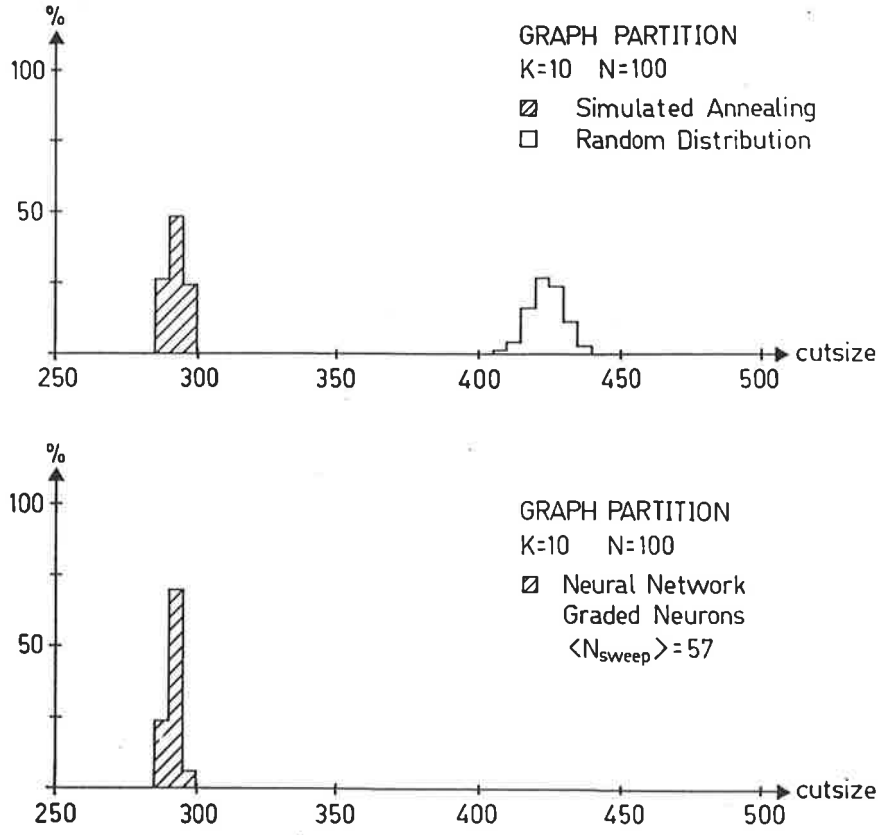


Figure 14. Comparison of neural network solutions versus simulated annealing and random distributions for a 10×100 GP problem. The histograms are based on 50 experiments for the neural network and simulated annealing algorithms and 1000 for the random distributions.

(2) The performance of both algorithms on the problem of two overlapping Gaussians is particularly impressive since it involves inconsistent training; the Bayesian theoretical limit is essentially reached!

One might wonder why MFT and BP behave so similarly despite their different foundations. It is related to the fact that for feature recognition the number of output (feature) units is much smaller than the number of input and hidden units. Since feedback is effectively only present between the hidden and output layers, the difference should be minor for these cases.

In situations with $O(n_O \approx O(n_I)$, e.g. time-prediction of many variables of a dynamical system, $x_i(t+\Delta t)=f(x_i(t))$, the algorithms should become significantly different.

For serial simulations MFT takes a factor 2–5 longer time that BP to learn. The real gain is in real-time applications when custom-made hardware is required.

Although BZ and MFT learning are formally derived for $T_{ij}=T_{ji}$ they can be extended to asymmetrical situations [50, 51]. This fact makes these approaches more plausible from a biological point of view. It also facilitates applications of processing time-sequenced data, e.g. speech, in a natural way.

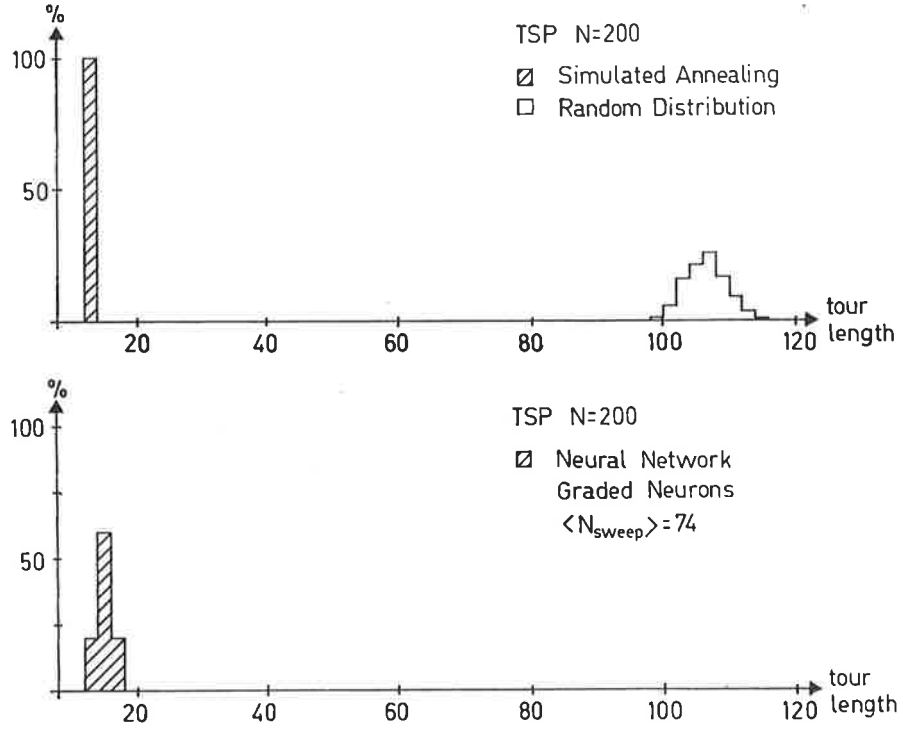


Figure 15. Comparison of neural network solutions versus simulated annealing and random distributions for a $N=200$ TSP. The historgrams are based on five experiments for the neural network and simulated annealing algorithms and 1000 for the random distributions.

7.2. A Content Addressable Memory

Since there is no inherent distinction between input and output units in MFT, its applications are not limited to feature recognition problems. In Section 5 it was convincingly demonstrated how to construct a CAM by building up internal representations of the patterns to be stored using MFT. With analogue neurons and weights, no clear theoretical limit on the storage capacity exists (except that set by the semantics of the task), and indeed it is possible to store at least $O(10\text{–}20)n_H$ patterns with good retrieval properties. This is a conservative estimate. It could very well be that the storage capacity is superlinear.

7.3. Optimization

Difficult optimization problems like graph partition and travelling salesman problems are easily mapped onto neural networks with multi-state spins (Potts spin) rather than the more commonly used spin multiplexed encodings (Ising spin). In this alternative formulation two major advantages have emerged.

(1) *Solution quality and parameter sensitivity.* The Potts glass formulation is superior to the Ising glass. The reason is that part of the contraints can be strictly embedded in the energy function, which reduces the solution space accordingly. The method is therefore far less sensitive to the choice of parameters.

(2) *Estimating* T_c. This quantity can be estimated for a given problem in advance within about 10% accuracy by finding the distribution of eigenvalues of the linearized updating equation. This makes the method even more parameter insensitive.

As a by-product of the eigenvalue analysis mentioned above we are able to consistently *avoid bifurcating behavior* in the case of synchronous updating (see, ref. [49]).

The neural network algorithm is benchmarked against simulated annealing for the graph partition and travelling salesman problems with respect to the quality of the solutions. Graph partition problems of sizes 4×100 and 10×100 were investigated and for TSP the corresponding problem sizes were $N = 50, 100$ and 200. Very good quality solutions were consistently found. The results compare favourably with other distributed parallel approaches (for a recent benchmark see ref. [52]).

This K-valued neuron formalism is also powerful for feature recognition problems, where exclusive classes are to be identified. Such an extension is not limited to the MFT framework but could also be used for the output layer in BP [53, 54].

Note Added in Proof

The comparisons made in refs [20, 25] between BZ, BP and LVQ (learning vector quantization) were not done in a reasonable manner. Only eight hidden units were used in the BP case in contrast to LVQ, where the authors allowed for the number of hidden units to increase with the dimensionality of the problem. It is thus not surprising that BP had a hard time with the eight-dimensional overlaps, and indeed, when adding input–output connections as was done in ref. [4], the BP performance is very close to the Bayes limit (see Table 1). Furthermore, if the authors of refs [20, 25] had allowed for the number of feature nodes to vary as a power of the dimension of the problem, LVQ would very likely have reached that limit as well—to fill a d-dimensional volume requires something that grows exponentially with d. Sigmoidal networks like BP, MFT and BZ on the other hand cut out planes, which is much more economical in terms of nodes for high-dimensional problems of this kind. Another factor that made the results of ref. [4] superior to those of refs [20, 24] for BZ was the fact that in ref. [4] 'Manhattan' updating was used.

Notes

1. Feedback networks may also exhibit non- fixed-point dynamics (limit cycles, chaotic behaviour). Such networks are not relevant for the applications dealt with in this review.
2. Resistors are of course always positive. Positive and negative T_{ij} elements are implemented by having a pair of connections between the amplifiers, T_{ij}^{+} and T_{ij}^{-}. Each amplifier is given two outputs, a normal (>0) and an inverted one (<0). T_{ij}^{+} and T_{ij}^{-} are both positive but interpreted with different signs depending on the sign of the outgoing voltage from the amplifier.
3. In Section 5 we will discuss how its use can be 'extended' to the case of no hidden units in the context of content addressable memories.
4. This factorization property of MFT was wrongly stated as an additional assumption in refs [3, 4].
5. This is not meant to imply that MFT cannot learn analogue values.
6. Varying the gain for BP in the $[-1, 1]$ case (fixed for a given run) did not improve generalization.
7. No change in performance was observed in trials using two output units.
8. In BP, the hidden units cannot be symmetrically interconnected, so they were connected asymmetrically: imagining the hidden units in a row, each hidden unit was connected to all hidden units to its right.

9. Throughout this review we refer to standard BP, not to recurrent BP [28] and other variants.
10. There exist two other functional possibilities, partial-contents CAM and schemata-completion (see ref. [4] for details).
11. The only remaining difference between the two algorithms is then that the MFT CAM has modifiable bias connections.
12. The $\frac{1}{2}[1+\ldots]$-form of equation (40) originates from the [0, 1] notation of equation (38).
13. The idea of using the Potts glass for GP was first introduced in ref. [47].
14. A similar approach was employed in ref. [48].
15. This is of course not the only way of coding the problem. One could, for example, interchange the roles of the labels i and a.

References

[1] Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986) Learning internal representations by error propagation. In D. E. Rumelhard and J. L. McClelland (Eds) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1.* Cambridge, MA: MIT Press.

[2] Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. (1985) A learning algorithm for Boltzmann machines. *Cognitive Science,* **9,** 147.

[3] Peterson, C. & Anderson, J. R. (1987) A mean field theory learning algorithm for neural networks. *Complex Systems,* **1,** 995.

[4] Peterson, C. & Hartman, E. (1989) Explorations of the mean field theory learning algorithm. *Neural Networks,* **2,** 475.

[5] Hartman, E. (1989) A large storage capacity neural network content-addressable memory. Dissertation Thesis. University of Texas at Austin.

[6] Hartman, E. (1990) A high storage capacity neural network content-addressable memory. MCC Technical Report MCC-ACT-NN-173-90 (submitted to *Network: Computation in Neural Systems*).

[7] Hopfield, J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science USA,* **79,** 2554.

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

[9] Peterson, C. & Söderberg, B. (1989) A new method for mapping optimization problems onto neural networks. *International Journal of Neural Systems,* **1,** 3.

[10] Gislén, L., Peterson, C. & Söderberg, B. (1989) Teachers and classes with neural networks. *International Journal of Neural Systems,* **1,** 167.

[11] Peterson, C. & Anderson, J. R. (1988) Neural networks and NP-complete optimization problems; a performance study on the graph bisection problem. *Complex Systems,* **2,** 59.

[12] Glauber, R. J. (1963) Time-dependent statistics of the Ising model, *Journal of Mathematical Physics,* **4,** 294.

[13] Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P. (1983) Optimization by simulated annealing. *Science,* **220,** 671.

[14] See, e.g. Mézard, M., Parisi, G. & Virasoro, M. (1986) *Spin Glass Theory and Beyond.* Singapore: World Scientific.

[15] Hopfield, J. J. (1984) Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science USA,* **81,** 3088.

[16] Kullback, S. (1989) *Information Theory and Statistics.* New York: Wiley.

[17] Hinton, G. E. (1989) Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural Computation,* **1,** 153.

[18] Yang, C.-P. (1963) *Proceedings of Symposia in Applied Mathematics,* Vol XV. Providence, RI: American Mathematical Society.

[19] Sejnowski, T. J., Klienker, P. & Hinton. G. E. (1986) Learning symmetry groups with hidden units: beyond the perceptron. *Physica,* **22D,** 260.

[20] Kohonen, T., Barna, G. & Chrisley, R. (1988) Statistical pattern recognition: benchmarking studies. *Proceedings of the IEEE Second International Conference on Neural Networks,* San Diego, California. New York: IEEE.

[21] Minsky, M. & Papert, S. (1969) *Perceptrons.* Cambridge, MA: MIT Press.

[22] Perez-Vicente, C. J. (1989) Sparse coding and information in hebbian neural networks. *Europhysics Letters,* **10,** 621; Perez-Vicente, C. J. (1989) Optimized network for sparsely coded patterns. *Journal of Physics A,* **22,** 559.

[23] See, e.g. Duda, R. & Hart, P. E. (1973) *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons.
[24] Tesauro, G. & Sejnowski, T. J. A neural network that learns to play backgammon. In D. Anderson (Ed.), *Neural Information Processing*. New York: American Institute of Physics.
[25] Barna, G. & Kaski, K. (1989) Variations on the Boltzmann machine. *Journal of Physics A*, **22**, 5143.
[26] Hopfield, J. J. (1987) Learning algorithms and probability distributions in feed-forward and feedback networks. *Proceedings of the National Academy of Science USA*, **84**, 8429.
[27] Baum, E. B. & Wilcek, F. (1988) Supervised learning of probability distributions in neural networks. In D. Z. Anderson (Ed.) *Neural Information Processing Systems*. New York: American Institute of Physics.
[28] Pineda, F. J. (1987) Generalization of backpropagation to recurrent neural networks. *Physical Review Letters*, **18**, 2229.
[29] Blahut, R. E. (1984) *Theory and Practice of Error-Control Codes*. New York: Addison-Wesley.
[30] Hebb, D. O. (1949) *The Organization of Behaviour*. New York: Wiley.
[31] Amit, D. J., Gutfreund, H. & Sompolinsky, H. (1985) Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters*, **55**, 1530.
[32] Canning, A. & Gardner, E. (1988) Partially connected models of neural networks. *Journal of Physics A*, **21**, 3275.
[33] Hopfield, J. J., Feinstein, D. I. & Palmer, R. G. (1983) Unlearning has stabilizing effects in collective memories. *Nature*, **304**, 158.
[34] Wallace, D. J. (1986) Memory and learning in a class of neural network models. In B. Bunk & K. H. Mutter (Eds), *Lattice Gauge Theory—A Challenge to Large Scale Computing*. New York: Plenum.
[35] Kanter, I. & Sompolinsky, H. (1987) Associative recall of memories without errors. *Physical Reveiw A*, **35**, 380.
[36] Bruce, A. D., Canning, A., Forrest, B., Gardner, E. & Wallace, D. J. (1986) Learning and memory properties in fully connected networks. *Proc Conf. on Neural Networks for Computing, Snowbird UT (AIP Conf. 151)*. New York: AIP.
[37] Crick, F. & Mitchison, G. (1983) The function of dream sleep. *Nature*, **304**, 111.
[38] Kleinfield, D. & Pendergraft, D. B. (1987) Unlearning the storage capacity of content addressable memories. *Biophysical Journal*, **51**, 47–53.
[39] McEliece, R. J., Posner, E. C., Rodemich, E. R. & Venkatesh (1984) The capacity of the Hopfield associative memory. *IEEE Transactions on Information Theory*, **IT-33**, 461.
[40] Gardner, E. J., Stroud, N. & Wallace, D. J. (1989) Training with noise and the storage of correlated patterns in a neural network model. *Journal of Physics A*, **22**, 2019.
[41] Wong, K. Y. M. & Sherrington, D. (1989) Training noise adaption in attractor neural networks. *Journal of Physics A*, **23**, L175.
[42] Fu, Y. & Anderson, P. W. (1986) Application of statistical mechanics to NP-complete problems in combinatorial optimisation. *Journal of Physics A*, **19**, 1605.
[43] Mézard, M. & Parisi, G. (1985) Replicas and optimisation. *Journal de Physique*, **46**, L771.
[44] Mézard, M. & Parisi, G. (1986) A replica analysis of the travelling salesman problem. *Journal de Physique*, **47**, 1285.
[45] Wilson, G. V. & Pawley, G. S. (1988) On the stability of the travelling salesman problem algorithm of Hopfield and Tank. *Biological Cybernetics*, **58**, 63.
[46] Wu, F. Y. (1983) The Potts model. *Review of Modern Physics*, **54**, 235.
[47] Kanter, I. & Sompolinsky, H. (1987) Graph optimization problems and the Potts glass. *Journal of Physics A*, **20**, L673.
[48] Van den Bout, D. E. & Miller III, T. K. (1988) A traveling salesman objective function that works. *Proceedings of the IEEE International Conference on Neural Networks*, pp. 299. New York: IEEE.
[49] Choi, M. Y. & Huberman, B. A. (1983) Digital dynamics and the simulation of magnetic systems. *Physical Review B*, **28**, 2547.
[50] Allen, R. B., & Alspector, J. (1990) Learning of stable states in stochastic asymmetric networks. *IEEE Transactions on Neural Networks*, **1**, 233.
[51] Galland, C. C. & Hinton, G. E. (1990) Deterministic Boltzmann learning in asymmetric networks. In D. Touretzky, J. Elman, T. Sejnowksi & G. Hinton (Eds) *Proceedings of the 1990 Connectionist Models Summer School*, San Mateo, CA: Morgan Kauffman.
[52] Peterson, C. (1990) Parallel distributed approaches to combinatorial optimization problems—benchmark studies on TSP. *Neural Computation*, **2**, 261.
[53] Rumelhart, D. (1989) unpublished.
[54] Lönnblad, L., Peterson, C. & Rögnvaldson, T. (1991) Using neural networks to identify jets, *Nuclear Physics B*, **349**, 675.