Using Boltzmann Machines for probability estimation: A general framework for neural network learning

Hilbert J. Kappen[1]

Department of Medical Physics and Biophysics, University of Nijmegen, Geert Grooteplein 21, 6525 EZ Nijmegen, The Netherlands

In this paper, efficient learning rules are developed for a large class of neural network archictectures within the Boltzmann Machine framework. The advantage of this appoach is that a unified view is created, in which supervised feed-forward learning, unsupervised learning and clustering appear as special cases. It is shown that temperature dependent spontaneous symmetry breaking occurs in the hidden layer of these networks. This opens the possibility to study the generalization performance of the network as a function of temperature instead of the number of hidden units.

1. INTRODUCTION

In this paper, I introduce Boltzmann Machines as a general computational framework for neural network modeling. There are several reasons why this is an attractive approach.

In most neural network applications, the generic idea is to find some complex non-linear mapping between an input domain and an output domain. This mapping can be deterministic but is in general probabilistic. Such a probabilistic map is formally described by the conditional probability $p(\vec{y}|\vec{x})$ to observe an output $\vec{y}$ given an input $\vec{x}$. Examples of deterministic mappings are Multi-Layered Perceptrons (MLPs) [1], which minimize a quadratic error between the output of the network and the desired output. The output of such a network has in general no clear interpretation in terms of conditional probability.

A description of probabilistic mappings in terms of conditional probabilities can be obtained by deriving learning rules from a Log-likelihood or Kulback-Leibler error rate [2]. In the context of neural networks, this has been done by Solla et al. [3] and Baum and Wilczek [4] for MLPs, and by Hopfield [5] and Kappen [6] for BMs. In [6] also a comparison of these methods was given.

There are many important applications for which the marginal probability for occurence of data points is needed in addition to the conditional probability. Some generic examples are:

Classification with confidence. One of the well known attractive features of neural networks is that they generalize well to data that were not part of the training set. Nevertheless, when data are presented that are far from the training set in the input

[1] This work was partly sponsored by the Dutch Foundation for Neural Networks (SNN) and the Japanese Real World Computing Program

domain, the network is unlikely to give correct classifications. Thus, what is needed is an indication of the confidence of the output that the network produces. Such a confidence measure depends on whether the input pattern belongs to a part of the input space that was well sampled during training, and, if so, whether for that part of the input space a clear distinction between classes can be made. The former is given by the marginal probability and the latter by the conditional probability. Both probabilities should be learned by the neural network to perform reliable classification with confidence.

Inverse modeling. After a network has been succesfully trained, the question often arises to find the set of inputs that maximizes the output of the network, or that corresponds to another desired output value. Applications exist in for instance direct mailing, where a company wants to reach a group of potential customers with the largest expected interest in their products, or process control, where one wants to know all possible combinations of machine set points and raw material characteristics that will yield a product with the desired specifications. A good model of the marginal probabilities is needed to constrain this search task to probable input values.

As we will see below, by a simple modification of the cost function, one can easily choose in the Boltzmann Machine framework, whether to incorporate the modeling of marginal probabilities.

Neural network theory has been well established to describe the learning and execution of single networks. This holds equally well for Multi-layered perceptrons, topological maps and associative memories. However, no neural network paradigm currently exists in which combinations of these strategies can be modeled and analysed. A particular difficulty is to assess the global learning performance of the network, when different modules use different learning rules (each possibly minimizing different cost criteria).

The need for hybrid and modular architectures comes from the fact that for large problems, the complexity of finding the set of weights that will solve the problem becomes prohibitively large. By splitting the neural network into several modules, or by combining different types of architectures, one may reduce the complexity of the learning task. It is, however, not obvious how such a modularization should be done.

In this paper, I propose to use the Boltzmann Machine [7] formalism to provide a unified framework for studying combinations of different neural network architectures.

For a general Boltzmann Machine, the execution of the learning rule is very slow. This is because the total weight vector defines a probability distribution over the neuron states (Gibbs distribution), and a change in the weights must ensure that this probability distribution remains normalized. Thus the normalisation, i.e. the partition function, of the probability distribution must be calculated at each learning step, which consists of a sum over all 2^n states of the network, where n is the number of unclamped neurons. The partition function can also be calculated by simulation of the equilibrium state of the network using Glauber dynamics. In either case, the execution of the learning rules is very time-consuming. Using a mean field approximation [8], the equilibrium distribution of the neurons can be approximated, which leads to significa
However, a coupled set of non-linear mean field equations must be solved iteratively during each learning step, which reduces the charm of this learning procedure considerably. It can also be shown, that the mean field solution is incompatible with many probability estimation problems [6].

By introducing lateral inhibition in the network, the number of *permissible states* of the network can be strongly reduced. As a result, for a variety of neural network architectures, the partition function can be easily calculated, resulting in very fast learning rules.

In Section 2, I will introduce the general concept. In Section 3, I will briefly sketch how to apply this idea to solve a feed-forward task by introducing lateral inhibition in the output layer of the network. Following [9], these networks will be called Boltzmann Perceptrons. It can be proven, that BPs are universal classifiers, i.e. they can classify any classification problem given enough hidden units [6]. In Section 4, we consider a joint probability estimation task. Here, lateral inhibition is introduced in the hidden layer of the network. In both cases, the resulting architecture is such, that analytic expressions for the learning rules can be obtained. This makes time consuming simulation of the Glauber dynamics unnecessary. The similarity and differences of these networks with MLPs and Radial Basis networks is described in [6]. With lateral inhibition in the hidden layer, temperature dependent spontaneous symmetry breaking occurs. This opens the possibility to study the generalization performance of the network as a function of temperature instead of the number of hidden units.

2. BOLTZMANN MACHINES WITH RESTRICTED STATE SPACE

Let a vector $(\vec{x}, \vec{y})$ denote the values of the visible units of the Boltzmann Machine: $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_1, \ldots, y_m)$. The Boltzmann Machine can be used to estimate joint probabilities or conditional probabilities. The latter is also known as feed-forward mappings. For feed-forward mapping, $\vec{x}$ and $\vec{y}$ will denote the values of the input nodes and output nodes, respectively. For joint probability estimation, $\vec{x}$ and $\vec{y}$ have no individual significance and $(\vec{x}, \vec{y})$ is simply a data vector.

Let for each training pattern $(\vec{x}, \vec{y})$, the probability for occurence be $q(\vec{x}, \vec{y})$. Let $\vec{s} = (s_1, \ldots, s_h)$ denote the values of the hidden units of the Boltzmann Machine. Then the total neuron state of the network may be written as $\vec{S} = (\vec{x}, \vec{s}, \vec{y})$ with components $S_I, I = 1, \ldots, n+h+m$. The generic architecture is given in Fig. 1. For given symmetric weights W_{IJ}, the equilibrium distribution of the Boltzmann Machine is given by a Gibbs distribution:

$$p(\vec{S}) = \frac{1}{Z} \exp(\beta \sum_{(IJ)} W_{IJ} S_I S_J) \qquad (1)$$

$$Z = \sum_{\vec{S}} \exp(\beta \sum_{(IJ)} W_{IJ} S_I S_J) \qquad (2)$$

The observed probabilities on the visible units are:

$$p(\vec{x}, \vec{y}) = \sum_{\vec{s}} p(\vec{x}, \vec{s}, \vec{y})$$

For joint probability estimation (JPE), the Boltzmann Machine minimizes the Kullback divergence between q and p on the visible units:

$$d_{JPE}(p, q) = \sum_{\vec{x}, \vec{y}} q(\vec{x}, \vec{y}) \log \left(\frac{q(\vec{x}, \vec{y})}{p(\vec{x}, \vec{y})} \right).$$

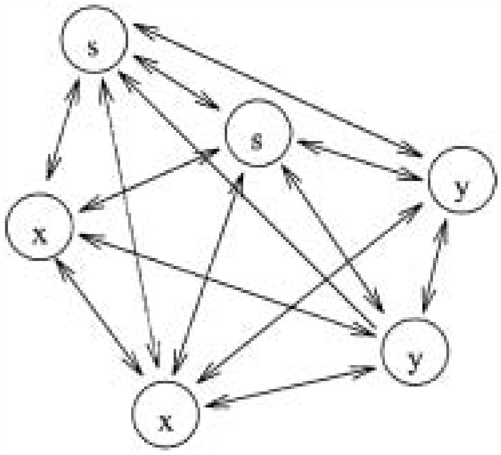

Figure 1: The architecture of a generic Boltzmann Machine with visible units $\vec{x}$ and $\vec{y}$ and hidden units $\vec{s}$. The weigths between units are symmetric.

$$= \sum_{\vec{x}} q(\vec{x}) \log\left(\frac{q(\vec{x})}{p(\vec{x})}\right) + \sum_{\vec{x},\vec{y}} q(\vec{x},\vec{y}) \log\left(\frac{q(\vec{y}|\vec{x})}{p(\vec{y}|\vec{x})}\right). \tag{3}$$

The first term attempts to make $p(\vec{x})$ of the Boltzmann Machine equal to the a priori probability $q(\vec{x})$. The second term attempts to make the conditional probability $p(\vec{y}|\vec{x})$ equal to the conditional probability $q(\vec{y}|\vec{x})$, averaged over $\vec{x}$. For feed-forward mappings only the last term is minimized [10]:

$$d_{FF}(p,q) = \sum_{\vec{x},\vec{y}} q(\vec{x},\vec{y}) \log\left(\frac{q(\vec{y}|\vec{x})}{p(\vec{y}|\vec{x})}\right). \tag{4}$$

When a pattern $\vec{x}$ is clamped to the input nodes, the conditional probability $p(\vec{y}|\vec{x})$ should have the desired value $q(\vec{y}|\vec{x})$. The probability to observe $\vec{x}$ is then not modeled by the network.

In both cases the learning rules are given by gradient descent on d [7]. The result for joint probability estimation is:

$$\Delta W_{IJ} = -\eta \frac{\partial d_{JPE}(p,q)}{\partial W_{IJ}} = \eta \left(\sum_{\vec{x},\vec{y}} q(\vec{x},\vec{y}) < S_I S_J >_{\vec{x},\vec{y}} - < S_I S_J > \right) \tag{5}$$

and for feed-forward mapping:

$$\Delta W_{IJ} = -\eta \frac{\partial d_{FF}(p,q)}{\partial W_{IJ}} = \eta \sum_{\vec{x}} q(\vec{x}) \left(\sum_{\vec{y}} q(\vec{y}|\vec{x}) < S_I S_J >_{\vec{x},\vec{y}} - < S_I S_J >_{\vec{x}} \right) \tag{6}$$

where $< S_I S_J >_A$ denotes the two point correlations between neurons S_I and S_J, when the units specified by A are clamped:

$$< S_I S_J >_A = \sum_{\vec{S} \in A} S_I S_J p(\vec{S}) \tag{7}$$

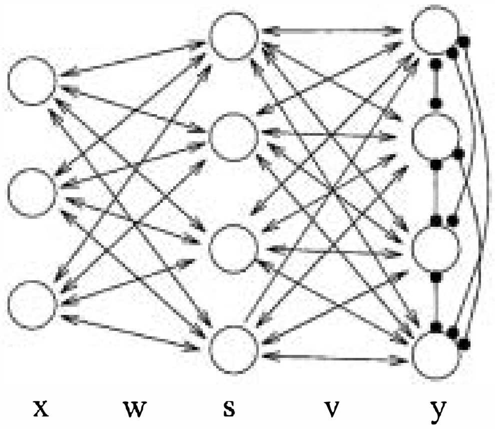

Figure 2: The architecture of a Boltzmann Perceptron for feed-forward mapping as proposed by Hopfield. Lateral inhibition in the output layer is an example of how fast learning rules can be obtained for this architecture.

So we see, that for instance for a weight between two hidden units i and j, $< s_i s_j >_{\vec{x},\vec{y}}$, $< s_i s_j >_{\vec{x}}$ and $< s_i s_j >$ contain 2^h, 2^{h+m} and 2^{h+m+n} terms, respectively. Instead of calculating this exponential number of terms, it is usually faster to *simulate* the network dynamics. Using Glauber dynamics, the network will visit only the most probable states, which make up the largest contributions to the sum in Eq (7). Nevertheless, whether simulated or calculated, the resulting Boltzmann Machine learning rule is extremely slow.

The main idea of this paper is that by introducing lateral inhibition, various specialized architectures and hybrid architectures can be constructed. In addition, the number of states in A can be reduced from an exponential to a linear or polynomial number. As a result, fast learning rules can be obtained for these architectures, which makes it possible to study their behaviour on simple sequential machines. I will illustrate this idea on a Multi-layered Boltzmann Perceptron for feed-forward mapping and on a Radial Basis Boltzmann Machine for probability estimation.

3. THE BOLTZMANN PERCEPTRON

The Boltzmann Perceptron (BP) architecture is sketched in Fig. 2a.

We will use this architecture for feed-forward mappings, i.e. we will use it to generate the conditional probability $p(\vec{y}|\vec{x})$ in Eq. (4). We use $x_i \in R$ and $s_j, y_k = \pm 1$. Usually, one takes $x_i = \pm 1$ as well. However, in the BP the input neurons are clamped during training as well as during execution, so that this constraint can be relaxed. Thresholds at hidden and output neurons are included in w_{ij} and u_{ik} respectively, by adding an input neuron $i = 0$ with $x_0 = 1$. The BP differs from the fully connected Boltzmann Machine in several ways:

- There are no connections between the neurons of the input layer. This is not necessary for feed-forward tasks, because the inputs are always clamped.
- There are no connections between the neurons in the hidden layer, because that would prevent us to calculate the learning rule explicitly. I will show, that the

architecture without these connections is rich enough to be able to perform any classification task.

- There are no learnable connections between units in the output layer. These could be added without excessive computational costs and may lead to interesting applications. They are not considered here.

For a given input $\vec{x}$, the states $(\vec{y}, \vec{s})$ will reach the equilibrium distribution

$$\begin{aligned} p(\vec{y}, \vec{s}|\vec{x}) &= \frac{p(\vec{x}, \vec{s}, \vec{y})}{p(\vec{x})} \\ p(\vec{x}) &= \sum_{\vec{y}, \vec{s}} p(\vec{x}, \vec{s}, \vec{y}) \\ p(\vec{x}, \vec{s}, \vec{y}) &= \frac{1}{Z} \exp\{\sum_{i=0}^{n} \sum_{k=1}^{m} u_{ik} x_i y_k + \sum_{i=0}^{n} \sum_{j=1}^{h} w_{ij} x_i s_j + \sum_{j=1}^{h} \sum_{k=1}^{m} v_{jk} s_j y_k - J(\sum_{k' \neq k}^{m} y_{k'} - 1)^2\} \end{aligned}$$

with Z a normalization constant such that $\sum_{\vec{x}, \vec{y}, \vec{s}} p(\vec{x}, \vec{s}, \vec{y}) = 1$. Learning consists of gradient descent on d_{FF}.

The last term in Eq. (8) is the contribution from the lateral inhibition in the output layer [6]. When J is large, only those states with total activity on output equal to 1 are probable. This reduces the number of possible output states from 2^m to m. This will be of direct benefit for the efficiency fo the learning rules. In [6], a generalization to the case where m_0 output units can be simultaneously 'on' is given.

We are interested in the conditional probability $p(\vec{y}|\vec{x})$ to observe an output $\vec{y}$ given an input $\vec{x}$. Define the *effective field* at output neuron k due to stimulus $\vec{x}$ as

$$H_k(\vec{x}) = \sum_{i=0}^{n} u_{ik} x_i + \sum_{j=1}^{h} \log\cosh(h_j + v_{jk}), \quad k = 1, \ldots, m$$

Let $\vec{y}_k$ denote the permissible output state with unit k 'on', i.e. $(\vec{y}_k)_l = \delta_{kl}$.

Then

$$p(\vec{y}_k|\vec{x}) = \frac{\exp(H_k(\vec{x}))}{Z'(\vec{x})} = < y_k >_{\vec{x}} . \tag{8}$$

with

$$Z'(\vec{x}) = \sum_{k=1}^{m} \exp(H_k(\vec{x}))$$

$< y_k >_{\vec{x}}$ denotes the expectation value of the k-th output neuron averaged over the ensemble of permissible states, in the presence of stimulus $\vec{x}$. Note that for large weights, Eq. (8) implements a winner-take-all mechanism: for the neuron for which H_k is maximal, the probability of firing is one, and for the others zero.

Let training pattern $\vec{x}$ belong to class $\alpha(\vec{x}) = 1, \ldots, m$. The Kullback divergence is then

$$d_{FF} = -\sum_{\vec{x}} q(\vec{x}) \log(p(\vec{y}_{\alpha(\vec{x})}|\vec{x})). \tag{9}$$

The learning rules for u_{ik}, v_{jk} and w_{ij} are now given by gradient descent on d_{FF}:

$$\begin{aligned}
\Delta u_{ik} &= \eta \sum_{\vec{x}} q(\vec{x}) x_i \left((\vec{y}_{\alpha(\vec{x})})_k - < y_k >_{\vec{x}} \right) \\
\Delta v_{jk} &= \eta \sum_{\vec{x}} q(\vec{x}) \tanh(h_j + v_{jk}) \left((\vec{y}_{\alpha(\vec{x})})_k - < y_k >_{\vec{x}} \right) \\
\Delta w_{ij} &= \eta \sum_{\vec{x}} q(\vec{x}) x_i \left(\tanh(h_j + v_{j\alpha(\vec{x})}) - < \tanh(h_j + g_j) >_{\vec{x}} \right).
\end{aligned}$$

The expectation value $< y_k >_{\vec{x}}$ is given by Eq. (8) and

$$< \tanh(h_j + g_j) >_{\vec{x}} = \sum_{k=1}^{m} \tanh(h_j + v_{jk}) p(\vec{y}_k | \vec{x})$$

and involves a sum over m terms only.

3.1. Some numerical results

A numerical study was performed to comparc thc pcrformance of the MLP and the BP [11]. The data consisted of 48.000 handwritten digits. The data were collected on data cntry forms and were preprocessed (segmentation, filtering, normalization and compression). The resulting digits were represented by 64 rcal numbers. Both MLP1 and BP with 64 inputs and 10 outputs were trained on 40.000 digits and tested on 8.000 digits. After extensive search for the optimal number of hidden units, the best results for the MLP1 gave 97.0 % correct on the test sct with two laycrs of hidden units with 42 and 24 units per laycr. Thc bcst result for the BP gave 97.8 % correct on the test set with 20 hidden units. The improvement of almost 1 % is a significant improvement in the context of this application.

4. JOINT PROBABILITY ESTIMATION

In this section, we consider the task of joint probability estimation with Boltzmann Machines. In this case there is no distinction betwcen input and output vectors. $\vec{x}$ and $\vec{y}$ will dcnote real valued and binary valued external values, respectively. Let the probability to observe $(\vec{x}, \vec{y})$ be given by $q(\vec{x}, \vec{y})$. Given the weights of the network, the BM will implement a probability density $p(\vec{x}, \vec{y})$. We will train a Boltzmann Machine such that p approximates q as close as possible, by minimizing Eq. (3). Once p is known, any marginal probability, such as $p(\vec{x}) = \sum_{\vec{y}} p(\vec{x}, \vec{y})$ or conditional probability, such as $p(\vec{y}|\vec{x}) = \frac{p(\vec{x},\vec{y})}{p(\vec{x})}$ can be easily calculated.

Let us consider a network with h hidden units: $s_j = 0$ or $1, j = 1, \ldots, h$. For the architecturc in Fig. 3, the Boltzmann distribution becomes

$$p(\vec{x}, \vec{y}, \vec{s}) = \frac{1}{Z} \exp\{ \sum_{i=0}^{n} \sum_{j=1}^{h} w_{ij} x_i s_j + \sum_{j=1}^{h} \sum_{k=0}^{m} v_{jk} s_j y_k - J(\sum_{j=1}^{h} s_j - h_0)^2 \} \qquad (10)$$

The last term in Eq. (10) is the result of the lateral inhibition in the hidden layer, which for hidden unit j is given by $-J(\sum_{j' \neq j}^{h} s_{j'} - 2h_0 + 1)$.

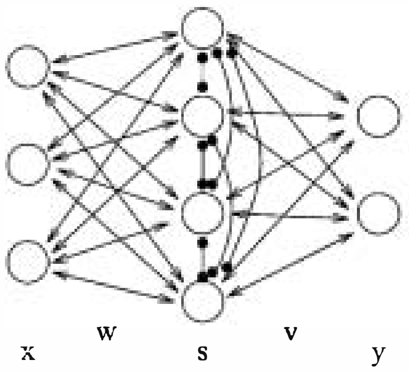

Figure 3: Boltzmann Machine for probability density estimation. There is no distinction between input and output neurons. The activity of the real-valued neurons are collectively denoted $\vec{x}$ and the activity of the binary-valued neurons are collectively denoted $\vec{y}$. The hidden layer has fully connected lateral inhibition of strength $-J$.

The use of real-valued neurons poses two problems at this point: The probability $p(\vec{x}, \vec{y}, \vec{s})$ is not normalizable with respect to $\vec{x}$. Although we used real-valued neurons in the previous section, this problem was not encountered there, because no probability estimation on the $\vec{x}$ space was performed: only conditional probability estimation was done, *given* $\vec{x}$. Secondly, what kind of stochastic dynamics should be used, such that the above equation correctly describes the equilibrium probability?

The way to solve the first problem is to observe that the simplest exponential function of $\vec{x}$ which is normalizable is the Gaussian. We therefore propose to change the local field contribution from $\vec{x}$ to unit s_j:

$$\sum_{i=0}^{n} w_{ij} x_i \rightarrow -\beta \|\vec{w}_j - \vec{x}\|^2, \tag{11}$$

where $(\vec{w}_j)_i = w_{ij}$. Note, that we have reintroduced temperature, because with this change it can no longer be scaled away.

This substitution solves the second problem as well. Note that the Boltzmann distribution is related to the energy as $p \propto \exp(-E)$. Therefore, the suggested change implies a change in energy contribution

$$-\sum_{i=0}^{n} \sum_{j=1}^{h} w_{ij} x_i s_j \rightarrow \beta \sum_{j=1}^{h} \|\vec{w}_j - \vec{x}\|^2 s_j. \tag{12}$$

The old energy term is not bounded from below in $\vec{x}$. Therefore, any stochastic process that has a mean tendency to minimize energy will give run away solutions in $\vec{x}$. The new energy term is bounded from below. The Boltzmann distribution becomes

$$p(\vec{x}, \vec{y}, \vec{s}) = \frac{1}{Z} \exp\{-\beta \sum_{j=i}^{h} \|\vec{w}_j - \vec{x}\|^2 s_j + \sum_{j=1}^{h} \sum_{k=0}^{m} v_{jk} s_j y_k - J(\sum_{j=1}^{h} s_j - h_0)^2\} \tag{13}$$

A stochastic process that has Eq. (13) as equilibrium distribution can be simply constructed using Glauber dynamics for the neurons $\vec{s}$ and $\vec{y}$, and Metropolis dynamics for $\vec{x}$ [12, 6].

We will study the behaviour of this network for the case $h_0 = 1$. The probabilities on the visible units become:

$$
\begin{aligned}
p(\vec{x}, \vec{y}) &= \frac{1}{Z} \sum_{j=1}^{h} \exp\{-\beta \|\vec{w}_j - \vec{x}\|^2 + \sum_{k=0}^{m} v_{jk} y_k\} \\
&= \frac{1}{Z} \sum_{j=1}^{h} \exp\{H_j(\vec{x}, \vec{y})\} \qquad (14)
\end{aligned}
$$

with

$$
\begin{aligned}
Z &= \int d\vec{x} \sum_{\vec{y}} \sum_{j=1}^{h} \exp\{-\beta \|\vec{w}_j - \vec{x}\|^2 + \sum_{k=0}^{m} v_{jk} y_k\} \qquad (15) \\
&= \left(\frac{\pi}{\beta}\right)^{n/2} \sum_{j=1}^{h} \exp(v_{j0}) \Pi_{k=1}^{m} 2 \cosh(v_{jk}) \qquad (16)
\end{aligned}
$$

Note that Z is independent of $\vec{w}_j$.

In the presence of continuous valued neurons ($n > 0$), this architecture is similar to the well known Radial Basis Networks (RBNs) (see for instance [13]). We will refer to this architecture as a Radial Basis Boltzmann Machine (RBBM). The main advantages of RBBM are: 1) The 'output' $p(\vec{y}|\vec{x})$ of the RBBM naturally encodes a probability, whereas the output of the RBN encodes some continuous number; 2) The joint probability $p(\vec{x}, \vec{y})$ is modeled in the RBBM, whereas in the RBN only a feed-forward mapping is modeled. For instance, the distribution of 'inputs' that correspond to a particular 'output' is given by $p(\vec{x}|\vec{y})$; 3) The optimal values of the weights of the RBBM minimize the Kullback divergence which has a firm information theoretic basis. Instead, for the RBN different criteria are used to find the cluster centers and their relative weights. Thus, the solutions of different RBBM architectures (and other architectures derived in the BM framework) can be compared objectively on the basis of their Kullback divergence; 4) Situations where both 'input' and 'output' consists of a combination of continuous and binary values can be naturally handled. (Variables that take values in a finite alphabet can also be included in the BM framework, but require an extension of the RBBM);

It is easy to calculate the learning rules for w_{ij}, v_{jk} and v_{j0} by minimizing Eq. (3) [6]. Their computational complexity is $O(h(n+m))$ for single pattern presentation. This allows this BM to be used for practical applications.

4.1. Clustering and symmetry breaking

An interesting symmetry breaking phenomenon occurs in the hidden layer of the RBBM. Consider the network Fig. 3 with one continuous valued neuron, 16 hidden neurons and no discrete neurons. The network performs unsupervised learning on the data set depicted in Fig. 4a. Learning starts from random weights at a high temperature (small β). After the weights are converged the values of the weights are plotted and temperature is lowered.

The Boltzmann Machine finds qualitatively different solutions for different values of β. For high temperature (low β) the solution after learning is such that all weight values are identical. The hidden units simply copy each others behaviour. At lower temperatures, a number of symmetry breakings occurs, during which neurons specialize to particular

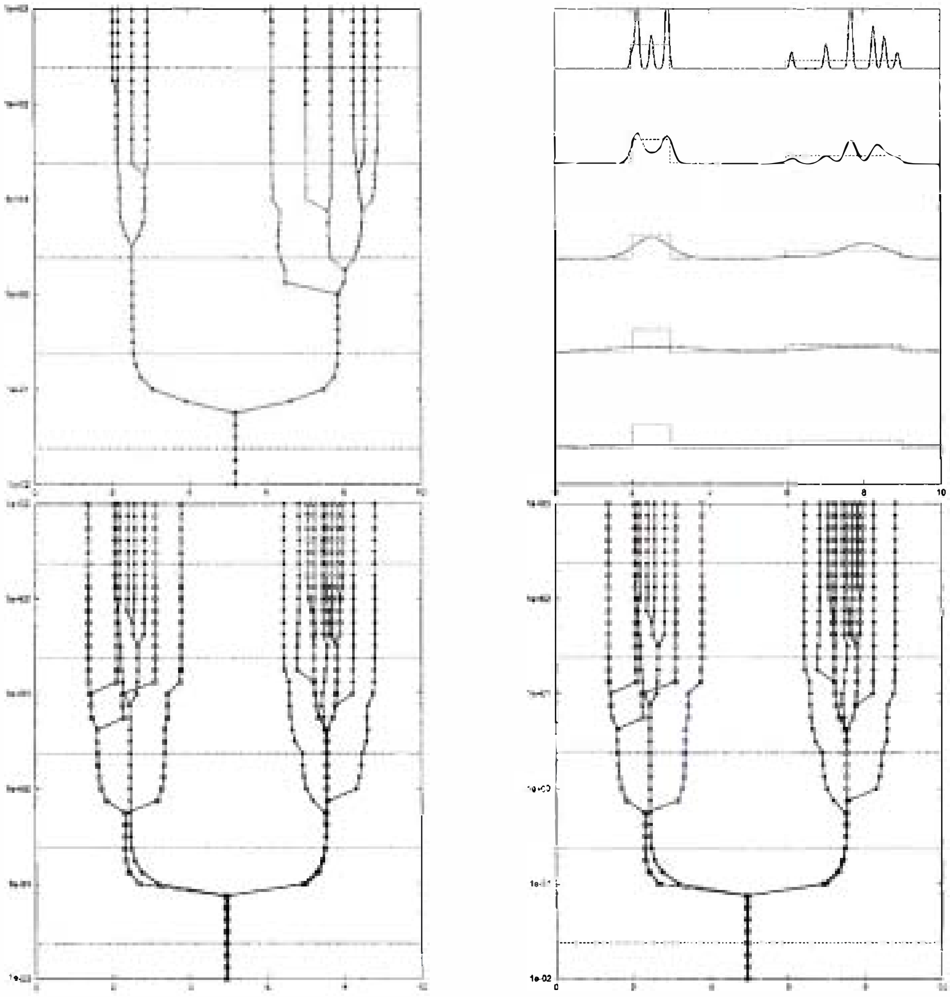

Figure 4: a) *Unsupervised clustering.* Top. One dimensional data set consisting of 240 points. Data were generated with probability 0.5 from one of 2 overlapping distributions Bottom. Optimal weight configuration vs. $\log(\beta)$ for RBBM with one continuous neuron and 20 hidden neurons. Optimal weights for consecutive values of β are connected by lines. b) *Supervised clustering.* Top. As in a) but class labels of the two distributions were also provided. The network consists of one continuous neuron, one discrete neuron (class label) and 20 hidden neurons. Circles and squares indicate specialization of hidden units to a specific class.

subregions of the input space. At some temperature, the number of clusters is correctly identified. At lower temperature, further specialization occurs in order to better fit the shape of the probability distribution of the individual clusters and finally specializes on individual data points. Thus the effective number of neurons in the hidden layer, that participate in the task at a given β can be dynamically adjusted by varying β, without the need to physically add or remove neurons.

This symmetry breaking mechanism has a close resemblance to the statistical mechanics approach to clustering, discussed by Rose et al. [14]. In fact, the Kullback divergence for the RBBM, with only continuous valued neurons in the absense of the thresholds v_{j0}, is *identical* to the free energy minimized in [14]. The temperature at which the first symmetry breaking occurs can be calculated analytically [14].

In the RBBM in its general form, the symmetry breaking phenomena will persist as long as continuous valued neurons are present. The specialization with respect to y is roughly independent of β and the specialization with respect to x is similar as in the unsupervised case.

For simplicity, the weights v_{j0} were not included in these simulations. As can be seen from Eq. (14), the role of v_{j0} is to determine the relative weight of the different Gaussian kernels. Their presence will not alter the observed symmetry breaking phenomenon but may improve the solutions.

For the RBBM with many external units and with $h_0 > 1$ active units in the hidden layer, a similar symmetry breaking pattern will occur. The specialization as a function of temperature will then occur at the level of cell assemblies. The role of individual neurons will be much less obvious, since each neuron will participate in many assemblies. At high (low) noise level, cell assemblies will describe course (detailed) features of the data. Thus, by adjusting the level of noise in the Glauber dynamics, the optimal weights can be made to represent global or detailed features of the dynamics.

In standard RBNs, it is a problem to determine the optimal number of Gaussian kernels. This problem is partly due to the fact that the number of clusters in the data is unknown, and is partly a question of optimal generalization. In the RBBN, the effective number of hidden units is controlled by β. Therefore, it becomes the problem of finding the optimal β. A practical approach is to start with low β and calculate the optimal Kullback divergence on a training set and on a test set. Increase β as above and continue until the Kullback divergence on the test set starts to increase. This approach was simulated for the 80X problem. The training cost decreases as a function of β, indicating that increased specialization of the hidden units leads to improved training set performance. However, the test cost has an optimal value for $\beta = 0.916$. Inspection of the receptive fields of the hidden units shows a specialization in 15 hidden units.

5. DISCUSSION

It was shown that by introducing lateral inhibition, hybrid architecture involving different computational principles such as feed-forward mapping, unsupervised learning and associative memory, can be modeled and analysed in the Boltzmann Machine framework. This is of great advantage for getting a better understanding of the capability of the Boltzmann Machine, and for the study of hybrid architectures in the context of neuro-biology as well as in engineering.

The analytic learning rules that can be derived for these networks allow for fast simulation on sequential machines. It should be emphasized that these learning rules are *identical* to the original BM learning rule, in the sense that for identical architectures the same weights are found after learning (except for local minima). The advantage is that for networks with restricted number of permissible states, the Glauber dynamics need not be invoked to calculate the learning rule: it can be done analytically. This makes fast simulations of these networks on conventional workstations possible.

An attractive feature of the Ackley learning rule is that it only requires locally available information. Because of the locality property, it allows for hardware realizations that scale well with problem size. This opens the possibility to design and simulate small Boltzmann Machines in software, using the approach outlined in this paper, and then to implement large real world applications in hardware, using the learning rules proposed by Ackley et al..

In this paper, we saw two examples how inhibitory connections lead to interesting architectures for which fast learning rules can be derived. For Boltzmann Perceptrons, it was shown that these architectures can solve complex real-world applications.

The Radial Basis Boltzmann Machine is an example of how different computational principles, such as feed-forward mapping and unsupervised learning, can be combined and analysed in a unified way. The possibility to include associative memory in this same network was mentioned but not further explored. The important advantage of this approach is that only one criterion (Kullback divergence) is minimized during learning. This gives a firm theoretical foundation to the issue how a hybrid architecture must allocate its resources to simultaneously implement different, possibly competing, functions.

From the construction of the Boltzmann distribution for the examples discussed in this paper it should be clear, how to construct modular architectures. For instance, for a modular architecture consisting of winner-take-all modules, the appropriate terms in Eq. (8) must be repeated for each module, and terms describing connections between modules must be added. Learning rules are immediately obtained by inserting the Boltzmann distribution in the Kullback divergence and taking the derivatives with respect to all the adaptive weights. The complexity of these learning rules is $O(h^m)$, where h is the number of permissible states per module, and m is the number of modules.

For Radial Basis Boltzmann Machines, we discovered a symmetry breaking phenomenon that is reminiscent of statistical mechanics. Thus generalization can be studied as a function of β for these networks. The shape of the Kullback divergence after training as a function of β can be calculated for simple data distributions. Comparison with the actual data may give some insight in the distribution of clusters, and thus on optimal generalization.

ACKNOWLEDGEMENTS

I would like to thank Marcel Nijman and Pierre van der Laar for providing some of the numerical results of this paper.

REFERENCES

[1] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[2] S. Kullback. *Information theory and statistics.* Wiley, New York, 1959.

[3] S. A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Systems*, 2:625–640, 1988.

[4] E. B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D.Z. Anderson, editor, *Neural Information Processing Systems*, pages 52–61, New York, 1987. American Institute of Physics.

[5] J.J. Hopfield. Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proc. Natl. Acad. Sci. USA*, 84:8429–8433, 1987.

[6] H.J. Kappen. Deterministic learning rules for boltzmann machines. *Neural Networks*, 1994. Accepted for publication.

[7] D. Ackley, G. Hinton, and T. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[8] C. Peterson and E. Hartman. Explorations of the mean field theory learning algorithm. *Neural Networks, Vol. 2*, pages 475–494, 1989.

[9] E. Yair and A. Gersho. The boltzmann perceptron network: A soft classifier. *Neural Networks*, 3:203–221, 1990.

[10] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the theory of neural computation*, volume 1 of *Santa Fe Institute.* Addison-Wesley, Redwood City, 1991.

[11] V. Lopez. Handwritten character recognition: Benchmark test for neural network engineering. Summary of talk presented at ICANN'93, 1993.

[12] M. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1091, 1953.

[13] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[14] K. Rose, E. Gurewitz, and G. Fox. Statistical mechanics of phase transitions in clustering. *Physical Review Letters*, 65:945–948, 1990.