Learning Stochastic Feedforward Networks

Radford M. Neal
Department of Computer Science
University of Toronto

November, 1990

Abstract. Connectionist learning procedures are presented for "sigmoid" and "noisy-OR" varieties of stochastic feedforward network. These networks are in the same class as the "belief networks" used in expert systems. They represent a probability distribution over a set of visible variables using hidden variables to express correlations. Conditional probability distributions can be exhibited by stochastic simulation for use in tasks such as classification. Learning from empirical data is done via a gradient-ascent method analogous to that used in Boltzmann machines, but due to the feedforward nature of the connections, the negative phase of Boltzmann machine learning is unnecessary. Experimental results show that, as a result, learning in a sigmoid feedforward network can be faster than in a Boltzmann machine. These networks have other advantages over Boltzmann machines in pattern classification and decision making applications, and provide a link between work on connectionist learning and work on the representation of expert knowledge.

Introduction

The work reported here began with the desire to find a network architecture that shared with Boltzmann machines [6, 1, 7] the capacity to learn arbitrary probability distributions over binary vectors, but that did not require the negative phase of Boltzmann machine learning. It was hypothesized that eliminating the negative phase would improve learning performance.

This goal was achieved by replacing the Boltzmann machine's symmetric connections with feedforward connections. In analogy with Boltzmann machines, the sigmoid function was used to compute the conditional probability of a unit being on from the weighted input from other units. Stochastic simulation of such a network is somewhat more complex than for a Boltzmann machine, but is still possible using local communication. Maximum likelihood, gradient-ascent learning can be done with a local Hebb-type rule.

These networks turn out to fall within the general class of "belief networks" studied by Pearl [11] and others as a means of representing probabilistic knowledge in expert systems. However, the specific network architectures considered by Pearl do not use a sigmoid probability function. Rather, they employ a "noisy-OR" model for the probability of a unit being on, based on the states of units feeding into it. It is natural to ask whether a learning procedure can be developed for this model as well. A local learning rule was indeed found for a generalization of the noisy-OR model, though this time the gradient-ascent procedure must be constrained to avoid an invalid region of weight-space.

The representational power of the two types of feedforward network were investigated, and compared to that of the Boltzmann machine. It turns out that each of these three networks can represent probability distributions over the full set of units that the other two networks cannot. With the help of "hidden" units, all these networks can represent arbitrary distributions over a set of "visible" units.

The learning performance of these networks was evaluated using a simple mixture distribution and an associated classification task. The sigmoid feedforward network was found to be capable of learning at a significantly higher rate than the Boltzmann machine. The noisy-OR network performed less well, however, and in other tasks it showed a strong tendency to get stuck at a local maximum. Additional experiments established that the sigmoid feedforward network's advantage in learning speed over the Boltzmann machine is due to the elimination of the negative phase.

This paper begins with a review of Boltzmann machines and belief networks. I then define the sigmoid and noisy-OR varieties of stochastic feedforward network, derive gradient-ascent learning rules for them, and investigate their representational power. The experiment comparing the learning performance of these networks with Boltzmann machines is then described. Finally, I show how stochastic feedforward networks relate to other connectionist approaches to statistical modeling and to work on the representation of probabilistic knowledge in expert systems, and I discuss how these networks open up new possibilities for decision making, alternative learning procedures, and biological modeling.

A review of Boltzmann machines

The Boltzmann machine [6, 1, 7] is most naturally viewed as a device for modeling a probability distribution, from which conditional distributions for use in pattern completion and classification may be derived. In the limit as probabilities approach zero and one, deterministic input-output mappings can be represented as well. These capabilities would make the Boltzmann machine attractive in many applications, were it not that its learning procedure is generally seen as being painfully slow.

Definition of Boltzmann machines. A Boltzmann machine consists of some fixed number of two-valued units linked by symmetrical connections.[1] In some formulations, the two possible values of a unit are 0 and 1; in other formulations the two values are -1 and $+1$. These alternate formulations are representationally equivalent, but can differ somewhat in learning performance.

The states of the units will be denoted by the vector $\overline{s}$, with the state of unit i being s_i. This state vector will often be regarded as a realization of a corresponding random variable $\overline{S}$. The weight on the connection between unit i and unit j will be denoted by w_{ij}. Since connections are symmetrical, $w_{ij} = w_{ji}$. Units do not connect to themselves. "Bias" weights, w_{i0}, from a fictitious unit 0 whose value is aways 1 are also assumed to be present.

The "energy" of a network with state $\overline{s}$ is defined as follows:

$$E(\overline{s}) \;=\; -\beta \sum_{j<i} s_i s_j w_{ij}$$

[1] Generalizations to units with continuous values and to networks with higher-order interactions are possible, but will not be considered in this paper.

where β is the constant 1 if units take on values of 0 and 1 or the constant $\frac{1}{2}$ if units take on values of -1 and $+1$.

This energy function induces a Boltzmann distribution over states, in which low-energy states are more probable than high-energy states. Specifically,

$$P(\overline{S} = \overline{s}) \;=\; \exp(-E(\overline{s}))\,/\,Z$$

where Z is a normalization factor needed to make the distribution sum to one:

$$Z \;=\; \sum_{\overline{s}} \exp(-E(\overline{s}))$$

Typically, some of the units in the network are "hidden", and we are interested only in the marginal distribution of the other "visible" units. We then consider the state vector $\overline{s}$ to be split into the pair $\langle \overline{h}, \overline{v} \rangle$, and similarly the random variable $\overline{S}$ becomes $\langle \overline{H}, \overline{V} \rangle$. The distribution over the visible units is then

$$P(\overline{V} = \overline{v}) \;=\; \sum_{\overline{h}} P(\overline{S} = \langle \overline{h}, \overline{v} \rangle)$$

Stochastic simulation of Boltzmann machines. Since Z is the sum of an exponentially large number of terms, directly computing the probability of a given state vector is infeasible for networks of significant size. Even if this calculation could be performed efficiently, we would still need time exponential in the number of hidden units to calculate the marginal probability of a visible vector, or the probability distribution for a subset of visible units conditional on given values for the other visible units. These distributions can, however, be exhibited via stochastic simulation, a process which is fundamental to the operation of all the networks considered in this paper.

The simulation starts with the network in an arbitrary state. Units are then repeatedly visited in turn, with a new value being selected on each visit according to the unit's probability distribution conditional on the values of all other units. For Boltzmann machines, this conditional distribution for unit i is as follows:

$$P(S_i = x \mid S_j = s_j \; : \; j \neq i) \;=\; \sigma(x^* \sum_{j \neq i} s_j w_{ij})$$

For $-1/+1$ valued units, $x^* = x$, while for $0/1$ valued units $x^* = 2x - 1$. The "sigmoid" function, $\sigma(t)$, is defined as $1/(1 + \exp(-t))$. Note that $\sigma(-t) = 1 - \sigma(t)$.

To produce a sample from the distribution over state vectors, the simulation is allowed to run for a length of time sufficient for it to settle to "equilibrium". A collection of state vectors taken at sufficiently widely separated times as the simulation continues to run will then form a sample from the distribution for $\overline{S}$. Conditional distributions can be exhibited by clamping certain units to fixed values during the simulation and updating only the values of the remaining units. This allows the network to perform pattern completion and classification tasks.

Unfortunately, it is difficult to say how much time should be allowed for the simulation to reach equilibrium, or at what interval state vectors should subsequently be taken to form the sample. The technique of "simulated annealing" is often used to reach equilibrium faster. In this method, the probability distribution used in the stochastic simulation is made more uniform by raising the probability of each state to the power $1/T$ (and then renormalizing). Here T is a "temperature" parameter, which is initially set high in order to make equilibrium easy to reach, and is then gradually reduced to 1, at which point one hopes that the equilibrium distribution for the original probabilities will have been reached.

Learning in Boltzmann machines. The learning problem for Boltzmann machines is to adjust the weights so as to make the distribution over visible units match as closely as possible the distribution of some real-world attributes, as evidenced by a set of training cases.

Adopting the maximum-likelihood approach to such estimation, we attempt to maximize the log-likelihood given the training cases, defined as

$$L \;=\; \log \prod_{\overline{v}\in\mathcal{T}} P(\overline{V}=\overline{v}) \;=\; \sum_{\overline{v}\in\mathcal{T}} \log P(\overline{V}=\overline{v})$$

where $\mathcal{T}$ is the collection of training cases (which may contain repetitions). The partial derivative of L with respect to a particular weight can be expressed as follows:

$$\frac{\partial L}{\partial w_{ij}} \;=\; \beta \sum_{\overline{v}\in\mathcal{T}} \Big(\sum_{\overline{s}} P(\overline{S}=\overline{s} \mid \overline{V}=\overline{v})\; s_i s_j \;-\; \sum_{\overline{s}} P(\overline{S}=\overline{s})\; s_i s_j \Big)$$

The above formula provides the basis for a gradient-ascent learning procedure involving two parallel stochastic simulations for each training case. In the "positive phase" simulation, the visible units are clamped to the values they take in the training case, with the result that the simulation produces a sample from the conditional distribution of $\overline{S}$ given $\overline{V}=\overline{v}$. In the "negative phase" simulation, no units are clamped, producing a sample from the unconditional distribution for $\overline{S}$. For each state vector $\overline{s}^+$ in the positive phase sample, the weight w_{ij} is incremented by a small amount proportional to $s_i^+ s_j^+$. For each state vector $\overline{s}^-$ in the negative phase sample, w_{ij} is decremented by an amount in the same proportion to $s_i^- s_j^-$. This procedure is repeated until convergence is reached.

Need for the negative phase. Intuitively, the need for a negative as well as a positive phase in Boltzmann machine learning arises from the presence of the normalizing factor, Z, in the expression for the probability of a state vector. Because of this, the direction of steepest descent in energy is not the same as that of steepest ascent in probability. The negative phase of the learning procedure is needed to account for this effect.

Looked at another way, the negative phase provides the mechanism by which the learning comes to a stop — once the correct distribution over visible units has been learned, this distribution is exhibited in the negative phase, just as it is forced in the positive phase. The positive phase increments and negative phase decrements then balance, and the weights become stable.

The presence of the negative phase has several disadvantages:

1) It directly increases computation by a factor of more than two.

2) It may make the learning procedure more sensitive to statistical errors.

3) It may reduce any biological plausibility the scheme possesses.

Regarding point (1), since the negative phase simulations have more unclamped units, they take longer to run than the positive phase simulations. Regarding point (2), the presence of a negative phase may make it necessary to collect a larger sample of state vectors from the simulations in order to reduce the variance in the estimate of the gradient of L, which will be the sum of the variances of the positive and the negative phase statistics. Taking the difference of the statistics from two phases may also exacerbate the ill-effects of not reaching equilibrium in the simulations.

On the other hand, the negative phase can be exploited to control how network resources are utilized. In particular, the network can be forced to learn an input-output mapping rather than the distribution of the input itself by clamping inputs in the negative as well as the positive phase. It will turn out that in stochastic feedforward networks, where the negative phase has been eliminated, control over what the network learns can be exercised by other means.

A look at belief networks

Belief networks, also known as "Bayesian networks", "causal networks", "influence diagrams", and "relevance diagrams", are designed, like Boltzmann machines, to represent a probability distribution over a set of attributes. Study of these networks by Pearl [11] and others [9] has been motivated principally by the desire to represent knowledge obtained from human experts, however. Accordingly, hard-to-interpret parameters such as the weights in a Boltzmann machine have been avoided in favour of more intuitive representations of conditional probabilities.

Definition of belief networks. Sticking as closely as possible to the terminology of the previous sections, we can view the state of a belief network as a vector, $\overline{s}$, with s_i being the state of unit i. In this paper, the units will always be two-valued.

The probability of a state vector is defined in terms of what I will call "forward condition probabilities" — the probability of a unit having a particular value conditional on the values of the units that precede it:

$$P(\overline{S} = \overline{s}) \;=\; \prod_i P(S_i = s_i \mid S_j = s_j \;:\; j < i)$$

The conditional probabilities above are assumed to have been given by an expert. Typically, only a subset of the units preceding unit i will be "connected" to it, and only these will be relevant in specifying its forward conditional probabilities. Note that the ordering of units in the state vector is crucial, since it determines which conditional probabilities must be specified.

Stochastic simulation of belief networks. In contrast with Boltzmann machines, computing the probability of a particular state vector for a belief network is straightforward. One can also easily generate a sample from the distribution for $\overline{S}$. However, computing conditional probabilities and sampling from conditional distributions are in general difficult problems. Various methods for computing exact conditional probabilities in belief networks have been proposed [11, 13, 8], but all are either restricted to special forms of network or have exponential time complexity in the worst case.

It appears that the only plausible method of sampling from conditional distributions in belief networks with high connectivity is stochastic simulation, described by Pearl [10, 11]. As with Boltzmann machines, a step in the simulation requires selecting a new value for unit i from its distribution conditional on the values of the other units. For a belief network, this distribution is given by the proportionality

$$P(S_i = x \mid S_j = s_j \;:\; j \neq i) \;\;\propto\;\; P(S_i = x \mid S_j = s_j \;:\; j < i) \prod_{j>i} P(S_j = s_j \mid S_i = x \;\&\; S_k = s_k \;:\; k < j,\, k \neq i)$$

For this procedure to be guaranteed to work (in the limit as the number of simulation passes grows), the forward conditional probabilities should be non-zero. The time to reach equilibrium in the simulation can be reduced by using simulated annealing, as described for Boltzmann machines.

The noisy-OR model of conditional probabilities. So far, forward conditional probabilities have been assumed to be given explicitly. In fact, this will generally not be feasible, since explicitly specifying the conditional distribution for S_i given the values of the preceding units requires 2^{i-1} parameters. Even if some of the preceding units are not connected to unit i, more compact specifications will generally be necessary.

One method, termed the "noisy-OR" model [11, 5], views the units as 0/1 valued OR-gates with the preceding units as inputs. An input of 1 does not invariably force a unit to take on the value 1, however. Rather, there is a certain probability, q_{ij}, that even though unit j has the value 1, it will fail to force a unit i that it feeds into to go to 1. Under this model, the forward conditional probabilities can be expressed in terms of the q_{ij} as follows:

$$P(S_i = 1 \mid S_j = s_j \;:\; j < i) \;\;=\;\; 1 - \prod_{j<i,\, s_j=1} q_{ij}$$

Once again, a fictitious unit 0 whose value is always 1 has been assumed, along with associated parameters q_{i0}. Note that if q_{ij} is one, unit j is effectively not connected to unit i.

Two varieties of stochastic feedforward network

We are now in a position to describe the two types of stochastic feedforward network that are investigated in this paper. The first, "sigmoid", variety was designed in analogy with Boltzmann machines. When the connection with belief networks was realized, the second variety was developed as a generalization of the "noisy-OR" model for specifying conditional probabilities.

Definition of sigmoid networks. Two formulations of sigmoid feedforward networks will be considered. In one, units take on the values 0 and 1, in the other, they take on the values -1 and $+1$. The weight on the directed connection in the feedforward network from unit j to unit i will be denoted by w_{ij}. A bias unit, 0, set permanently to 1 is assumed, with associated weights, w_{i0}. The forward conditional probabilities for sigmoid networks can then be defined as follows:

$$P(S_i = s_i \mid S_j = s_j \;:\; j < i) \;\;=\;\; \sigma(s_i^* \sum_{j<i} s_j w_{ij})$$

Here again, for $-1/+1$ valued units, $s_i^* = s_i$, while for 0/1 valued units, $s_i^* = 2s_i - 1$.

One can easily verify that a network of 0/1 valued units with forward conditional probabilities defined as above can be converted to an equivalent network of $-1/+1$ valued units with weights w'_{ij} by the transformation:

$$\begin{aligned} w'_{i0} &= w_{i0} + \sum_{0<j<i} w_{ij}/2 \\ w'_{ij} &= w_{ij}/2, \quad \text{for } 0 < j < i \end{aligned}$$

This transformation is easily inverted. The two formulations thus have equal representational power, though their performance with gradient-ascent learning may differ. A similar equivalence applies to Boltzmann machines.

The probability of a state vector, $\overline{s}$, is defined in terms of the forward conditional probabilities:

$$\begin{aligned} P(\overline{S} = \overline{s}) &= \prod_i P(S_i = s_i \mid S_j = s_j \; : \; j < i) \\ &= \prod_i \sigma(s_i^* \sum_{j<i} s_j w_{ij}) \end{aligned}$$

As with Boltzmann machines, we are often interested in the marginal distribution over a subset of "visible" units, given by

$$P(\overline{V} = \overline{v}) = \sum_{\overline{h}} P(\overline{S} = \langle \overline{h}, \overline{v} \rangle)$$

where $\overline{S}$ has been split into $\langle \overline{H}, \overline{V} \rangle$. We are also interested in conditional distributions involving subsets of visible units, as these allow one to perform tasks such as pattern completion and classification.

Stochastic simulation of sigmoid networks. To exhibit these marginal and conditional distributions via stochastic simulation, we must repeatedly select a new value for each unit from its distribution conditional on the rest of the network. This distribution is given by the proportionality

$$P(S_i = x \mid S_j = s_j \; : \; j \neq i) \;\propto\; \sigma(x^* \sum_{j<i} s_j w_{ij}) \prod_{j>i} \sigma(s_j^*(x w_{ji} + \sum_{k<j,\, k\neq i} s_k w_{jk}))$$

To select a new value from the above distribution, unit i must have available both its own total input: $\sum_{j<i} s_j w_{ij}$, and the input to each unit, j, that it feeds into, exclusive of its own contribution: $\sum_{k<j,\, k\neq i} s_k w_{jk}$. The procedure is thus somewhat more complex than that for a Boltzmann machine, but the information required can still be made available through local network communication, provided data can pass both ways along the directed connections.

A "short-cut" simulation method is possible when we wish to sample from the distribution conditional on values for some subset of visible units, and these clamped units happen to be the first ones in the state vector. In this case, rather than employ the full stochastic simulation procedure, we can simply select new values for each unclamped unit in a single forward pass, using the forward conditional probabilities. The selection for each unit depends only on the values for preceding units, and the values in the previous state vector have no effect on the result. Accordingly, no settling to equilibrium is required, and the state vectors obtained in successive passes are all independent. This short-cut can be exploited when feedforward networks are used for pattern classification or for other tasks that have the form of an input-output mapping.

Noisy-OR networks. In the "noisy-OR" form of belief network that was described previously, the probabilities, q_{ij}, that an input of 1 from unit j into unit i will be ineffective in forcing unit i to 1 can be replaced with weights defined by $w_{ij} = -\log q_{ij}$. The forward conditional probabilities can then be written as follows:

$$P(S_i = 1 \mid S_j = s_j \ : j < i) \ = \ 1 - \exp(-\sum_{j<i} s_j w_{ij})$$

Here, units take on values of 0 and 1, and a unit 0 set permanently to 1 exists.

In the above formulation, all weights are non-negative. However, the conditional probability specification will be valid even when some weights are negative, provided that the weighted input to a unit cannot be negative, no matter what states the preceding units have. For a network with 0/1 valued units, this is equivalent to the constraint that, for all i:

$$w_{i0} \ + \sum_{j<i,\, w_{ij}<0} w_{ij} \ \geq \ 0$$

With this generalization, units can behave not only as OR gates, but also as OR gates with some or all inputs negated. For example, if unit 3 has input weights of $w_{30} = +20$, $w_{31} = -10$, and $w_{32} = -10$, it will behave as a slightly noisy OR gate with negated inputs from units 1 and 2 (i.e. as a NAND gate).

Noisy-OR networks can also be formulated with $-1/+1$ valued units. Forward conditional probabilities are defined as above, with the constraint that to be valid, the weights must satisfy the following, for all i:

$$w_{i0} \ - \sum_{0<j<i} |w_{ij}| \ \geq \ 0$$

The same equivalence between 0/1 and $-1/+1$ formulations that applied to sigmoid networks applies to noisy-OR networks as well.

Conditional probability distributions for noisy-OR networks can be exhibited via a stochastic simulation process entirely analogous to that described above for sigmoid networks.

Learning stochastic feedforward networks

The feedforward networks that have been described are of interest principally because they can be learned from empirical data. Except for the lack of a negative phase, their learning procedures are similar to that used for the Boltzmann machine.

These learning procedures are all based on the widely-used method of maximum-likelihood estimation. One should realize that this method is prone to "overfitting" the data when the amount of training data is small in relation to the number of free network parameters, with the result that the network generalizes poorly to future cases. Also, all these procedures use gradient-ascent to try to find a set of weights maximizing the likelihood. This may lead to the learning getting stuck at a point that is a local but not global maximum of the likelihood.[2]

[2] Since the gradients are evaluated stochastically, there is in fact some non-zero probability of the weights being driven to any given set of values. Consequently, in the long run, the learning procedure will spend most of its time in the vicinity of the global maximum. The "long run" in this case can be very long indeed, however, and in practice the learning procedures occasionally end up in local maxima that are effectively stable.

Learning in sigmoid networks. In the learning scenario assumed here, we have a collection, $\mathcal{T}$, of training cases drawn from the distribution of interest. Each training case consists of the values for certain attributes, assumed here to be two-valued. Exact repetitions are possible, indeed expected, in proportion to how common a particular combination of attributes is.

We wish to model the distribution from which the training sample was drawn. To do this, we decide on some size for a state vector, $\overline{S}$, for our network, and then select some subset, $\overline{V}$, of units in the state vector to represent the attributes in the training cases. The remaining, "hidden", units will constitute the set $\overline{H}$. Note that since the ordering of units in the state vector is significant for feedforward nets, different selections for the subset of visible units may give different results. This is discussed further below.

We now wish to find values for the network weights that maximize the likelihood given the training cases. However, to avoid overfitting or to reduce computational expense, we might decide to fix certain weights at zero, based on our *a priori* knowledge of the task. Other weights will be set to zero (or to small random values if we wish to break symmetry faster) and then adjusted by gradient-ascent so as to maximize the log-likelihood:

$$L \;=\; \log \prod_{\overline{v} \in \mathcal{T}} P(\overline{V} = \overline{v}) \;=\; \sum_{\overline{v} \in \mathcal{T}} \log P(\overline{V} = \overline{v})$$

For a sigmoid feedforward network, the partial derivatives of the log-likelihood with respect to the weights may be found as follows:

$$\begin{aligned}
\frac{\partial L}{\partial w_{ij}} &= \sum_{\overline{v} \in \mathcal{T}} \frac{1}{P(\overline{V} = \overline{v})} \frac{\partial P(\overline{V} = \overline{v})}{\partial w_{ij}} \\
&= \sum_{\overline{v} \in \mathcal{T}} \frac{1}{P(\overline{V} = \overline{v})} \sum_{\overline{h}} \frac{\partial P(\overline{S} = \langle \overline{h}, \overline{v} \rangle)}{\partial w_{ij}} \\
&= \sum_{\overline{v} \in \mathcal{T}} \sum_{\overline{h}} P(\overline{S} = \langle \overline{h}, \overline{v} \rangle |\; \overline{V} = \overline{v}) \; \frac{1}{P(\overline{S} = \langle \overline{h}, \overline{v} \rangle)} \; \frac{\partial P(\overline{S} = \langle \overline{h}, \overline{v} \rangle)}{\partial w_{ij}} \\
&= \sum_{\overline{v} \in \mathcal{T}} \sum_{\overline{s}} P(\overline{S} = \overline{s} \mid \overline{V} = \overline{v}) \; \frac{1}{P(\overline{S} = \overline{s})} \; \frac{\partial P(\overline{S} = \overline{s})}{\partial w_{ij}} \\
&= \sum_{\overline{v} \in \mathcal{T}} \sum_{\overline{s}} P(\overline{S} = \overline{s} \mid \overline{V} = \overline{v}) \; \frac{1}{\sigma(s_i^* \sum_{k<i} s_k w_{ik})} \; \frac{\partial \sigma(s_i^* \sum_{k<i} s_k w_{ik})}{\partial w_{ij}} \\
&= \sum_{\overline{v} \in \mathcal{T}} \sum_{\overline{s}} P(\overline{S} = \overline{s} \mid \overline{V} = \overline{v}) \; s_i^* s_j \;\; \sigma(-s_i^* \sum_{k<i} s_k w_{ik})
\end{aligned}$$

The last step uses the fact that $\sigma'(t) = \sigma(t)\sigma(-t)$.

These partial derivatives can be evaluated by running a separate stochastic simulation of the network for each training case, clamping the visible units to the values they take in that training case and observing the state vectors that arise as a result. If the simulation is run "long enough", these observations will form a sample from the conditional distribution for $\overline{S}$ given the values in the current training case. Incrementing each weight, w_{ij}, by a small amount proportional to the average value of $s_i^* s_j \; \sigma(-s_i^* \sum_{k<i} s_k w_{ik})$ over the samples for all training cases will then move the weights along the gradient toward a local maximum of the likelihood.

Various detailed implementations of this procedure are possible, as will be discussed in the section on experimental results.

Intuitively, only a single phase is needed to learn a sigmoid feedforward network because normalization of the probability distribution over state vectors is accomplished locally at each unit via the sigmoid function, rather than globally via the hard-to-compute normalization constant, Z. The role of the Boltzmann machine's negative phase in stopping learning once the distribution has been correctly modeled is taken over by the factor $\sigma(-s_i^* \sum_{k<i} s_k w_{ik})$ used to weight the learning increments. In the limiting case where unit i is learning a deterministic function of the preceding units, for example, this factor is the probability that unit i would be set to the wrong value in an unclamped network. As the correct function is approached, this factor becomes zero, and the learning stops.

Learning in noisy-OR networks. Learning in noisy-OR networks is analogous to that in sigmoid networks, with the added complication that the gradient-ascent procedure must be constrained to the region of weight-space that produces valid probabilities for state vectors.

The partial derivatives of the log-likelihood with respect to the weights in a noisy-OR network can be expressed as follows:

$$\begin{aligned}
\frac{\partial L}{\partial w_{ij}} &= \sum_{\overline{v}\in\mathcal{T}} \sum_{\overline{s}} P(\overline{S}=\overline{s} \mid \overline{V}=\overline{v}) \, \frac{1}{P(\overline{S}=\overline{s})} \, \frac{\partial P(\overline{S}=\overline{s})}{\partial w_{ij}} \\
&= \sum_{\overline{v}\in\mathcal{T}} \sum_{\overline{s}} P(\overline{S}=\overline{s} \mid \overline{V}=\overline{v}) \left\{ \begin{array}{ll} -s_j + s_j/(1-\exp(-\sum\limits_{k<i} s_k w_{ik})) & \text{if } s_i = 1 \\ -s_j & \text{if } s_i \neq 1 \end{array} \right\}
\end{aligned}$$

This formula is valid both for networks with $0/1$ valued units and for those with $-1/+1$ valued units.

These derivatives are computed via stochastic simulation and used to perform gradient-ascent learning as described above for sigmoid networks. For noisy-OR networks, however, we must also ensure that the weights always define a valid probability distribution. In fact, in order for the simulations to reach equilibrium in a reasonable amount of time, it is desirable to further constrain the weights so that the conditional probability of unit i being 1 given the values of the preceding units is at least some minimum. For a noisy-OR network with $0/1$ valued units, this will be so provided that

$$w_{i0} \; + \sum_{j<i,\, w_{ij}<0} w_{ij} \;\geq\; \eta$$

Where η is some small positive constant. This can be ensured by applying the procedure in Figure 1 to the weights for unit i after each movement along the gradient. One can show[3] that this procedure moves the weights to the set of valid values that is closest in Euclidean distance to the previous set. Since the valid region of weight space is convex, it follows that if one starts with a valid initial set of weights, moves in the direction of the gradient, and then applies the above procedure, the resulting total movement will have a positive projection in the direction of the gradient whenever this is possible.

An analogous constraint procedure exists for noisy-OR networks with $-1/+1$ valued units.

[3] See claim 1 in the appendix.

Loop:
  $C \leftarrow \{0\} \cup \{j : 0 < j < i \;\&\; w_{ij} < 0\}$
  $t \leftarrow \sum_{j \in C} w_{ij}$
  If $t \geq \eta$ then exit loop
  $d \leftarrow (\eta - t)/|C|$
  For each $j \in C - \{0\}$: if $|w_{ij}| < d$ then $d \leftarrow |w_{ij}|$
  For each $j \in C$: $w_{ij} \leftarrow w_{ij} + d$
End loop

Figure 1: Procedure to move the weights into unit i to the valid region.

Controlling what is learned. When training a Boltzmann machine, one can control what the network learns by clamping certain units in the negative as well as the positive phase. This is commonly done when only the mapping from a set of "input" attributes to a set of "output" attributes is of interest. Clamping the input units in both phases forces the hidden units to model the conditional distribution of the output given the input, rather than the distribution of the input itself.

The same technique could be used with stochastic feedforward networks, but this would naturally require re-introduction of a negative learning phase, the elimination of which was the main motivation for adopting a feedforward structure. Fortunately, one can achieve similar control via judicious placement of input, output, and hidden units within a feedforward network.

Three network architectures that illustrate the control possible are shown in Figure 2, using a medical diagnosis problem as an example. In all cases, a set of visible "symptom" units are used to represent various attributes of a patient, and a set of visible "disease" units are used to encode a diagnosis. There are assumed to be no connections among the units within each visible set.

In network (a), the hidden units feed into both sets of visible units. As a result of training, these units may come to model correlations among symptom units, among disease units, or between the symptoms and the disease. If the network succeeds in modeling the total distribution perfectly, it will be capable of performing any sort of pattern completion task. For example, one could clamp a set of symptoms and then observe the most likely diseases as the network is stochastically simulated, or, conversely, one could clamp a disease and observe the most likely symptoms. However, if the number of hidden units is insufficient to model the total distribution, the net will end up modeling whichever correlations are strongest, and these might not be the ones that are most important for diagnosis.

In network (b), the hidden units are placed between the symptom units and the disease units. This forces the hidden units to learn to model the conditional distribution of the disease units given the symptoms. One could then clamp a set of symptoms and observe the most likely diseases. In fact, this can be done using the short-cut simulation procedure described previously, since the clamped symptom units precede all the unclamped units (the full simulation procedure is still required during learning). The converse operation of clamping a disease and observing likely symptoms no longer works well, however, since there are no hidden units in a position to model correlations among symptoms.

Network (c) adds a set of hidden units prior to the symptom units in order to capture such

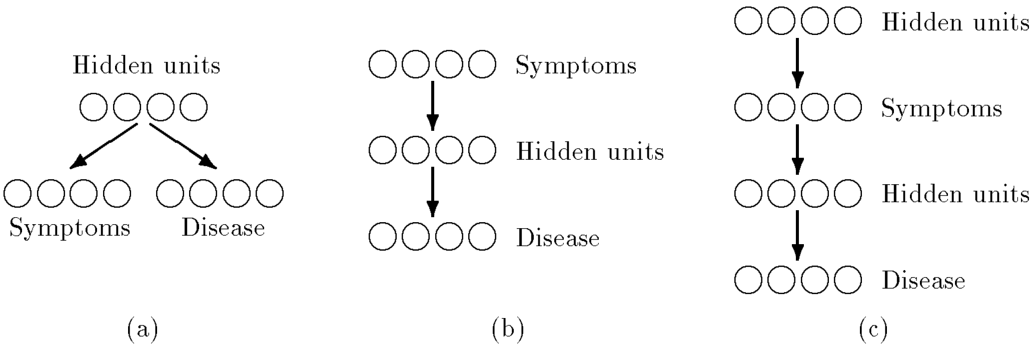

Figure 2: Three network architectures for a medical diagnosis problem.

correlations. This network has capabilities comparable to those of network (a), with the difference that the number of hidden units devoted to modeling each type of correlation is under the control of the network designer. Network (c) might be appropriate for a diagnosis application in which knowledge of correlations among symptoms is sometimes needed in order to fill in missing symptom values.

Representational power of feedforward networks

In this section I will investigate how powerful the various forms of stochastic feedforward network are at representing probability distributions. I will first consider sigmoid feedforward networks, and then discuss the noisy-OR variety. Recall that as shown earlier, there is no difference in the representational power of networks with 0/1 valued units and those with $-1/+1$ valued units.

Power of sigmoid feedforward nets and Boltzmann machines. Let us consider first the relative capacity of sigmoid feedforward networks and Boltzmann machines to represent probability distributions over the full state vector, $\overline{S}$. The ability of these networks to produce marginal distributions over a subset, $\overline{V}$, of visible units will be discussed later.

One can show[4] that any distribution over one or two units can be approximated arbitrarily closely by either a Boltzmann machine or a sigmoid feedforward network. For networks of three units, the restricted set of possible probability distributions turns out to be the same for the two types of network.[5] With four or more units, however, both Boltzmann machines and sigmoid feedforward networks can represent probability distributions that the other cannot.

This is illustrated in Figure 3, which shows a Boltzmann machine than cannot be translated into a sigmoid feedforward network, and a sigmoid feedforward network that cannot be translated into a Boltzmann machine. In both cases, 0/1 valued units are used, and absent connection weights are assumed to be zero. An unattached connection is from the bias unit.

To show that Boltzmann machine of Figure 3(a) cannot be translated to an equivalent sigmoid feedforward network, one can proceed as follows. First, since the Boltzmann network is symmetrical, there is no significant choice of ordering in trying to construct an equivalent sigmoid

[4] See claim 2 in the appendix.
[5] See claim 3 in the appendix

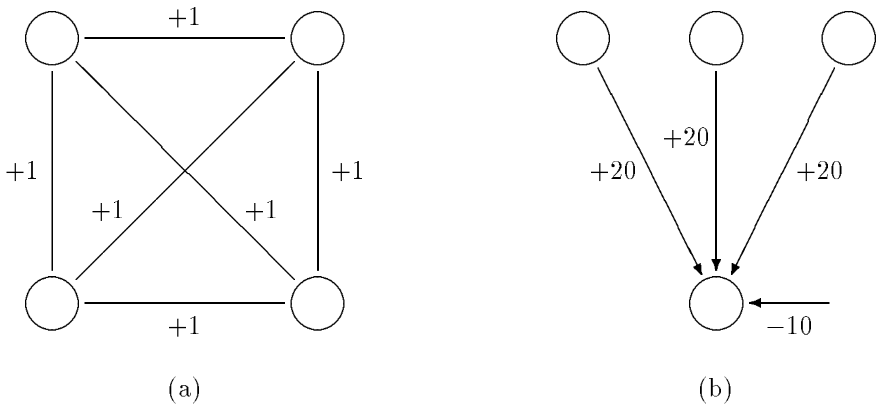

Figure 3: An untranslatable Boltzmann machine (a) and sigmoid network (b).

net. Next, note that the weights into the last unit in the sigmoid net must be identical to the weights in the Boltzmann machine in order to produce the correct conditional probability distributions for the last unit's value given the values of the other units. One can then show[6] that there is no way to set the weights into the second-to-last unit so as to produce the required conditional distributions given the various settings of the other units.

Turning to the sigmoid feedforward network of Figure 3(b), note that the lower unit is (almost) the OR of the three upper units. This mapping can be duplicated in a Boltzmann machine by simply using the same connection weights, but one can verify[7] that it is then impossible to arrange for the eight possible states of the upper three units to be equally probable, as they are in the original feedforward network.

Representing mixture distributions. Suppose we are interested in the probability distribution over a vector of "visible" units, $\overline{V}$. As seen above, in general, not all such distributions will be representable in a net consisting of these visible units alone. This problem can be overcome by including additional "hidden" units in a network.

In particular, sigmoid feedforward networks (as well as Boltzmann machines) can use hidden units to represent visible distributions that are "mixtures" of several other distributions. Such a mixture distribution can be written as follows:

$$P(\overline{V} = \overline{v}) \;=\; \sum_m P(\overline{V} = \overline{v} \mid M = m)\, P(M = m)$$

The hidden variable M identifies a "component" of the mixture. Each component produces its own distribution for $\overline{V}$, and these component distributions are then combined in the proportions $P(M)$. Mixture distributions are commonly encountered and much studied [15].

To represent a mixture distribution in a network, we need first to represent the mixture variable, M. For a mixture of n components, one way to do this is via a cluster of n units, exactly one of which is on at any time. Figure 4 shows how a three-unit cluster of this sort can be constructed for both a Boltzmann machine and a sigmoid feedforward network (with 0/1 valued units). In both cases, the three state vectors with exactly one unit on have nearly equal probabilities, and

[6] See claim 4 in the appendix.
[7] See claim 5 in the appendix.

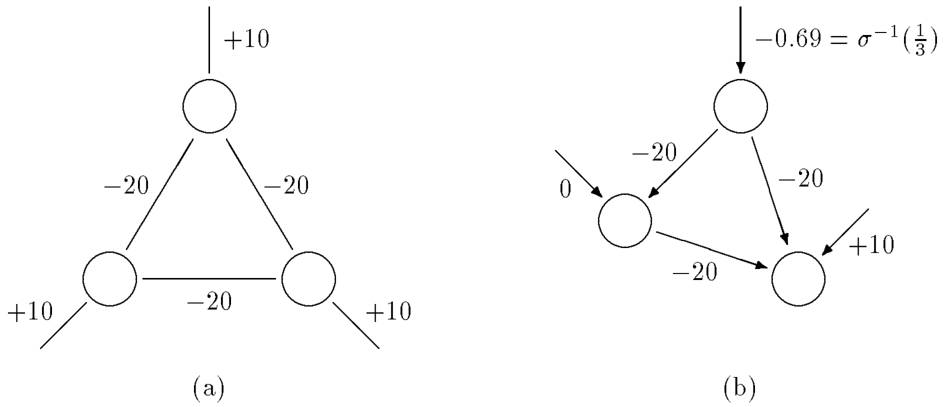

Figure 4: 1-in-3 clusters in a Boltzmann machine (a) and a sigmoid network (b).

all other state vectors have very small probability. The constructions generalize to clusters of any size, and to clusters in which the possible state vectors have unequal probabilities.

A 1-in-n cluster can directly implement a mixture in which the component distributions assign independent probabilities to the various visible units. For feedforward networks, all that is required is to connect each cluster unit to the visible units using weights that produce the required conditional probabilities. For Boltzmann machines, after making these connections to visible units, one must also adjust the bias weights to the cluster units in order to re-create the correct mixture proportions.

With more complex mixtures, each component distribution would be represented by a subnetwork. The results of one subnetwork would then be selected for routing to the visible units according to which of the cluster units is on, using simulations of AND and OR gates.

Note that any distribution over k visible units can be represented as a mixture of 2^k component distributions, each of which generates but a single vector. It follows that sigmoid feedforward networks, and Boltzmann machines, can approximate any distribution over k visible units arbitrarily closely, provided one is prepared to employ 2^k hidden units.

Power of noisy-OR networks. Unlike sigmoid feedforward units, the ability of a noisy-OR network to represent a distribution is sensitive to negation of the unit values. For example, there is no way to make a noisy-OR unit behave as an AND gate, but one can make one behave as a NAND gate. This sensitivity is of significance only for visible units, since the output of a hidden unit can always be implicitly negated by negating the weights on all its out-going connections.

In view of the above, there are certainly distributions over $\overline{S}$ that both a Boltzmann machine and a sigmoid feedforward network can implement but which a noisy-OR network cannot. Conversely, one can show[8] that the distribution produced by a three-unit noisy-OR network with 0/1 valued units in which $w_{31} = w_{32} = 1$ and all other weights are zero cannot be duplicated by either a Boltzmann machine or a sigmoid feedforward network with only three units.

[8] See claim 6 in the appendix

A 1-in-n cluster similar to that of Figure 3(b) but with the unit values negated can be constructed from noisy-OR units. Such a cluster can be used to represent a mixture distribution over visible units using a noisy-OR network, just as with sigmoid feedforward networks and Boltzmann machines.

Empirical comparison with Boltzmann machines

In this section, I will describe an experiment in which the learning procedures for stochastic feedforward networks and for Boltzmann machines were compared on the task of learning a simple mixture distribution and classifying items derived from it.

Objectives of the experiment. This experiment is intended to answer the following questions:

1) Are the learning procedures for stochastic feedforward networks capable in practice of learning an approximation to a non-trivial distribution, as evidenced by a set of training cases?

2) If so, how does the speed of learning in sigmoid feedforward networks compare to the speed of learning in a Boltzmann machine?

3) Can differences in learning speed between sigmoid feedforward networks and the Boltzmann machine be attributed to the lack of a negative phase in the learning procedure for the feedforward networks?

4) Are there differences in the learning performances of networks with 0/1 valued units and those with $-1/+1$ valued units?

5) How does learning in noisy-OR networks compare to learning in sigmoid feedforward networks?

6) How well do the solutions learned by the various networks on the basis of training data generalize to the true distribution?

Regarding points (2) and (3), the expectation is that the negative phase adds additional noise to the estimation of the gradient, and that this noise is detrimental to the learning process in Boltzmann machines. The magnitude of this effect is hard to judge, however. The added noise could even be beneficial, if it allows the network to escape local maxima during learning.

Concerning point (4), there is reason to think that learning performance might be worse for networks with 0/1 valued units than for networks with $-1/+1$ units. With 0/1 valued units, weight w_{ij} is modified only at times when unit j has the value 1. Superficially, at least, this seems inefficient. Also, since the transformation between 0/1 valued and $-1/+1$ valued networks affects the bias weights differently from the others, gradient ascent will operate differently in the two formulations.

Regarding point (6), one would expect any problems with overfitting to be similar for the various networks, since they all have the same number of free parameters.

The learning procedure used. Numerous variations of the Boltzmann machine learning procedure have been tried [4], each of which requires fixing a number of parameters, such as the learning rate, and the temperatures in an annealing schedule. This presents a problem in comparing learning in Boltzmann machines to learning in stochastic feedforward networks — a valid comparison would require searching for the optimal parameter settings for each type of network, which would be a rather large undertaking.

The approach I have adopted is to train both types of network using a simple method that has only one adjustable parameter — the learning rate, ϵ. A complete picture of the performance of each type of network for various values of ϵ can be obtained with a reasonable number of runs, and the relative performance of the different networks with their best ϵ can then be compared.

The procedure used can be characterized as follows:

1) Learning was done in "batch" mode — i.e. each change to the weights was made on the basis of the entire set of training cases.

2) Each training case was clamped into a separate copy of the network, where a separate stochastic simulation was run.[9] For Boltzmann machines, there was also an unclamped negative phase copy of the network associated with each training case.

3) No annealing was done.

4) The state of each copy of the network was retained after each change to the weights, on the assumption that if the weight changes are "small", these existing simulation states will be close to equilibrium, and that a single pass over the units in each simulation will thus produce state vectors that are reasonable for the new weights.

5) Changes to the weights were made after each simulation pass, based on the sample consisting of the current state vectors from the simulations for all training cases (plus the state vectors from the negative phase simulations, for Boltzmann machines).

6) Weight changes were scaled by a learning rate parameter, ϵ.

7) Weights were set to zero initially. Symmetry was broken by the stochastic nature of the simulation.

In detail, the weights in the Boltzmann machines were changed by

$$\Delta w_{ij} \;=\; \beta\frac{\epsilon}{N}\left(\sum_{\bar{s}^+\in\mathcal{T}^+} s_i^+ s_j^+ \;-\; \sum_{\bar{s}^-\in\mathcal{T}^-} s_i^- s_j^-\right)$$

Here, $\mathcal{T}^+$ is the set of current state vectors from the positive phase simulations (one per training case), and $\mathcal{T}^-$ is the set of state vectors from the corresponding negative phase simulations. N is the number of training cases.

Similarly, the weights in the sigmoid feedforward networks were changed by

$$\Delta w_{ij} \;=\; \frac{\epsilon}{N}\sum_{\bar{s}\in\mathcal{T}} s_i^* s_j \;\sigma(-s_i^* \sum_{k<i} s_k w_{ik})$$

and those in the noisy-OR networks by

$$\Delta w_{ij} \;=\; \frac{\epsilon}{N}\sum_{\bar{s}\in\mathcal{T}}\left\{\begin{array}{ll} -s_j + s_j/(1-\exp(-\sum_{k<i} s_k w_{ik})) & \text{if } s_i = 1 \\ -s_j & \text{if } s_i \neq 1 \end{array}\right\}$$

[9] This aspect of the learning procedure appears to be advantageous from an engineering point of view, but is quite implausible in a biological context.

m	$P(M=m)$	$P(V_i = v_i \mid M = m),\ i = 1..9$								
1	0.25	0.8	0.8	0.8	0.8	0.2	0.2	0.2	0.2	1.0
2	0.25	0.2	0.2	0.2	0.2	0.8	0.8	0.8	0.8	1.0
3	0.25	0.8	0.8	0.2	0.2	0.8	0.8	0.2	0.2	0.0
4	0.25	0.2	0.2	0.8	0.8	0.2	0.2	0.8	0.8	0.0

Table 1: The mixture distribution to be learned.

Weight changes in noisy-OR networks were limited to a magnitude of no more than 1, to avoid the possibility of huge weight changes resulting from the division above. After all changes were made, the constraint procedure of Figure 1 with $\eta = 2^{-7}$ was applied.[10]

The lack of annealing in this procedure is unconventional, as is the changing of weights based on a single state vector from each training case. The rationale behind these choices is that as ϵ approaches zero, the simulations will necessarily approach equilibrium, as they will run for many passes with the weights essentially unchanged. Furthermore, the cumulative effect of many changes with a small ϵ that are based on a single state vector from each training case will be equivalent to a single change with a larger ϵ that is based on a larger sample. As ϵ approaches zero, the learning procedure used will thus "do the right thing".

Whether this procedure is better or worse than previous methods is not important for this experiment, however, provided only that any differences in learning performance between the various networks seen using this procedure will show up in some guise in any other implementation. It is difficult to see how this can be guaranteed, but some assurance was sought by running additional experiments to see whether the differences observed were sensitive to adding annealing to the procedure or to collecting a larger sample before changing the weights.

The task learned. The networks were evaluated on the task of learning the mixture distribution shown in Table 1. There are four equally-probable mixture components, each of which produces a distribution over nine visible attributes in which each attribute is independent of the others (given knowledge of the mixture component).

All the networks tested had a similar structure. Six interconnected hidden units were provided to allow the network to model the mixture variable, using a cluster such as in Figure 4. (Four hidden units would have sufficed; six were provided to help avoid problems with local maxima.) These hidden units were connected to a set of nine visible units. For the feedforward networks, these connections were directed from the hidden units, to the visible units. The visible units were not connected to each other. All units had a bias connection.

Since the task is to model the entire distribution, the negative phase in Boltzmann machines was left completely unclamped.

The entropy of the target distribution is 7.67 bits. For this experiment, the number of training

[10] Further implementation details: Weights were kept as fixed-point values with a precision of 2^{-12} and limited in magnitude to no more than 16. This allowed computation by table look-up, and prevented round-off accumulation during incremental maintenance of the total input to each unit. Weights were rounded to a precision of 2^{-12} by adding a random number in the range 0 to 2^{-12} and then taking the floor. Very small gradients can then still have an effect.

cases used, N, was 250. The particular set of cases generated at random and used in these runs had an average value of $-\log_2 P(\overline{V} = \overline{v})$ of 7.87 bits, where $P(\cdot)$ is the true probability distribution. This is close to the entropy, as expected. This value is the target for $-L/N$ in network training, but due to overfitting, the training procedures might well reach values even lower than this.

The networks were also evaluated on the task of guessing the last attribute given the values of the other eight attributes. With knowledge of the real distribution, the optimal error rate on this classification task is 18.6%. Note that performance on this task is not the formal learning objective, and need not, in fact, be monotonically related to the actual objective of maximizing the likelihood.

Evaluation method. Typically, stochastic simulation is used when applying networks such as these to a problem instance, as well as when training them. For example, the classification task would be performed by clamping the values of the eight known attributes and observing which value for the unknown ninth attribute shows up most often as the network is simulated.

This method was *not* used for most of the evaluations in this experiment, however. Instead, the exact probabilities of all 2^{15} states of the trained network were computed, and from these, the log-likelihood given the training data, the cross-entropy with the real distribution, and performance on the classification task for the training data and for items drawn from the real distribution were all calculated.

Of course, this method is infeasible for networks that are even slightly larger than the ones used here. It is convenient for this experiment since it eliminates statistical noise from the evaluations. In the tests of generalization performance, the classification task was performed using stochastic simulation as well as with exact probabilities, and results were similar, as reported below.

Comparing sigmoid feedforward nets and Boltzmann machines. The 250 training cases drawn from the mixture distribution were used to train both sigmoid feedforward networks and Boltzmann machines, using values for ϵ of $\frac{1}{4}$, $\frac{1}{2}$, 1, 2, and so on until network behaviour became unstable. Networks with 0/1 valued units and those with $-1/+1$ valued units were both tried.

Illustrative results are shown in Figure 5, for $-1/+1$ valued units and ϵ of $\frac{1}{4}$ and 1. Three runs are shown, in which different random seeds were used in the simulations. During each run, the log-likelihood, L, was computed exactly after 25, 50, 100, 200, 400, and 800 simulation passes. (Recall that each pass consists of a (potential) change to each unit value in each simulation, and that weights are changed after each pass.) The value of $-L/N$ in bits (i.e. using base 2 logarithms) is plotted. It is 9 bits initially, since with zero weights all of the 9-element visible vectors are equally probable.

With $\epsilon = \frac{1}{4}$, the Boltzmann machine and the sigmoid feedforward network behaved similarly. As ϵ was increased, however, the Boltzmann machine became unstable. This is seen in the figure for $\epsilon = 1$, where the Boltzmann machine reached the 8.25 bit performance level, but thereafter failed to improve consistently. In contrast, the sigmoid feedforward network with $\epsilon = 1$ simply learned at four times the rate that it did with $\epsilon = \frac{1}{4}$. For larger ϵ, the Boltzmann machine became even more unstable, while the sigmoid feedforward network tolerated learning rates up to $\epsilon = 4$ before becoming unstable at $\epsilon = 8$ and above.

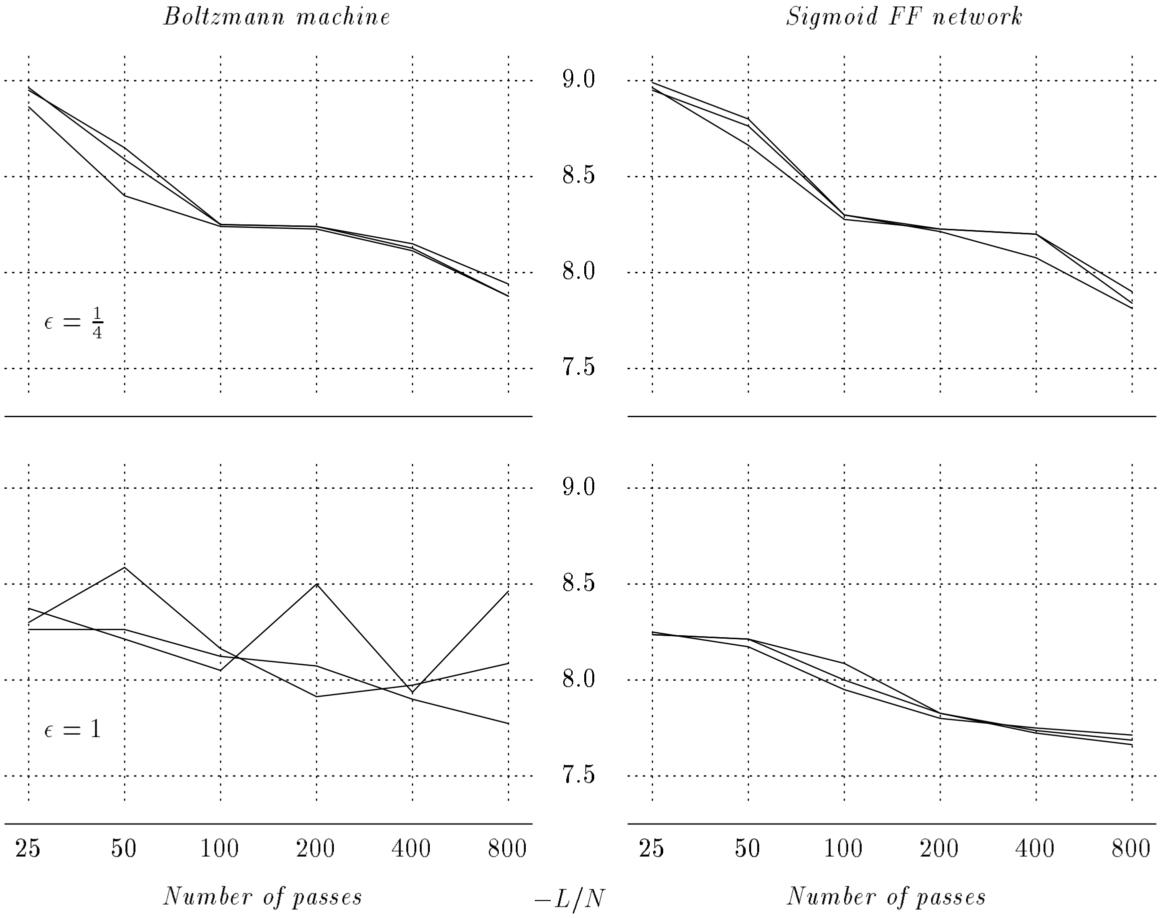

Figure 5: Learning performance of Boltzmann machines and sigmoid feedforward networks with $-1/+1$ valued units for ϵ of $\frac{1}{4}$ and 1. Three runs with different random number seeds are shown.

Type of network	*Time/pass*	*Best* ϵ	*Values of* $-L/N$	*Error rates*
Boltzmann machine (0/1 units)	0.39 s	$\frac{1}{2}$	8.30 8.22 8.23	35% 30% 33%
Boltzmann machine ($-1/+1$ units)	0.58 s	$\frac{1}{2}$	7.94 8.25 8.10	19% 37% 25%
Sigmoid FF network (0/1 units)	0.30 s	4	7.76 7.82 7.77	19% 19% 17%
Sigmoid FF network ($-1/+1$ units)	0.35 s	2	7.72 7.74 7.74	17% 17% 16%

Table 2: Best performances after 200 learning passes (three runs each).

The instability of the Boltzmann machine with $\epsilon = 1$ was examined at a finer time scale by evaluating the network after every learning pass for one of the runs, as performance went from 8.29 bits for $-L/N$ at pass 25 to 8.58 bits at pass 50. Changes in $-L/N$ of as much as 0.41 bits were seen after single learning passes, and $-L/N$ ranged in value from 8.25 bits to 8.83 bits during this interval. Examination at this time scale of learning in a sigmoid feedforward network simply shows steady improvement.

Results using 0/1 valued units were similar, except that learning was slower for a given value of ϵ in both types of network. This was largely compensated for with sigmoid feedforward networks by the fact that a larger ϵ could be used before instability set in. With Boltzmann machines, however, there appeared to be some net advantage for the $-1/+1$ formulation.[11]

The relative performance of these networks is shown in Table 2, under the assumption that learning must be stopped after 200 passes. This would produce a fair comparison if the computation time per pass was the same for all networks. In fact, Boltzmann machine passes require somewhat more time, as would be expected from the need to simulate negative phase cases, so the comparison is somewhat biased in favour of Boltzmann machines. (The times shown, measured on a machine rated at approximately 20 MIPS, should not be taken too seriously, since they are affected by many implementation factors that may not be of general significance.)

The entries in Table 2 were produced by selecting the value of ϵ that gave the best value of $-L/N$ after 200 passes, averaged over the three runs that were done. These values are shown along with the corresponding error rates when guessing the last attribute of the training cases from the first eight attributes. The superiority of the sigmoid feedforward networks is evident. The high error rates for the Boltzmann machine (especially using 0/1 valued units) is due to the fact that all these networks initially learn correlations among the first eight attributes, and only later discover how the ninth attribute relates to these. Four of the six Boltzmann machine runs had not progressed far into the second stage after 200 passes.

[11] Clear differences between the 0/1 and $-1/+1$ formulations are seen in other problems. For example, with both Boltzmann machines and sigmoid feedforward networks, learning to assign high probability to only those 4-bit visible vectors with odd parity, using four hidden units to express correlations, is much easier with $-1/+1$ units than with 0/1 units.

Further experiments. Additional experiments were done to check whether the superiority seen for sigmoid feedforward networks was related to the particular learning procedure used.

One unconventional aspect of the learning procedure is that weights were changed based on a single state vector from each simulation. An experiment was done to compare this method with one of equal cost in which changes where based on more than one state vector, under the assumption that the simulation passes dominate the computational cost. In this experiment, weight changes were calculated as before, but actual updates were performed only every ten passes, at which time the previously calculated, but deferred, changes where all made simultaneously. The changes to weights were thus based on a sample of ten state vectors from each training case simulation, rather than on a single state vector.

This method resulted in uniformly poorer results for both Boltzmann machines and sigmoid feedforward networks. For small values of ϵ, learning was slower than when changes were made immediately. This might be expected, since information is not being exploited as early. Furthermore, learning became unstable at a smaller value of ϵ. This might also be expected, since the maximum amount by which a weight can change without feedback is greater.

Experiments were also performed to determine whether the Boltzmann machine's instability at values of ϵ for which the sigmoid feedforward network was stable was due to the lack of annealing in the learning procedure. When the simulations were annealed before using the state vectors for learning,[12] the stability of the Boltzmann machine appeared to be slightly improved. However, using $-1/+1$ valued units, there were still signs of instability at $\epsilon = 1$, and this became severe at $\epsilon = 2$. Training a sigmoid feedforward network using the same annealing procedure produced results that were not appreciably different from those seen without annealing (for $\epsilon = 1$ and $\epsilon = 2$).

Interpretation of the results. These experiments show that a sigmoid feedforward network can learn the target mixture distribution faster than a Boltzmann machine. This difference is due to the feedforward network's tolerance of a high learning rate that causes instability in the Boltzmann machine. Since this instability is apparent at the time-scale of a single learning pass, and since it is little affected by the introduction of annealing, it appears that it is due simply to the sampling noise in the calculation of the gradient from the results of positive and negative phase stochastic simulations.

One advantage of a feedforward network in this respect may be seen clearly when there are no hidden units. In this case, the positive-phase, clamped simulations are completely deterministic, while the negative-phase, unclamped simulations remain stochastic. Learning in a feedforward network, for which only the positive phase simulation is necessary, will then take place with no noise disturbing the measurement of the gradient. Learning in a Boltzmann machine, which requires a negative phase, will still be subject to noise. When hidden units are present, the estimate of the gradient in the feedforward network will have some noise, but still not as much as in the Boltzmann machine.

This is not the full explanation of the difference, however, as was seen in experiments where the sigmoid feedforward network was trained with a redundant unclamped phase, using the "short-cut" simulation method to ensure that state vectors came from the true equilibrium

[12] The annealing schedule was as follows: One pass at infinite temperature, 90 passes at temperatures declining geometrically from 8 to 1, and finally 10 passes at a temperature of 1. The training time with annealing was, of course, many times that required without annealing.

distribution. Regardless of whether this redundant phase was negative (as in Boltzmann machines) or positive (equivalent to a second set of 250 unclamped training cases) its inclusion did *not* induce instability, but merely introduced a bit more variability in the progress of learning (see Figure 6, below).

The difference appears to result from the way weights are changed in the two networks. In a feedforward network, each change to w_{ij} is weighted by the forward conditional probability of S_i having a value different from that it presently has. As learning progresses, these weighting factors tend to decrease, leading to stability. In Boltzmann machines, the magnitude of each change remains constant; it is only the balance between positive phase increments and negative phase decrements that, in theory, brings learning to a stable halt, but this balance is sensitive to noise in the samples.

Effect of failure to reach equilibrium. Although it was not a major factor in the experiment described here, failure to reach equilibrium in the simulations is noted as a problem in [7], where it is observed that after a period of good progress learning can "go sour", as weights are built up to values where they form large energy barriers that inhibit settling to the equilibrium distribution. The authors prescribe "weight-decay" as a partial solution.

One would think that learning could "go sour" in stochastic feedforward networks as well as in Boltzmann machines, but such problems have not been observed. However, it *is* possible to make the sigmoid feedforward network go sour in the mixture distribution experiment by adding a redundant, unclamped negative phase, simulated in the normal manner (i.e. with no short-cut). This is seen in Figure 6. Using $-1/+1$ valued units with $\epsilon = 2$, learning with the redundant negative phase closely matches that without a negative phase for about the first 100 passes, but then becomes unstable. Interestingly, adding a redundant, unclamped *positive* phase does *not* cause the learning to go sour.

These results can be understood by picturing the effects of failing to sample from the true equilibrium distribution in the various phases. In a clamped positive phase, the effect will be to confine the state vectors seen to a subset of those high-probability state vectors that are compatible with the clamped visible units. The learning increments that result from this sample will still tend to increase the probability of the clamped training data, albeit at a lesser rate than would be the case if *all* the compatible high-probability state vectors had been seen.

In an unclamped negative phase, failure to sample from the equilibrium distribution will produce state vectors that do not represent all those of high probability. Once learning has made some progress, there will be a group of high-probability state vectors compatible with each training case. In a non-representative sample, some of these groups may not be sampled from at all, while other groups will contribute more than their share of state vectors to the sample. The learning decrements that occur in the negative phase will then unfairly decrease the probability of the training cases compatible with the over-sampled groups, more than offsetting the increments in the positive phase and producing instability.

Similarly, an extra unclamped positive phase results in some training cases being over-sampled, and thus weighted more heavily in the learning. This may produce sub-optimal progress, but not instability. In fact, runs done with an extra unclamped positive phase (simulated without use of the short-cut) were notable for a high variability — some runs did significantly better than those without the extra phase, while others did rather worse. Nevertheless, even the less successful runs showed nearly steady improvement as the simulations progressed, in contrast

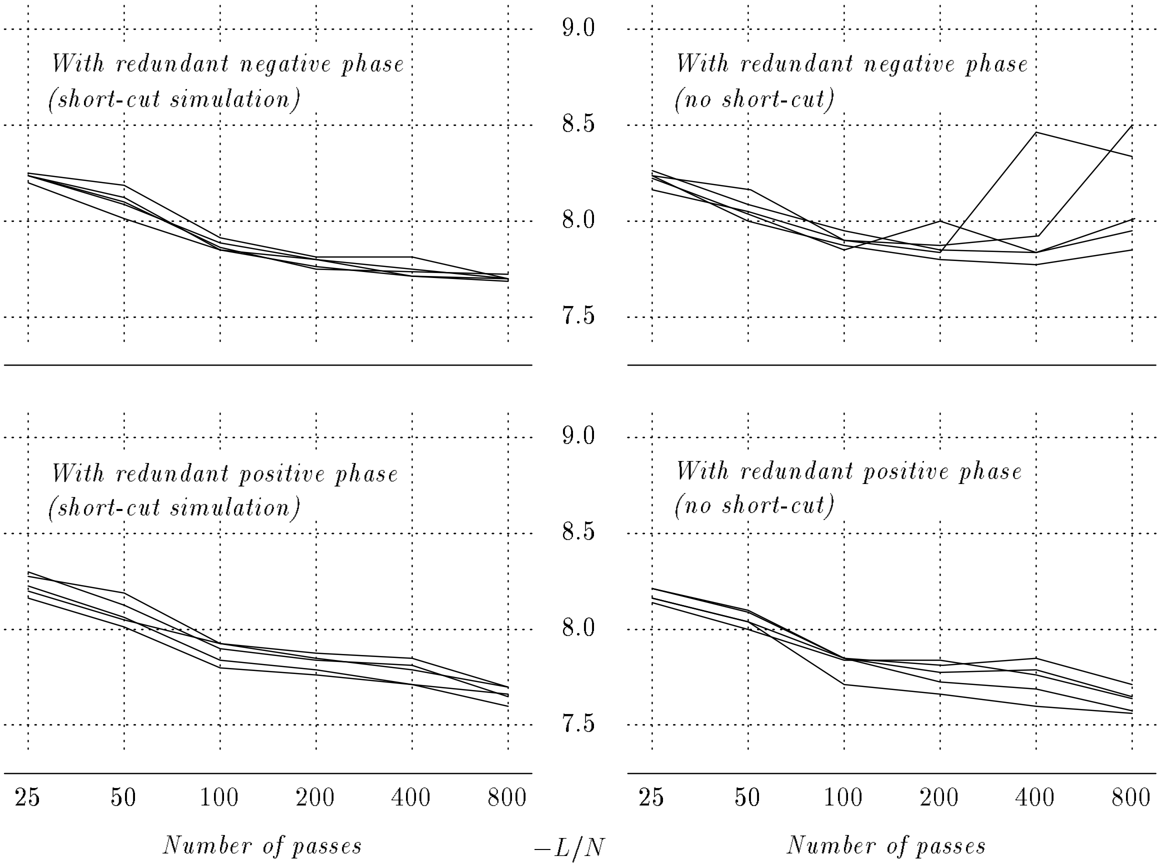

Figure 6: Effect of redundant phases on learning performance in sigmoid feedforward networks ($-1/+1$ valued units, $\epsilon = 2$). Five runs with different random number seeds are shown.

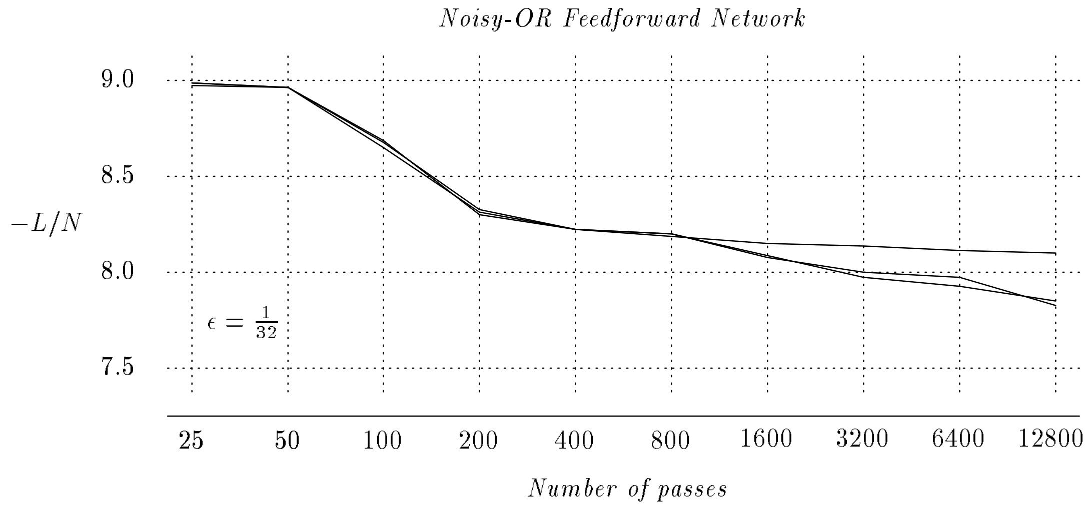

Figure 7: Learning performance of a noisy-OR feedforward network with $-1/+1$ valued units for ϵ of $\frac{1}{32}$. Three runs with different random number seeds are shown.

to the drastic worsening seen at times with an extra negative phase.

Thus, it appears that the consequences of failure to reach equilibrium are more serious in a negative phase than in a positive phase. This gives stochastic feedforward networks a qualitative advantage in circumstances where equilibrium is hard to reach — learning may be adversely affected, but the instability that can occur with Boltzmann machines does not arise.

Performance of noisy-OR networks. Noisy-OR feedforward networks were also applied to the task of learning the mixture distribution, with rather disappointing results. Performance was both poorer and more erratic than for sigmoid feedforward networks.

Figure 7 shows the progress of three runs using $-1/+1$ valued units, with $\epsilon = \frac{1}{32}$. The networks appear to have difficulty learning to reduce $-L/N$ to less than 8 bits. Increasing ϵ sometimes improved learning speed, but not reliably so. Performance of noisy-OR networks with $0/1$ valued units was essentially similar, except that a higher value of ϵ was desirable.

In other experiments, noisy-OR networks sometimes showed a strong tendency to get stuck at a local maximum (or at a point where the gradient was so small that learning essentially stopped). For example, attempts to train a noisy-OR network with $0/1$ valued units to compute XOR using two hidden units between inputs and output (the minimum required) succeeded in only 1 out of 20 tries.[13] Somewhat better results were obtained using $-1/+1$ valued units — success in 10 out of 20 attempts. Sigmoid networks almost never get stuck when solving this problem, even with only one hidden unit (the minimum needed with sigmoid networks).

It seems possible that some re-parameterization of noisy-OR networks would improve gradient-ascent learning, perhaps by eliminating the need for constraints. Attempts along these lines have so far failed to produce better results, however.

[13] Some details: The two hidden units were connected to the inputs, but not to each other. The output unit was connected to the inputs and to the hidden units. Training was done for 5000 passes with $\epsilon = \frac{1}{8}$.

Generalization to the real distribution. All the results shown so far concern the performance of the networks on the training cases. Generally, the true objective is good performance on items drawn from the real distribution of which the training cases are a sample.

Table 3 shows the performance of all the network types on both the training data and on items from the real distribution. Each type of network was trained with a reasonable value of ϵ until performance on the training data approached convergence. (The choice of ϵ and the point of near-convergence were both subjectively determined.) The value of $-L/N$ and the classification error rates for the training data are shown, along with the corresponding figures for the real distribution. (The analogue of $-L/N$ for the real distribution is the cross-entropy.)

Classification error rates shown in the table were calculated in two ways. The first calculation uses the exact, real distribution, and assumes that classification is based on the exact probabilities defined by the networks. The second calculation uses a sample of 1000 test items drawn from the real distribution, and uses classifications based on simulation results for each test item, in which the first eight attributes were clamped, and the resulting values for the ninth attribute observed.[14] Results were similar for all except one run of the noisy-OR network with 0/1 valued units, showing that classification performance is generally not dependent on very small differences in probabilities that would be swamped by noise in the simulations.

For comparison, results from a maximum-likelihood fit of a mixture model with six components to the training data using the EM algorithm [3, 15] are given as well, evaluated on a test sample of 5000 items.

The sigmoid feedforward networks, the Boltzmann machine with $-1/+1$ valued units, and the mixture model all show signs of overfitting the data, since their values for $-L/N$ on the training data are less than the value of 7.87 bits that the true model would give. Accordingly, it is not surprising that their performance on the real distribution is not quite as good as on the training data. The mixture model appears to have overfitted to a lesser extend than the networks. This is expected, since it is a more restricted model that nevertheless can exactly model this particular distribution. The penalty in overfitting paid for the generality of the network models does not seem large, however.[15]

The noisy-OR feedforward networks and the Boltzmann machine with 0/1 valued units do not show such definite signs of overfitting the training data. This appears to be a reflection of their generally inferior learning performance, not of any intrinsic superiority in generalization. Nevertheless, the best value for cross-entropy with the real distribution seen in any of these runs is 7.81 for one run of the noisy-OR network with 0/1 valued units, though the other two runs of this network were rather poor.

Generalization performance for all these networks might well be improved by stopping learning before convergence using a cross-validation criterion [14].

As an aside, it is interesting that the weights found during network training generally bear only a vague resemblance to those that would result from manually solving the problem using the clusters of Figure 4 to represent mixture components.

[14] Data was collected from 100 simulation passes, with the annealing schedule of note (12) being applied before each group of ten passes.

[15] The EM algorithm does take considerably less time than any of the network training procedures.

Type of network	*Passes*	ϵ	*Values of* $-L/N$	*Error rates (exact* $\sim$ *simulated)*
Boltzmann machine	1600	$\frac{1}{4}$	7.98 7.94 7.94	20% 20% 21% $\sim$ 22% 21% 21%
(0/1 units)			7.95 7.95 7.93	19% 20% 20% $\sim$ 19% 22% 21%
Boltzmann machine	1600	$\frac{1}{4}$	7.80 7.81 7.77	19% 17% 18% $\sim$ 20% 18% 18%
($-1/+1$ units)			7.90 7.85 7.84	19% 19% 19% $\sim$ 20% 17% 20%
Sigmoid FF network	800	4	7.67 7.72 7.68	16% 19% 16% $\sim$ 20% 20% 18%
(0/1 units)			7.90 7.85 7.89	20% 19% 19% $\sim$ 20% 19% 19%
Sigmoid FF network	800	2	7.64 7.66 7.70	15% 17% 18% $\sim$ 18% 18% 20%
($-1/+1$ units)			7.91 7.89 7.87	19% 20% 19% $\sim$ 19% 19% 20%
Noisy-OR FF network	12800	$\frac{1}{8}$	7.79 8.06 7.93	20% 20% 20% $\sim$ 20% 28% 24%
(0/1 units)			7.81 8.11 7.92	19% 19% 20% $\sim$ 19% 26% 18%
Noisy-OR FF network	12800	$\frac{1}{32}$	7.81 7.85 8.08	19% 19% 35% $\sim$ 18% 19% 33%
($-1/+1$ units)			7.82 7.86 8.12	19% 19% 35% $\sim$ 18% 20% 36%
Mixture model	100	–	7.73 7.74 7.72	18% 18% 20%
(EM algorithm)			7.82 7.85 7.86	19% 22% 19%

Table 3: Performance on training data (first line) and on items from the real distribution (second line) for various networks trained to near-convergence. Results from three runs with different random number seeds are shown.

Discussion

I conclude by discussing how stochastic feedforward networks relate to other connectionist approaches to statistical modeling and to work on the representation of expert knowledge. I also outline some areas in which stochastic feedforward networks appear to open up new possibilities.

Relation to other connectionist statistical methods. Problems such as speech or handwriting recognition are fundamentally statistical in nature. Although some *a priori* knowledge of the task may be available, much of the information required to solve the problem must come from training data. The preferred output for such classification problems is a probability distribution over possible classes, conditional on the attributes presented as input.

A deterministic feedforward network, trained by a method such as backpropagation [12], can represent a distribution over two classes by simply producing the probability of one of the classes as its output. Such a network that uses the sigmoid function to compute the output of a unit from its weighted input appears very similar to a sigmoid stochastic feedforward network with the same structure. Indeed, the two networks are essentially equivalent if there are no hidden units. However, in the general case, this is not so, since the hidden units in the deterministic network take on fixed numeric values, while those in the stochastic network represent a distribution over binary vectors.

Distributions over more than two classes can be represented in a deterministic network using a cluster of output units, one for each possible classification. The output of unit c, representing $P(\text{Class} = c \mid \text{Input})$, is set to $\exp(X_c)/\sum_i \exp(X_i)$, where X_i is the total input of unit i [2]. Distributions over a vector of output attributes can be represented using several such clusters, under the assumption that the probabilities for the various attributes are independent.

Stochastic networks have the more general capacity to exhibit distributions over a large output vector in which there are arbitrary dependencies among attributes. The improved learning speed of stochastic feedforward networks over Boltzmann machines may make this approach feasible in practice. Furthermore, in a stochastic feedforward network where the input units precede all the hidden and output units, as in Figure 2(b), the short-cut simulation method can be used to produce possible classifications for a given input without the need to settle to equilibrium, at a speed comparable to that of a deterministic feedforward network. Settling to equilibrium is still necessary during learning, when the output units are clamped.

A further advantage of stochastic over deterministic networks is their superior ability to cope with missing data, especially when architectures such as those of Figure 2(a) and (c) are used.

Relation to expert systems. In applications such as medical diagnosis, experts have extensive knowledge relevant to the task. Empirical data, while valuable, may be of limited extent, or may have been acquired under circumstances different from those currently prevailing. The need in such applications to integrate knowledge derived from experts with that derived from empirical data has been recognized by workers in the area (see the discussion in [8], for example). The learning procedures described in this paper may contribute to solving this problem.

One possible approach would be for the expert to specify the structure of a belief network (i.e. the causal connections), while leaving the numeric values of the forward conditional probabilities to be estimated empirically. If training data is available in which all attributes are known, this will be straightforward. It is likely, however, that the belief network will contain units whose values were not always measured, or which are not directly observable (such as the true underlying disease a patient suffered from). In this case, the gradient-ascent learning procedures of this paper could be applied, perhaps starting with weight values derived from an expert's tentative assessment of the probabilities. The expert might also constrain probabilities to some interval in order to guard against training data that is not extensive enough, or that is not representative of all possible contexts in which the system might be used.

More ambitiously, in parts of the network where causal connections are not clear to the expert a pool of hidden units could be included and their weights trained from empirical data. A problem with this approach is that the resulting networks may be hard to interpret. Using "weight decay" [7] to encourage some weights to go to zero might help.

The desire to keep the network's operation intelligible to the experts might also lead one to use the noisy-OR model for conditional probabilities, despite the superior learning performance seen using sigmoid units. The particular properties of the noisy-OR model might also be desirable for technical reasons; they are exploited in the heuristic diagnostic search algorithm of [5], for example. Noisy-OR and sigmoid units can also be mixed in the same network, and for that matter, encorporating a Boltzmann machine as a sub-network is not impossible.

Making decisions. Stochastic feedforward networks are compatible with the "influence diagrams" used to formulate decision problems. An algorithm of Shachter [13] exploits the structure of these diagrams to find decisions that maximize expected utility. Unfortunately, this algorithm can sometimes take exponential time. I will describe here a method of making simple decisions using stochastic simulation that also exploits the feedforward structure.

Consider a network with three sets of visible units — a "context" set, $\overline{C}$, a "action" set $\overline{A}$, and a "result" set, $\overline{R}$. Using empirical data, we can train this network to represent the conditional probabilities that $\overline{R}$ will result given that we perform action $\overline{A}$ in context $\overline{C}$. Suppose now that we wish to bring about some "goal", $\overline{g}$, at a time when the context is $\overline{c}$. Our best bet is to perform an action $\overline{a}$ that maximizes $P(\overline{R}=\overline{g} \mid \overline{A}=\overline{a} \;\&\; \overline{C}=\overline{c})$.

We could find the action that maximizes the probability of our goal by running a separate stochastic simulation for every possible action. In each simulation, the action and context units would be clamped, and we would observe how often the goal shows up in the result units. We would then chose the action that leads to the goal showing up most often. However, this method is infeasible if there are many actions, represented by a large number of units. (Consider the number of possible medical treatments when twenty drugs can be given in combination, for example.)

However, we can transform the problem by rewriting the probability to be maximized using Bayes' rule:

$$P(\overline{R}=\overline{g} \mid \overline{A}=\overline{a} \;\&\; \overline{C}=\overline{c}) \;=\; \frac{P(\overline{A}=\overline{a} \mid \overline{R}=\overline{g} \;\&\; \overline{C}=\overline{c})\; P(\overline{R}=\overline{g} \mid \overline{C}=\overline{c})}{P(\overline{A}=\overline{a} \mid \overline{C}=\overline{c})}$$

Now, *provided* that $P(\overline{A}=\overline{a} \mid \overline{C}=\overline{c})$ is the same for all $\overline{a}$, we can choose the best action by running a single stochastic simulation in which we clamp the context units to $\overline{c}$ and the result units to $\overline{g}$, and then observe which value of $\overline{a}$ turns up most often in the action units.

Ensuring that $P(\overline{A}=\overline{a} \mid \overline{C}=\overline{c})$ is the same for all $\overline{a}$ is easy in a stochastic feedforward network — we simply set up the network so that units in $\overline{A}$ have no incoming connections, ensuring equal probabilities for each $\overline{a}$ in an unclamped network, and we further arrange that there is no directed path from a unit in $\overline{A}$ to a unit in $\overline{C}$, which ensures that clamping $\overline{C}$ will not change these probabilities. Producing these equal probabilities in a Boltzmann machine is not so easy. We could try to train the Boltzmann machine to satisfy the constraint, but there is no guarantee that we will succeed very well, and the attempt may interfere with learning the distribution of $\overline{R}$ given $\overline{A}$ and $\overline{C}$.

Unfortunately, the transformed method does not completely solve this decision problem. It is possible that with a particular context and goal clamped, a different action will show up after every simulation pass, even during a long simulation. We will then have no basis for deciding which (if any) of these actions is best. The method is most applicable in situations where only a small, but unknown, subset of actions have a significant probability of producing the goal.[16]

Potential for new learning procedures. Gradient-ascent learning has the advantage that it is simple, and that it can be performed in an "on-line" manner if desired. However, it can be rather slow, and can get stuck at a local maximum. For Boltzmann machines, there appears to be no reasonable alternative to gradient-ascent, but for stochastic feedforward networks the fact that the probability of a full state vector can be explicitly calculated allows one to contemplate other learning procedures.

For noisy-OR networks, one possibility is to apply a stochastic version of the EM algorithm [3]. This seems feasible provided that the efficacy of each input to a unit in forcing the unit to take on the value 1 is made explicit in a set of auxiliary units that are stochastically simulated along with the main units. Probabilities can then be iteratively estimated from co-occurrence counts.

It also seems feasible to implement true Bayesian learning for these networks, in which the output for a test case is obtained by integrating the outputs of the network over all possible sets of connection weights, modified by the posterior probability of each weight set given the training cases. For stochastic feedforward networks, this integration can be done by applying stochastic simulation to the learning process as a whole. This method may avoid both the problem of overfitting the training data and the possibility of getting stuck in a local maximum. It may work especially well for noisy-OR networks with the auxiliary units described above, since it turns out that one can then avoid having to simulate distributions over continuous parameters.

Biological modeling. Stochastic feedforward networks also provide additional options for modeling of real neural processes. Although analogies between the negative phase of Boltzmann machine learning and dream sleep are speculated upon in [7], it may well turn out that the negative phase is biologically implausible. The work here shows that this would not necessarily be fatal to the idea that stochastic simulation plays some role in learning in the brain. The somewhat complex simulation procedure for feedforward networks may be a barrier to their encorporation in models of neural processing, however.

[16] It is tempting to try to solve this problem using simulated annealing by "cooling" the simulation down to a temperature of zero in order to find the most probable state vector compatible with the clamped context and goal. However, the action part of the most probable state vector is guaranteed to represent the best decision only if there are no hidden units in the network.

Appendix — Proofs of claims

Claim 1 *The procedure of Figure 1 moves the weights into unit i to the point closest in Euclidean distance that satisfies the constraint*

$$w_{i0} \; + \sum_{j<i,\, w_{ij}<0} w_{ij} \;\geq\; \eta$$

Proof. Let w_{ij} be the original set of weights, and let $w'_{ij} = w_{ij} + \delta_j$ be the set of weights satisfying the constraint for which $\Delta^2 = \sum_j \delta_j^2$ is minimal. We can prove a number of properties of the δ_j.

First, all the δ_j are non-negative, since decreasing a weight will certainly not help satisfy the constraint. Also, for $j > 0$, $\delta_j = 0$ if $w_{ij} \geq 0$ and $\delta_j \leq |w_{ij}|$ if $w_{ij} < 0$, since once δ_j is large enough to make w'_{ij} zero, making it any larger does not help satisfy the constraint.

Next, if $w_{ij} \leq w_{ik} < 0$, then $\delta_j \geq \delta_k$. Otherwise replacing both δ_j and δ_k by $(\delta_j + \delta_k)/2$ would reduce Δ^2 while keeping the constraint satisfied. Similarly, $\delta_0 \geq \delta_j$ for all j, since otherwise there would be an advantage in replacing them both by $(\delta_0 + \delta_j)/2$.

We can therefore renumber the units before i in such a way that for some n:

$$\begin{array}{ccccccccc} \delta_0 & \geq & \delta_1 & \geq & \cdots & \geq & \delta_n & > & 0 \\ & & w_{i1} & \leq & \cdots & \leq & w_{in} & < & 0 \end{array}$$

and $w_{ij} \geq 0$ and $\delta_j = 0$ for $j > n$. One can now show, by arguments similar to those above, that there is an m such that for all j less than m, $\delta_j = \delta_0$ and $\delta_j < |w_{ij}|$, while for $m < j \leq n$, $\delta_j = |w_{ij}|$.

The entire set of optimal changes, δ_j, is therefore determined by the value of δ_0. The other δ_j are either equal to δ_0, or are less, if a lesser value suffices to make w'_{ij} non-negative.

The procedure of Figure 1 is now easily seen to be a search for the appropriate value of δ_0.

Claim 2 *The weights in both a Boltzmann machine and in a sigmoid feedforward network consisting of only one or two units can be set so as to produce any probability distribution over state vectors. (Except that distributions in which some state vectors have zero probability can only be approached as the weights go to infinity.)*

Proof. We need only consider networks with 0/1 valued units. Clearly, any distribution over a network of one unit can be produced by simply adjusting the single bias weight.

To produce a given distribution over a sigmoid feedforward network with two units, start by setting the bias weight for the first unit to produce the required marginal probability distribution for that unit. Then set the bias weight for the second unit to produce the required conditional probability distribution given that the first unit has value 0. Lastly, set the weight on the connection from the first to second unit to produce the correct conditional probability distribution given that the first unit has value 1, taking into account the value of bias weight determined earlier.

For a two-unit Boltzmann machine, we must find weights that give energies to the four possible states that produce the required distribution. The energy of state $\langle 0,0\rangle$ is zero irrespective of the weight values. We can arrange for states $\langle 1,0\rangle$ and $\langle 0,1\rangle$ to have the appropriate energies relative to that of $\langle 0, 0\rangle$ by adjusting their bias weights. The energy of state $\langle 1, 1\rangle$ can then be made whatever we wish by setting the weight on the connection between the two units, taking account of the bias weights.

Claim 3 *The set of probability distributions that can be produced over a network of three units is the same for Boltzmann machines and sigmoid feedforward networks.*

Proof. To translate a three-unit Boltzmann machine to a sigmoid feedforward network, start by setting up the first two units of the sigmoid network so as to duplicate the marginal distribution over (any) two units of the Boltzmann machine. Claim 2 guarantees that this is possible. Now add a third unit after these two, connected to the first two using the same weights as in the Boltzmann machine. This duplicates the required conditional probabilities for the third unit, without disturbing the distribution over the first two units.

To translate a three-unit sigmoid feedforward net to a Boltzmann machine, start by setting the weights to the third unit in the Boltzmann machine to be the same as those into the last unit in the sigmoid network. This duplicates the conditional probabilities for this unit given the values of the other two units. Now we need to set up the weights between the remaining two units so as to produce the same marginal distribution as in the sigmoid network, taking into account the biasing effects of the third unit. This can be done because, again, all things are possible with only two units.

Claim 4 *The probability distribution produced by the 0/1 valued Boltzmann machine of Figure 3(a) cannot be duplicated by a sigmoid feedforward network with the same number of units.*

Proof. We can assume that the feedforward network also uses 0/1 valued units. Due to the symmetry of the Boltzmann machine, there is no choice in ordering the units when trying to find an equivalent sigmoid feedforward network. The last unit in the feedforward net (unit 4) must have the same weights as in the Boltzmann machine in order to reproduce the conditional probabilities for that unit's value given the values in the rest of the network.

Now consider how we must set the weights into the second-to-last unit in the feedforward network (unit 3). By symmetry, the two weights from the earlier units must be equal; call their value w. There is also a bias weight, b. Consider the odds in favour of unit 3 having the value 1 when unit 4 has the value 1 and there are zero, one, or two units with value 1 before unit 3. Equating these odds in the feedforward network to the odds in the Boltzmann machine produces the constraints, respectively:

$$\begin{aligned} \exp(b)\,\frac{\sigma(1)}{\sigma(0)} &= \exp(1) \\ \exp(b+w)\,\frac{\sigma(2)}{\sigma(1)} &= \exp(2) \\ \exp(b+2w)\,\frac{\sigma(3)}{\sigma(2)} &= \exp(3) \end{aligned}$$

By taking logarithms, one obtains a system of linear equations in w and b which numerical calculation shows to be inconsistent.

Claim 5 *The probability distribution produced by the 0/1 valued sigmoid feedforward network of Figure 3(b) cannot be duplicated by a Boltzmann machine with the same number of units.*

Proof. We can restrict consideration to Boltzmann machines with 0/1 valued units. Consider the unit in a candidate Boltzmann machine corresponding to the bottom unit in the sigmoid network. The weights into this unit must be the same as those in the sigmoid network, in order to reproduce the conditional probabilities for this unit given the various combinations of other unit values. Note that the value of the bottom unit is effectively a deterministic function of the values of the upper three units — i.e. there are only eight state vectors with significant probability.

Now consider the constraints placed on the weights in the Boltzmann machine by the requirement that all combinations of values for the upper three units be equally probable, as they are in the sigmoid network. In the Boltzmann machine, this translates to the requirement that the energy of the network be the same for all eight possible state vectors. In particular, since the energy of the state vector $\langle 0,0,0,0 \rangle$ is zero, the energy of the other seven state vectors must be zero as well. Applying this constraint to the three state vectors $\langle 1,0,0,1 \rangle$, $\langle 0,1,0,1 \rangle$, and $\langle 0,0,1,1 \rangle$, we find that the bias weights into the three upper units must be -10. Applying it to the the three state vectors $\langle 0,1,1,1 \rangle$, $\langle 1,0,1,1 \rangle$, and $\langle 1,1,0,1 \rangle$, we get that the weights between the upper units must all be -10 as well. The energy of the state $\langle 1,1,1,1 \rangle$ is now determined to be -10, showing that a proper set of weights is impossible.

Claim 6 *The probability distribution produced by a three-unit noisy-OR network with* 0/1 *valued units in which* $w_{31} = w_{32} = 1$ *and all other weights zero cannot be duplicated by either a Boltzmann machine or a sigmoid feedforward network with only three units.*

Proof. In view of Claim 3, it suffices to show that the noisy-OR net cannot be duplicated by a Boltzmann machine with 0/1 valued units.

Consider the unit in a candidate Boltzmann machine corresponding to unit 3 in the noisy-OR net. In the noisy-OR net, this unit is always zero when the other units are both zero. To approximate this in the Boltzmann machine, the bias weight for this unit must be very large and negative. The probability of this unit being zero when one of the other two units is one is only e^{-1}, however. The weights from the other two units must therefore be very large and positive, in order to nearly cancel out the large negative bias in this case. Now, however, the probability of the unit being one when *both* other units are one is nearly 1 in the Boltzmann machine, but only $1 - e^{-2}$ in the noisy-OR net. Thus no set of weights for the Boltzmann machine can produce (or even closely approximate) the required distribution.

Acknowledgements

I thank Geoff Hinton and the other members of the connectionist research group at the University of Toronto for many helpful discussions. This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Ontario Information Technology Research Centre.

References

[1] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985) A learning algorithm for Boltzmann machines, *Cognitive Science*, vol. 9, pp. 147-169. Also found in D. Waltz and J. A. Feldman (editors), *Connectionist Models and Their Implications: Readings from Cognitive*

Science, Norwood, New Jersey: Ablex.

[2] Bridle, J. S. (1989) Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition, in F. Fougelman-Soulie and J. Hérault (editors) *Neuro-computing: Algorithms, Architectures, and Applications*, Springer-Verlag.

[3] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977) Maximum likelihood from incomplete data via the EM algorithm (with discussion), *Journal of the Royal Statistical Society B*, vol. 39, pp. 1-38.

[4] Derthick, M. (1984) Variations on the Boltzmann machine learning algorithm, Technical Report CMU-CS-84-120, Pittsburg: Carnegie-Mellon University, Department of Computer Science.

[5] Henrion, M. (1988) Towards efficient probabilistic diagnosis in multiply connected belief networks, in R. M. Oliver and J. Q. Smith (editors) *Influence Diagrams, Belief Nets and Decision Analysis* (proceedings of a conference entitled 'Influence diagrams for decision analysis, inference, and prediction', Berkeley, USA, 1988), Chichester, England: John Wiley.

[6] Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. (1984) Boltzmann machines: Constraint satisfaction networks that learn, Technical Report CMU-CS-84-119, Pittsburg: Carnegie-Mellon University, Department of Computer Science.

[7] Hinton, G. E. and Sejnowski, T. J. (1986) Learning and relearning in Boltzmann machines, in D. E. Rumelhart and J. L. McClelland (editors), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, Cambridge, Massachusetts: MIT Press.

[8] Lauritzen, S. L. and Spiegelhalter, D. J. (1988) Local computations with probabilities on graphical structures and their application to expert systems (with discussion), *Journal of the Royal Statistical Society B*, vol. 50, no. 2, pp. 157-224.

[9] Oliver, R. M. and Smith, J. Q. (1990) *Influence Diagrams, Belief Nets and Decision Analysis* (proceedings of a conference entitled 'Influence diagrams for decision analysis, inference, and prediction', Berkeley, USA, 1988), Chichester, England: John Wiley.

[10] Pearl, J. (1987) Evidential reasoning using stochastic simulation of causal models, *Artificial Intelligence*, vol. 32, no. 2, pp. 245-257.

[11] Pearl, J. (1988) *Probabilistic Reasoning in Intelligent System: Networks of Plausible Inference*, San Mateo, California: Morgan Kaufmann.

[12] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back-propagating errors, *Nature*, vol. 323, pp. 533-536.

[13] Shachter, R. D. (1988) Probabilistic inference and influence diagrams, *Operations Research*, vol. 36, no. 4, pp. 589-604.

[14] Stone, M. (1974) Cross-validatory choice and assessment of statistical predictions (with discussion), *Journal of the Royal Statistical Society B*, vol. 36, pp. 111-147.

[15] Titterington, D. M., Smith, A. F. M., and Makov, U. E. (1985) *Statistical Analysis of*

Finite Mixture Distributions, Chichester, New York: Wiley.