Tractable Learning of Probability Distributions Using the Contrastive Hebbian Algorithm

Craig E. L. Stark
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
cs6h@crab.psy.cmu.edu

James L. McClelland
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
jlm@crab.psy.cmu.edu

Abstract

In some tasks (e.g., assigning meanings to ambiguous words) humans produce multiple distinct alternatives in response to a particular stimulus, apparently mirroring the environmental probabilities associated with each alternative. For this purpose, a network architecture is needed that can produce a distribution of outcomes, and a learning algorithm is needed that can lead to the discovery of ensembles of connection weights that reproduce the environmentally specified probabilities. Stochastic symmetric networks such as Boltzmann machines and networks that use graded activations perturbed with Gaussian noise can exhibit such distributions at equilibrium, and they can be trained to match environmentally specified probabilities using Contrastive Hebbian Leaning, the generalized form of the Boltzmann Learning algorithm. Learning distributions exacts a considerable computational cost as processing time is used both in settling to equilibrium and in sampling equilibrium statistics. The work presented here examines the extent of this cost and how it may be minimized, and produces speed-ups of roughly a factor of 5 compared to previously published results.

In recent years, we have gained an understanding both of the power and of the limitations of the backpropagation learning algorithm (MacKay, 1992; Rumelhart, Durbin, Golden, and Chauvin, in press.) Backpropagation finds a set of weights **W** such that when given some input $\bar{x}_i$, the network will produce an output $\bar{y}_i$ that minimizes some measure of the difference between the network's output and the desired output $\bar{d}_i$ (often construed as the "environment"). The relationship between $\bar{x}_i$ and $\bar{d}_i$ may be stochastic, and $\bar{d}_i$ may have a wide range of distributions, but the network's task is deterministic. If, given the shape of $\bar{d}_i$, the appropriate activation functions and error measures are used, the minimum value of the error measure is obtained when $\bar{y}_i$ is equal to the expected value of $\bar{d}_i$. This expected value is essentially a deterministic function of the input. In some cases, such expected values may be sufficient, but for the purpose of modeling human performance, or for the purpose of adequately characterizing a wide range of input-output functions, the expected values of the individual output values have serious limitations (see Movellan and McClelland, 1993 for discussion). As one example, if we ask human subjects to generate definitions of polysemous words such as *bank* on a number of different occasions, subjects will come up with different meanings, and each meaning will have a frequency approximately equal to its frequency of use. Adopting as we do a parallel-distributed processing perspective on the nature of the representations used, we assume that each meaning corresponds to a distributed pattern of activation over a population of units. The fact that people produce a number of different meanings suggests that the population settles to a number of different patterns, rather than simply to the pattern that represents the expected value of each unit involved in the representation. While this behavior can be accounted for either by truly stochastic processing or by effectively random contextual influences, capturing the distribution of patterns rather than their expected values is key.

Recently, some progress has been made into this problem. Movellan and McClelland (1993) studied a class of networks known as symmetric diffusion nets (SDNs) and were able to demonstrate the ability to learn to produce distinct output patterns with probabilities corresponding to their relative frequency in the training environment. SDNs are similar to Boltzmann machines in that they make use of symmetrical connections, but differ in that the units use continuous-valued activations perturbed by Gaussian noise:

$$\Delta a_i = \lambda (net_i - \hat{net}_i) + \sigma \sqrt{\lambda} Z_i \qquad \text{(EQ 1)}$$

Here λ is a time constant, net_i is the net input ($\mathbf{a}^T\mathbf{w}$), Z_i is a standard independent random Gaussian variable, σ controls the amount of noise, and $\hat{net}_i$ is a scaled version of the inverse (generalized) logistic function given by:

$$\hat{net}_i = \frac{1}{gain_i} \log\left(\frac{a_i - min}{max - a_i}\right) \qquad \text{(EQ 2)}$$

where *max* and *min* bound the activation values. In the absence of noise the activations settle to values equal to the generalized logistic of the net input; in the presence of noise activations are subject to perturbations both during settling and at equilibrium; at equilibrium the probability of finding the network in a particular state is proportional to the exponential of the Goodness of the state (see Movellan and McClelland, 1993, for details).

Both Boltzmann machines and SDN's can be trained to reproduce desired distributions of output states using what Galland and Hinton (1989) call the 'Contrastive Hebbian Learning Algorithm' (CHL). CHL is the general version of

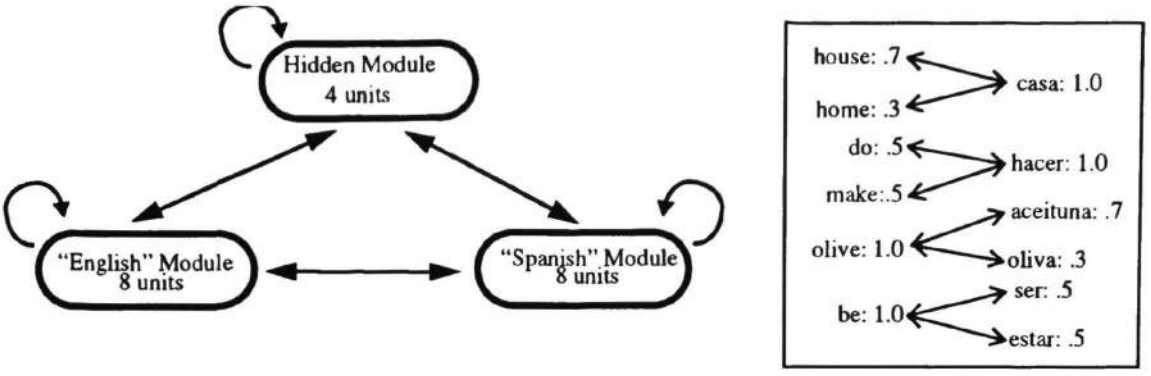

Figure 1: Figure 1: Architecture and I/O patterns used in Movellan and McClelland's (1993) translation problem

the Boltzmann algorithm; the network is run in two phases, a minus phase in which only input units are clamped, and a plus phase in which input and output units are clamped. Weights are adjusted according to:

$$\Delta w_{ij} = \frac{\varepsilon}{\sigma^2}\left(E_{xyd}(a_i a_j) - E_x(a_i a_j)\right) \qquad \text{(EQ 3)}$$

where ε is the learning rate parameter, σ is the standard deviation of the noise, a_i and a_j are the activations of units i and j, x is the vector of inputs, y is the vector of outputs, $E_{\overline{xyd}}()$ is the expected value calculated during the plus phase, and $E_{\overline{x}}()$ is the expected value calculated in the minus phase. In practice the expected values are estimated by settling repeatedly to equilibrium in each phase and averaging a sample the relevant co-products collected at equilibrium.

The Contrastive Hebbian algorithm, when used with either type of stochastic, symmetric network, minimizes a quantity known as the *information gain* (IG). In the present paper, we are concerned with the problem of learning to settle to one of a small number of discrete alternatives. For this case, either the Boltzmann machine or the SDN can in principle be used. Here we focus on the SDN, in part because its use of graded activations appears to capture a key aspect of human information processing (McClelland, 1993), which is our ultimate concern. For the case of discrete alternatives, the information gain becomes:

$$IG = \sum_{alts} P_{xd} \ln\left(\frac{P_{xd}(y)}{P_x(y)}\right) \qquad \text{(EQ 4)}$$

Where the state of the output units is considered to be equivalent to the desired state if the activation of each output unit it within some *tolerance* of the value specified in the desired output pattern. The quantity shown in Equation 4 is measured separately for each input pattern; the sum over all inputs is called the *Total Information Gain (TIG)*.

A key property of CHL is the fact that the network's position on the error surface (and therefore the current TIG and Δw) must be determined by a process of sampling. Sampling can be thought of as producing estimates of the true TIG and Δw. This has two implications. First, if we use an estimate of TIG as a measure of performance, we must keep in mind that it is an estimate of performance and thereby limited in its accuracy to the expected variance in the estimate. Second, in order to generate a estimated Δw vector that accurately demonstrates the error gradient, we need an adequate sample of the gradient. Not only must the network go through a settling process, but the process must be extended and repeated several times to generate accurate statistics. For example, M&M use a small 'translation' problem, described below. In this simulation, they used twenty settles of a hundred cycles (50 of settling to equilibrium and 50 for calculating statistics) per input pattern per phase per epoch, for a total of 48,000 updates of the entire network's activation in a single epoch.

If these values represent what is actually needed to learn probability distributions, this seriously limits the appeal of CHL, both from the point of view of tractability and from the point of view of psychological plausibility. From a psychological point of view one assumes that information processing involves some settling, but the suggestion that a fairly large number of separate repeats of the settling process is needed to estimate the direction of the gradient is somewhat troubling: it effectively amounts to a dramatic increase in the number of passes through of the training environment that are required for learning. For many problems, back propagation is surely already slow enough!

The work presented here was motivated by the desire to apply CHL to psychological problems in which humans do select one of a number of distinct alternatives. For CHL to be a viable learning algorithm for such problems, an understanding of which parameters have the largest effect on learning speed is vital. This paper reflects several steps we have taken toward developing such an understanding. We use the M&M's translation problem to examine the role of the number of repeat settles, the amount of noise in processing as well as in the units' initial activations, and the number of cycles per settle with the aim of optimizing the algorithm for use in connectionist modeling. We find that considerable optimization is possible when this is done, and that simula-

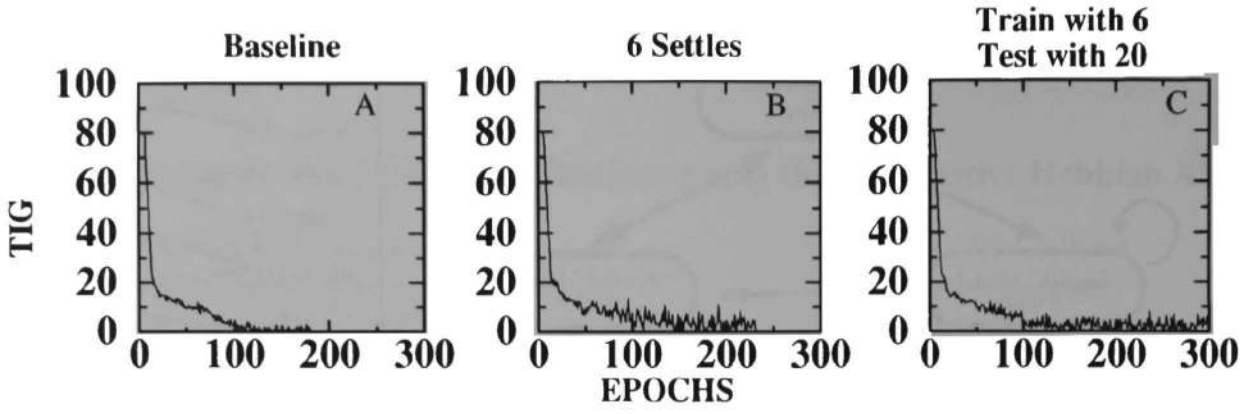

Figure 2: Effect of the number of settles on training time. Figure 2A shows the total information gain as a function of epochs of training for a typical network in the baseline condition. Figure 2B shows the same network trained with 6 as opposed to 20 settles per pattern per epoch. Note that while noisier, the shape of the curve is identical. Figure 2C demonstrates that during initial training, the source of this noise is primarily in the estimate of the current TIG not in the learning itself.

tions of the complexity required by psychological models become possible.

The Translation Problem

The main simulations presented here used M&M's translation problem, designed to test an algorithm's ability to learn to deal with a probabilistic environment in which an input has possibly more than one output. The translation problem has a network translate "words" (8-bit random patterns) from one "language" (layer of units) to another. In their example, the network was to translate Spanish words to and from their English counterparts (see Figure 1.) What makes this problem interesting is the lack of a one-to-one correspondence between the English and Spanish words. For example, the English word "olive" has two Spanish counterparts, "oliva" (derived from Latin) and "aceituna" (derived from Arabic.) If given either "oliva" or "aceituna" the network should respond with "olive." However, if given "olive" the network is supposed to respond with "aceituna" 70% of the time and "oliva" 30% of the time. The average of the two or a random selection of bits from each is not a valid response. The architecture, I/O mapping, and desired probabilities are presented in Figure 1.

Baseline condition

M&M broke learning into two stages, an initial stage designed to achieve approximate matching of the desired distribution and a final phase designed to achieve virtually exact matching. We have concentrated on the first stage, using M&M's parameters for this stage as a baseline condition: a timestep (λ) of 0.1, a learning rate (ε) of 0.01,[1] a noise constant (σ) of 0.1, 20 settling repeats per pattern, zeroing the network's activation between settles, 50 activation cycles of initial settling, 50 cycles for gathering statistics, and a stopping criterion on the TIG of 0.1. The tolerance for unit activations was set to 0.8. Since the desired activations used in each pattern were ± 0.9, this meant that the output had to fall within the right corner of the 8-dimensional hypercube defined by the activations of the output units by at least 0.1 on each unit, in order to be taken as a match to a particular desired output. No momentum was used.

A typical learning curve showing TIG as a function of epochs is shown in Figure 2A[2]: Clearly, the learning in this condition is relatively smooth and fairly rapid using epochs as a measure of training time. One interesting aspect of the learning is that for the last 75 epochs or so, the network's average performance remains close to the minimum, though there is some variability. This variability can come from two sources. Either the actual TIG that the network produces across epochs varies (since the weights vary from epoch to epoch), or our estimate of the actual TIG varies across epochs. Given a reasonable learning rate, the latter is the primary source of variability in the TIG measure, which opens the door for significant optimization.

Coarse Estimates of the Error Gradient

If the network never moves very far along the error surface on any given weight update, it stands to reason that we might be able to use a very coarse estimate of the weight-change vector to drive learning. Since a coarse estimate is computationally cheap, we should be able to speed training considerably. This idea is similar to the basis of Manhattan updating, which can be quite effective in the early stages of training.

1. In the text of the article, 0.0025 was the specified ε, but the actual value employed was 0.01.

2. For this and almost all subsequent simulations, a trio of networks starting with three different sets of starting weights were run. There were no notable differences between the three sets in any of the conditions.

Figure 2B shows the result of training the same network from Figure 2A but using only six settles per pattern instead of twenty. While the tail of the graph certainly shows more noise, their basic shapes are the same. Keep in mind that given a sample size of only six, we expect to see more variance in the TIG than with a sample size of twenty *even if the networks are in the same position in weight space*. That is to say that the two networks shown in Figures 2A and 2B could be following nearly identical trajectories through weight-space and the network in 2B would naturally look noisier.

This point is demonstrated by the training curve shown in Figure 2C. Here, the same network was trained using the Δw generated from six settles, but the TIG plotted was generated by testing the network without learning for twenty settles at every epoch. Clearly, the tail of the graph shows more noise than the baseline condition, and, by chance it doesn't happen upon a TIG sample below the 0.1 stopping criterion by 300 epochs. Equally clear, however, is that during the initial stage of training, six settles per pattern is almost indistinguishable from 20 settles per pattern when the variance in the estimate of the current TIG is controlled for.

For our purposes of demonstrating optimization of the training time, there are two problems. First, the amount of time to run an epoch is based on the number of cycles per settle and the number of settles per pattern. Since both variables are to be manipulated, we report training time in thousands of cycles per pattern (kCs). Second, the TIG stopping criterion of 0.1 in a single epoch introduces a high degree of variability in the apparent stopping time, and actually corresponds to a somewhat higher true TIG. For more stable comparisons across runs, we adopted a criterion of a 2.64 average over five epochs. This corresponds to producing each correct output pattern with a probability that is within 20% of its probability as specified in the training corpus.

Table 1 shows the training time for both the original and the six settle conditions, and we can now see the merit of this approach. Training time has been cut by over 50%. Like the M&M network, this net is ready to be fine-tuned to cleanly match the desired output distributions. All subsequent networks were trained using six settles per pattern.

Table 1: Training Times

Condition	Epoch	kCs	Condition	Epoch	kCs
Original (20)	100	200	50/10 cycles	157	57
6 settles	144	86	30/50 cycles	135	65
train 6 / test 20	105	~63	50/30 cycles	153	69
10 init/10 stat cyc.	263	52	100/100 cyc.	99	118
30/30 cycles	158	57	Final	103	37

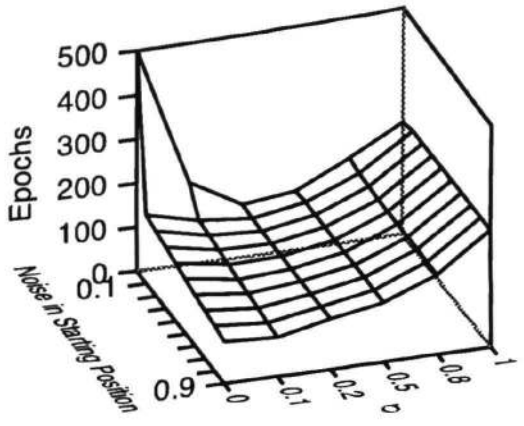

Figure 3: Effect of noise in processing and noise in the starting position on training time.

Role of Noise

Without some source of variability to make the settling process stochastic, a network cannot settle into differing outputs when presented a single input. It's a trivial statement, for by definition, it's an impossible problem. However, determining the optimal source of variability and the optimal amount of variability is far from trivial. Here we examine the effect of noise in two relatively standard locations: noise in the activation function and noise in the starting position of the network in activation space. While not an exhaustive list of places we might add noise to break the symmetry of the settling process, the results are basic enough that generalization to other sources of noise should be possible.

Noise in the Activation Function

When considering the amount of noise to be added to the activation function, there are at least two conflicting goals. If, as in M&M's network, the only source of noise is in the activation function (the σZ term in Equation 1) it must be of sufficient magnitude to let the network visit different attractors so that it may generate the desired probability distribution. However, working against this desire for stochastic processing is the need to remain stable enough to generate reliable statistics. Simulations using no noise in the starting position but σ values of 0.0, 0.1, 0.2, 0.5, 0.8, and 1.0 support this notion of a trade-off in the ideal amount of noise. No noise gives the expected result of failing to learn the 1-2 mappings and the extremal values of 0.8 and 1.0 both show slowed and incomplete learning.

Noise in the Starting Position

When the TIG and corresponding weight change vector are calculated, the network's estimated output distributions are computed across cycles and settles, giving the network the opportunity to exhibit the desired distribution of output

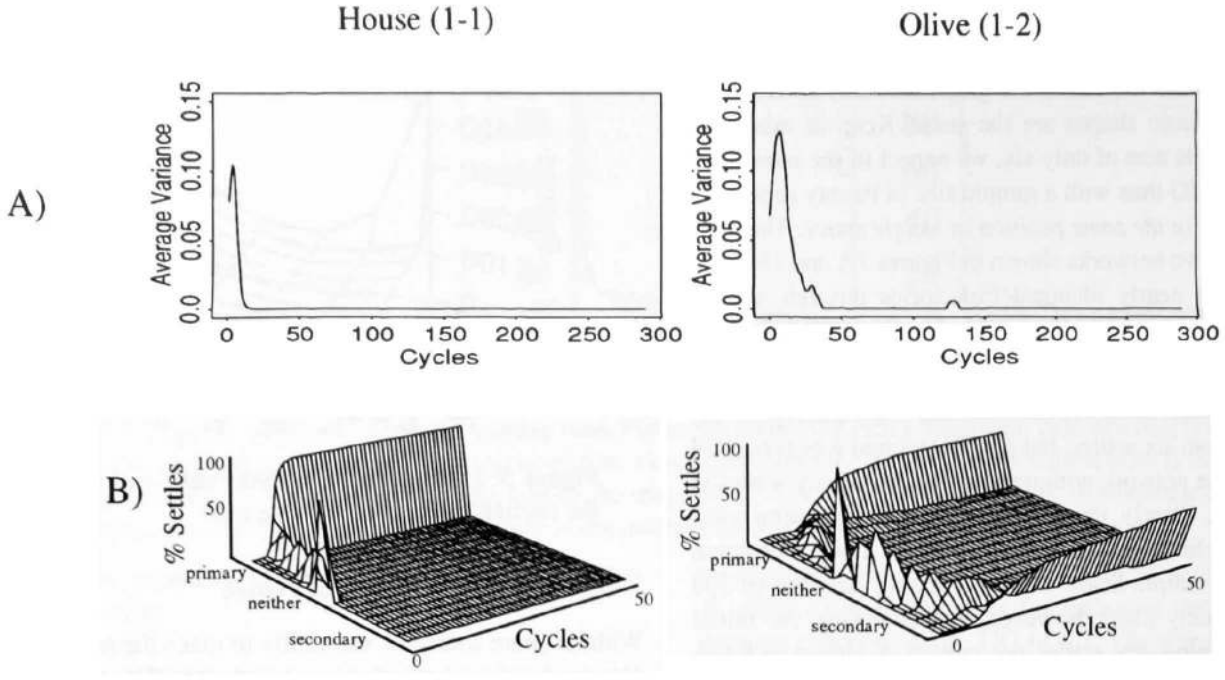

Figure 4: Settling behavior of the network when given a 1-1 mapping as input (e.g. "house") and a 1-2 mapping (e.g. "olive"). Figure 3A shows the variance in position in activation space in a ten cycle window during settling and Figure 3B shows the percentage of settles that are inside either attractor as a function of the number of cycles of settling.

states in two possible ways. Like the Boltzmann machine, the network could jump between the n desired states with the appropriate probabilities p_n within a single settle. Or, the network could settle to a single attractor with probability p_n and thereby generate the distribution across settles. A prediction of the former is that noise in the starting position should have little effect on learning, whereas the latter might predict a significant effect, especially when σ is small.

What we find when we train networks using starting activations ranges of ± 0.1 and ± 0.9 and σ values of 0.0, 0.1, 0.2, 0.5, 0.8, and 1.0 is that there is evidence to support the conclusion that the network is generating the distributions across settles. For large values of σ, noise in the starting position has no effect as it is quickly subsumed by the noise in the activation function. However, when σ is small (and especially when zero), random starting positions help the network considerably by allowing it to form trajectories from starting regions in activation space to the desired attractors.

We can see the effect of this trade-off in Figure 3 where the training times to criterion are plotted as a function of σ and noise in the starting position. When we examine the settling behavior of the network closely, we do find, in fact, that it is extremely unlikely for the network to jump from one desired state to another. Even with large amounts of noise in the activation function, the network settles towards one attractor and generates the distributions across settles. Figure 4A and 4B show the settling behavior of a network trained for 100 epochs using a σ of 0.2 and starting activation ranges of ± 0.9.[3] In both figures, the trained network was presented "house" (a 1-1 mapping) and "olive" (a 1-2 mapping) and allowed to settle 100 times while the network's activation was monitored.[4] In Figure 4A, we plot the variance in a sliding ten cycle window, averaged across the 100 settles. What this shows is that for both 1-1 and 1-2 mappings, the amount of change in position in activation space drops off rapidly during settling. That is to say that within a given settle, the network settles to a fixed point within 50 cycles and remains in that attractor, even if the desired output distribution is not unimodal. However, as Figure 4B demonstrates, when we look at the distribution of output states relative to the desired attractor(s) as a function of settling time, we see that across settles, the network generates the desired distribution of states quite well.

In fact, this settling behavior suggests that if we keep the number of initial settling cycles where it is, we can cut the number of subsequent statistical generation cycles sharply and gain a significant speed increase. While we do reach a

3. Other noise parameters including σ=1.0 and starting activations of 0.0 show the same basic effect, though the data are noisier.

4. In this network the olive → *oliva* and *aceituna* was trained on a 50/50 not 30/70 mapping.

point of diminishing returns, Table 1 shows that an additional 15% speed increase is quite feasible.

From this data, we can make two basic conclusions about the effect of noise during training. First, a little goes a long way: it takes very little noise to allow the network to behave stochastically, and too much noise is harmful. Second, where the noise comes from, while not vital, can certainly affect training time. As we saw from Figure 3, high values of noise in the starting position and low values in the activation function produce the fastest learning.

Putting it all Together

Now that we have an understanding of the individual optimizing effects of coarse statistics, limited settling time, and noise, it is useful to see how they work together. A final series of networks was trained using six settles, σ of 0.1, initial activations ranging between ± 0.9, 50 cycles of initial settling and 10 of statistic collection. Compared with the 200kCs required in M&M's original simulation, this network reached criterion in only 37kCs -- a speedup of 82% or roughly a factor of 5. The next step is to train networks to learn substantially larger problems. So far the results are quite promising: We have been able to train a number of larger networks of varying architectures to generate probability distributions as outputs when the initial training was done using coarse estimates of the error gradient. One such network consisted of 23 input units, 40 hidden units, and 50 output units and was trained on a spelling-to-meaning problem like that studied by Hinton and Shallice (1991). The training corpus consisted of 21 words with one meaning and 15 words with two distinct meanings. The network reached a stopping criterion comparable to that used for the translation problem in 1014 epochs, or a total of 608 kCs. If given an opportunity to fine tune it's weights during several epochs of subsequent training with more accurate estimates of the gradient, this network, and all the others we've examined, show no adverse side effects of the initial coarse training.

Conclusions

A number of general conclusions can be drawn from this research. First and foremost, despite the daunting theoretical computational needs of CHL, it can be optimized so that probability distributions can be learned in a reasonable amount of time. Although accurate statistics may call for a larger number of settles per pattern per epoch, learning can progress up to all but the fine tuning stage use the coarse estimate of the error gradient generated with a small number of settles. The rapid settling behavior and lack of a within-settle distribution of outputs allow us to use a limited number of cycles within each settle, further speeding learning. Furthermore, while noise in the network needs to be present, it can come from various sources and its magnitude can be small, assisting learning using gradients based on a small sample.

References

Galland, C., & Hinton, G. E. (1989). Deterministic Boltzmann Learning in networks with asymmetric connectivity. Department of Computer Science, University of Toronto. (Tech. Rep. No. CRG-TR-89-6).

Hinton, G. E., & Sejnowski, T. J. (1986). Learning in Boltzmann Machines. In D. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the microstructure of cognition. Volume 1.* Cambridge, MA: MIT Press

Hinton, G. E., & Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review*, **98**(1), 74-95.

MacKay, D. (1992). A practical Bayesian framework for backpropagation networks. *Neural Computation*, **4**, 448-472.

McClelland, J. L. (1993). Toward a theory of information processing in graded random interactive networks. In *Attention and Performance* (Vol. 14, pp. 655-689). Cambridge, MA

Movellan, J. R., & McClelland, J. L. (1993). Learning continuous probability distributions with symmetric diffusion networks. *Cognitive Science*, **17**, 463-496.

Rumelhart, D. E., Durbin, R., Golden, R., & Chauvin, Y. (in press) Backpropagation: The Basic Theory. In Y. Chauvin and D. E. Rumelhart (Eds.), *Back propagation: Theory, Architectures, and Applications*, Erlbaum.