In *Network: Computation in Neural Systems* 9(1) 1998.

Cascaded Redundancy Reduction

Virginia R. de Sa*and Geoffrey E. Hinton
Department of Computer Science
University of Toronto
Toronto, Ontario, M5S 1A4, Canada

August 2, 1998

*Current Address and Address for Communication: Virginia R. de Sa, Sloan Center for Theoretical Neurobiology, Dept. Physiology–Box 0444, 513 Parnassus Ave., San Francisco, CA 94143-0444

Abstract

We describe a method of incrementally constructing a hierarchical generative model of an ensemble of binary data vectors. The model is composed of stochastic, binary, logistic units. Hidden units are added to the model one at a time with the goal of minimizing the information required to describe the data vectors using the model. In addition to the top-down generative weights that define the model, there are bottom-up recognition weights that determine the binary states of the hidden units given a data vector. Even though the stochastic generative model can produce each data vector in many ways, the recognition model is forced to pick just one of these ways. The recognition model therefore underestimates the ability of the generative model to predict the data, but this underestimation greatly simplifies the process of searching for the generative and recognition weights of a new hidden unit.

1 Introduction

Unsupervised learning algorithms attempt to extract statistical structure from an ensemble of input vectors without the help of an additional teaching signal. One way of defining what it means to extract statistical structure is to appeal to a stochastic generative model that produces data vectors from hidden stochastic variables. If a relatively simple generative model is likely to have produced the observed data vectors, then we can view the underlying states that the model uses to generate a particular data vector as an economical representation of that vector.

In this paper we consider the problem of fitting a generative model that is composed of multiple layers of stochastic binary units. The bottom layer consists of "visible" units which correspond to the individual components of a binary data vector. The model generates data by starting at the top layer and working downwards. At the top level, each unit turns on randomly and independently with a probability that depends only on its generative bias. At each subsequent level, a unit, i, receives top-down input that depends on the already decided binary states, s_j of units in the layer above and the generative weights, $g_{j,i}$. This top-down input is combined with the unit's generative bias[1], $g_{0,i}$ to produce the unit's "total generative input", x_i.

$$x_i = g_{0,i} + \sum_{j>i} s_j g_{j,i} \tag{1}$$

The total generative input is then put through a logistic function to determine the probability, p_i, of turning on the unit:

$$p_i = \sigma(x_i) = \frac{1}{1+e^{-x_i}} \tag{2}$$

Because the units are stochastic, each top-down generative pass through the model will typically produce a different data vector and each data vector can typically be produced in many different ways.

[1]The bias is equivalent to a generative weight from a "bias" unit that is always on. For notational convenience we define this node as unit 0.

If we start with a multilayer model with a fixed architecture and we want to learn the generative weights and biases, the obvious objective function is the log likelihood of the observed data under the generative model:

$$\sum_d \log p(d|\theta) = \sum_d \log \left(\sum_\alpha p(\alpha|\theta) p(d|\alpha, \theta) \right) \tag{3}$$

where d is an index over data vectors, α is an index over all possible assignments of binary states to the hidden units and θ is the generative weights and biases.

If units receive top-down generative connections from many units in the layer above, it is intractable to maximize this objective function using either the EM algorithm or gradient ascent. Both of these algorithms need to compute the posterior distribution over exponentially many hidden state vectors, α, given a data vector, d, and the generative parameters, θ. Instead of computing the posterior distribution exactly, it can be approximated using Gibbs sampling (Neal 1992) or mean field methods (Jaakkola, Saul and Jordan 1996), though neither of these approaches seems to have much biological plausibility.

A Helmholtz machine (Dayan, Hinton, Neal and Zemel 1995) uses a separate set of recognition connections to compute an approximation to the posterior distribution. It can be shown that the use of an approximation still allows EM to maximize a lower bound on the log probability of the data, with the tightness of the bound depending on the quality of the approximation (Neal and Hinton 1998). For one version of the Helmholtz machine (Hinton, Dayan, Frey and Neal 1995), there is a very simple "wake-sleep" learning algorithm that uses only locally available information.

An extreme version of the idea behind Helmholtz machines is for the recognition connections to approximate the posterior distribution using a single vector of hidden states. Although this provides a much worse bound than the other approximations it greatly simplifies the search for the recognition and generative weights because there is no need to integrate across the many alternative ways of producing the data. The hope is that the computational convenience will compensate for the looser bound. A secondary aim is to see how much is gained by allowing the recognition connections of a Helmholtz machine to produce a whole distribution rather than a single hidden state vector[2]. In this form, the algorithm is similar to that of Redlich (1993). The biggest difference is in the search strategy and the construction in this case of a stochastic generative model.

Given a data vector, d, the recognition connections produce a binary state vector, s^d, over the hidden units. Using s^d we get an upper bound on the negative log probability of the data:

$$\begin{aligned} -\log p(d|\theta) &= -\log \sum_{\alpha} p(\alpha|\theta) p(d|\alpha, \theta) \\ &\leq -\log p(s^d|\theta) p(d|s^d, \theta) \end{aligned} \tag{4}$$

This bound and its derivatives can be computed in a time that is linear in the number of connections, so it is much more efficient to perform gradient descent in the upper bound than in the true negative log probability of the data.

2 A Minimum Description Length (MDL) interpretation of the objective function

There is an alternative interpretation of the upper bound which is mathematically equivalent but conceptually different. We think in terms of a communication game in which a sender must communicate each data vector to a receiver. Rather than sending the individual components of each data vector separately, the sender first uses the recognition connections to produce a hierarchical representation of the data vector. Then she sends this representation starting at the top level and working downwards. The idea is that if the sender has found a good model for representing the data, it should take fewer bits to send the data using the representations prescribed by the model than to send the raw data. In the full MDL framework, we

[2]This is essentially the same debate as Viterbi versus Baum-Welch in the fitting of Hidden Markov Models, except that Baum-Welch computes the true posterior rather than just an approximation to it.

must also include the one-time cost of communicating the model itself. In this paper we ignore the model cost, but it could be used as a criterion for deciding when to stop enlarging the model (only add the new unit when the improvement in coding cost is more than the cost of communicating the new unit; this is dependent on the coding strategy for the weights).

According to Shannon's coding theorem, if a sender and a receiver have agreed upon a probability distribution under which a discrete event has probability p, the event can be communicated using $-\log_2 p$ bits. This is an asymptotic result that involves using block coding techniques but the details are irrelevant here. On average, it is best for the sender and receiver to agree on the correct probability distribution, but this is not required. They can use any distribution they like. For communicating the binary state, s_i^d, of unit i that is produced by the recognition connections, the sender and receiver use the Bernoulli distribution p_i^d, $1 - p_i^d$ that is obtained by applying the generative model to the states in the layer above (see Eq. 2). This distribution is available to the receiver because the states of the units in the layer above have already been communicated. So it requires $-s_i^d \log_2 p_i^d - (1 - s_i^d)\log_2(1 - p_i^d)$ bits to communicate s_i^d. The cost of communicating a data vector is simply the sum of this quantity over all units, including the visible units that represent the data. So the description length is:

$$\begin{aligned} C(d) &= \sum_i -s_i^d \log_2 p_i^d - (1 - s_i^d)\log_2(1 - p_i^d) \\ &= -\log p(s^d|\theta) p(d|s^d, \theta) \end{aligned} \tag{5}$$

3 Why a hidden unit helps

Consider the cost of describing the data using no hidden units. The only adjustable parameters in the model are the generative biases of the visible units. The cost in Eq. 5 is minimized by setting these biases so that $p_i = \sigma(g_{0,i})$ is equal to the fraction of the data vectors in which unit i is on. We call this the "base rate" model. The base rate model makes good use of the individual frequencies with which visible units come on, but it ignores all correlations. Now, suppose we have a single hidden unit, k. If a subset of the visible units tend to come on together, this redundancy can be captured by setting the recognition weights, $r_{i,k}$, of the hidden unit so that it comes on in these circumstances and setting its generative weights, $g_{k,i}$, so as to increase p_i for all the units in the subset. By modifying the generative biases of the visible units it is also possible to reduce p_i when the hidden unit is off. In effect, the hidden unit allows the base rates to be dependent on which data vector is being described. With just one hidden unit we obtain a mixture of two base rate models.

After adding one hidden unit, we could split the data into two disjoint subsets and apply the same algorithm again to each subset separately. This would create a tree structure and would be a natural extension of decision tree algorithms like CART (Breiman, Friedman, Olshen and Stone, 1984) to the task of probability density estimation. Unfortunately, splitting the data in this way is usually a bad strategy when there are multiple different regularities in the data. Suppose, for example, that components of the data vector can be split into two subsets. Within each subset there are weak correlations, but between subsets components are independent. The first split in a tree can use all of the data to detect the redundancy within one of the subsets, but at the next level down the other redundancy must be discovered separately in each half of the tree using only half the data. To avoid this problem, we consider all of the data when adding each new hidden unit, but we take into account the work that is already being done by the previous hidden units in creating appropriate top-down probabilities for describing the states of lower units. We also allow new hidden units to receive recognition connections from existing hidden units and to send generative connections to them. In adding a new hidden unit, k, we greedily attempt to minimize the cost of describing the data using the new unit and all the previous ones. This cost includes the cost of describing the state of k and the cost of describing the state of every pre-existing unit, i, using the new generative probabilities created by combining the new bias for i with the generative weight from k and the pre-existing generative weights from units above i.

$$\begin{aligned} C &= \sum_d \left(-s_k^d \log \sigma(g_{0,k}) - (1 - s_k^d) \log[1 - \sigma(g_{0,k})]\right) \\ &\quad + \sum_d \sum_i \left(-s_i^d \log[\sigma(x_i)]\right. \end{aligned}$$

$$- (1 - s_i^d) \log[1 - \sigma(x_i)]\Big) \tag{6}$$

where x_i is the generative input to unit i.

$$x_i = g_{0,i} + s_k^d g_{k,i} + \sum_{k>j>i} s_j^d g_{j,i}$$

The term $\sum_{k>j>i} s_j^d g_{j,i}$ is unaffected by adding unit k so it can be stored for each data vector in the training set, thus eliminating much computation. To emphasize this and to simplify subsequent equations we define

$$G_{k>j>i}^d \equiv \sum_{k>j>i} s_j^d g_{j,i} \tag{7}$$

In searching for the best generative and recognition weights for k, we allow the generative biases of all earlier units to be modified but not the other generative weights or the recognition weights or biases of other units.

4 Creating a smooth search space

Once a new hidden unit has been added to the network, it behaves deterministically. Its recognition weights cause it to be either on or off for each data vector. However, the search for a suitable set of recognition weights is easier if the weights have a smooth, differentiable effect on the unit's behaviour. This can be achieved by using a stochastic unit whose probability of being on is a smooth monotone function of its recognition weights. Each data vector determines a single state for all previous hidden units but for the new unit we consider both possible states and compute the expected value of the cost function in Eq. 6 given this stochastic behaviour. To make the cost function represent the negative log probability of the data vector allowing for both states of the new hidden unit, it is necessary to subtract the entropy of the state of the new hidden unit, as explained in Hinton and Zemel (1994). If this entropy term is omitted, the expected cost is always minimized by scaling up all of the units recognition weights so that it behaves deterministically.

While searching for the best recognition weights we want the hidden unit to behave stochastically in order to smooth the search space, but at the conclusion of the search we want the hidden unit to behave deterministically. This can be achieved by using a "temperature" parameter which scales both the entropy term and the softness of the logistic function used in recognition (but not the one used in generation). During the search the temperature is reduced from 1 to 0 in small steps.

At a given temperature, the cost function to be minimized is:

$$
\begin{aligned}
C &= \sum_d \sum_{i<k} \Big(q_k^d [-s_i^d \log(\sigma(G^d_{k>j>i} + g_{0,i} + g_{k,i})) \\
&\qquad\qquad - (1 - s_i^d) \log(1 - \sigma(G^d_{k>j>i} + g_{0,i} + g_{k,i}))] \\
&\qquad + (1 - q_k^d)[-s_i^d \log \sigma(G^d_{k>j>i} + g_{0,i}) \\
&\qquad\qquad - (1 - s_i^d) \log(1 - \sigma(G^d_{k>j>i} + g_{0,i}))] \Big) \\
&+ \sum_d [-q_k^d \log \sigma(g_{0,k}) - (1 - q_k^d) \log(1 - \sigma(g_{0,k}))] \\
&+ \sum_d T[(q_k^d \log q_k^d) + (1 - q_k^d) \log(1 - q_k^d)] \qquad (8)
\end{aligned}
$$

where q_k^d is the recognition probability of unit k for data vector d. The last line of Eq. 8 is the entropy of unit k (weighted by $-T$) and the penultimate line is the expected cost of coding the state of unit k given its generative bias. The first two lines of the equation are the cost of coding all the other units given that unit k is on, weighted by the probability that k is on. The next two lines are the cost if k is off, weighted by the corresponding probability.

5 The standard CRR learning procedure

We have explored several variants of the Cascaded Redundancy Reduction (CRR) learning procedure. To simplify the discussion we first describe the "standard" procedure.

The outermost loop of the procedure consists of adding hidden units one at a time in a cascaded fashion (shown in Figure 1) as in Fahlman and Lebiere (1990). Once a unit has been added, its recognition weights and bias are never changed.

In searching for the best generative and recognition weights for the new unit, CRR decreases the temperature from 1 to 0 in steps, and at each temperature it performs an alternating optimization that closely resembles the EM algorithm. Holding the recognition weights fixed, an iterative search is performed for the optimal generative weights and biases (an M-step). Then, holding the generative weights and biases fixed, the recognition weights and bias are iteratively improved (a partial E-step). The overall algorithm has the form:

```
Set the initial generative biases for visible units
Repeat adding hidden units until stopping criterion is met
  Create a new unit, k, with random recognition weights and recognition bias
  Set T = 1
  Repeat until T = 0
    Repeat N times
      Do Newton searches for new generative biases of all previous units, j
      Do Newton searches for generative weights from new unit, k
      Do a conjugate gradient search for recognition weights and bias of k
      Decrease T using a temperature schedule
  Add the new hidden unit to the network
  Set the generative bias of the new hidden unit
  Use conjugate gradient to backfit ALL generative weights and biases
```

5.1 Updating the recognition weights

The update rule for the recognition weights can be obtained by taking the partial derivative of the cost with respect to each modifiable recognition weight. This gives:

$$\frac{\partial C}{\partial r_{j,k}} = \frac{1}{T}\sum_{d}\frac{\partial C}{\partial q_k^d} q_k^d(1-q_k^d)s_j^d \tag{9}$$

where

$$\begin{aligned}
\frac{\partial C}{\partial q_k^d} &= \sum_{i<k} \left(-s_i^d \log \frac{\sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i})}{\sigma(G_{k>j>i}^d + g_{0,i})} \right. \\
&\qquad \left. - (1-s_i^d) \log \frac{1-\sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i})}{1-\sigma(G_{k>j>i}^d + g_{0,i})} \right) \\
&- \log \frac{\sigma(g_{0,k})}{(1-\sigma(g_{0,k}))} \\
&+ T \log \frac{q_k^d}{(1-q_k^d)} \qquad (10)
\end{aligned}$$

A conjugate gradient search technique is used to determine successive steps in the search.

5.2 Updating the generative weights

The generative weights could also be updated using a gradient method. However, we can do better by taking advantage of the independence between the individual weights and the monotonicity of the derivative. First note that the partial derivative with respect to one of the generative weights depends only on that generative weight and the bias weight to the same unit.

$$\frac{\partial C}{\partial g_{k,i}} = \sum_d -q_k^d(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i})) \qquad (11)$$

Similarly for the partial derivative with respect to one of the bias weights.

$$\frac{\partial C}{\partial g_{0,i}} = \sum_d -q_k^d(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i})) - (1-q_k^d)(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i})) \qquad (12)$$

This gives two equations (with two unknowns). However we will show below that finding the local extrema involves solving equations of only one variable.

Setting the derivative equal to zero in Equations 11 and 12 gives

$$0 = \sum_d q_k^d(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i}))$$

$$
\begin{aligned}
0 &= \sum_d \Big(q_k^d (s_i^d - \sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i})) \\
&\quad + (1 - q_k^d)(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i})) \Big)
\end{aligned}
$$

and combining gives

$$
0 = \sum_d (1 - q_k^d)(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i}))
$$

which we can solve for $g_{0,i}$. Unfortunately this equation can not be solved directly but as it is a monotonic function it could be solved using a binary search technique. We chose however, to use Newton's method.

Letting

$$
A = \sum_d (1 - q_k^d)(s_i^d - \sigma(G_{k>j>i}^d + g_{0,i}))
$$

we solve for $g_{0,i}$ using

$$
g_{0,i}(t+1) = g_{0,i}(t) + \mathrm{sign}(A) \times \min\left(2, \left|\frac{A(g_{0,i}(t))}{\frac{dA}{dg_{0,i}} - \epsilon}\right|\right) \tag{13}
$$

where ϵ is a positive safety factor used to avoid instability where the second derivative (first derivative of A) vanishes. This together with the constraint that the maximum step size have magnitude 2 gives a robust implementation of Newton's algorithm. Note that due to the monotonicity of A, $\frac{dA}{dg_{0,i}}$ is always non-positive.

Once we have solved for $g_{0,i}$ in Equation 13 we can use it to find $g_{k,i}$ similarly.

$$
B = \sum_d -q_k^d (s_i^d - \sigma(G_{k>j>i}^d + g_{0,i} + g_{k,i}))
$$

$$
g_{k,i}(t+1) = g_{k,i}(t) + \mathrm{sign}(B) \times \min\left(2, \left|\frac{B(g_{0,i}, g_{k,i}(t)))}{\frac{dB}{dg_{k,i}} - \epsilon}\right|\right) \tag{14}
$$

5.3 Backfitting the generative weights

Greedy algorithms have the disadvantage that a step taken early may lead to subsequent poor performance. Ideally we would like to go back and modify earlier connections with added hindsight. Unfortunately to do this for all connections would defeat the efficiency advantages of a greedy algorithm. Changing the recognition weights to a lower unit could invalidate the recognition weights to higher units as they depend on activity propagated from lower units. This would also invalidate the generative weights as they depend on the activated recognition pattern s^d.

The generative weights and biases, however, can be updated for little cost and without invalidating the recognition connections. The update rule is determined from the appropriate gradient.

$$\frac{\partial C}{\partial g_{j,i}} = s_j(s_i - \sigma(\sum_{k>i} s_k g_{k,i})) \tag{15}$$

$$\frac{\partial C}{\partial g_{0,i}} = s_i - \sigma(\sum_{k>i} s_k g_{k,i}) \tag{16}$$

The conjugate gradient algorithm is also used to update these weights. A few conjugate gradient steps of backfitting are performed after the addition of each new unit.

6 Algorithm Modifications

6.1 Adding a Unit more Carefully

One way to avoid some of the worst local minima while adding units is to consider a pool of units for each new input unit as in Cascade-Correlation (Fahlman and Lebiere, 1990). These units can be initialized with different random weights and trained independently in parallel. After training, the performance of each candidate unit can be assessed and the best unit added.

We found that for a fixed number of units added, this did improve the performance, but as discussed below, for better performance with a fixed amount of time, serial consideration of candidate units was better. Only one candidate unit was trained at any time but if it did not lead to a lower coding cost it was not included. To avoid overfitting, the coding costs were evaluated on a separate validation set, rather than on the data used to train the weights. This evaluation was performed before the back-fitting stage.

6.2 Mini-batches

All the iterative updates are done using batch algorithms. For the dataset we used, and most suitable datasets, the size of the set is very large and the patterns within it are quite redundant. This suggests that an on-line algorithm would be more efficient. To maintain the parameter-free advantages of the conjugate gradient batch algorithm while increasing the learning speed we investigated the effect of using mini-batches. For each considered unit 20% of the patterns, balanced for class content, were used for the recognition weight searches and the generative Newton searches.

Using this strategy, we achieved rapid initial decrease in coding cost. At this stage it became more efficient to increase the size of the mini-batches instead of increasing the number of minimizing steps pursued for each unit. The batch size trades off speed for appropriate descent direction. Early in the search, the exact estimate of the gradient is not important to allow significant improvement. When the search has evolved to a reasonable solution, further progress depends on progressively more accurate estimates of the gradient of the desired function. This suggests that an adaptively growing batch size that monitors the percentage of rejected units and increases the batch size when it gets too large would be a good strategy.

7 Results

We tested the algorithm using a dataset previously used to test similar algorithms (Hinton, Dayan, Frey and Neal, 1995; Frey, Hinton, and Dayan, 1996). This dataset consists of 13,000 normalized digits quantized into 8×8

binary images from the US Postal Service Office of Advanced Technology. As in Frey, Hinton, and Dayan, (1996) the data were divided into 6000 training examples, 2000 validation examples and 5000 examples for testing. The validation data were used to evaluate whether a unit should be added and also, as they are already loaded, for backfitting the generative weights after addition of a unit.

Training curves are shown in Figures 2 and 3 for training with the full (training) dataset, the 20% mini batches as well as a run with a gradually increasing batch size. The batches were 10% for the first 20 iterations, 20% for the next 20 iterations, 50% for the next 10 iterations, and 100% for the last 30 iterations. The curves plot the CRR calculated coding cost (an upper bound on the true coding cost) on the validation set. As mentioned, Figure 2 shows that the mini batches allow faster learning (particularly at the beginning) but are limited in their asymptotic accuracy. Increasing the batch size throughout learning, allows fast initial learning and good late learning. Figure 3 shows that when using the validation set, to decide about adding a new unit, the resulting efficiency of the networks are similar for all the training methods.

While the objective function of our network was specifically designed, for simplicity, to optimize the coding cost using a single hidden state per data pattern, the resulting generative model is a stochastic model capable of coding each pattern with a distribution over hidden states just like those constructed using Gibbs sampling, mean field methods, and Helmholtz machines. We can therefore estimate the *actual* cost of coding the data using the true posterior distribution of our trained networks. Testing the network trained with the increasing batch size, we found that the CRR estimated coding cost (using s^d as the sole posterior state) was significantly overestimating the true coding cost of the network, particularly as the size of the network grew. The Figure 4 shows the estimated coding cost using the CRR (single hidden state) cost on the validation set (as shown in Figure 2) reproduced beside the coding cost on the test set calculated using Gibbs sampling of the full posterior distribution of hidden states. Note that the Gibbs sampled estimate of the test error gives a smaller coding cost and that the difference increases with the size of the network. The Gibbs sampled coding cost, on the test data, for the network with 30 hidden units was 38.6 bits, compared to the CRR validation coding cost of 40.8 bits.

For comparison a Helmholtz machine trained using the wake-sleep algorithm with 72 hidden units (in a $16 \Rightarrow 24 \Rightarrow 32 \Rightarrow 64$ layered network architecture) achieved a test-set coding cost of 39.1 bits in 3120 seconds (Frey, Hinton and Dayan 1996). This coding cost was achieved using the recognition distribution learned by the algorithm. For networks of this size, it was not possible to compute the true log-likelihood of the data under the generative network learned by the wake-sleep algorithm. However, Brendan Frey (personal communication, 1997) has found that in a variety of cases in which the networks are small enough to compute the log-likelihood of the data *exactly*, the coding cost given by the recognition distribution was very close to the true log-likelihood under the generative model. Frey, Hinton and Dayan reported the total training time as 7200 seconds which included training several architectures from which they picked the best.

8 Discussion

The central idea of this training algorithm is to avoid the computational cost of computing the posterior distribution over the hidden states by using a single state approximation. This gives computational simplicity at the expense of increased coding cost. We found that even though the network was trained to optimize coding cost using a single hidden state, it had a reduced cost when the full posterior distribution over hidden states was considered. Considered in this way, the algorithm produced networks with similar coding cost to those created using the wake-sleep algorithm. The advantage of the CRR method is that the size of the network does not have to be predetermined. Also, it is hoped that as the networks are biased to performing well with single-state posteriors, they might lead to simpler more understandable representations. On the particular problem we tried, our networks of 30 units achieved a lower coding cost than the 72 hidden unit network trained with the wake-sleep algorithm.

Acknowledgements

We thank Brendan Frey for providing us with the code for the Gibbs sampling calculations and the performance results for the stochastic Helmholtz machine. This research was funded by the Institute for Robotics and Intelligent Systems, the Information Technology Research Center, and NSERC. Hinton is the Nesbitt-Burns fellow of the Canadian Institute for Advanced Research.

References

Breiman L, Friedman J H and Olshen R A and Stone C J 1984 *Classification and regression trees* (Belmont, CA:Wadsworth)

Dayan P, Hinton G E, Neal R M and Zemel R S 1995 The helmholtz machine *Neural Computation* **7** 889-904

Fahlman S E and Lebiere C 1990 The Cascade-Correlation Learning Architecture *Advances in Neural Information Processing Systems 2*, ed D S Touretzky (Morgan Kaufmann) pp 524-532

Frey B J 1997 personal communication

Frey B J, Hinton G E and Dayan P 1996 Does the wake-sleep algorithm produce good density estimators? *Advances in Neural Information Processing Systems 8*, ed D S Touretzky, M C Mozer and M E Hasselmo (Cambridge, MA: MIT Press) pp 661-667

Hinton G E, Dayan P, Frey B J and Neal R M 1995 The wake-sleep algorithm for unsupervised neural networks *Science* **268** 1158-1161

Hinton G E and Zemel R S 1994 Autoencoders, Minimum Description Length and Helmholtz Free Energy *Advances in Neural Information Processing Systems 6* ed J D Cowan, G Tesauro and J Alspector (San Mateo, CA: Morgan Kaufmann) pp 3-10

Jaakkola T, Saul L K and Jordan M I 1996 Fast learning by bounding likelihoods in Sigmoid Type Belief Networks *Advances in Neural Information Processing Systems 8*, ed D S Touretzky, M C Mozer and M E Hasselmo (Cambridge, MA: MIT Press) pp 528-534

Neal R M 1992 Connectionist learning of belief networks *Artificial Intelligence* **56** 71-113

Neal R M and Hinton G E 1998 A new view of the EM algorithm that justifies incremental, sparse and other variants to appear in *Learning in Graphical Models* ed M I Jordan (Kluwer Academic Press) (currently available from ftp://ftp.cs.utoronto.ca/pub/radford/emk.ps.Z)

Redlich A N 1993 Redundancy Reduction as a Strategy for Unsupervised Learning *Neural Computation* **5** 289-304

Figure Captions

Figure 1. The CRR network. The binary activity of unit i, resulting from application of pattern d to the network, is given by s_i^d. The unit currently being considered (k) has analogue activity given by q_k^d. The recognition weight from unit i to unit k is given by $r_{i,k}$ and the generative weights from k to i by $g_{k,i}$ The bias unit is defined to be unit 0. For a given input pattern d We represent the sum generative input from all *added* units j to input unit i by $G^d_{k>j>i}$.

Figure 2. Coding cost vs time. The x axis gives the number of seconds when run on a 200MHz R4400 chip with a 4MB secondary cache. Each plotted point represents the addition of one unit. The plot for the mini-batches was run for 13500 seconds but added no units after 5500 seconds. The y axis gives the average coding cost over the dataset (in bits) for the network at that stage.

Figure 3. Coding cost vs number of added units. This graph plots the same information as in Figure 2, but with respect to the number of units in the current network. This is not just a simple rescaled version of Figure 3, as the consideration of each unit did not always result in a unit being added to the network.

Figure 4. Coding cost vs time. This graph shows the effect of considering the whole posterior distribution over hidden states when calculating the coding cost. The whole posterior is approximated using prolonged Gibbs sampling.

Figure 1

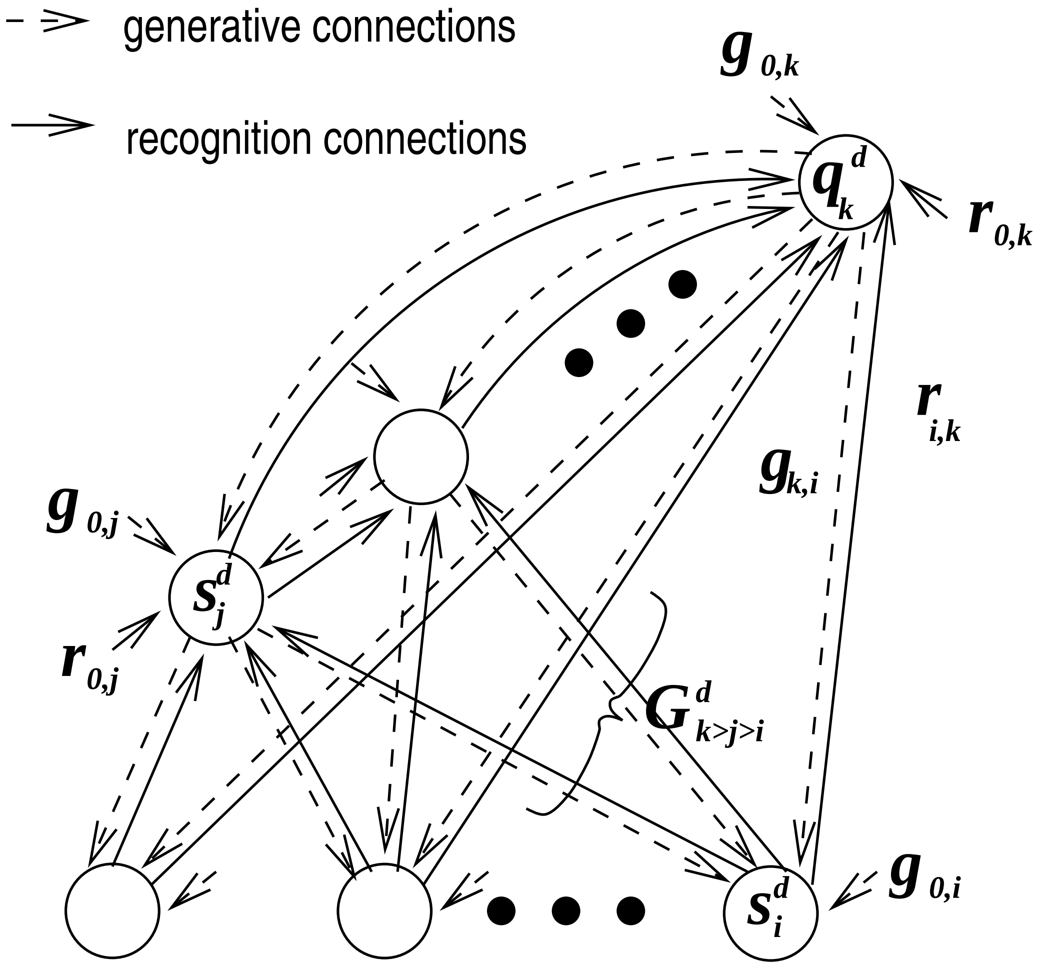

generative connections
recognition connections
g 0,k
q k d
r 0,k
r i,k
g k,i
g 0,j
s j d
r 0,j
G d k>j>i
s i d
g 0,i

Figure 2

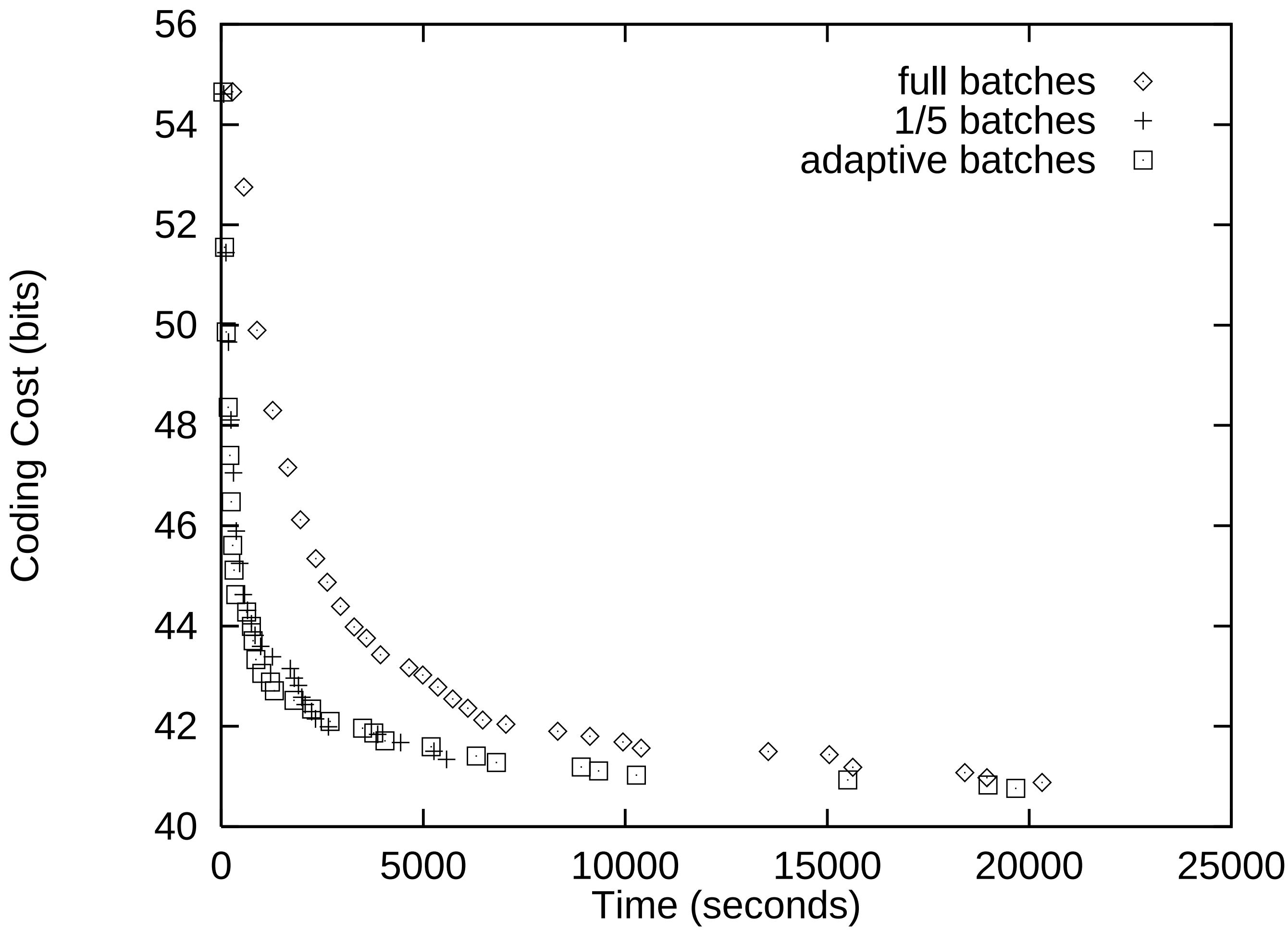

56
54
52
50
48
46
44
42
40
0
5000
10000
15000
20000
25000
Time (seconds)
Coding Cost (bits)
full batches
1/5 batches
adaptive batches

Figure 3

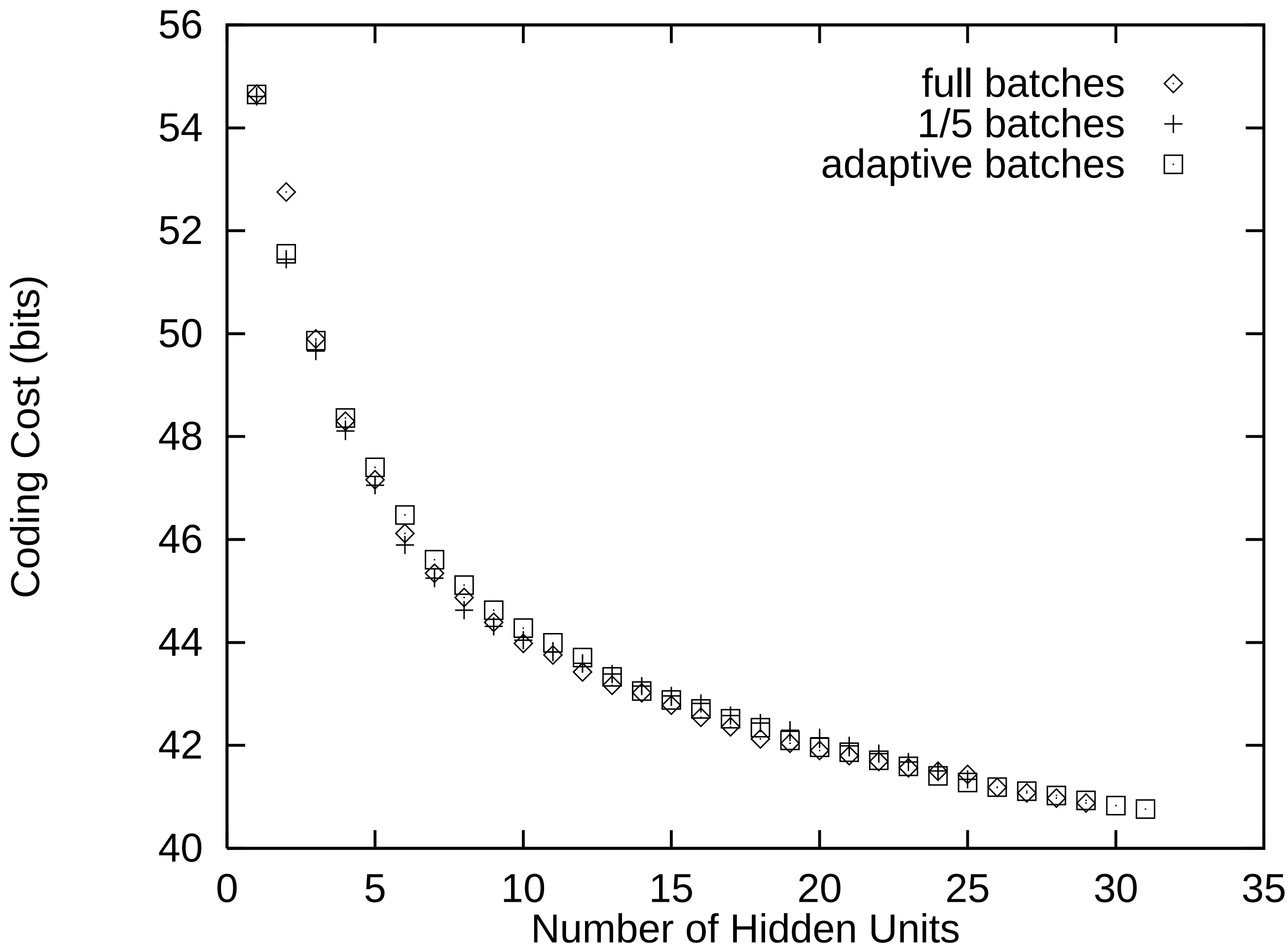
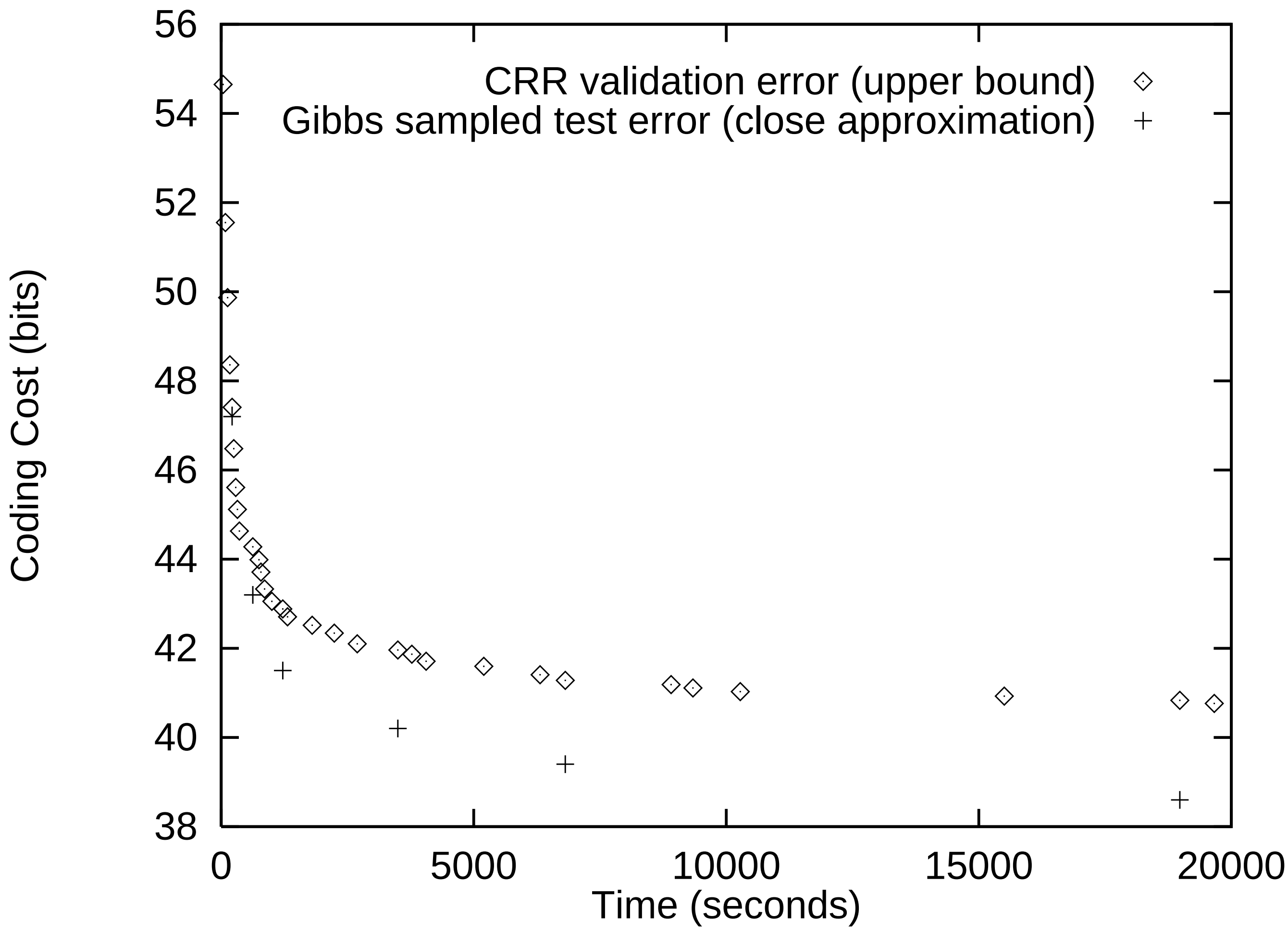

full batches
1/5 batches
adaptive batches
Coding Cost (bits)
Number of Hidden Units
56
54
52
50
48
46
44
42
40
0
5
10
15
20
25
30
35

Figure 4



CRR validation error (upper bound)
Gibbs sampled test error (close approximation)
Coding Cost (bits)
Time (seconds)
56
54
52
50
48
46
44
42
40
38
0
5000
10000
15000
20000