A HIERARCHICAL COMMUNITY OF EXPERTS

GEOFFREY E. HINTON
BRIAN SALLANS
AND
ZOUBIN GHAHRAMANI
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada M5S 3H5
{hinton,sallans,zoubin}@cs.toronto.edu

Abstract. We describe a directed acyclic graphical model that contains a hierarchy of linear units and a mechanism for dynamically selecting an appropriate subset of these units to model each observation. The non-linear selection mechanism is a hierarchy of binary units each of which gates the output of one of the linear units. There are no connections from linear units to binary units, so the generative model can be viewed as a logistic belief net (Neal 1992) which selects a skeleton linear model from among the available linear units. We show that Gibbs sampling can be used to learn the parameters of the linear and binary units even when the sampling is so brief that the Markov chain is far from equilibrium.

1. Multilayer networks of linear-Gaussian units

We consider hierarchical generative models that consist of multiple layers of simple, stochastic processing units connected to form a directed acyclic graph. Each unit receives incoming, weighted connections from units in the layer above and it also has a bias (see figure 1). The weights on the connections and the biases are adjusted to maximize the likelihood that the layers of "hidden" units would produce some observed data vectors in the bottom layer of "visible" units.

The simplest kind of unit we consider is a linear-Gaussian unit. To generate data from a model composed of these units we start at the top

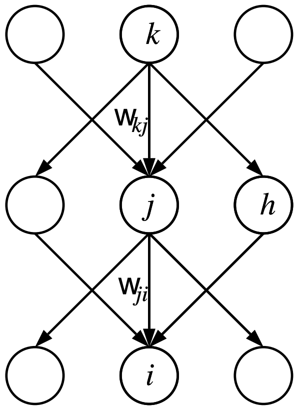

Figure 1. Units in a belief network.

layer and stochastically pick states for each top-level unit from a Gaussian distribution with a learned mean and variance. Once the states, y_k of units in the top layer have been chosen, we can compute the top-down input, $\hat{y}_j$ to each unit, j, in the next layer down:

$$\hat{y}_j = b_j + \sum_k w_{kj} y_k \tag{1}$$

where b_j is the bias of unit j, k is an index over all units in the layer above and w_{kj} is the weight on the top-down connection from k to j. The state of unit j is then picked randomly from a Gaussian distribution with mean $\hat{y}_j$ and a variance σ_j^2 that is learned from data.

The generative model underlying factor analysis (Everitt, 1984) consists of one hidden layer of linear-Gaussian units (the factors) that send weighted connections (the factor loadings) to a visible layer of linear-Gaussian units.

Linear models with Gaussian noise have two important advantages: They often provide good models of continuous data and they are easy to fit even when many of the linear variables are unobserved. Given the states of any subset of the linear units it is tractable to compute the posterior distribution across the unobserved units and once this distribution is known, it is straightforward to use the EM algorithm to update all the parameters of the model. Unfortunately, linear models ignore all the higher order statistical structure in the data so they are inappropriate for tasks like vision in which higher-order structure is crucial.

One sensible way to extend linear models is to use a mixture of M of them (Ghahramani and Hinton, 1996; Hinton et al., 1997). This retains tractability because the full posterior distribution can be found by computing the posterior across each of the M models and then normalizing.

However, a mixture of linear models is not flexible enough to represent the kind of data that is typically found in images. If an image can have several different objects in it, the pixel intensities cannot be accurately modelled by a mixture unless there is a separate linear model for each possible *combination* of objects. Clearly, the efficient way to represent an image that contains n objects is to use a "distributed" representation that contains n separate parts, but this cannot be achieved using a mixture because the non-linear selection process in a mixture consists of picking *one* of the linear models. What we need is a non-linear selection process that can pick arbitrary subsets of the available linear-Gaussian units so that some units can be used for modelling one part of an image, other units can be used for modelling other parts, and higher level units can be used for modelling the redundancies between the different parts.

2. Multilayer networks of binary-logistic units

Multilayer networks of binary-logistic units in which the connections form a directed acyclic graph were investigated by Neal (1992). We call them logistic belief nets or LBN's. In the generative model, each unit computes its top-down input, $\hat{s}_j$, in the same way as a linear-Gaussian unit, but instead of using this top-down input as the mean of a Gaussian distribution it uses it to determine the probability of adopting each if the two states 1 and 0:

$$\hat{s}_j = b_j + \sum_k w_{kj} s_k \tag{2}$$

$$p(s_j = 1 | \{s_k : k \in \mathrm{pa}_j\}) \;=\; \sigma(\hat{s}_j) \;=\; \frac{1}{1 + e^{-\hat{s}_j}} \tag{3}$$

where pa_j is the set of units that send generative connections to unit j (the "parents" of j), and $\sigma(\cdot)$ is the logistic function. A binary-logistic unit does not need a separate variance parameter because the single statistic $\hat{s}_j$ is sufficient to define a Bernouilli distribution.

Unfortunately, it is exponentially expensive to compute the exact posterior distribution over the hidden units of an LBN when given a data point, so Neal used Gibbs sampling: With a particular data point clamped on the visible units, the hidden units are visited one at a time. Each time hidden unit u is visited, its state is stochastically selected to be 1 or 0 in proportion to two probabilities. The first, $P^{\alpha \backslash s_u = 1} = p(s_u = 1, \{s_k^\alpha : k \neq u\})$ is the joint probability of generating the states of all the units in the network (including u) if u has state 1 and all the others have the state defined by the current configuration of states, α. The second, $P^{\alpha \backslash s_u = 0}$, is the same quantity if u has state 0. When calculating these probabilities, the states of all the other units are held constant. It can be shown that repeated application of this stochastic decision rule eventually leads to hidden state configurations being selected according to their posterior probabilities.

Because the LBN is acyclic it is easy to compute the joint probability P^α of a configuration, α, of states of all the units.

$$P^\alpha = \prod_i p(s_i^\alpha | \{s_k^\alpha : k \in \mathrm{pa}_i\} \tag{4}$$

where s_i^α is the binary state of unit i in configuration α.

It is convenient to work in the domain of negative log probabilities which are called energies by analogy with statistical physics. We define E^α to be $-\ln P^\alpha$.

$$E^\alpha = -\sum_u \left(s_u^\alpha \ln \hat{s}_u^\alpha + (1 - s_u^\alpha) \ln(1 - \hat{s}_u^\alpha)\right) \tag{5}$$

where s_u^α is the binary state of unit u in configuration α, $\hat{s}_u^\alpha$ is the top-down expectation generated by the layer above, and u is an index over all the units in the net.

The rule for stochastically picking a new state for u requires the ratio of two probabilities and hence the difference of two energies

$$\Delta E_u^\alpha = E^{\alpha \backslash s_u = 0} - E^{\alpha \backslash s_u = 1} \tag{6}$$

$$p(s_u = 1 | \{s_k^\alpha : k \neq u\}) = \sigma(\Delta E_u^\alpha) \tag{7}$$

All the contributions to the energy of configuration α that do not depend on s_j can be ignored when computing ΔE_j^α. This leaves a contribution that depends on the top-down expectation $\hat{s}_j$ generated by the units in the layer above (see Eq. 3) and a contribution that depends on both the states, s_i, and the top-down expectations, $\hat{s}_i$, of units in the layer below (see figure 1)

$$\begin{aligned}\Delta E_j^\alpha \quad = \quad & \ln \hat{s}_j^\alpha - \ln(1 - \hat{s}_j^\alpha) + \sum_i \left[s_i^\alpha \ln \hat{s}_i^{\alpha \backslash s_j = 1} + (1 - s_i^\alpha) \ln \left(1 - \hat{s}_i^{\alpha \backslash s_j = 1}\right) \right. \\ & \left. - \, s_i^\alpha \ln \hat{s}_i^{\alpha \backslash s_j = 0} - (1 - s_i^\alpha) \ln \left(1 - \hat{s}_i^{\alpha \backslash s_j = 0}\right) \right] \end{aligned} \tag{8}$$

Given samples from the posterior distribution, the generative weights of a LBN can be learned by using the online delta rule which performs gradient ascent in the log likelihood of the data:

$$\Delta w_{ji} = \epsilon s_j (s_i - \hat{s}_i) \tag{9}$$

3. Using binary units to gate linear units

It is very wasteful to use highly non-linear binary units to model data that is generated from continuous physical processes that behave linearly

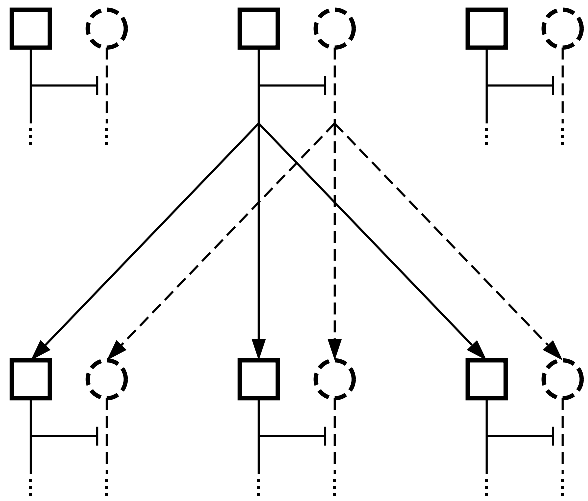

Figure 2. Units in a community of experts, a network of paired binary and linear units. Binary units (solid squares) gate the outputs of corresponding linear units (dashed circles) and also send generative connections to the binary units in the layer below. Linear units send generative connections to linear units in the layer below (dashed arrows).

over small ranges. So rather than using a multilayer binary network to generate data directly, we use it to synthesize an appropriate linear model by selecting from a large set of available linear units. We pair a binary unit with each hidden linear unit (figure 2) and we use the same subscript for both units within a pair. We use y for the real-valued state of the linear unit and s for the state of the binary unit. The binary unit gates the output of the linear unit so Eq. 1 becomes:

$$\hat{y}_j = b_j + \sum_k w_{kj} y_k s_k \tag{10}$$

It is straightforward to include weighted connections from binary units to linear units in the layer below, but this was not implemented in the examples we describe later. To make Gibbs sampling feasible (see below) we prohibit connections from linear units to binary units, so in the generative model the states of the binary units are unaffected by the linear units and are chosen using Eq. 2 and Eq. 3. Of course, during the inference process the states of the linear units do affect the states of the binary units.

Given a data vector on the visible units, it is intractable to compute the posterior distribution over the hidden linear and binary units, so an approximate inference method must be used. This raises the question of whether the learning will be adversely affected by the approximation errors that occur during inference. For example, if we use Gibbs sampling for inference and the sampling is too brief for the samples to come from the equilibrium distribution, will the learning fail to converge? We show in section 6 that it is not necessary for the brief Gibbs sampling to approach equilibrium. The only property we really require of the sampling is that it get us closer to equilibrium. Given this property we can expect the learning to improve a bound on the log probability of the data.

3.1. PERFORMING GIBBS SAMPLING

The obvious way to perform Gibbs sampling is to visit units one at a time and to stochastically pick a new state for each unit from its posterior distribution given the current states of all the other units. For a binary unit we need to compute the energy of the network with the unit on or off. For a linear unit we need to compute the quadratic function that determines how the energy of the net depends on the state of the unit.

This obvious method has a significant disadvantage. If a linear unit, j, is gated out by its binary unit (*i.e.*, $s_j = 0$) it cannot influence the units below it in the net, but it still affects the Gibbs sampling of linear units like k that send inputs to it because these units attempt to minimize $(y_j - \hat{y}_j)^2/2\sigma_j^2$. So long as $s_j = 0$ there should be no net effect of y_j on the units in the layer above. These units completely determine the distribution of y_j, so sampling from y_j would provide no information about their distributions. The effect of y_j on the units in the layer above during inference is unfortunate because we hope that most of the linear units will be gated out most of the time and we do not want the teeming masses of unemployed linear units to disturb the delicate deliberations in the layer above. We can avoid this noise by integrating out the states of linear units that are gated out. Fortunately, the correct way to integrate out y_j is to simply ignore the energy contribution $(y_j - \hat{y}_j)^2/2\sigma_j^2$.

A second disadvantage of the obvious sampling method is that the decision about whether or not to turn on a binary unit depends on the particular value of its linear unit. Sampling converges to equilibrium faster if we integrate over all possible values of y_j when deciding how to set s_j. This integration is feasible because, given all other units, y_j has one Gaussian posterior distribution when $s_j = 1$ and another Gaussian distribution when $s_j = 0$. During Gibbs sampling, we therefore visit the binary unit in a pair first and integrate out the linear unit in deciding the state of the binary unit. If the binary unit gets turned on, we then pick a state for the linear unit from the relevant Gaussian posterior. If the binary unit is turned off it is unnecessary to pick a value for the linear unit.

For any given configuration of the binary units, it is tractable to compute the full posterior distribution over all the selected linear units. So one interesting possibility is to use Gibbs sampling to stochastically pick states for the binary units, but to integrate out *all* of the linear units when making these discrete decisions. To integrate out the states of the selected linear units we need to compute the exact log probability of the observed data using the selected linear units. The change in this log probability when one of the linear units is included or excluded is then used in computing the energy gap for deciding whether or not to select that linear unit. We have not implemented this method because it is not clear that it is worth the computational effort of integrating out all of the selected linear units at the beginning of the inference process when the states of some of the binary units are obviously inappropriate and can be improved easily by only integrating out one of the linear units.

Given samples from the posterior distribution, the incoming connection weights of both the binary and the linear units can be learned by using the online delta rule which performs gradient ascent in the log likelihood of the data. For the binary units the learning rule is Eq. 9. For linear units the rule is:

$$\Delta w_{ji} = \epsilon\; y_j s_j (y_i - \hat{y}_i) s_i / \sigma_i^2 \tag{11}$$

The learning rule for the biases is obtained by treating a bias as a weight coming from a unit with a state of 1.[1]

The variance of the local noise in each linear unit, σ_j^2, can be learned by the online rule:

$$\Delta \sigma_j^2 = \epsilon\; s_j \left[(y_j - \hat{y}_j)^2 - \sigma_j^2 \right] \tag{12}$$

Alternatively, σ_j^2 can be fixed at 1 for all hidden units and the effective local noise level can be controlled by scaling the incoming and outgoing weights.

4. Results on the bars task

The noisy bars task is a toy problem that demonstrates the need for sparse distributed representations (Hinton et al., 1995; Hinton and Ghahramani, 1997). There are four stages in generating each $K \times K$ image. First a global orientation is chosen, either horizontal or vertical, with both cases being equally probable. Given this choice, each of the K bars of the appropriate orientation is turned on independently with probability 0.4. Next, each active bar is given an intensity, chosen from a uniform distribution. Finally, independent Gaussian noise is added to each pixel. A sample of images generated in this way is shown in figure 3(a).

[1] We have used w_{ji} to denote both the weights from binary units to binary units and from linear units to linear units; the intended meaning should be inferred from the context.

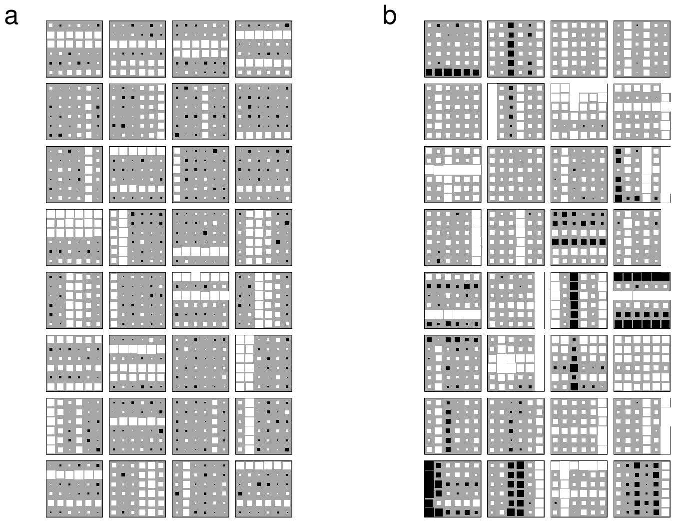

Figure 3. a) Training data for the noisy bars problem. b) Images generated by the trained network. The area of each square represents the value of the corresponding pixel in the 6×6 images. White represents positive values and black represents negative values.

We trained a 3-layer network on the 6×6 noisy bars problem. The network consisted of one pair of units in the top hidden layer, where each pair consists of a linear-Gaussian unit gated by its corresponding binary logistic unit; 24 pairs of units in the first hidden layer; and 36 linear-Gaussian units in the visible layer. The network was trained for 12 passes through a data set of of 1000 images, with a learning rate of 0.04 and a weight decay parameter of 0.04. The images were presented in a different, random order for each pass.

For each image presented, 16 Gibbs sampling iterations were performed. Gibbs sampling was performed by visiting each pair of units in a layer in random order, where for each pair the binary unit was visited first, followed by the linear unit. Of the 16 network states visited, the first four were discarded, and the next 12 were used for learning. The weights from the linear units in the first hidden layer to the units in the visible layer were constrained to be positive. Without this constraint, the trained model still generates images from the correct distribution, but the solution is not so easily interpreted. The result of training is shown in figure 4.

The trained network is using 12 of the linear-Gaussian units in the first hidden layer to represent each of the 12 possible horizontal and vertical bars. The top level binary unit is selecting the linear units in the first hidden layer that represent horizontal bars by exciting the corresponding

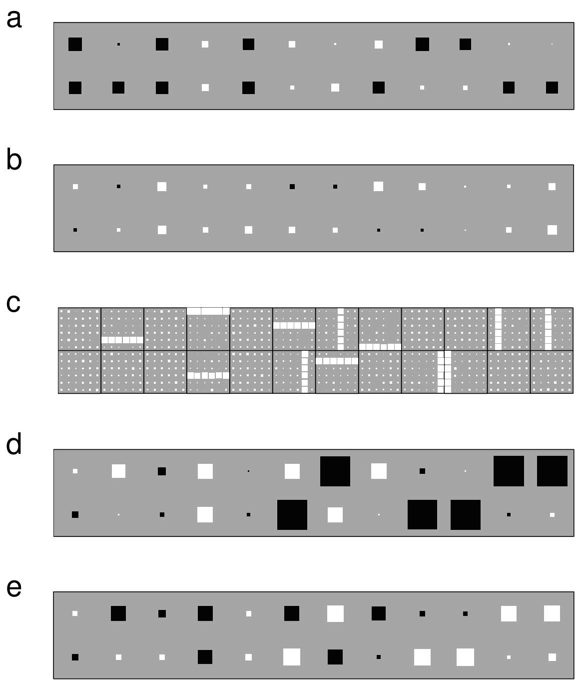

Figure 4. Generative weights and biases of a three-layered network after being trained on the noisy bars problem. a) Weights from the top layer linear-Gaussian unit to the 24 middle layer linear-Gaussian units. b) Biases of the middle layer linear units. c) Weights from the 24 middle layer linear units to the 36 visible units. d) Weights from the top layer binary logistic unit to the 24 middle layer binary logistic units. e) Biases of the middle layer binary logistic units.

binary units; these binary units are biased to be off otherwise. Similarly, the binary units that correspond to vertical bars, which are often active due to positive biases, are being inhibited by the top binary unit. The top linear unit is simply acting as an additional bias on the linear units in the first hidden layer. Examples of data generated by the trained network are shown in figure 3(b).

The network was shown novel images, and 10 iterations of Gibbs sampling were performed. After the final iteration, the top level binary unit was found to be off for 90% of vertical images, and on for 84% of horizontal images.

5. Results on handwritten digits

We trained a similar three-layer network on handwritten twos and threes from the CEDAR CDROM 1 database (Hull, 1994). The digits were scaled to an 8×8 grid, and the 256-gray-scale pixel values were rescaled to lie within $[0, 1]$. The 2000 digits were divided into a training set of 1400 digits, and a test set of 600 digits, with twos and threes being equally represented in both sets. A small subset of the training data is shown in figure 5(a).

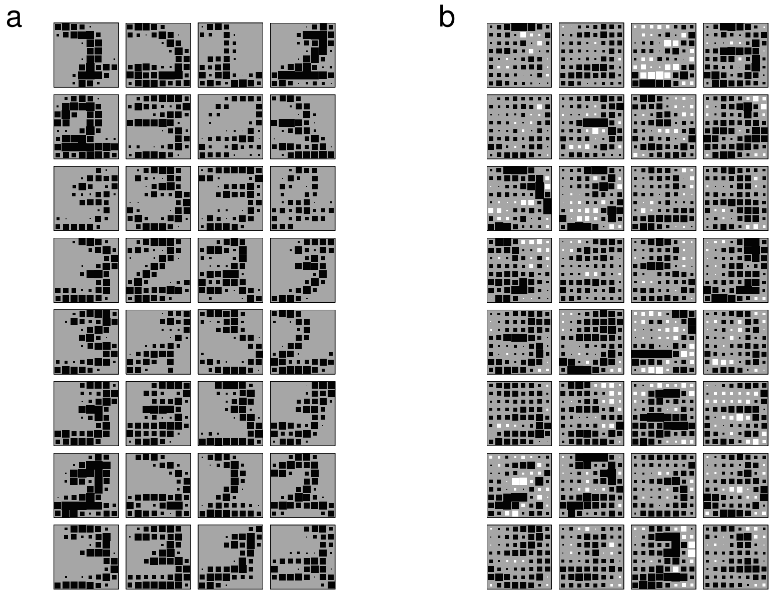

Figure 5. a) A subset of the training data. b) Images generated by the trained network. For clarity, black represents positive values in this figure.

The network consisted of a single pair of units in the top hidden layer, 24 pairs of units in the first hidden layer, and 64 linear-Gaussian units in the visible layer. During training, the network made 43 passes through the data set, with a learning rate of 0.01 and a weight decay parameter of 0.02. Gibbs sampling was performed as in the bars problem, with 4 discarded Gibbs sampling iterations, followed by 12 iterations used for learning. For this task, there were no constraints placed on the sign of the weights from the linear-Gaussian units in the first hidden layer to the units in the visible layer. The result of training is shown in figure 6.

In this case, the network uses all 24 linear units in the first hidden layer to represent digit features. Some of the features are global, while others are highly localized. The top binary unit is selecting the linear units in the first hidden layer that correspond to features found predominantly in threes, by exciting the corresponding binary units. Features that are exclusively used in twos are being gated out by the top binary unit, while features that can

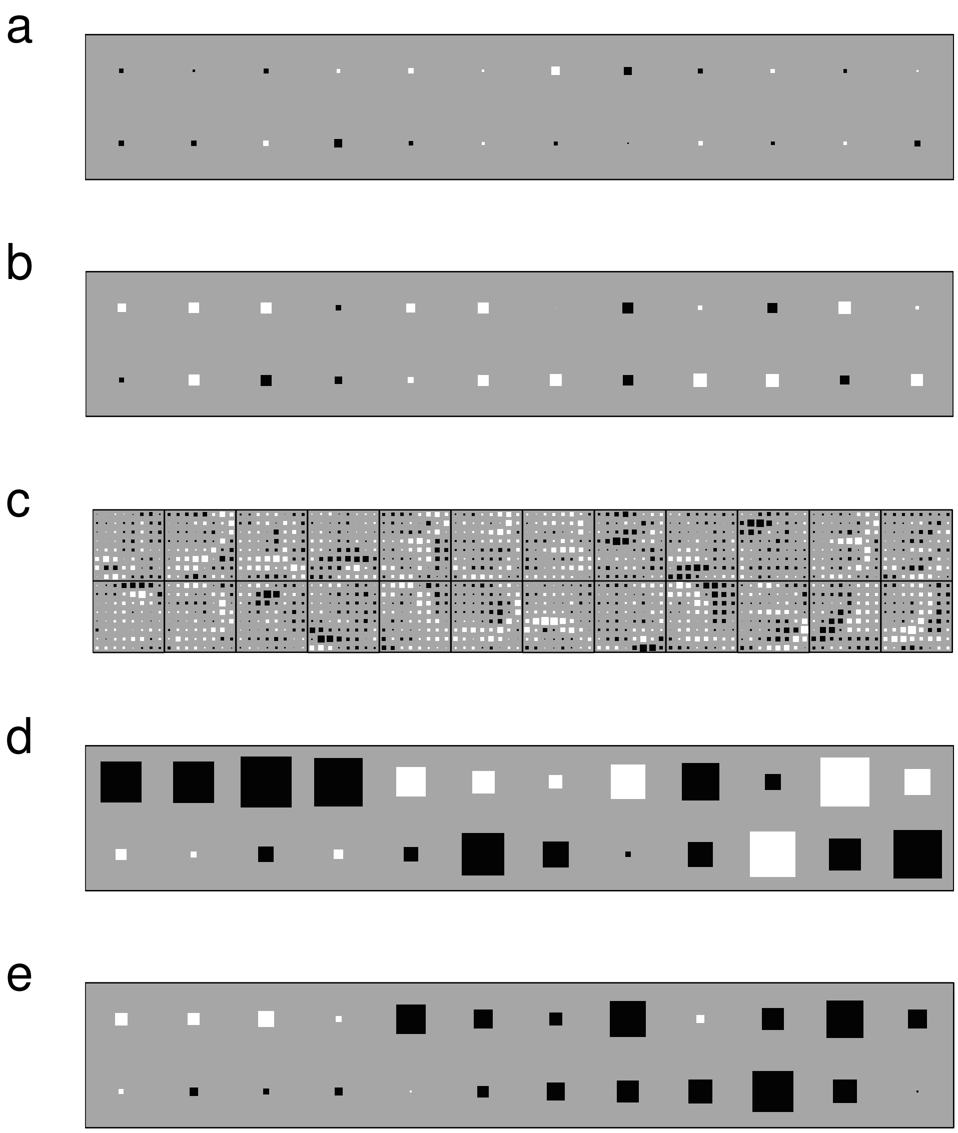

Figure 6. Generative weights and biases of a three-layered network after being trained on handwritten twos and threes. a) Weights from the top layer linear-Gaussian unit to the 24 middle layer linear-Gaussian units. b) Biases of the middle layer linear-Gaussian units. c) Weights from the 24 middle layer linear-Gaussian units to the 36 visible units. d) Weights from the top layer binary logistic unit to the 24 middle layer binary logistic units. e) Biases of the middle layer binary logistic units.

be shared between digits are being only slightly excited or inhibited. When the top binary unit is off, the features found in threes are are inhibited by strong negative biases, while features used in twos are gated in by positive biases on the corresponding binary units. Examples of data generated by the trained network are shown in figure 5(b).

The trained network was shown 600 test images, and 10 Gibbs sampling iterations were performed for each image. The top level binary unit was found to be off for 94% of twos, and on for 84% of threes. We then tried to improve classification by using prolonged Gibbs sampling. In this case, the first 300 Gibbs sampling iterations were discarded, and the activity of the top binary unit was averaged over the next 300 iterations. If the average activity of the top binary unit was above a threshold of 0.32, the digit was classified as a three; otherwise, it was classified as a two. The threshold was found by calculating the optimal threshold needed to classify 10 of the training samples under the same prolonged Gibbs sampling scheme. With prolonged Gibbs sampling, the average activity of the top binary unit was found to be below threshold for 96.7% of twos, and above threshold for 95.3% of threes, yielding an overall successful classification rate of 96% (with no rejections allowed). Histograms of the average activity of the top level binary unit are shown in figure 7.

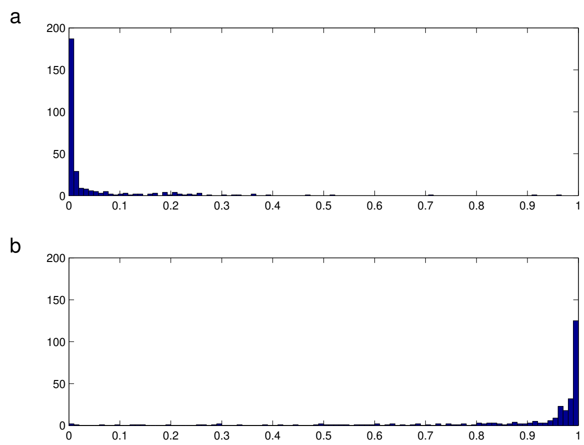

Figure 7. Histograms of the average activity of the top level binary unit, after prolonged Gibbs sampling, when shown novel handwritten twos and threes. a) Average activity for twos in the test set. b) Average activity for threes in the test set.

6. Why brief Gibbs sampling works

There are two major difficulties in using Gibbs sampling for maximum likelihood learning in a neural network:

1. The learning algorithm is usually derived by assuming that Gibbs sampling produces samples from the equilibrium distribution. But when the weights are large, there can be high energy barriers that make convergence to the equilibrium distribution very slow. Moreover, it is generally very hard to measure whether convergence has been achieved.
2. Even if the samples do come from the equilibrium distribution, non-uniform sampling noise can have unfortunate effects. The weights can be strongly repelled from regions where the sampling noise is high, even if the estimated gradient of the log likelihood with respect to the weights is unbiased. A familiar example of this phenomenon is that gravel accumulates at the sides of a road, even if the road is flat, because there is higher variance in the movement of the gravel where the traffic is. In networks with binary logistic units this effect causes the weights to be repelled from values that cause hidden units to be on about half the time, since they then have much higher variance than when they are firmly on or firmly off. This prevents uncommitted hidden units from sitting around in their middle range and following small gradients of the log likelihood. The variance repulsion causes them to wander into useless regions where they are always on or always off.

The sampling noise can easily be estimated by repeating exactly the same sampling procedure several times. It should then be possible for simple gradient methods to cancel out the effects of non-uniform variance by using a smaller learning rate when the variance in the estimated gradient is high.

The failure to approach equilibrium seems like a far less tractable problem than the sampling noise and makes Gibbs sampling seem an unpromising candidate as a model of real neural computation. Fortunately, the EM algorithm can be generalized so that each iteration improves a lower bound on the log likelihood (Neal and Hinton, 1993). In this form, the only property required of Gibbs sampling is that it get closer to equilibrium on each iteration. There is a sensible objective function for the learning that can be improved even if the sampling is far from equilibrium.

Suppose that Gibbs sampling produces a distribution Q over the hidden state configurations. We define the *free energy* of the network as the the expected energy under Q minus the entropy of Q:

$$F = \sum_{\alpha} Q_{\alpha} E_{\alpha} - \left(- \sum_{\alpha} Q_{\alpha} \ln Q_{\alpha} \right) \tag{13}$$

If Q is the posterior distribution over hidden configurations given E, then F is equal to the negative log probability of the configuration of the visible units under the model defined by E. Otherwise, F exceeds the negative log probability of visible configuration by the Kullback-Leibler divergence between Q and P:

$$F = -\ln p(\text{visible}) + \sum_{\alpha} Q_{\alpha} \ln \frac{Q_{\alpha}}{P_{\alpha}} \tag{14}$$

The EM algorithm consists of coordinate descent in F (Neal and Hinton, 1993): a full M step minimizes F with respect to the parameters that determine E, and a full E step minimizes F with respect to Q, which is achieved by setting Q equal to the posterior distribution over the hidden configurations given E.

A major advantage of viewing EM as coordinate descent in F is that it justifies partial E-steps which improve F without fully minimizing it with respect to the distribution Q. We define Q^t to be the distribution reached at the end of partial E-step t and E^t to be the energy function used during partial E-step t. Partial M-step t occurs after partial E-step t and updates the energy function to E^{t+1}.

To eliminate sampling noise, imagine that we have an infinite ensemble of identical networks so that we can compute the exact Q distribution produced by a few sweeps of Gibbs sampling. Provided we start the Gibbs sampling in each network from the hidden configuration at the end of the previous partial E-step we are guaranteed that $F^{t+1} \leq F^t$ because the gradient M-step ensures that:

$$\sum_{\alpha} Q^t_{\alpha} E^{t+1}_{\alpha} \leq \sum_{\alpha} Q^t_{\alpha} E^t_{\alpha} \tag{15}$$

while Gibbs sampling, however brief, ensures that:

$$\sum_{\alpha} Q^{t+1}_{\alpha} E^{t+1}_{\alpha} + Q^{t+1}_{\alpha} \ln Q^{t+1}_{\alpha} \leq \sum_{\alpha} Q^t_{\alpha} E^{t+1}_{\alpha} + Q^t_{\alpha} \ln Q^t_{\alpha}. \tag{16}$$

In practice, we try to approximate an infinite ensemble by using a very small learning rate in a single network so that many successive partial E-steps are performed using very similar energy functions. But it is still nice to know that with a sufficiently large ensemble it is possible for a simple learning algorithm to improve a bound on the log probability of the visible configurations even when the Gibbs sampling is far from equilibrium.

Changing the parameters can move the equilibrium distribution further from the current distribution of the Gibbs sampler. The E step ensures that the Gibbs sampler will chase this shifting equilibrium distribution. One worrisome consequence of this is that the equilibrium distribution may end up very far from the initial distribution of the Gibbs sampler. Therefore, when presented a new data point for which we don't have a previous remembered Gibbs sample, inference can take a very long time since the Gibbs sampler will have to reach equilibrium from its initial distribution.

There are at least three ways in which this problem can be finessed:

1. Explicitly learn a bottom-up initialization model. At each iteration t, the initialization model is used for a fast bottom-up recognition pass. The Gibbs sampler is initialized with the activities produced by this pass and proceeds from there. The bottom-up model is trained using the difference between the next sample produced by the Gibbs sampler and the activities it produced bottom-up.
2. Force inference to recapitulate learning. Assume that we store the sequence of weights during learning, from which we can obtain the sequence of corresponding energy functions. During inference, the Gibbs sampler is run using this sequence of energy functions. Since energy functions tend to get peakier during learning, this procedure should have an effect similar to annealing the temperature during sampling. Storing the entire sequence of weights may be impractical, but this procedure suggests a potentially interesting relationship between inference and learning.
3. Always start from the same distribution and sample briefly. The Gibbs sampler is initialized with the same distribution of hidden activities at each time step of learning and run for only a few iterations. This has the effect of penalizing models with an equilibrium distribution that is far from the distributions that the Gibbs sampler can reach in a few samples starting from its initial distribution.[2] We used this procedure in our simulations.

7. Conclusion

We have described a probabilistic generative model consisting of a hierarchical network of binary units that select a corresponding network of linear units. Like the mixture of experts (Jacobs et al., 1991; Jordan and Jacobs, 1994), the binary units gate the linear units, thereby choosing an appropriate set of linear units to model nonlinear data. However, unlike the mixture of experts, each linear unit is its own expert, and any subset of experts can

[2]The free energy, F, can be interpreted as a penalized negative log likelihood, where the penalty term is the Kullback-Leibler divergence between the approximating distribution Q_α and the equilibrium distribution (Eq. 14). During learning, the free energy can be decreased either by increasing the log likehood of the model, or by decreasing this KL divergence. The latter regularizes the model towards the approximation.

be selected at once, so we call this network a hierarchical community of experts.

Acknowledgements

We thank Peter Dayan, Michael Jordan, Radford Neal and Michael Revow for many helpful discussions. This research was funded by NSERC and the Ontario Information Technology Research Centre. GEH is the Nesbitt-Burns Fellow of the Canadian Institute for Advanced Research.

References

Everitt, B. S. (1984). *An Introduction to Latent Variable Models.* Chapman and Hall, London.

Ghahramani, Z. and Hinton, G. E. (1996). The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1 [ftp://ftp.cs.toronto.edu/pub/zoubin/tr-96-1.ps.gz], Department of Computer Science, University of Toronto.

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., Dayan, P., and Revow, M. (1997). Modeling the manifolds of Images of handwritten digits. *IEEE Trans. Neural Networks*, 8(1):65–74.

Hinton, G. E. and Ghahramani, Z. (1997). Generative models for discovering sparse distributed representations. *Phil. Trans. Roy. Soc. London B: Biol. Sci.*

Hull, J. J. (1994). A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixture of local experts. *Neural Computation*, 3:79–87.

Jordan, M. I. and Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214.

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

Neal, R. M. and Hinton, G. E. (1993). A new view of the EM algorithm that justifies incremental and other variants. Unpublished manuscript [ftp://ftp.cs.utoronto.ca/pub/radford/em.ps.z], Department of Computer Science, University of Toronto.