Training Products of Experts by Minimizing Contrastive Divergence

GCNU TR 2000-004

Geoffrey E. Hinton
Gatsby Computational Neuroscience Unit
University College London
17 Queen Square, London WC1N 3AR, U.K.

http://www.gatsby.ucl.ac.uk/

Abstract

It is possible to combine multiple probabilistic models of the same data by multiplying their probability distributions together and then renormalizing. This is a very efficient way to model high-dimensional data which simultaneously satisfies many different low-dimensional constraints because each individual expert model can focus on giving high probability to data vectors that satisfy just one of the constraints. Data vectors that satisfy this one constraint but violate other constraints will be ruled out by their low probability under the other experts. Training a product of experts appears difficult because, in addition to maximizing the probability that each individual expert assigns to the observed data, it is necessary to make the experts be as different as possible. This ensures that the product of their distributions is small which allows the renormalization to magnify the probability of the data under the product of experts model. Fortunately, if the individual experts are tractable there is an efficient way to train a product of experts.

1 Introduction

One way of modeling a complicated, high-dimensional data distribution is to use a large number of relatively simple probabilistic models and to somehow combine the distributions specified by each model. A well-known example of this approach is a mixture of Gaussians in which each simple model is a Gaussian and the combination rule consists of taking a weighted arithmetic mean of the individual distributions. This is equivalent to assuming an overall generative model in which each data vector is generated by first choosing one of the individual generative models and then allowing that individual model to generate the data vector. Combining models by forming a mixture is attractive for several reasons. It is easy to fit mixtures of tractable models to data using EM or gradient ascent and, if the individual models differ a lot, the mixture is likely to be a better fit to the true distribution of the data than a random choice among the individual models. Indeed, if sufficiently many models are included in the mixture, it is possible to approximate complicated smooth distributions arbitrarily accurately.

Unfortunately, mixture models are very inefficient in high-dimensional spaces. Consider, for example, the manifold of face images. It takes about 35 real numbers to specify the shape, pose, expression and illumination of a face and, under good viewing conditions, our perceptual systems produce a sharp posterior distribution on this 35-dimensional manifold. This cannot be done using a mixture of models each of which is tuned in the 35-dimensional space because the posterior distribution cannot be sharper than the individual models in the mixture and the individual models must be broadly tuned to allow them to cover the 35-dimensional space.

A very different way of combining distributions is to multiply them together and renormalize. High-dimensional distributions, for example, are often approximated as the product of one-dimensional distributions. If the individual distributions are uni- or multivariate Gaussians, their product will also be a multivariate Gaussian so, unlike mixtures of Gaussians, products of Gaussians cannot approximate arbitrary smooth distributions. If, however, the individual models are a bit more complicated and each contain one or more latent (*i.e.* hidden) variables, multiplying their distributions together (and renormalizing) can be very powerful. Individual models of this kind will be called "experts".

Products of Experts (PoE) have the advantage that they can produce much sharper distributions than the individual expert models. For example, each expert model can constrain a different subset of the dimensions in a high-dimensional space and their product will then constrain all of the dimensions. For modeling handwritten digits, one low-resolution model can generate images that have the approximate overall shape of the digit and other more local models can ensure that small image patches contain segments of stroke with the correct fine structure. For modeling sentences, each expert can enforce a nugget of linguistic knowledge. For example, one expert could ensure that the tenses agree, one could ensure that there is number agreement between the subject and verb and one could ensure that strings in which colour adjectives follow size adjectives are more probable than the the reverse.

Fitting a PoE to data appears difficult because it appears to be necessary to compute the derivatives, with repect to the parameters, of the partition function that is used in the renormalization. As we shall see, however, these derivatives can be finessed by optimizing a less obvious objective function than the log likelihood of the data.

2 Learning products of experts by maximizing likelihood

We consider individual expert models for which it is tractable to compute the derivative of the log probability of a data vector with respect to the parameters of the expert. We combine n individual expert models as follows:

$$p(\mathbf{d}|\theta_1 ... \theta_n) = \frac{\Pi_m p_m(\mathbf{d}|\theta_m)}{\sum_{\mathbf{c}} \Pi_m p_m(\mathbf{c}|\theta_m)} \tag{1}$$

where $\mathbf{d}$ is a data vector in a discrete space, θ_m is all the parameters of individual model m, $p_m(\mathbf{d}|\theta_m)$ is the probability of $\mathbf{d}$ under model m, and $\mathbf{c}$ indexes all possible vectors in the data space [1]. For continuous data spaces the sum is replaced by the appropriate integral.

For an individual expert to fit the data well it must give high probability to the observed data and it must waste as little probability as possible on the rest of the data space. A PoE, however, can fit the data well even if each expert wastes a lot of its probability on inappropriate regions of the data space provided different experts waste probability in different regions.

The obvious way to fit a PoE to a set of observed *iid* data vectors [2], is to follow the derivative of the log likelihood of each observed vector, $\mathbf{d}$, under the PoE. This is given by:

$$\frac{\partial \log p(\mathbf{d}|\theta_1 ... \theta_n)}{\partial \theta_m} = \frac{\partial \log p_m(\mathbf{d}|\theta_m)}{\partial \theta_m} - \sum_{\mathbf{c}} p(\mathbf{c}|\theta_1 ... \theta_n) \frac{\partial \log p_m(\mathbf{c}|\theta_m)}{\partial \theta_m} \tag{2}$$

The second term on the RHS of Eq. 2 is just the expected derivative of the log probability of an expert on fantasy data, $\mathbf{c}$, that is generated from the PoE. So, assuming that each of the

[1] The symbol p_m has no simple relationship to the symbol p used on the LHS of Eq. 1. Indeed, so long as $p_m(\mathbf{d}|\theta_m)$ is positive it does not need to be a probability at all, though it will generally be a probability in this paper.

[2] For time series models, $\mathbf{d}$ is a whole sequence.

individual experts has a tractable derivative, the obvious difficulty in estimating the derivative of the log probability of the data under the PoE is generating correctly distributed fantasy data. This can be done in various ways. For discrete data it is possible to use rejection sampling: Each expert generates a data vector independently and this process is repeated until all the experts happen to agree. Rejection sampling is a good way of understanding how a PoE specifies an overall probability distribution and how different it is from a causal model, but it is typically *very* inefficient. A Markov chain Monte Carlo method that uses Gibbs sampling is typically much more efficient. In Gibbs sampling, each variable draws a sample from its posterior distribution given the current states of the other variables. Given the data, the hidden states of all the experts can always be updated in parallel because they are conditionally independent. This is a very important consequence of the product formulation. If the individual experts also have the property that the components of the data vector are conditionally independent given the hidden state of the expert, the hidden and visible variables form a bipartite graph and it is possible to update all of the components of the data vector in parallel given the hidden states of all the experts. So Gibbs sampling can alternate between parallel updates of the hidden and visible variables. To get an unbiased estimate of the gradient for the PoE it is necessary for the Markov chain to converge to the equilibrium distribution.

Unfortunately, even if it is computationally feasible to approach the equilibrium distribution before taking samples, there is a second, serious difficulty. Samples from the equilibrium distribution generally have very high variance since they come from all over the model's distribution. This high variance swamps the derivative. Worse still, the variance in the samples depends on the parameters of the model. This variation in the variance causes the parameters to be repelled from regions of high variance even if the gradient is zero. To understand this subtle effect, consider a horizontal sheet of tin which is resonating in such a way that some parts have strong vertical oscillations and other parts are motionless. Sand scattered on the tin will accumulate in the motionless areas even though the time-averaged gradient is zero everywhere.

3 Learning by minimizing contrastive divergence

Maximizing the log likelihood of the data (averaged over the data distribution) is equivalent to minimizing the Kullback-Liebler divergence between the data distribution, Q^0, and the equilibrium distribution over the visible variables, Q^∞, that is produced by prolonged Gibbs sampling from the generative model [3].

$$Q^0||Q^\infty \;=\; \sum_{\mathbf{d}} Q^0_{\mathbf{d}} \log Q^0_{\mathbf{d}} \;-\sum_{\mathbf{d}} Q^0_{\mathbf{d}} \log Q^\infty_{\mathbf{d}} \;=\; -\,H(Q^0) - \;<\log Q^\infty_{\mathbf{d}}>_{Q^0} \tag{3}$$

where $||$ denotes a Kullback-Leibler divergence, the angle brackets denote expectations over the distribution specified as a subscript and $H(Q^0)$ is the entropy of the data distribution. Q^0 does not depend on the parameters of the model, so $H(Q^0)$ can be ignored during the optimization. Note that $Q^\infty_{\mathbf{d}}$ is just another way of writing $p(\mathbf{d}|\theta_1 ... \theta_n)$. Eq. 2, averaged over the data distribution, can be rewritten as:

$$\left\langle \frac{\partial \log Q^\infty_{\mathbf{d}}}{\partial \theta_m} \right\rangle_{Q^0} = \left\langle \frac{\partial \log p_m(\mathbf{d}|\theta_m)}{\partial \theta_m} \right\rangle_{Q^0} - \left\langle \frac{\partial \log p_m(\mathbf{c}|\theta_m)}{\partial \theta_m} \right\rangle_{Q^\infty} \tag{4}$$

There is a simple and effective alternative to maximum likelihood learning which eliminates almost all of the computation required to get samples from the equilibrium distribution and also eliminates much of the variance that masks the gradient signal. This alternative approach involves optimizing a different objective function. Instead of just minimizing $Q^0||Q^\infty$ we minimize

[3] Q^0 is a natural way to denote the data distribution if we imaging starting a Markov chain at the data distribution at time 0.

the difference between $Q^0||Q^\infty$ and $Q^1||Q^\infty$ where Q^1 is the distribution over the "one-step" reconstructions of the data vectors that are generated by one full step of Gibbs sampling.

The intuitive motivation for using this "contrastive divergence" is that we would like the Markov chain that is implemented by Gibbs sampling to leave the initial, distribution Q^0 over the visible variables unaltered. Instead of running the chain to equilibrium and comparing the initial and final derivatives we can simply run the chain for one full step and then update the parameters to reduce the tendency of the chain to wander away from the initial distribution on the first step. Because Q^1 is one step closer to the equilibrium distribution than Q^0, we are guaranteed that $Q^0||Q^\infty$ exceeds $Q^1||Q^\infty$ unless Q^0 equals Q^1, so the contrastive divergence can never be negative. Also, for Markov chains in which all transitions have non-zero probability, $Q^0 = Q^1$ implies $Q^0 = Q^\infty$ so the contrastive divergence can only be zero if the model is perfect.

The mathematical motivation for the contrastive divergence is that the intractable expectation over Q^∞ on the RHS of Eq. 4 cancels out:

$$-\frac{\partial}{\partial \theta_m}\left(Q^0||Q^\infty \;-\; Q^1||Q^\infty\right) \;=\; \left\langle \frac{\partial \log p_m(\mathbf{d}|\theta_m)}{\partial \theta_m} \right\rangle_{Q^0} - \left\langle \frac{\partial \log p_m(\hat{\mathbf{d}}|\theta_m)}{\partial \theta_m} \right\rangle_{Q^1} + \frac{\partial Q^1}{\partial \theta_m}\,\frac{\partial Q^1||Q^\infty}{\partial Q^1} \tag{5}$$

If each expert is chosen to be tractable, it is possible to compute the exact values of the derivatives of $\log p_m(\mathbf{d}|\theta_m)$ and $\log p_m(\hat{\mathbf{d}}|\theta_m)$. It is also straightforward to sample from Q^0 and Q^1, so the first two terms on the RHS of Eq. 5 are tractable. By definition, the following procedure produces an unbiased sample from Q^1:

1. Pick a data vector, $\mathbf{d}$, from the distribution of the data Q^0.
2. Compute, for each expert separately, the posterior probability distribution over its latent (i.e. hidden) variables given the data vector, $\mathbf{d}$.
3. Pick a value for each latent variable from its posterior distribution.
4. Given the chosen values of all the latent variables, compute the conditional distribution over all the visible variables by multiplying together the conditional distributions specified by each expert.
5. Pick a value for each visible variable from the conditional distribution. These values constitute the reconstructed data vector, $\hat{\mathbf{d}}$.

The third term on the RHS of Eq. 5 is problematic to compute, but extensive simulations (see section 10) show that it can safely be ignored because it is small and it seldom opposes the resultant of the other two terms. The parameters of the experts can therefore be adjusted in proportion to the approximate derivative of the contrastive divergence:

$$\Delta\theta_m \;\propto\; \left\langle \frac{\partial \log p_m(\mathbf{d}|\theta_m)}{\partial \theta_m} \right\rangle_{Q^0} - \left\langle \frac{\partial \log p_m(\hat{\mathbf{d}}|\theta_m)}{\partial \theta_m} \right\rangle_{Q^1} \tag{6}$$

This works very well in practice even when a single reconstruction of each data vector is used in place of the full probability distribution over reconstructions. The difference in the derivatives of the data vectors and their reconstructions has some variance because the reconstruction procedure is stochastic. But when the PoE is modelling the data moderately well, the one-step reconstructions will be very similar to the data so the variance will be very small. The close match between a data vector and its reconstruction reduces sampling variance in much the same way as the use of matched pairs for experimental and control conditions in a clincal trial. The low variance makes it feasible to perform online learning after each data vector is presented, though the simulations described in this paper use batch learning in which the parameter updates are based on the summed gradients measured on all of the training set or on relatively large mini-batches.

There is an alternative justification for the learning algorithm in Eq. 6. In high-dimensional datasets, the data nearly always lies on, or close to, a much lower dimensional, smoothly curved manifold. The PoE needs to find parameters that make a sharp ridge of log probability along the low dimensional manifold. By starting with a point on the manifold and ensuring that this point has higher log probability than the typical reconstructions from the latent variables of all the experts, the PoE ensures that the probability distribution has the right local curvature (provided the reconstructions are close to the data). It is possible that the PoE will accidentally assign high probability to other distant and unvisited parts of the data space, but this is unlikely if the log probabilty surface is smooth and if both its height and its local curvature are constrained at the data points. It is also possible to find and eliminate such points by performing prolonged Gibbs sampling without any data, but this is just a way of improving the learning and not, as in Boltzmann machine learning, an essential part of it.

4 A simple example

PoE's should work very well on data distributions that can be factorized into a product of lower dimensional distributions. This is demonstrated in figure 1. There are 15 "unigauss" experts each of which is a mixture of a uniform and a single, axis-aligned Gaussian. In the fitted model, each tight data cluster is represented by the intersection of two Gaussians which are elongated along different axes. Using a conservative learning rate, the fitting required 2,000 updates of the parameters. For each update of the parameters, the following computation is performed on every observed data vector:

1. Given the data, $\mathbf{d}$, calculate the posterior probability of selecting the Gaussian rather than the uniform in each expert and compute the first term on the RHS of Eq. 6.
2. For each expert, stochastically select the Gaussian or the uniform according to the posterior. Compute the normalized product of the selected Gaussians, which is itself a Gaussian, and sample from it is used to get a "reconstructed" vector in the data space.
3. Compute the negative term in Eq. 6 using the reconstructed vector as $\hat{\mathbf{d}}$.

5 Learning a population code

A PoE can also be a very effective model when each expert is quite broadly tuned on every dimension and precision is obtained by the intersection of a large number of experts. Figure 3 shows what happens when experts of the type used in the previous example are fitted to 100-dimensional synthetic images that each contain one edge. The edges varied in their orientation, position, and the intensities on each side of the edge. The intensity profile across the edge was a sigmoid. Each expert also learned a variance for each pixel and although these variances varied, individual experts did not specialize in a small subset of the dimensions. Given an image, about half of the experts have a high probability of picking their Gaussian rather than their uniform. The products of the chosen Gaussians are excellent reconstructions of the image. The experts at the top of figure 3 look like edge detectors in various orientations, positions and polarities. Many of the experts further down have even symmetry and are used to locate one end of an edge. They each work for two different sets of edges that have opposite polarities and different positions.

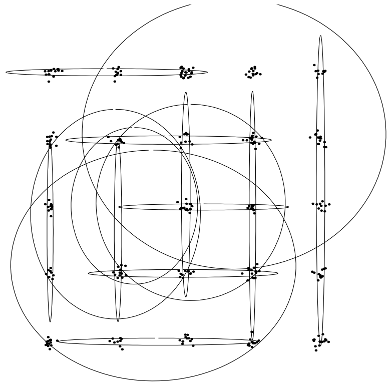

Figure 1: Each dot is a datapoint. The data has been fitted with a product of 15 experts. The ellipses show the one standard deviation contours of the Gaussians in each expert. The experts are initialized with randomly located, circular Gaussians that have about the same variance as the data. The five unneeded experts remain vague, but the mixing proportions of their Gaussians remain high.

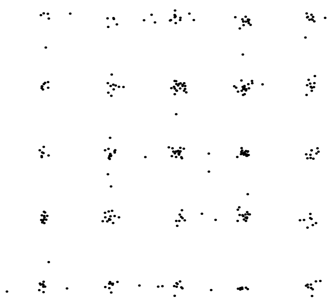

Figure 2: 300 datapoints generated by prolonged Gibbs sampling from the 15 experts fitted in figure 1. The Gibbs sampling started from a random point in the range of the data and used 25 parallel iterations with annealing. Notice that the fitted model generates data at the grid point that is missing in the real data.

6 Initializing the experts

One way to initialize a PoE is to train each expert separately, forcing the experts to differ by giving them different or differently weighted training cases or by training them on different subsets of the data dimensions, or by using different model classes for the different experts. Once each expert has been initialized separately, the individual probability distributions need to be raised to a fractional power to create the initial PoE.

Separate initialization of the experts seems like a sensible idea, but simulations indicate that the PoE is far more likely to become trapped in poor local optima if the experts are allowed to specialize separately. Better solutions are obtained by simply initializing the experts randomly with very vague distributions and using the learning rule in Eq. 6.

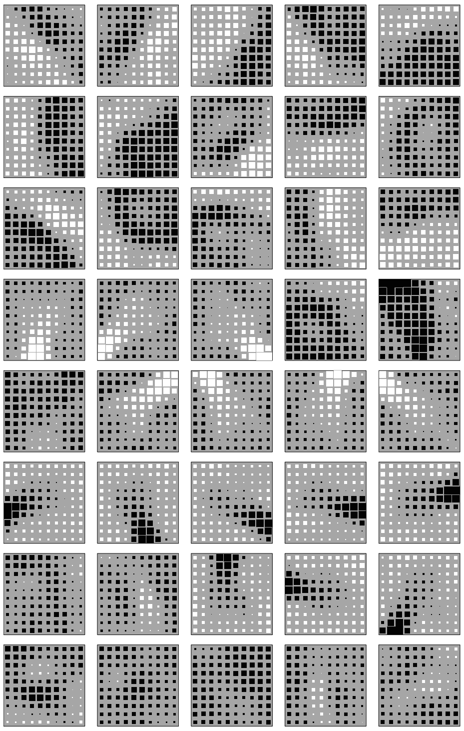

Figure 3: The means of all the 100-dimensional Gaussians in a product of 40 experts, each of which is a mixture of a Gaussian and a uniform. The PoE was fitted to 10×10 images that each contained a single intensity edge. The experts have been ordered by hand so that qualitatively similar experts are adjacent.

7 PoE's and Boltzmann machines

The Boltzmann machine learning algorithm (Hinton and Sejnowski, 1986) is theoretically elegant and easy to implement in hardware, but it is very slow in networks with interconnected hidden units because of the variance problems described in section 2. Smolensky (1986) introduced a restricted type of Boltzmann machine with one visible layer, one hidden layer, and no intralayer connections. Freund and Haussler (1992) realised that in this restricted Boltzmann machine (RBM), the probability of generating a visible vector is proportional to the product of the probabilities that the visible vector would be generated by each of the hidden units acting alone. An RBM is therefore a PoE with one expert per hidden unit[4]. When the hidden unit of an expert is off it specifies a factorial probability distribution in which each visible unit is equally likely to be on or off. When the hidden unit is on, it specifies a different factorial distribution by using the weight on its connection to each visible unit to specify the log odds that the visible unit is on. Multiplying together the distributions over the visible states specified by different experts is achieved by simply adding the log odds. Exact inference is tractable in an RBM because the states of the hidden units are conditionally independent given the data.

The learning algorithm given by Eq. 2 is exactly equivalent to the standard Boltzmann learning algorithm for an RBM. Consider the derivative of the log probability of the data with respect to the weight w_{ij} between a visible unit i and a hidden unit j. The first term on the RHS of Eq. 2 is:

[4]Boltzmann machines and Products of Experts are very different classes of probabilistic generative model and the intersection of the two classes is RBM's

$$\frac{\partial \log p_j(\mathbf{d}|\mathbf{w}_j)}{\partial w_{ij}} \quad = \quad <s_i s_j>_{\mathbf{d}} - <s_i s_j>_{Q^\infty(j)} \tag{7}$$

where $\mathbf{w}_j$ is the vector of weights connecting hidden unit j to the visible units, $<s_i s_j>_{\mathbf{d}}$ is the expected value of $s_i s_j$ when $\mathbf{d}$ is clamped on the visible units and s_j is sampled from its posterior distribution given $\mathbf{d}$, and $<s_i s_j>_{Q^\infty(j)}$ is the expected value of $s_i s_j$ when alternating Gibbs sampling of the hidden and visible units is iterated to get samples from the equilibrium distribution in a network whose only hidden unit is j.

The second term on the RHS of Eq. 2 is:

$$\sum_{\mathbf{c}} p(\mathbf{c}|\mathbf{w}) \frac{\partial \log p_j(\mathbf{c}|\mathbf{w}_j)}{\partial w_{ij}} \quad = \quad <s_i s_j>_{Q^\infty} - <s_i s_j>_{Q^\infty(j)} \tag{8}$$

where $\mathbf{w}$ is all of the weights in the RBM and $<s_i s_j>_{Q^\infty}$ is the expected value of $s_i s_j$ when alternating Gibbs sampling of *all* the hidden and all the visible units is iterated to get samples from the equilibrium distribution of the RBM.

Subtracting Eq. 8 from Eq. 7 and taking expectations over the distribution of the data gives:

$$\left\langle \frac{\partial \log Q^\infty_{\mathbf{d}}}{\partial w_{ij}} \right\rangle_{Q^0} \quad = \quad -\frac{\partial Q^0||Q^\infty}{\partial w_{ij}} \quad = \quad <s_i s_j>_{Q^0} - <s_i s_j>_{Q^\infty} \tag{9}$$

The time required to approach equilibrium and the high sampling variance in $<s_i s_j>_{Q^\infty}$ make learning difficult. It is much more effective to use the approximate gradient of the contrastive divergence. For an RBM this approximate gradient is particularly easy to compute:

$$-\frac{\partial}{\partial w_{ij}} \left(Q^0||Q^\infty \;-\; Q^1||Q^\infty \right) \quad \approx \quad <s_i s_j>_{Q^0} - <s_i s_j>_{Q^1} \tag{10}$$

where $<s_i s_j>_{Q^1}$ is the expected value of $s_i s_j$ when one-step reconstructions are clamped on the visible units and s_j is sampled from its posterior distribution given the reconstruction.

8 Learning the features of handwritten digits

When presented with real, high-dimensional data, a restricted Boltzmann machine trained to minimize the contrastive divergence using Eq. 10 should learn a set of probabilistic binary features that model the data well. To test this conjecture, an RBM with 500 hidden units and 256 visible units was trained on 8000 16×16 real-valued images of handwritten digits from all 10 classes. The images, from the training set on the USPS Cedar ROM, were normalized but highly variable in style. The pixel intensities were normalized to lie between 0 and 1 so that they could be treated as probabilities and Eq. 10 was modified to use probabilities in place of stochastic binary values for both the data and the one-step reconstructions:

$$-\frac{\partial}{\partial w_{ij}} \left(Q^0||Q^\infty \;-\; Q^1||Q^\infty \right) \quad \approx \quad <p_i p_j>_{Q^0} - <p_i p_j>_{Q^1} \tag{11}$$

Stochastically chosen binary states of the hidden units were still used for computing the probabilities of the reconstructed pixels, but instead of picking binary states for the pixels from those probabilities, the probabilities themselves were used as the reconstructed data vector $\hat{\mathbf{d}}$.

It took two days in matlab on a 500MHz workstation to perform 658 epochs of learning. In each epoch, the weights were updated 80 times using the approximate gradient of the contrastive divergence computed on mini-batches of size 100 that contained 10 exemplars of each digit class. The learning rate was set empirically to be about one quarter of the rate that caused divergent oscillations in the parameters. To further improve the learning speed a momentum method was used. After the first 10 epochs, the parameter updates specified by Eq. 11 were supplemented by adding 0.9 times the previous update.

The PoE learned localised features whose binary states yielded almost perfect reconstructions. For each image about one third of the features were turned on. Some of the learned features had on-center off-surround receptive fields or vice versa, some looked like pieces of stroke, and some looked like Gabor filters or wavelets. The weights of 100 of the hidden units, selected at random, are shown in figure 4.

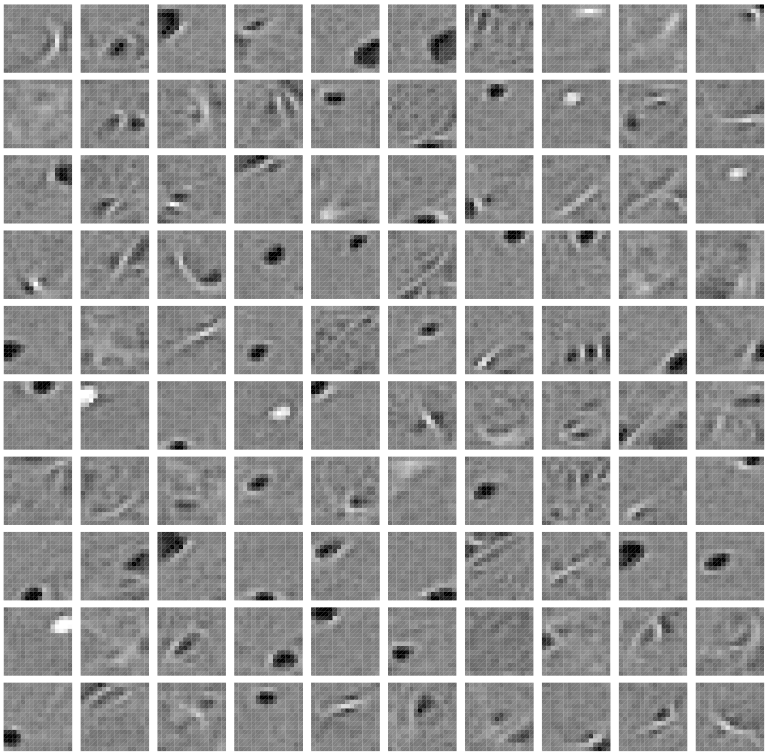

Figure 4: The receptive fields of a randomly selected subset of the 500 hidden units in a PoE that was trained on 8000 images of digits with equal numbers from each class. Each block shows the 256 learned weights connecting a hidden unit to the pixels. The scale goes from +2 (white) to −2 (black).

9 Using learned models of handwritten digits for discrimination

An attractive aspect of PoE's is that it is easy to compute the numerator in Eq. 1 so it is easy to compute the log probability of a data vector up to an additive constant, $\log Z$, which is the log of the denominator in Eq. 1. Unfortunately, it is very hard to compute this additive constant. This does not matter if we only want to compare the probabilities of two different data vectors under the PoE, but it makes it difficult to evaluate the model learned by a PoE. The obvious way to measure the success of learning is to sum the log probabilities that the PoE assigns to test data vectors that are drawn from the same distribution as the training data but are not used during training.

An alternative way to evaluate the learning procedure is to learn two different PoE's on different datasets such as images of the digit 2 and images of the digit 3. After learning, a test image, $\mathbf{t}$, is presented to PoE_2 and PoE_3 and they compute $\log p(\mathbf{t}|\theta_2) + \log Z_2$ and $\log p(\mathbf{t}|\theta_3) + \log Z_3$ respectively. If the difference between $\log Z_2$ and $\log Z_3$ is known it is easy to pick the most likely class of the test image, and since this difference is only a single number it is quite easy to estimate it discriminatively using a set of validation images whose labels are known.

Figure 5 shows features learned by a PoE that contains a layer of 100 hidden units and is trained on 800 images of the digit 2. Figure 6 shows some previously unseen test images of 2's and their one-step reconstructions from the binary activities of the PoE trained on 2's and from an identical PoE trained on 3's.

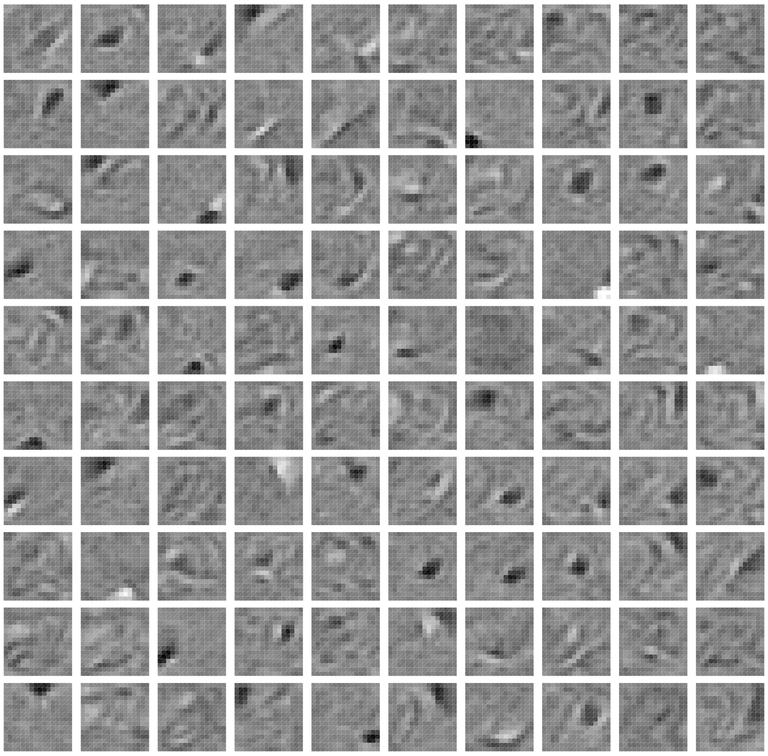

Figure 5: The weights learned by 100 hidden units trained on 16 x 16 images of the digit 2. The scale goes from +3 (white) to −3 (black). Note that the fields are mostly quite local. A local feature like the one in column 1 row 7 looks like an edge detector, but it is best understood as a local deformation of a template. Suppose that all the other active features create an image of a 2 that differs from the data in having a large loop whose top falls on the black part of the receptive field. By turning on this feature, the top of the loop can be removed and replaced by a line segment that is a little lower in the image.

Figure 7 shows the unnormalized log probability scores of some training and test images under a model trained on 825 images of the digit 4 and a model trained on 825 images of the digit 6. Unfortunately, the official test set for the USPS digits violates the standard assumption that test data should be drawn from the same distribution as the training data, so the test images were drawn from the unused portion of the official **training** set. Even for the previously unseen test images, the scores under the two models allow perfect discrimination. To achieve this excellent separation, it was necessary to use models with two hidden layers and to average the scores from two separately trained models of each digit class. For each digit class, one model had 200 units in its first hidden layer and 100 in its second hidden layer. The other model had 100 in the first hidden layer and 50 in the second. The units in the first hidden layer were

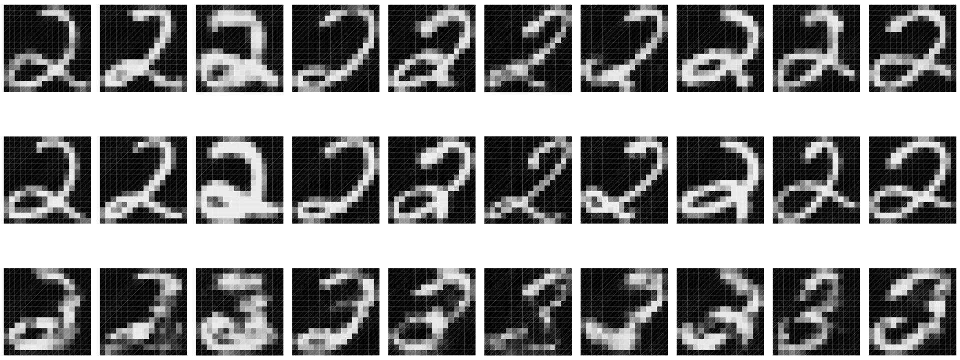

Figure 6: The center row is previously unseen images of 2's. The top row shows the pixel probabilities when the image is reconstructed from the binary activities of 100 feature detectors that have been trained on 2's. The bottom row shows the reconstruction probabilities using 100 feature detectors trained on 3's.

trained without regard to the second hidden layer. After training the first hidden layer, the second hidden layer was then trained using the probabilities of feature activation in the first hidden layer as the data.

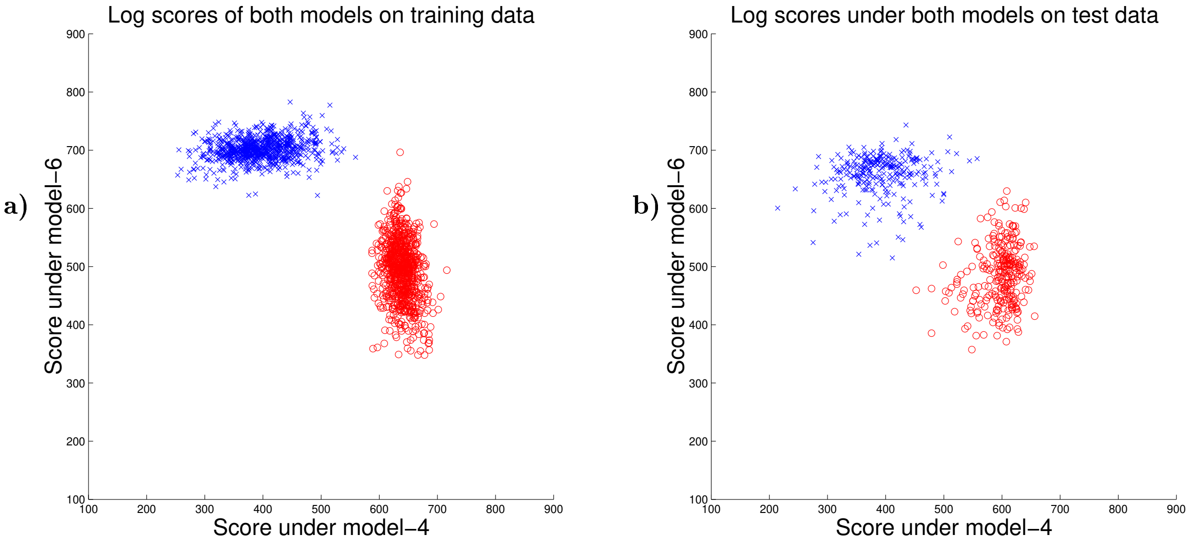

Figure 7: a) The unnormalised log probability scores of the training images of the digits 4 and 6 under the learned PoE's for 4 and 6. b) The log probability scores for previously unseen test images of 4's and 6's. Note the good separation of the two classes.

Figure 8 shows the unnormalized log probability scores for images of 7's and 9's which are the most difficult classes to discriminate. Discrimination is not perfect on the test images, but it is encouraging that all of the errors are close to the decision boundary, so there are no confident misclassifications.

9.1 Dealing with multiple classes

If there are 10 different PoE's for the 10 digit classes it is slightly less obvious how to use the 10 unnormalized scores of a test image for discrimination. One possibility is to use a validation set to train a logistic regression network that takes the unnormalized log probabilities given by

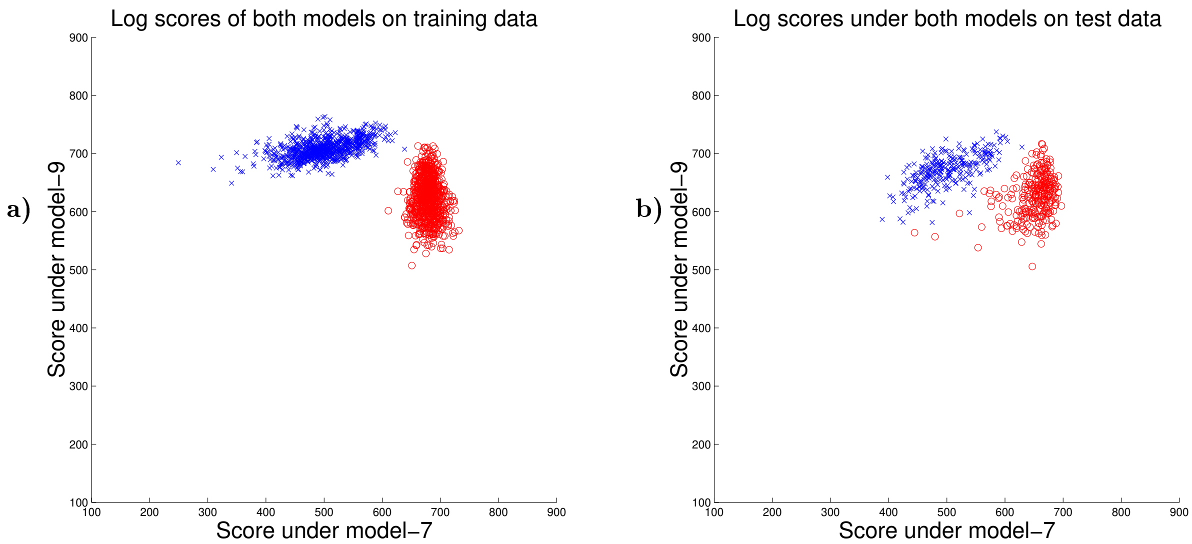

Figure 8: a) The unnormalised log probability scores of the training images of the digits 7 and 9 under the learned PoE's for 7 and 9. b) The log probability scores for previously unseen test images of 7's and 9's. Although the classes are not linearly separable, all the errors are close to the best separating line, so there are no very confident errors.

the PoE's and converts them into a probability distribution across the 10 labels. Figure 9 shows the weights in a logistic regression network that is trained after fitting 10 PoE models to the 10 separate digit classes. In order to see whether the second hidden layers were providing useful discriminative information, each PoE provided two scores. The first score was the unnormalized log probability of the pixel intensities under a PoE model that consisted of the units in the first hidden layer. The second score was the unnormalized log probability of the probabilities of activation of the first layer of hidden units under a PoE model that consisted of the units in the second hidden layer. The weights in figure 9 show that the second layer of hidden units provides useful additional information. Presumably this is because it captures the way in which features represented in the first hidden layer are correlated.

The error rate is 1.1% which compares very favorably with the 5.1% error rate of a simple nearest neighbor classifier on the same training and test sets and is about the same as the very best classifier based on elastic models of the digits (Revow, Williams and Hinton, 1996). If 7% rejects are allowed (by choosing an appropriate threshold for the probability level of the most probable class), there are no errors on the 2750 test images.

Several different network architectures were tried for the digit-specific PoE's and the results reported are for the architecture that did best on the test data. Although this is typical of research on learning algorithms, the fact that test data was used for model selection means that the reported results are a biased estimate of the performance on genuinely unseen test images. Mayraz and Hinton (*in preparation*) report good comparative results for the larger MNIST database at `www.research.att.com/~yann/ocr/mnist` and they were careful to do all the model selection using subsets of the training data so that the official test data was used only to measure the final error rate.

10 How good is the approximation?

The fact that the learning procedure in Eq. 6 gives good results in the simulations described in sections 4, 5, and 9 suggests that it is safe to ignore the final term in the RHS of Eq. 5 that comes from the change in the distribution Q^1.

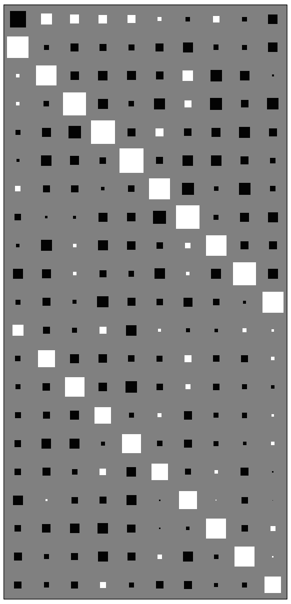

Figure 9: The weights learned by doing multinomial logistic regression on the training data with the labels as outputs and the unnormalised log probability scores from the trained, digit-specific, PoE's as inputs. Each column corresponds to a digit class, starting with digit 1 The top row is the biases for the classes. The next ten rows are the weights assigned to the scores that represent the log probability f the pixels under the model learned by the first hidden layer of each PoE. The last ten rows are the weights assigned to the scores that represent the log probabilities of the probabilities on the first hidden layer under the the model earned by the second hidden layer. Note that although the weights in the last ten rows are smaller, they are still quite large which shows that the scores from the second hidden layers provide useful, additional, discriminative information.

To get an idea of the relative magnitude of the term that is being ignored, extensive simulations were performed using restricted Boltzmann machines with small numbers of visible and hidden units. By performing computations that are exponential in the number of hidden units and exponential in the number of visible units it is possible to compute the exact values of $<s_i s_j>_{Q^0}$ and $<s_i s_j>_{Q^1}$. It is also possible to measure what happens to $Q^0||Q^\infty - Q^1||Q^\infty$ when the approximation in Eq. 10 is used to update the weights by an amount that is large compared with the numerical precision of the machine but small compared with the curvature of the contrastive divergence.

The RBM's used for these simulations had random training data and random weights. They did not have biases on the visible or hidden units. The main result can be summarized as follows: For an individual weight, the RHS of Eq. 10, summed over all training cases, occasionally differs in sign from the LHS. But for networks containing more than 2 units in each layer it is almost certain that a parallel update of all the weights based on the RHS of Eq. 10 will improve the contrastive divergence. In other words, when we average over the training data, the vector of parameter updates given by the RHS is almost certain to have a positive cosine with the true gradient defined by the LHS. Figure 10a is a histogram of the improvements in the contrastive divergence when Eq. 10 was used to perform one parallel weight update in each of a hundred thousand networks. The networks contained 8 visible and 4 hidden units and their weights were chosen from a Gaussian distribution with mean zero and standard deviation 20. With smaller weights or larger networks the approximation in Eq. 10 is even better.

Figure 10b shows that the learning procedure does not always improve the log likelihood of the training data, though it has a strong tendency to do so. Note that only 1000 networks were

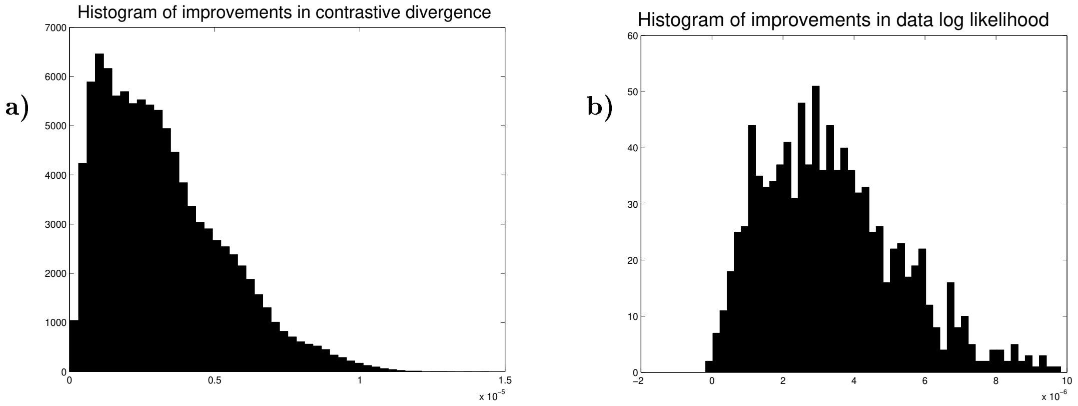

Figure 10: a) A histogram of the improvements in the contrastive divergence as a result of using Eq. 10 to perform one update of the weights in each of 10^5 networks. The expected values on the RHS of Eq. 10 were computed exactly. The networks had 8 visible and 4 hidden units. The initial weights were randomly chosen from a Gaussian with mean 0 and standard deviation 20. The training data was chosen at random. b) The improvements in the log likelihood of the data for 1000 networks chosen in exactly the same way as in figure 10a. Note that the log likelihood decreased in two cases. The changes in the log likelihood are the same as the changes in $Q^0||Q^\infty$ but with a sign reversal.

used for this histogram.

Figure 11 compares the contributions to the gradient of the contrastive divergence made by the RHS of Eq. 10 and by the term that is being ignored. For the vector of weight updates given by Eq. 10 to make the contrastive divergence worse, the dots in figure 11 would have to be above the diagonal line, so it is clear that in these networks the approximation in Eq. 10 is quite safe. Intuitively, we expect Q^1 to lie between Q^0 and Q^∞ so when the parameters are changed to move Q^∞ closer to Q^0 the changes should also move Q^1 towards Q^0 and away from the previous position of Q^∞. So the ignored changes in Q^1 should cause an increase in $Q^1||Q^\infty$ and thus an improvement in the contrastive divergence.

In an earlier version of this paper, the learning rule in Eq. 6 was interpreted as approximate optimization of the contrastive log likelihood:

$$\langle \log Q^\infty_{\mathbf{d}} \rangle_{Q^0} \;-\; \langle \log Q^\infty_{\hat{\mathbf{d}}} \rangle_{Q^1}$$

Unfortunately, the contrastive log likelihood can achieve its maximum value of 0 by simply making all possible vectors in the data space equally probable. The contrastive divergence differs from the contrastive log likelihood by including the entropies of the distributions Q^0 and Q^1 (see Eq. 10) and so the high entropy of Q^1 rules out the solution in which all possible data vectors are equiprobable.

11 Other types of expert

Binary stochastic pixels are not unreasonable for modeling preprocessed images of handwritten digits in which ink and background are represented as 1 and 0. In real images, however, there is typically very high mutual information between the real-valued intensity of one pixel and the real-valued intensities of its neighbors. This cannot be captured by models that use binary stochastic pixels because a binary pixel can never have more than 1 bit of mutual information with anything. It is possible to use "multinomial" pixels that have n discrete values. This is a

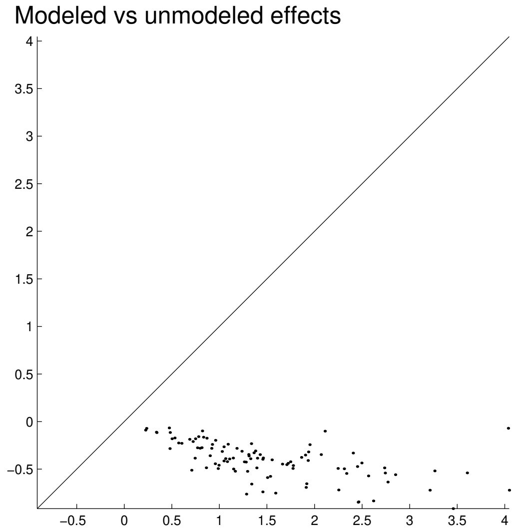

Figure 11: A scatterplot that shows the relative magnitudes of the modeled and unmodeled effects of a parallel weight update on the contrastive divergence. The 100 networks used for this figure have 10 visible and 10 hidden units and their weights are drawn from a zero-mean Gaussian with a standard deviation of 10. The horizontal axis shows $\left(Q^0 || Q^\infty_{old} - Q^1_{old} || Q^\infty_{old}\right) - \left(Q^0 || Q^\infty_{new} - Q^1_{old} || Q^\infty_{new}\right)$ where *old* and *new* denote the distributions before and after the weight update. This differs from the improvement in the contrastive divergence because it ignores the fact that changing the weights changes the distribution of the one-step reconstructions. The ignored term, $Q^1_{old} || Q^\infty_{new} - Q^1_{new} || Q^\infty_{new}$, is plotted on the vertical axis. Points above the diagonal line would correspond to cases in which the weight updates caused a decrease in the contrastive divergence. Note that the unmodeled effects are almost always helpful rather than being in conflict with the modeled effects.

clumsy solution for images because it fails to capture the continuity and one-dimensionality of pixel intensity, though it may be useful for other types of data. A better approach is to imagine replicating each visible unit so that a pixel corresponds to a whole set of binary visible units that all have identical weights to the hidden units. The number of active units in the set can then approximate a real-valued intensity. During reconstruction, the number of active units will be binomially distributed and because all the replicas have the same weights, the single probability that controls this binomial distribution only needs to be computed once. The same trick can be used to allow replicated hidden units to approximate real-values using binomially distributed integer states. A set of replicated units can be viewed as a computationallty cheap approximation to units whose weights actually differ or it can be viewed as a stationary approximation to the behaviour of a single unit over time in which case the number of active replicas is a firing rate.

An alternative to replicating hidden units is to use "unifac" experts that each consist of a mixture of a uniform distribution and a factor analyser with just one factor. Each expert has a binary latent variable that specifies whether to use the uniform or the factor analyser and a real-valued latent variable that specifies the value of the factor (if it is being used). The factor analyser has three sets of parameters: a vector of factor loadings which specify the direction of the factor in image space, a vector of means in image space, and a vector of variances in image space[5]. Experts of this type have been explored in the context of directed acyclic graphs (Hinton, Sallans and Ghahramani, 1998) but they should work better in a product of experts.

An alternative to using a large number of relatively simple experts is to make each expert

[5]The last two sets of parameters are exactly equivalent to the parameters of a "unigauss" expert introduced in section 4, so a "unigauss" expert can be considered to be a mixture of a uniform with a factor analyser that has no factors.

as complicated as possible, whilst retaining the ability to compute the exact derivative of the log likelihood of the data *w.r.t.* the parameters of an expert. In modeling static images, for example, each expert could be a mixture of many axis-aligned Gaussians. Some experts might focus on one region of an image by using very high variances for pixels outside that region. But so long as the regions modeled by different experts overlap, it should be possible to avoid block boundary artifacts.

11.1 Products of Hidden Markov Models

Hidden Markov Models (HMM's) are of great practical value in modeling sequences of discrete symbols or sequences of real-valued vectors because there is an efficient algorithm for updating the parameters of the HMM to improve the log likelihood of a set of observed sequences. HMM's are, however, quite limited in their generative power because the only way that the portion of a string generated up to time t can constrain the portion of the string generated after time t is via the discrete hidden state of the generator at time t. So if the first part of a string has, on average, n bits of mutual information with the rest of the string the HMM must have 2^n hidden states to convey this mutual information by its choice of hidden state. This exponential inefficiency can be overcome by using a product of HMM's as a generator. During generation, each HMM gets to pick a hidden state at each time so the mutual information between the past and the future can be linear in the number of HMM's. It is therefore exponentially more efficient to have many small HMM's than one big one. However, to apply the standard forward-backward algorithm to a product of HMM's it is necessary to take the cross-product of their state spaces which throws away the exponential win. For products of HMM's to be of practical significance it is necessary to find an efficient way to train them.

Andrew Brown (Brown and Hinton, *in preparation*) has shown that for a toy example involving a product of four HMM's, the learning algorithm in Eq. 6 works well. The forward-backward algorithm is used to get the gradient of the log likelihood of an observed or reconstructed sequence *w.r.t.* the parameters of an individual expert. The one-step reconstruction of a sequence is generated as follows:

1. Given an observed sequence, use the forward-backward algorithm in each expert separately to calculate the posterior probability distribution over paths through the hidden states.
2. For each expert, stochastically select a hidden path from the posterior given the observed sequence.
3. At each time step, select an output symbol or output vector from the product of the output distributions specified by the selected hidden state of each HMM.

If more realistic products of HMM's can be trained successfully by minimizing the contrastive divergence, they should be far better than single HMM's for many different kinds of sequential data. Consider, for example, the HMM shown in figure 12. This expert concisely captures a non-local regularity. A single HMM which must also model all the other regularities in strings of English words could not capture this regularity efficiently because it could not afford to devote it entire memory capacity to remembering whether the word "shut" had already occurred in the string.

12 Discussion

There have been previous attempts to learn representations by adjusting parameters to cancel out the effects of brief iteration in a recurrent network (Hinton and McClelland, 1988; O'Reilly, 1996, Seung, 1998), but these were not formulated using a stochastic generative model and an appropriate objective function.

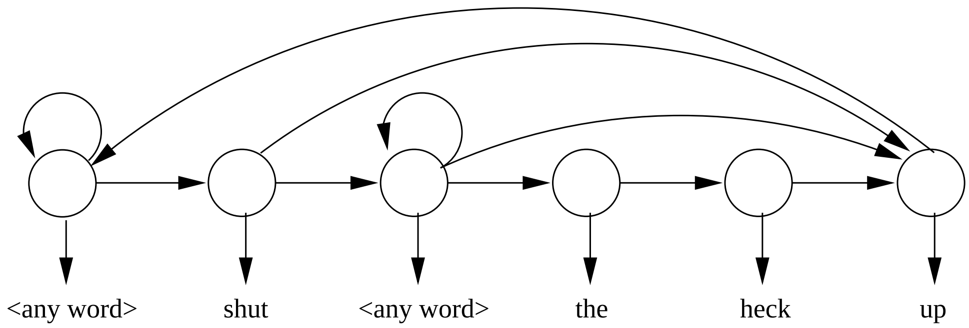

Figure 12: A Hidden Markov Model. The first and third nodes have output distributions that are uniform across all words. If the first node has a high transition probability to itself, most strings of English words are given the same low probability by this expert. Strings that contain the word "shut" followed directly or indirectly by the word "up" have higher probability under this expert.

Minimizing contrastive divergence has an unexpected similarity to the learning algorithm proposed by Winston (1975). Winston's program compared arches made of blocks with "near misses" supplied by a teacher and it used the differences in its representations of the correct and incorrect arches to decide which aspects of its representation were relevant. By using a stochastic generative model we can dispense with the teacher, but it is still the differences between the real data and the near misses generated by the model that drive the learning of the significant features.

12.1 Logarithmic opinion pools

The idea of combining the opinions of multiple different expert models by using a weighted average in the log probability domain is far from new (Genest and Zidek, 1986; Heskes 1998), but research has focussed on how to find the best weights for combining experts that have already been learned or programmed separately (Berger, Della Pietra and Della Pietra, 1996) rather than training the experts cooperatively. The geometric mean of a set of probability distributions has the attractive property that its Kullback-Liebler divergence from the true distribution, P, is smaller than the average of the Kullback-Liebler divergences of the individual distributions, Q:

$$P|| \frac{\Pi_m Q_m^{w_m}}{Z} \leq \sum_m w_m P||Q_m \tag{12}$$

where the w_m are non-negative and sum to 1, and $Z = \sum_{\mathbf{c}} \Pi_m Q_m^{w_m}(\mathbf{c})$. When all of the individual models are identical, $Z = 1$. Otherwise, Z is less than one and the difference between the two sides of 12 is $-\log Z$. This makes it clear that the benefit of combining experts comes from the fact that they make $\log Z$ small by disagreeing on unobserved data.

It is tempting to augment PoE's by giving each expert, m, an additional adaptive parameter, w_m, that scales its log probabilities. However, this makes inference much more difficult (Yee-Whye Teh, personal communication). Consider, for example, an expert with $w_m = 100$. This is equivalent to having 100 copies of an expert but with their latent states all tied together and this tying affects the inference. So it is easier just to fix $w_m = 1$ and allow the PoE learning algorithm to determine the appropriate sharpness of the expert.

12.2 Comparison with directed acyclic graphical models

Inference in a PoE is trivial because the experts are individually tractable and the product formulation ensures that the hidden states of different experts are conditionally independent given the data. This makes them relevant as models of biological perceptual systems, which must be able to do inference very rapidly. Alternative approaches based on directed acyclic graphical models suffer from the "explaining away" phenomenon. When such graphical models are densely connected exact inference is intractable, so it is necessary to resort to clever but implausibly slow iterative techniques for approximate inference (Saul and Jordan, 1998) or to use crude approximations that ignore explaining away during inference and rely on the learning algorithm to find representations for which the shoddy inference technique is not too damaging (Hinton, Dayan, Frey and Neal, 1995).

Unfortunately, the ease of inference in PoE's is balanced by the difficulty of generating fantasy data from the model. This can be done trivially in one ancestral pass in a directed acyclic graphical model but requires an iterative procedure such as Gibbs sampling in a PoE. If, however, Eq. 6 is used for learning, the difficulty of generating samples from the model is not a major problem.

In addition to the ease of inference that results from the conditional independence of the experts given the data, PoE's have a more subtle advantage over generative models that work by first choosing values for the latent variables and then generating a data vector from these latent values. If such a model has a single hidden layer and the latent variables have independent prior distributions, there will be a strong tendency for the posterior values of the latent variables to be approximately marginally independent after the model has been fitted to data[6]. For this reason, there has been little success with attempts to learn such generative models one hidden layer at a time in a greedy, bottom-up way. With PoE's, however, even though the experts have independent priors the latent variables in different experts will be marginally *dependent*: They can have high mutual information even for fantasy data generated by the PoE itself. So after the first hidden layer has been learned greedily there may still be lots of statistical structure in the latent variables for the second hidden layer to capture.

The most attractive property of a set of orthogonal basis functions is that it is possible to compute the coefficient on each basis function separately without worrying about the coefficients on other basis functions. A PoE retains this attractive property whilst allowing non-orthogonal experts and a non-linear generative model.

PoE's provide an efficient instantiation of the old psychological idea of analysis-by-synthesis. This idea never worked properly because the generative models were not selected to make the analysis easy. In a PoE, it is difficult to generate data from the generative model, but, given the model, it is easy to compute how any given data vector might have been generated and, as we have seen, it is relatively easy to learn the parameters of the generative model.

Acknowledgements

This research was funded by the Gatsby Foundation. Thanks to Zoubin Ghahramani, David MacKay, David Lowe, Yee-Whye Teh, Guy Mayraz, Andy Brown, and other members of the Gatsby unit for helpful discussions and to Peter Dayan for improving the manuscript and disproving some expensive speculations.

[6]This is easy to understand from a coding perspective in which the data is communicated by first specifying the states of the latent variables *under an independent prior* and then specifying the data given the latent states. If the latent states are not marginally independent this coding scheme is inefficient, so pressure towards coding efficiency creates pressure towards independence.

References

Berger, A., Della Pietra, S. and Della Pietra, V. (1996) A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, **22**.

Freund, Y. and Haussler, D. (1992) Unsupervised learning of distributions on binary vectors using two layer networks. *Advances in Neural Information Processing Systems 4.* J. E. Moody, S. J. Hanson and R. P. Lippmann (Eds.), Morgan Kaufmann: San Mateo, CA.

Genest, C. & Zidek, J. V. (1986) Combining probability distributions: A critique and an annotated bibliography. *Statistical Science* **1**, 114-148.

Heskes, T. (1998) Bias/Variance decompositions for likelihood-based estimators. *Neural Computation* **10**, 1425-1433.

Hinton, G., Dayan, P., Frey, B. & Neal, R. (1995) The wake-sleep algorithm for self-organizing neural networks. *Science*, **268**, 1158-1161.

Hinton, G. E. Sallans, B. and Ghahramani, Z. (1998) Hierarchical Communities of Experts. In M. I. Jordan (Ed.) *Learning in Graphical Models.* Kluwer Academic Press.

Hinton, G. E. & McClelland, J. L. (1988) Learning representations by recirculation. In D. Z. Anderson, editor, *Neural Information Processing Systems*, 358–366, American Institute of Physics: New York.

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

O'Reilly, R. C. (1996) Biologically Plausible Error-Driven Learning Using Local Activation Differences: The Generalized Recirculation Algorithm. /it Neural Computation, /bf 8, 895-938.

Revow, M., Williams, C. K. I. and Hinton, G. E. (1996) Using Generative Models for Handwritten Digit Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**, 592-606.

Saul, L. K., Jaakkola, T. & Jordan, M. I. (1996) Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, **4** 61-76.

Seung, H. S. (1998) Learning continuous attractors in a recurrent net. *Advances in Neural Information Processing Systems 10.* M. I. Jordan, M. J. Kearns, and S. A. Solla (Eds.) MIT Press, Cambridge Mass.

Smolensky, P. (1986) Information processing in dynamical systems: Foundations of harmony theory. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, MIT Press

Winston, P. H. (1975) Learning structural descriptions from examples. In Winston, P. H. (Ed.) *The Psychology of Computer Vision*, McGraw-Hill Book Company, New York.