Modelling High-Dimensional Data by Combining Simple Experts

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 non-linear probabilistic models of the same data by multiplying the probability distributions together and then renormalizing. A "product of experts" is a very efficient way to model data that simultaneously satisfies many different constraints. It is difficult to fit a product of experts to data using maximum likelihood because the gradient of the log likelihood is intractable, but there is an efficient way of optimizing a different objective function and this produces good models of high-dimensional data.

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 because it is easy to fit mixtures of tractable models to data using EM or gradient ascent and, 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 simple models each of which is tuned in the 35-dimensional manifold 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 manifold.

A very different way of combining distributions is to multiply them together and renormalize. 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.

The idea of combining the opinions of multiple different expert models by using a weighted average in the log probability domain (i.e. a product) 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 separately rather than training the experts cooperatively. This may be because fitting a PoE to data appears very difficult. 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.

Learning PoE's 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 is to compute 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 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.

[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.

Unfortunately, even if it is computationally feasible to approach equilibrium 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 strongly repelled from regions of high variance even if the gradient is zero. To understand this subtle but powerful 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.

Learning by maximizing contrastive likelihood

There is a simple and very effective alternative to maximum likelihood learning which eliminates almost all of the computation required to get samples from the equilibrium distribution and also eliminates almost all of the variance that masks the gradient signal. Instead of maximizing the log likelihood of the data, we maximize the *difference* between the log likelihood of the data vectors and the log likelihood of "one-step" reconstructions of the data vectors. This objective function will be called the "contrastive log likelihood". The reconstructions are generated by one full step of Gibbs sampling, which involves the following four stages:

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

Each expert is chosen to be tractable, so it is possible to compute the exact value of the first term on the RHS of Eq. 2 for both $\mathbf{d}$ and $\hat{\mathbf{d}}$. The second term on the RHS is the same for $\mathbf{d}$ and $\hat{\mathbf{d}}$. So if we ignore the fact that, unlike the data, the distribution of the reconstructions depends on the parameters of the experts, we get a simple rule for approximately following the gradient of the contrastive log likelihood:

$$\Delta\theta_m \propto \frac{\partial \log p_m(\mathbf{d}|\theta_m)}{\partial \theta_m} - \frac{\partial <\log p_m(\hat{\mathbf{d}}|\theta_m)>_{\hat{\mathbf{d}}}}{\partial \theta_m} \tag{3}$$

where the angle brackets denote expectations over the distribution of the one-step reconstructions of $\mathbf{d}$. 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. Changing θ_m changes the distribution of $\hat{\mathbf{d}}$ but this effect only makes a small contribution to the derivative of the contrastive log likelihood so the expected vector of changes in the parameters given by Eq. 3 almost always has a positive cosine with the true gradient of the contrastive log likelihood (see Hinton (1999) for details).

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.

The idea of simultaneously maximizing the log likelihood of the data and minimizing the log likelihood of the one-step reconstructions is central to this paper. Its main justification is that it is computationally easy and it works well in practice, but it also has a number of intuitive justifications.

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.

Perhaps the best intuitive justification comes from considering the convergence of the Gibbs sampling procedure towards the equilibrium distribution. To maximize the log likelihood of the observed data we need to minimize the Kullback-Liebler divergence between the data distribution and the marginal over the visible variables of the equilibrium distribution. If we already have a perfect model and we use the data distribution to initialize the Gibbs sampling, we will already be at equilibrium. So maybe we can find a good model by initializing the Gibbs sampling with the data, observing how one step of convergence to equilibrium causes us to diverge from the initial distribution, and adjusting the parameters to cancel out this divergence. The obvious way to reduce the tendency to wander away from the data distribution is to increase the log likelihood of the data vectors we start with and to decrease the log likelihood of the reconstructed data vectors. Because Gibbs sampling moves towards the equilibrium distribution of the model, the reconstructed vectors cannot, on average, be less likely under the model than the data vectors and they can only have the same average likelihood when the model is perfect (assuming that the Markov chain mixes).

Although these intuitive justifications are each somewhat vague, their product is sharper.

PoE's and Boltzmann machines

The Boltzmann machine learning algorithm (Hinton and Sejnowski, 1986) is easy to implement in hardware, but it is very slow in networks with interconnected hidden units. S-molensky (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[2]. 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:

$$\frac{\partial \log p_j(\mathbf{d}|\mathbf{w}_j)}{\partial w_{ij}} = <s_i s_j>^0 - <s_i s_j>^{j\infty} \tag{4}$$

where $\mathbf{w}_j$ is the vector of weights connecting hidden unit j to the visible units, $<s_i s_j>^0$ 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 $< s_i s_j >^{j\infty}$ is the expected value of $s_i s_j$ when alternating Gibbs sampling of the single hidden unit and the 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:

$$\sum_{\mathbf{c}} p(\mathbf{c}|\mathbf{w}) \frac{\partial \log p_j(\mathbf{c}|\mathbf{w}_j)}{\partial w_{ij}} = <s_i s_j>^{\infty} - <s_i s_j>^{j\infty} \tag{5}$$

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

where $\mathbf{w}$ is all of the weights in the RBM and $<s_i s_j>^{\infty}$ 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. 5 from Eq. 4 gives the gradient of the log likelihood of d:

$$\frac{\partial \log p(\mathbf{d}|\mathbf{w})}{\partial w_{ij}} = <s_i s_j>^0 - <s_i s_j>^{\infty} \tag{6}$$

The high sampling variance in $<s_i s_j>^{\infty}$ makes learning difficult. It is much more effective to follow the approximate gradient of the contrastive log likelihood. For an RBM this approximate gradient is particularly easy to compute:

$$\frac{\partial \log p(\mathbf{d}|\mathbf{w})}{\partial w_{ij}} - \frac{\partial <\log p(\hat{\mathbf{d}}|\mathbf{w})>_{\hat{\mathbf{d}}}}{\partial w_{ij}} \approx <s_i s_j>^0 - <s_i s_j>^1 \tag{7}$$

where $<s_i s_j>^1$ is the expected value of $s_i s_j$ when a one-step reconstruction of $\mathbf{d}$ is clamped on the visible units and s_j is sampled from its posterior given the reconstruction.

Learning the features of handwritten digits

When presented with real, high-dimensional data, a restricted Boltzmann machine trained to maximize the contrastive log likelihood using Eq. 7 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. 7 was modified to use probabilities in place of stochastic binary values for both the data and the hidden units. The binary values of the hidden units were still used for generating the one-step reconstructions.

It took two days in matlab on a second millenium workstation to perform 658 epochs of learning. In each epoch, the weights were updated 80 times using the approximate gradient of the contrastive log likelihood computed on mini-batches of size 100 that contained 10 exemplars of each digit class. To improve the learning speed a momentum method was used. Except for the first 10 epochs, the parameter updates 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 1.

Learning to discriminate handwritten digits

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

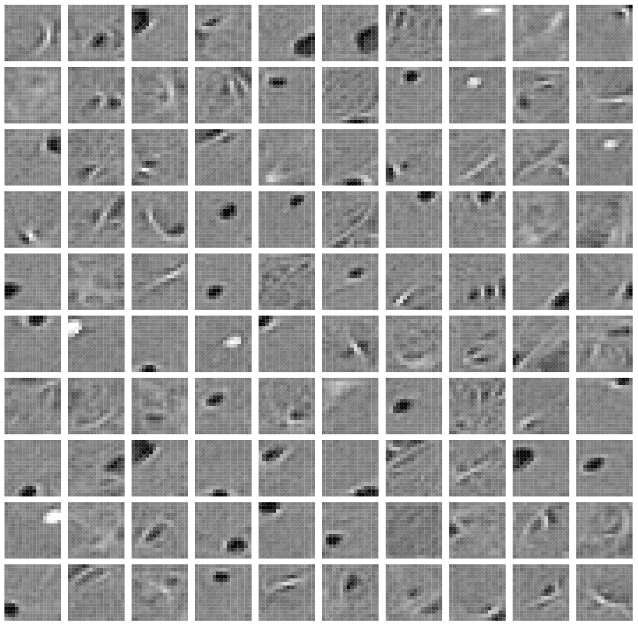

Figure 1: The receptive fields of a randomly selected subset of the 500 hidden units in a PoE trained on 8000 images of digits with equal numbers from each class. Each block shows the 256 learned weights connecting a hidden unit to the pixels. The scale goes from $+2$ (white) to -2 (black).

is the log of the denominator in Eq. 1. Unfortunately, it is 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 by summing the log probabilities that the PoE assigns to test data vectors.

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 2 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 3 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 4 shows the unnormalized log probability scores of some test images under a model trained on 825 images of the digit 7 and a model trained on 825 images of the digit 9. These two classes were chosen because they are the most difficult 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 mis-

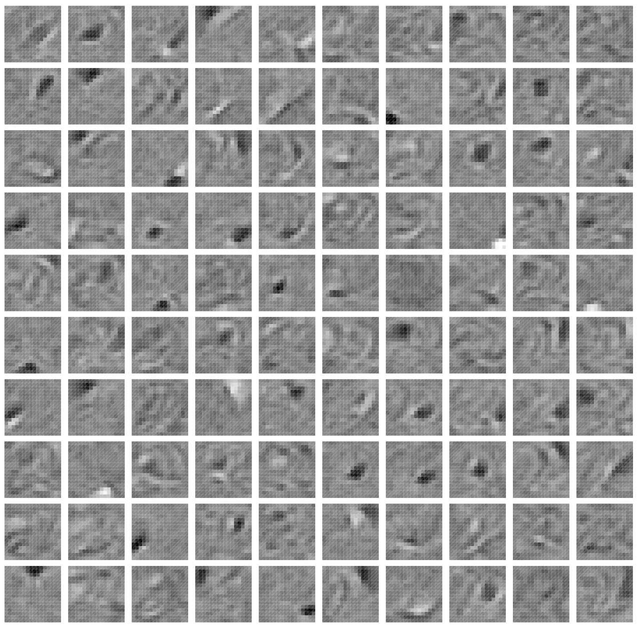

Figure 2: The weights learned by 100 hidden units trained on 16 x 16 images of the digit 2. The scale goes from +3 (white) to −3 (black). Note that the fields are mostly quite local. A local feature like the one in column 1 row 7 looks like an edge detector, but it is best understood as a local deformation 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.

classifications. 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 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.

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 the PoE's and converts them into a probability distribution across the 10 labels (see Hinton (1999) for details). This gives an error rate of 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.

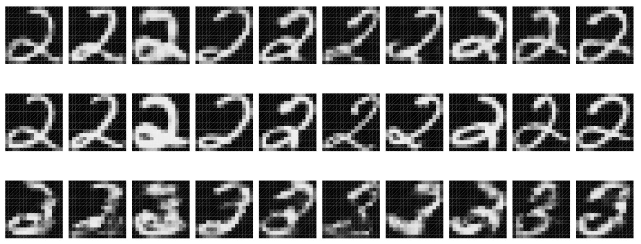

Figure 3: The center row is previously unseen images of 2's. The top row shows the pixel probabilities when the image is reconstructed from the binary activities of 100 feature detectors that have been trained on 2's. The bottom row shows the reconstruction probabilities using 100 feature detectors trained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

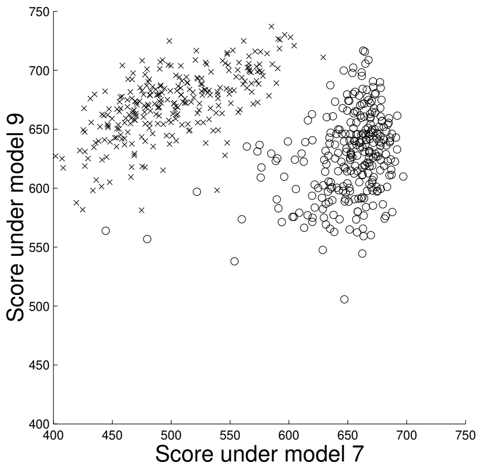

Figure 4: The unnormalised log probability scores of the previously unseen test images of 7's and 9's. Although the classes are not linearly separable, all the errors are close to the best separating line.

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. An interesting approach is to use 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. 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.

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. 3 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 by using the forward-backward algorithm in each expert separately to calculate the posterior probability distribution over paths through the hidden states, then stochastically selecting a hidden path in each expert from the posterior. At each time step in the reconstructed sequence, an output symbol or output vector is then chosen from the product of the output distributions specified by the hidden state selected for that time step in each HMM. If more realistic products of HMM's can be trained successfully by maximizing the contrastive likelihood, they should be far better than single HMM's for many different kinds of sequential data.

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).

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. 3 is used for learning, the difficulty of generating samples from the model does not impede learning.

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. If the generative model is causal, this attractive property can only be achieved by using basis functions that are orthogonal. A product of experts, however, retains this attractive property whilst allowing non-orthogonal experts and non-linear generative models.

Acknowledgements

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

References

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. E. (1999) Training Products of Experts by Maximizing Contrastive Likelihood. Technical Report: GCNU TR1999-001. Gatsby Computational Neuroscience Unit, University College London (www.gatsby.ucl.ac.uk).

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. & 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.

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.

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.