Recognizing Handwritten Digits Using Hierarchical Products of Experts

Guy Mayraz and Geoffrey E. Hinton

Abstract—The product of experts learning procedure [1] can discover a set of stochastic binary features that constitute a nonlinear generative model of handwritten images of digits. The quality of generative models learned in this way can be assessed by learning a separate model for each class of digit and then comparing the unnormalized probabilities of test images under the 10 different class-specific models. To improve discriminative performance, a hierarchy of separate models can be learned for each digit class. Each model in the hierarchy learns a layer of binary feature detectors that model the probability distribution of vectors of activity of feature detectors in the layer below. The models in the hierarchy are trained sequentially and each model uses a layer of binary feature detectors to learn a generative model of the patterns of feature activities in the preceding layer. After training, each layer of feature dectectors produces a separate, unnormalized log probabilty score. With three layers of feature detectors for each of the 10 digit classes, a test image produces 30 scores which can be used as inputs to a supervised, logistic classification network that is trained on separate data. On the MNIST database, our system is comparable with current state-of-the-art discriminative methods, demonstrating that the product of experts learning procedure can produce effective hierarchies of generative models of high-dimensional data.

Index Terms—Neural networks, products of experts, handwriting recognition, feature extraction, shape recognition, Boltzmann machines, model-based recognition, generative models.

1 Learning Products of Stochastic Binary Experts

Hinton [1] describes a learning algorithm for probabilistic generative models that are composed of a number of experts. Each expert specifies a probability distribution over the visible variables and the experts are combined by multiplying these distributions together and renormalizing.

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

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 c is an index over all possible vectors in the data space.

It is very difficult to generate data from the "Product of Experts" generative model defined by (1). One very inefficient approach, based on rejection sampling, is to allow each expert separately to choose values for its internal latent variables from their prior distributions and then to produce a data vector from these latent values. If all of the experts happen to exactly agree on which data vector to produce, that vector is produced as output. Otherwise, we try again. The fraction of times that we must try again is given by the intractable denominator on the RHS of (1). Fortunately, we do not want to use the model as a generator. The purpose of the generative model is to provide a basis for doing inference and learning. Having a clearly specified generative model allows us to infer the probability distribution over the states of the latent variables inside each expert when data is observed. It also allows us to specify how the parameters of each expert should be changed to maximize the probability that the generative model would produce the observed data. For inference and learning, the product of experts model can be very efficient, though, as we shall see, the learning needs to optimize a function which differs from the standard log likelihood of the data.

A Restricted Boltzmann Machine (RBM) [2], [3] is a special case of a product of experts in which each expert is a single, binary stochastic hidden unit that has symmetrical connections to a set of visible units and connections between the hidden units are forbidden. Inference in an RBM is much easier than in a general Boltzmann machine because there is no need to perform any iteration to determine the activities of the hidden units. The hidden states, s_j, are *conditionally independent* given the visible states, s_i, and the distribution of s_j is given by the standard logistic function:

$$p(s_j = 1) \ = \ \frac{1}{1 + \exp(-\sum_i w_{ij} s_i)} \ . \tag{2}$$

Conversely, the hidden states of an RBM are *marginally dependent*: If we average over data that is generated by the model, hidden units will typically have highly correlated activities. So, it is easy for an RBM to learn population codes in which the activities of units may be highly redundant, thus providing robustness against hardware failures. It is hard to learn redundant population codes in causal belief nets with one hidden layer because the generative model of a causal belief net assumes that the hidden units choose their activities independently (i.e., they are *marginally*

• *The authors were with Gatsby Computational Neuroscience Unit, University College London, 17 Queen Square, London WC1N 3AR, UK. E-mail: {gmz, hinton}@gatsby.ucl.ac.uk.*

Manuscript received 01 Aug. 2000; revised 13 Mar. 2001; accepted 01 May 2001.
Recommended for acceptance by T.K. Ho.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 112633.

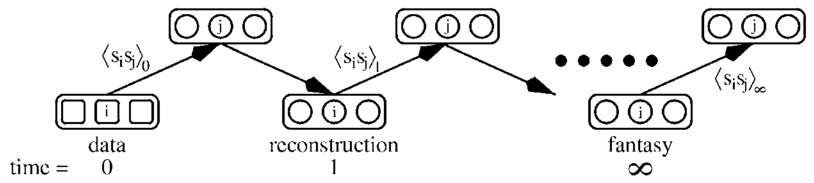
$\langle s_i s_j \rangle_0$
$\langle s_i s_j \rangle_1$
$\langle s_i s_j \rangle_\infty$
data
reconstruction
fantasy
time = 0
1
∞

Fig. 1. A visualization of alternating Gibbs sampling. At time 0, the visible variables represent a data vector and the hidden variables of all the experts are updated in parallel with samples from their posterior distribution given the visible variables. At time 1, the visible variables are all updated to produce a reconstruction of the original data vector from the hidden variables and then the hidden variables are again updated in parallel. If this process is repeated sufficiently often, it is possible to get arbitrarily close to the equilibrium distribution. Samples from this distribution are called "fantasies." The correlations $< s_i s_j >$ shown on the connections between visible and hidden variables are the statistics used for learning in Restricted Boltzmann Machines.

independent).[1] It is also much harder to infer the states of the hidden variables in a causal belief net becasue they are *conditionally dependent* given the observed data. For causal belief nets that are nonlinear and densely connected, this typically means that the hidden variables need to communicate iteratively to decide on their posterior distributions given the data.

An RBM can be trained using the standard Boltzmann machine learning algorithm which follows a noisy, but unbiased estimate of the gradient of the log likelihood of the data. One way to implement this algorithm is to start the network with a data vector on the visible units and then to alternate between updating all of the hidden units in parallel and updating all of the visible units in parallel (see Fig. 1). Each update picks a binary state for a unit from its posterior distribution given the current states of all the units in the other set. If this alternating Gibbs sampling is run to equilibrium, there is a very simple way to update the weights so as to minimize the Kullback-Leibler divergence, $P^0||P^\infty_\theta$, between the data distribution, P^0, and the equilibrium distribution over the visible units of samples from the generative model, P^∞_θ, produced by the RBM with parameters θ [4]:

$$\Delta w_{ij} \propto < s_i s_j >_{P^0} - < s_i s_j >_{P^\infty_\theta} , \tag{3}$$

where $< s_i s_j >_{P^0}$ is the expected value of $s_i s_j$ when data is clamped on the visible units and the hidden states are sampled from their conditional distribution given the data and $< s_i s_j >_{P^\infty_\theta}$ is the expected value of $s_i s_j$ after prolonged Gibbs sampling.

This learning rule does not work very well because it can take a long time to approach thermal equilibrium and the sampling noise in the estimate of $< s_i s_j >_{P^\infty_\theta}$ can swamp the gradient. It is far more effective to minimize the *difference* between $P^0||P^\infty_\theta$ and $P^1_\theta||P^\infty_\theta$, where P^1_θ is the distribution of the one-step reconstructions of the data that are produced by first picking binary hidden states from their conditional distribution given the data and then picking binary visible states from their conditional distribution given the hidden states. Using p_{θ_m} to denote a random variable representing the probability of a data vector under a model with parameters θ_m, the exact gradient of this "contrastive divergence" is

$$-\frac{\partial}{\partial \theta_m}\left(P^0||P^\infty_\theta - P^1_\theta||P^\infty_\theta\right) = \left\langle \frac{\partial \log p_{\theta_m}}{\partial \theta_m} \right\rangle_{P^0} - \left\langle \frac{\partial \log p_{\theta_m}}{\partial \theta_m} \right\rangle_{P^1_\theta} + \frac{\partial P^1_\theta}{\partial \theta_m} \frac{\partial (P^1_\theta||P^\infty_\theta)}{\partial P^1_\theta} \tag{4}$$

The exact gradient is complicated because the distribution P^1_θ depends on the weights, but [1] shows that the last term in (4) can safely be ignored to yield a learning rule for following the approximate gradient of the contrastive divergence. In an RBM, this learning rule is particularly simple:

$$\Delta w_{ij} \propto < s_i s_j >_{P^0} - < s_i s_j >_{P^1_\theta} \tag{5}$$

This learning rule is simpler to implement and considerably faster than the Boltzmann Machine learning rule. The learning scales up well to large networks. As we shall see, it can learn of the order of a million weights in about one day on a 500MHz pentium.

For images of digits, it is possible to apply (5) directly if we use stochastic binary pixel intensities, but it is more effective to normalize the intensities to lie in the range $[0, 1]$ and then to use these real values as the inputs to the hidden units. During reconstruction, the stochastic binary pixel intensities, s_i required by (5) are also replaced by real-valued probabilities, p_i. Finally, the learning rule can be made less noisy by replacing the stochastic binary activities of the hidden units by their expected values. So, the learning rule we actually use is

$$\Delta w_{ij} \propto < p_i p_j >_{P^0} - < p_i p_j >_{P^1_\theta} . \tag{6}$$

Stochastically chosen binary states of the hidden units are still used for computing the probabilities of the reconstructed pixels, so the hidden probabilities cannot be used to convey an unbounded amount of information to the reconstruction.

The rest of this paper shows how RBM's trained to minimize contrastive divergence can be used for handwritten digit recognition. It is possible to achieve very good discrimination on a standard test set by using a three-layer hierarchy of models for each digit class and by basing the final decision on a small supervised network that learns how to combine unnormalized log probability scores that are produced by all three layers in the hierarchy for each of the 10 digit classes.

1. The activities of hidden units are also chosen independently in a Product of Experts, but all the rejected attempts in which the experts fail to agree on the data vector mean that the hidden states are typically highly dependent in the small subset of successful attempts.

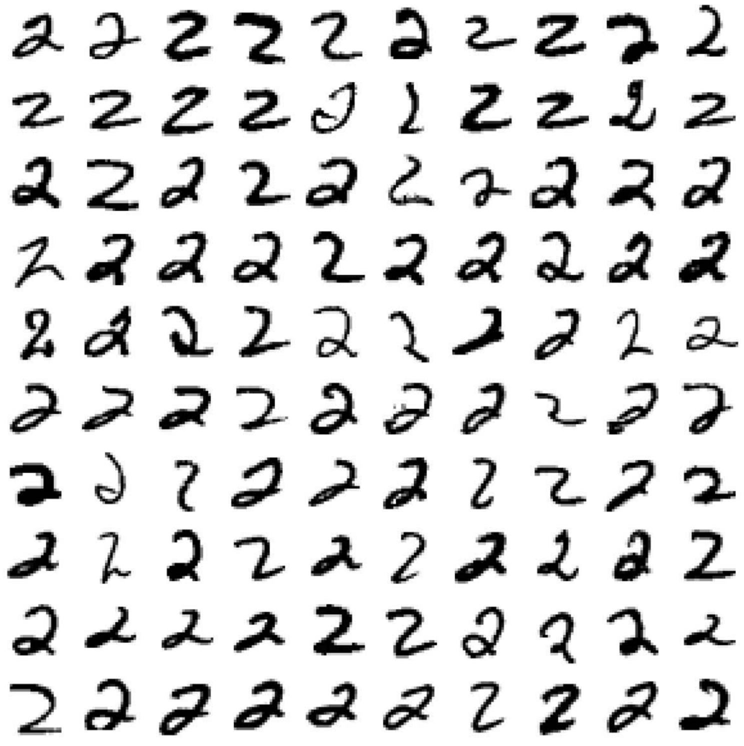

Fig. 2. Examples of 2s in the MNIST database (first 100 examples). Note, the very diverse handwriting styles.

2 The MNIST Database

Handwritten digit recognition is a convenient and important subproblem in optical character recognition (OCR). It has been used as a test case for theories of human or artificial pattern recognition since the days of the perceptron learning procedure. During the long history of research, several standard databases have emerged in which the handwritten digits are presegmented and approximately normalized so that researchers can compare recognition results without worrying about pre or post-processing. The freely available MNIST database of handwritten digits is now a standard for testing digit recognition algorithms. MNIST was constructed by Y. Le Cun of AT&T Labs out of the NIST database. There are 60,000 training images and 10,000 test images which are drawn from the same distribution as the training set. Images are size-normalized and translated so that the center of gravity of their intensity lies at the center of a fixed-size image of 28 by 28 pixels. Fig. 2 shows some examples of MNIST digits. The MNIST database can be found by searching for "mnist" using www.google.com.

A number of well-known learning algorithms have been run on the MNIST database [5], so it is easy to assess the relative performance of a novel algorithm. The main contenders include standard backpropagation, Support Vector Machines (SVMs), and Le Cun's own Le Net, which uses backpropagation training in a highly structured, multilayer network that has local receptive fields and averages the responses of feature detectors that have similarly located receptive fields and shared weights. The results in Table 1 show Le Net as the best algorithm, closely followed by SVM. Standard backpropagation without the domain-specific modifications used in Le Net does considerably worse but is still much better than simple statistical methods such as K-nearest-neighbors or a linear classifier. However, the results in Table 1 should be treated with caution. Some attempts to replicate the SVM results have produced slightly higher error rates of around 1.4 percent [6] and standard backpropagation can be carefully tuned to achieve under 2 percent (John Platt, personal communication).

Table 1 shows that it is possible to achieve a result that is comparable with the best discriminative techniques by using hierarchical PoE models of each digit class to extract scores that represent unnormalized log probabilities. These scores are then used as the inputs to a simple logistic classifier. The rest of this paper describes this system in detail.

Some of the experiments in [5] included deskewing images by computing the principal axis of the shape that is closest to the vertical and then transforming the image to make this principal axis vertical. In other experiments, the training set was augmented with distorted versions of the original training images. The distortions were small affine transformations of the digit or changes in the stroke thickness. Deskewing and distortions improve the performance of all methods. We did not use deskewing or distortions in our main experiments and, so, we only compare our results with other methods that did not use them. We return to this issue at the end of the paper.

3 Training the Individual PoE Models

The MNIST database contains an average of 6,000 training examples per digit, but these examples are unevenly distributed among the digit classes. In order to simplify

TABLE 1
Performance of Various Learning Methods on the MNIST Test Set

METHOD	% ERRORS
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
1000 RBF + linear classifier	3.6
Best Back-Prop: 3-layer NN, 500+150 hidden units	2.95
Reduced Set SVM deg 5 polynomial	1.0
LeNet-1 [with 16x16 input]	1.7
LeNet-5	0.95
Product of Experts (separate 3-layer net for each model)	**1.7**

the research, we produced a balanced database by using only 5,400 examples of each digit. The first 4,400 examples were the *unsupervised training set* used for training the individual PoE models. The remaining training examples of each of the 10 digits constituted the *supervised training set* used for training the logistic classification net that converts the scores of all the PoE models into a classification. The supervised training set was further subdivided into two equal halves and the second half was used as a validation set to determine the best PoE network size and the stopping point when training the discrimination network. Only then was the discrimination network retrained on the entire supervised training set and tested on the official test set.

The original intensity range in the MNIST images was 0 to 255. This was normalized to the range 0 to 1 so that we could treat intensities as probabilities. The normalized pixel intensities were used as the initial activities of the 784 visible units corresponding to the 28 by 28 pixels. The visible units were fully connected to a single layer of hidden units. The weights between the input and hidden layer were initialized to small, zero-mean, Gaussian-distributed, random values. The 4,400 training examples were divided into 44 minibatches. One epoch of learning consisted of a pass through all 44 minibatches in fixed order with the weights being updated after each minibatch. We used a momentum method with a small amount of weight decay, so the change in a weight after the tth minibatch was

$$\Delta w_{ij}^t = \mu \Delta w_{ij}^{t-1} + 0.1\Big(\langle p_i p_j \rangle_{Q_t^0} - \langle p_i p_j \rangle_{Q_t^1} - 0.0001 w_{ij}^t\Big), \quad (7)$$

where Q_t^0 and Q_t^1 are averages over the data or the one-step reconstructions for minibatch t and the momentum, μ, was 0 for the first 50 weight changes and 0.9 thereafter. The hidden and visible biases were initialized to zero. Their values were similarly altered (by treating them like connections to a unit that was always on) but with no weight decay.

After testing different sized networks on the validation set we determined that the largest network was the best, even though each digit model contains 392,500 parameters trained on only 4,400 images. The receptive fields learned by the hidden units are quite local (see Fig. 3). Since the hidden units are fully connected and have random initial weights, the learning procedure must infer the spatial proximity of pixels from the statistics of their joint activities. Fig. 4 shows the mean goodness scores of all 10 models on all 10 digit classes.

Fig. 5 shows reconstructions produced by models on previously unseen data from the digit class they were trained on and also on data from a different digit class. With 500 hidden units, the 7s model is almost perfect at reconstructing 9s. This is because a model gets better at reconstructing more or less any image as its set of available features becomes more varied and more local. Despite this, the larger networks give better discriminative information.

3.1 Multilayer Models

Networks that use a single layer of hidden units and do not allow connections within a layer have some major advantages over more general networks. With an image clamped on the visible units, the hidden units are conditionally independent. So, it is possible to compute an unbiased sample of the binary states of the hidden units without any iteration. This property makes PoEs easy to train and it is lost in more general architectures. If, for example, we introduce a second hidden layer that is symmetrically connected to the first hidden layer, it is no longer straightforward to compute the posterior expected activity of a unit in the first hidden layer when given an image that is assumed to have been generated by the multilayer model at thermal equilibrium. The posterior distribution can be

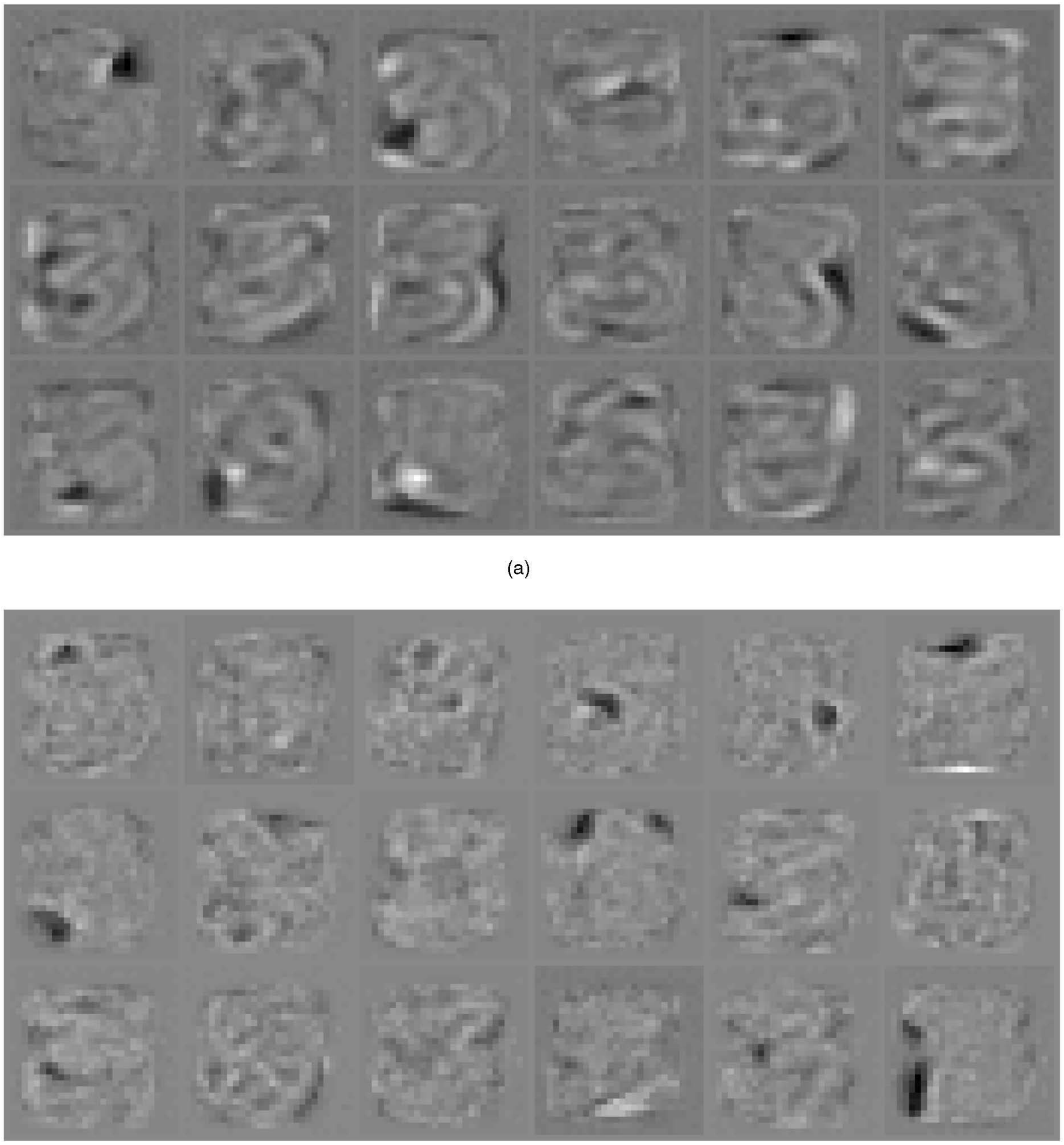

Fig. 3. First layer receptive fields of the PoE model trained on samples of 3s from the MNIST database with (a) 100 units trained for 100 epochs and (b) 500 units trained for 500 epochs. The receptive fields are for 18 hidden units chosen at random. The gray levels represent the values of the weights with white being positive and black negative. The most extreme weights in (a) and (b) have values of -4.6 and -4.5, respectively.

computed by alternating Gibbs sampling between the two hidden layers, but this is slow and noisy.

Fortunately, if our ultimate goal is discrimination, there is a computationally convenient alternative to using a multilayer Boltzmann machine. Having trained a one-hidden-layer PoE on a set of images, it is easy to compute the expected activities of the hidden units on each image in the training set. These hidden activity vectors will themselves have interesting statistical structure because a PoE is not attempting to find independent causes and has no implicit penalty for using hidden units that are marginally highly correlated. So, we can learn a completely separate PoE model in which the activity vectors of the hidden units are treated as the observed data and a new layer of hidden units learns to model the structure of this "data." It is not entirely clear how this second level PoE model helps as a way of modeling the original image distribution, but it is clear that, if a PoE is trained on images of 2s, we would expect the vectors of hidden activities to be very different when it is presented with a 3, even if the features it has learned are quite good at reconstructing the 3. So, a second level model should be able to assign high scores to the vectors of hidden activities that are typical of the 2 model when it is given images of 2s and low scores to the hidden

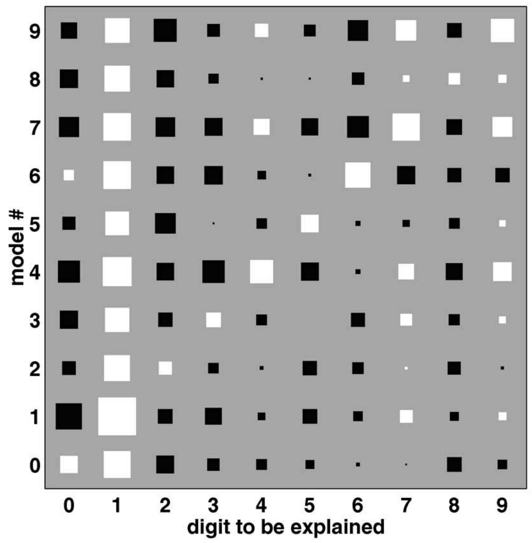

Fig. 4. The mean goodness of validation set digits using the first hidden layer of the 500 unit models. A different constant is added to all the goodness scores of each model so that rows sum to zero. White squares represent positive values, black squares represent negative values, and the area of a square represents the absolute value. Successful discrimination depends on models being better on their own class than other models are. The converse is not true: Models can be better reconstructing other easier classes of digits than their own class.

activities of the 2 model when it is given images that contain combinations of features that are not normally present at the same time in a 2.

We used a three-layer hierarchy of hidden features in each digit model.[2] The layers were trained sequentially and, to simplify the research, we always used the same number of hidden units in each layer. We trained models of five different sizes with 25, 100, 200, 400, and 500 units per layer.

4 The Logistic Classification Network

An attractive aspect of PoEs is that it is easy to compute the numerator in (1) so it is easy to compute a goodness score which is equal to the log probability of a data vector up to an additive constant. Fig. 6 shows the goodness of the 7s and 9s models (the most difficult pair of digits to discriminate) when presented with test images of both 7s and 9s. It can be seen that a line can be passed that separates the two digit sets almost perfectly. It is also encouraging that all of the errors are close to the decision boundary, so there are no confident misclassifications.

The classification network had 10 output units, each of which computed a total input, x, that was a linear function of the goodness scores, g, of the various PoE models, m, on an image, c. The probability assigned to class j was then computed by taking a "softmax" of the total inputs:

$$p_j^c = \frac{e^{x_j^c}}{\sum_k e^{x_k^c}} \qquad x_j^c = b_j + \sum_m g_m^c w_{mj} \ . \tag{8}$$

2. Strictly speaking, each layer of hidden features is a separate generative model of the activities in the layer below but it is more convenient to describe all three hidden layers as a single multilayer model that produces three goodness scores, one per hidden layer.

There were 10 PoE models with three layers each, so the classification network had 30 inputs and, therefore, 300 weights and 10 output biases. Both weights and biases were initialized to zero. The weights were learned by a momentum version of gradient ascent in the log probability assigned to the correct class. Since there were only 310 weights to train, little effort was devoted to making the learning efficient.

$$\Delta w_{mj}(t) \ = \ \mu \Delta w_{mj}(t-1) + 0.0002 \sum_c g_m^c (t_j^c - p_j^c) \ , \tag{9}$$

where t_j^c is 1 if class j is the correct answer for training case c and 0 otherwise. The momentum μ was 0.9. The biases were treated as if they were weights from an input that always had a value of 1 and were learned in exactly the same way.

In each training epoch, the weight changes were averaged over the whole supervised training set.[3] We used separate data for training the classification network because we expect the goodness score produced by a PoE of a given class to be worse and more variable on exemplars of that class that were not used to train the PoE and it is these poor and noisy scores that are relevant for the real, unseen test data.

The training algorithm was run using goodness scores from PoE networks with different numbers of hidden units. The results in Table 2 show a consistent improvement in classification error as the number of units in the hidden layers of each PoE increase. There is no evidence for over-fitting, even though large PoEs are very good at reconstructing images of other digit classes. It is possible to reduce the error rate by a further 0.1 percent by averaging together the goodness scores of corresponding layers of all the networks with 100 or more units per layer, but this model averaging is not nearly as effective as using extra layers.

Fig. 7 shows the images from the MNIST test set that were misclassified by the PoE network. Some images are impossible to categorize with confidence, but the network also makes errors on many images that are easy for people.

5 Training Time

Taking into account the 10 digit classes and three hidden layers, the complete system that uses PoEs with 500 hidden units in each layer contains 8,942,840 parameters in the 30 PoEs that are trained nondiscriminatively[4] and only 310 in the logistic classification net that is trained discriminatively. The 8,942,840 parameters of the PoE models are trained on only 44,000 images and the total training time is less than two weeks in Matlab on a 500MHz pentium II. If we regard each hidden layer as a separate model, no one model has more than 393,284 parameters and each model is only trained on 4,400 images or hidden activity vectors. Moreover, many different models can be trained in parallel on different

3. We held back part of the supervised training set to use as a validation set in determining the optimal number of epochs to train the classifiaction net, but once this was decided we retrained on all the supervised training data for that number of epochs.

4. This is about the number of synapses in 0.02 mm^3 of mouse cortex.

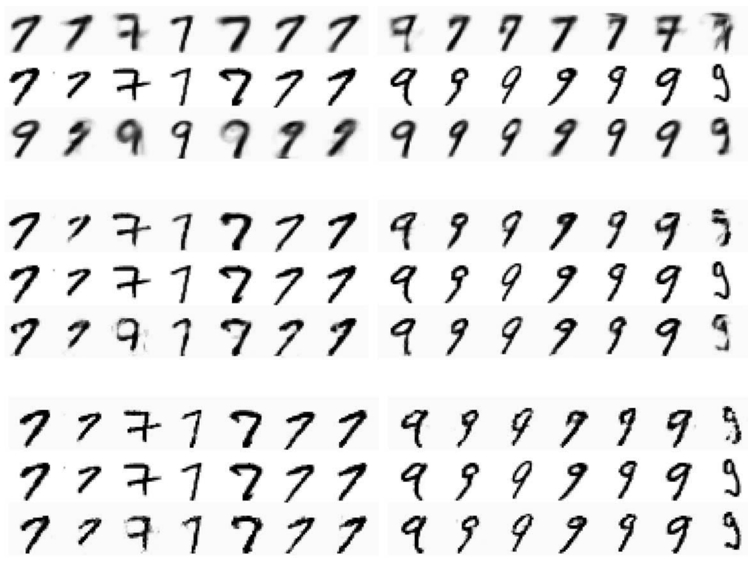

Fig. 5. Cross reconstructions of 7s and 9s with networks of 25, 100, and 500 units (top to bottom). The central horizontal line in each block contains originals and the lines above and below are reconstructions by the 7s and 9s models, respectively. Both models produce stereotyped digits in the small net and much better reconstructions in the larger ones for both the digit classes. There are also some examples of the 9s model trying to close the loop in 7s and the 7s model to open the loop in examples of 9s.

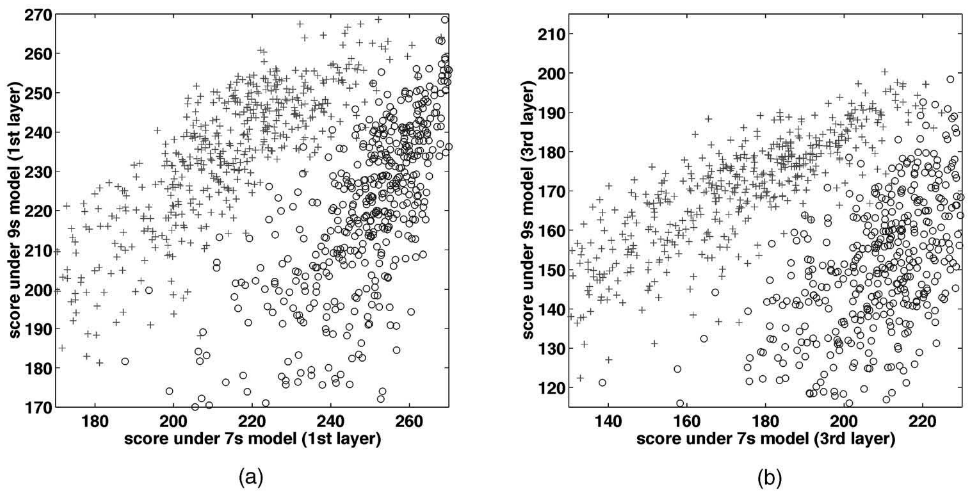

Fig. 6. Validation set cross goodness results of the first (a) and third (b) layers in 7s and 9s models with 500 units per layer. Higher layers clearly contribute significant discriminative information.

machines, though successive hidden layers for one digit class must be trained sequentially.

6 Shared Preprocessing

One gross simplification in our current system is that the density models for each digit class are totally separate and do not make any use of shared preprocessing. An optimistic approach is to have one big density model for all digit classes and to hope that hidden units in the higher layers become very selective for specific classes or subsets of the classes. Preliminary experiments suggest that higher-level hidden units do become moderately class-specific, but not enough to provide a really good classification. Separate experiments performed on the smaller USPS digit database show that, if the hidden activities of a one-hidden-layer PoE model trained on all the digit classes are used as inputs to a logistic discrimination net, the error rate is almost twice the rate achieved by using the scores provided by separate PoE models of each class.

A more promising approach to feature sharing is to train a single density model, but using 10 additional visible units. During training, exactly one of these units is initially turned on to represent the correct class label.[5] During one-step reconstruction, the softmax function is used to ensure that the activities of the 10 extra visible units sum to 1. Curiously, the learning algorithm is completely unaffected by this use of the softmax function to constrain the activities of a subset of the binary units and it does not even need to know about it. During testing, the goodness score is

5. This sounds remarkably like supervised training, but notice that the label is treated in just the same way as the pixels and almost all the capacity of the network is used for reconstructing the pixels because the network is learning a joint density model and there are a lot more pixels than classes.

TABLE 2
MNIST Test Set Error Rate as a Function of the Number of Hidden Units per Layer

Network size	Learning epochs	% Errors
25	25	3.8
100	100	2.3
200	200	2.2
400	200	2.0
500	500	1.7

computed with each of the "label" units clamped on in turn. This requires only slightly more computation than computing the goodness for the image alone because the total effect of the other visible units on the hidden features does not change. We have tried this approach and preliminary results are disappointing. The error rates are much greater than 1.7 percent. Also, keeping all the training examples from all 10 classes takes up a lot of memory and the computation is much less easy to parallelize the computation than in our current system.

7 Model-Based Normalization

The results of our current system are still not nearly as good as human performance. In particular, it appears the network has only a very limited understanding of image invariances. This is not surprising since it is trained on prenormalized data. Dealing with image invariances better will be essential for approaching human performance. The fact that we are using generative models suggests an interesting way of refining the image normalization. If the normalization of an image is slightly wrong, we would expect it to have lower probability under the correct class-specific model. So, we should be able to use the goodness score as an objective function for comparing many slightly different normalizations. As an initial experiment, we restricted ourselves to comparing five possible normalizations of the image: the given normalization and the four possible translations by a single pixel horizontally or vertically. For each generative model, we allow it to select whichever of these five normalizations gives the best goodness score. Obviously, this means that the goodness scores can only improve and the question is whether they improve more for the right model than for the wrong ones, particularly in marginal cases. A preliminary experiment using only the two most confusable classes (7s and 9s) showed that this reduces the error rate by 30 percent. Model-based deskewing should make a similar

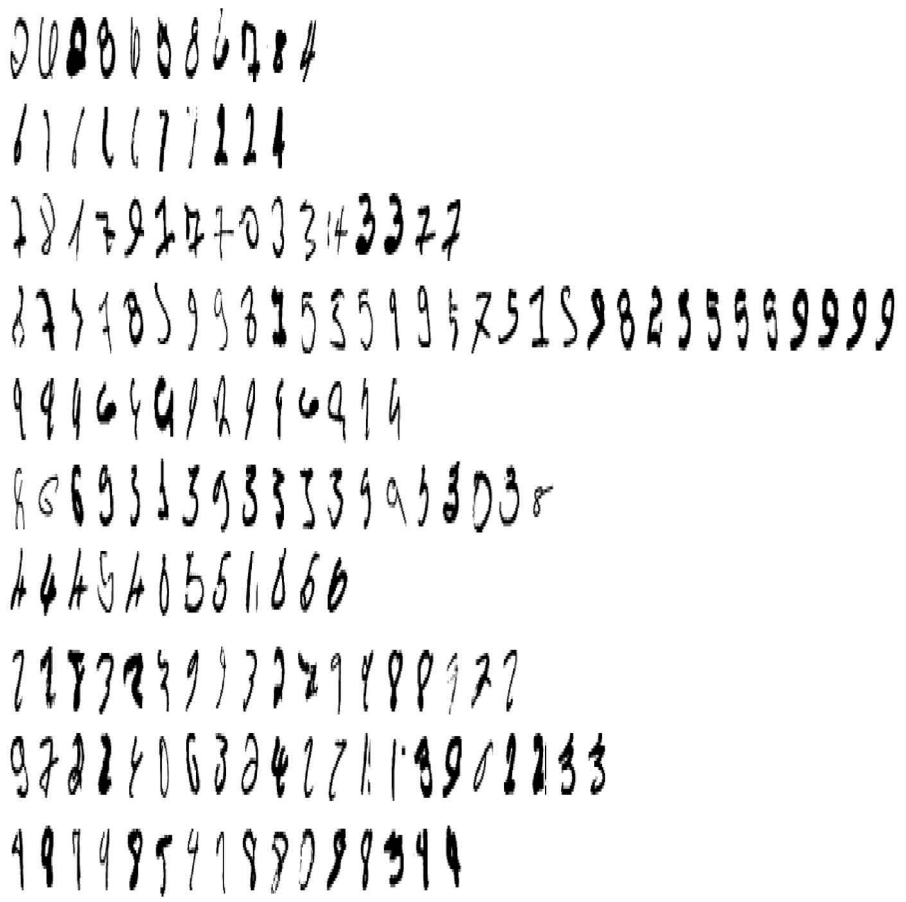

Fig. 7. The images in the MNIST test set that were misclassified by the combined network that was trained on all the validation data. All the images in the top row were misclassified as 0, all the images in the second row were misclassified as 1, etc.

improvement. It should be even more effective to search for the best normalization by taking the easily-computed gradient of the goodness with regard to the image intensities and projecting this gradient vector into the tangent space of affine image transformations [7].

8 Discussion

The main result is that it is possible to achieve discriminative performance comparable with state-of-the-art methods using a system in which nearly all of the work is done by separate density models of each digit class. This does not prove that the density models are actually good models of the digit classes but it strongly suggests it.

The way in which the individual experts are combined in a Product of Experts generative model has strong similarities to the way in which conditional density models are combined in approaches like stacking [8], bagging [9], or boosting [10]. For regression tasks in which a single real-valued output must be predicted from an input vector, it is possible to learn a number of different models, each of which is trained on different, or differently weighted, subsets of the training data. The predictions of all the experts are then combined by using a weighted average. This is equivalent to assuming that each model outputs the mean of a Gaussian distribution and the weights used for the averaging are the inverse variances of the Gaussians. The weighted average is then the mean of the product of the Gaussians [11]. This is quite different to a mixture of experts [12] which combines the Gaussian distributions of each expert using addition rather than multiplication.

Acknowledgments

The authors would like to thank Y. Le Cun, M. Revow, and members of the Gatsby Computational Neuroscience Unit for their helpful discussions. This research was funded by the Gatsby Charitable Foundation.

References

[1] G.E. Hinton, "Training Products of Experts by Minimizing Contrastive Divergence," Technical Report GCNU TR 2000-004, Gatsby Computational Neuroscience Unit, Univ. College London, 2000.
[2] P. Smolensky, "Information Processing in Dynamical Systems: Foundations of Harmony Theory," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* D.E. Rumelhart and J.L. McClelland, eds., vol. 1, 1986.
[3] Y. Freund and D. Haussler, "Unsupervised Learning of Distributions of Binary Vectors Using 2-Layer Networks," *Advances in Neural Information Processing Systems,* J.E. Moody, S.J. Hanson, and R.P. Lippmann, eds., vol. 4, pp. 912-919, 1992.
[4] G.E. Hinton and T.J. Sejnowski, "Learning and Relearning in Boltzmann Machines," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* D.E. Rumelhart and J.L. McClelland, eds., vol. 1, 1986.
[5] Y. LeCun, L.D. Jackel, L. Bottou, A. Brunot, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Comparison of Learning Algorithms for Handwritten Digit Recognition," *Proc. Int'l Conf. Artificial Neural Networks,* pp. 53-60, 1995.
[6] C.J.C. Burges and B. Schölkopf, "Improving the Accuracy and Speed of Support Vector Machines," *Advances in Neural Information Processing Systems,* M.C. Mozer, M.I. Jordan, and T. Petsche, eds., vol. 9, p. 375, 1997.
[7] P. Simard, Y. LeCun, J. Denker, and B. Victorri, "An Efficient Algorithm for Learning Invariances in Adaptive Classifiers," *Proc. Int'l Conf. Pattern Recognition (IAPR '92),* 1992.
[8] D. Wolpert, "Stacked Generalization," *Neural Networks,* vol. 5, pp. 241-259, 1992.
[9] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 26, pp. 123-140, 1996.
[10] R.S. Zemel and T. Pitassi, "A Gradient-Based Boosting Algorithm for Regression Problems," *Advances in Neural Information Processing Systems,* V. Tresp, T. Leen, and T. Dietterich, eds., vol. 13, 2001.
[11] T. Heskes, "Bias/Variance Decompositions for Likelihood-Based Estimators," *Neural Computation,* vol. 10, pp. 1425-1433, 1998.
[12] R. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation,* vol. 3, pp. 79-87, 1991.

Guy Mayraz received the BSc (Hons.) degree in mathematics and physics (Talpiot Programme) from the Hebrew University, the MSc (Hons.) degree in computer science from Tel-Aviv University, and he studied for the PhD degree at the Gatsby Computational Neuroscience Unit, University College London, working on unsupervised learning in vision under the guidance of Dr. Geoffrey Hinton. He is a founder of Linkadoo Communications, a software company where he now works.

Geoffrey E. Hinton received the BA degree in experimental psychology from Cambridge in 1970 and the PhD degree in artificial intelligence from Edinburgh in 1978. He was a member of the PDP group at the University of California, San Diego, an assistant professor at Carnegie-Mellon University, and a professor at the University of Toronto. He was the director of the Gatsby Computational Neuroscience Unit at the University College, London, and has now returned to the University of Toronto. He is a fellow of the Royal Society of London and the Royal Society of Canada, and a former president of the Cognitive Science Society. He has received the IEEE Neural Networks Pioneer award and the David E. Rumelhart prize. He does research on ways of using neural networks for learning, memory, perception, and symbol processing, and has more than 150 publications in these areas.