Handwritten Digit Recognition Using a Hierarchical Bayesian Network

Jonathan Mugan
November 30, 2005

Abstract

Probabilistic methods provide one set of tools for building perceptual systems. Lee and Mumford [2003] have recently put forth a Bayesian model of visual perception that is based on electrophysiological observations of the early visual neurons in monkeys. For this project a simplified version of the Lee and Mumford model has been implemented in Matlab and tested on the MNIST dataset of handwritten digits. This implementation performed poorly on the dataset relative to other methods, but it shows promise in its ability to scale to larger images. For comparison, nearest neighbor and recognition using the eigenface approach were also implemented.

1 Introduction

1.1 An Ideal Observer Perception Model

The "Bayesian coding hypothesis" [Knill and Pouget, 2004] states that the human brain uses probabilities and probability distributions to represent sensory information. In Bayesian statistical decision theory, decisions that minimize Bayes error are referred to as Bayes optimal. Within the study of perception, such Bayes optimal observers are referred to as "ideal observers" [Geisler and Diehl, 2003]. Lee and Mumford [2003] have recently proposed an ideal-observer model of the visual cortex. Their model has the structure of a hierarchical Bayesian network, with each node performing Bayesian inference based on evidence from the nodes lower in the hierarchy and background information from nodes higher in the hierarchy. This inference at each node is represented by the equation

$$P(\omega_i|x_O, x_B) = \frac{P(x_O|\omega_i, x_B)P(\omega_i|x_B)}{P(x_O|x_B)} \quad (1)$$

where ω_i is the state of nature (or in this case the feature produced by this node), x_O is the observation that comes from the nodes below, and x_B is the background or context information that comes from higher in the the hierarchy. For the leaf nodes, x_O comes directly from the environment (or in the case of the human visual system the lateral geniculate nucleus (LGN)), and for intermediate nodes x_O comes from the level below.

1.2 Summary of the Implementation

For this project a simplified version of the Lee and Mumford model was implemented in Matlab and tested on the MNIST handwritten digit dataset [LeCun and Cortes, 1998]. The records in this dataset consist of greyscale images of size 28×28; an example digit is shown in Figure 1. The results were somewhat disappointing. It was able to correctly identify the handwritten digit about 44% of the time. Both the nearest neighbor and the eigenface method that were implemented for this project were able to correctly identify the handwritten digit better than 90% of the time, and the current best methods using specialized neural networks and support vector machines are able to classify with greater than 99% accuracy [LeCun *et al.*, 1998].

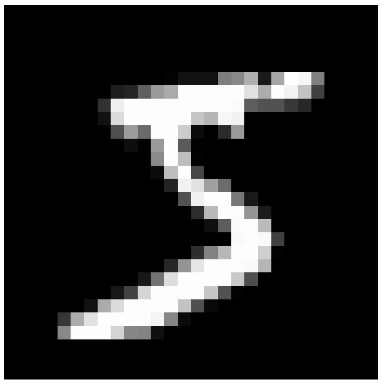

Figure 1: An example handwritten digit from the MNIST dataset.

1.3 The Goal of this Implementation

The main objective of this project was not to accurately classify handwritten digits, but rather to lay a groundwork for future research in feature acquisition. Of interest is a model of feature learning that accounts for various phenomena of common experience, such as the idea that features are not always noticed immediately, but once they are noticed they are always seen, and the idea that experts in a domain notice more features than non-experts.

One possibility is that analogous phenomena can be represented in a hierarchical generative model by incrementally adding discriminative ability to nodes as experience is gained. If vector quantization is used throughout the hierarchy, then every node has a finite set of states it can take, these states could then be the features available for discrimination. The common experience of suddenly noticing a new feature in the environment could correspond to adding a new code vector to some node in the hierarchy. As humans, we notice more features as we gain more experience, and this may account for why experts can make finer distinctions than non-experts. Additionally, humans can also add distinctions if pointed out by someone else. In the proposed model, when a code vector is added to a node it is there permanently and from then on it can be identified in the scene and its probabilities tracked.

An interesting possible research direction then is to devise a principled way for adding code vectors to the hierarchy. The difficulty with feature leaning is that there are an infinite number of possible features, the challenge then is to extract the ones that are in some sense "useful."

2 The Learning Algorithm

2.1 The Network Structure

The Bayesian network used for this project consists of four levels $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, and $\mathcal{I}$, as is shown in Figure 2. Level $\mathcal{I}$ contains 16 vector-valued random evidence variables. The handwritten digit images in the MNIST dataset are of size 28×28, and when the network is presented with a handwritten digit image, the image is copied to level $\mathcal{I}$ such that each $I_{ij} \in \mathcal{I}$ is a vector of size $7 \times 7 = 49$ and $\bigcup_{ij} I_{ij}$ covers the image with no overlap.

Level $\mathcal{C}$ contains 16 scalar random variables. The value of each $c_{ij} \in \mathcal{C}$ is a code vector index that approximates the received vector I_{ij}. Similarly, level $\mathcal{B}$ contains 4 scalar random variables. The value of each $b_i \in \mathcal{B}$ is a code vector index that approximates the vector formed by the received values of its children $\langle c_{i1}, c_{i2}, c_{i3}, c_{i4} \rangle$.

Level $\mathcal{A}$ contains a vector-valued random variable a of size 10 that represents the belief distribution of the correct identity of the current handwritten digit input.

2.2 The Learning Algorithm

The network is trained in stages from the bottom up. No learning occurs at level $\mathcal{I}$ as nodes at that level only partition the image and send it up to level $\mathcal{C}$.

$$\begin{tikzcd}[column sep=small]
 & & & & & & & & a & & & & & & & \\
 & & b_1 & & & & b_2 & & & b_3 & & & & b_4 & & \\
c_{11} & c_{12} & c_{13} & c_{14} & c_{21} & c_{22} & c_{23} & c_{24} & c_{31} & c_{32} & c_{33} & c_{34} & c_{41} & c_{42} & c_{43} & c_{44} \\
I_{11} & I_{12} & I_{13} & I_{14} & I_{21} & I_{22} & I_{23} & I_{24} & I_{31} & I_{32} & I_{33} & I_{34} & I_{41} & I_{42} & I_{43} & I_{44}
\arrow[from=1-9, to=2-3, leftrightarrow]
\arrow[from=1-9, to=2-7, leftrightarrow]
\arrow[from=1-9, to=2-10, leftrightarrow]
\arrow[from=1-9, to=2-14, leftrightarrow]
\arrow[from=2-3, to=3-1, leftrightarrow]
\arrow[from=2-3, to=3-2, leftrightarrow]
\arrow[from=2-3, to=3-3, leftrightarrow]
\arrow[from=2-3, to=3-4, leftrightarrow]
\arrow[from=2-7, to=3-5, leftrightarrow]
\arrow[from=2-7, to=3-6, leftrightarrow]
\arrow[from=2-7, to=3-7, leftrightarrow]
\arrow[from=2-7, to=3-8, leftrightarrow]
\arrow[from=2-10, to=3-9, leftrightarrow]
\arrow[from=2-10, to=3-10, leftrightarrow]
\arrow[from=2-10, to=3-11, leftrightarrow]
\arrow[from=2-10, to=3-12, leftrightarrow]
\arrow[from=2-14, to=3-13, leftrightarrow]
\arrow[from=2-14, to=3-14, leftrightarrow]
\arrow[from=2-14, to=3-15, leftrightarrow]
\arrow[from=2-14, to=3-16, leftrightarrow]
\arrow[from=4-1, to=3-1]
\arrow[from=4-2, to=3-2]
\arrow[from=4-3, to=3-3]
\arrow[from=4-4, to=3-4]
\arrow[from=4-5, to=3-5]
\arrow[from=4-6, to=3-6]
\arrow[from=4-7, to=3-7]
\arrow[from=4-8, to=3-8]
\arrow[from=4-9, to=3-9]
\arrow[from=4-10, to=3-10]
\arrow[from=4-11, to=3-11]
\arrow[from=4-12, to=3-12]
\arrow[from=4-13, to=3-13]
\arrow[from=4-14, to=3-14]
\arrow[from=4-15, to=3-15]
\arrow[from=4-16, to=3-16]
\end{tikzcd}$$

Figure 2: The Bayesian network.

2.2.1 Training Level $\mathcal{C}$

Each node at level $\mathcal{C}$ performs two types of unsupervised learning. The first learning task is to discretize the space of input vectors received from its child in level $\mathcal{I}$. Based on the training data, this discretization is carried out individually at each node in $\mathcal{C}$, and at each node this discretization process determines the unique code vectors whose indices form the possible values for its random variable c_{ij}. Two different discretization methods were tried. The first was to use k-means with the number of clusters determined by a parameter k_c. The second was to use a self-organizing map (SOM) of size $\lceil\sqrt{k_c}\rceil \times \lceil\sqrt{k_c}\rceil$.[1]

The second learning task is to learn the probability mass functions $P(c_{ij}|x_{b_{ij}})$, where $x_{b_{ij}}$ is the vector $\langle c_{ip}, c_{iq}, c_{ir}\rangle$ where $p = j + 1 \text{ mod } 4$, $q = i + 2 \text{ mod } 4$, and $r = i + 3 \text{ mod } 4$, and is simply the concatenation of the values of the other random values in $\mathcal{C}$ whose nodes feed to the same node in $\mathcal{B}$.

Learning $P(c_{ij}|x_{b_{ij}})$ was done using the nonparametric method outlined in [George and Hawkins, 2005]. To approximate $P(c_{ij}|x_{b_{ij}})$, for each $c_{ij} \in C$ a lookup table $T_{c_{ij}}$ was created of size $k_c^3 \times k_c$. The process then looped through the training data, and for each training record, $c_{ij} \in \mathcal{C}$ was given the value of the index of its node's closest code vector to I_{ij} using Euclidean distance, and c_{ij} was passed up to the node corresponding to b_i. The node corresponding to b_i then collected inputs from its other children and passed back $x_{b_{ij}}$. At that point each $T_{c_{ij}}[x_{b_{ij}}, c_{ij}]$ was then incremented by 1. After the completion of this process, $T_{c_{ij}}$ serves as an approximation for $P(c_{ij}|x_{b_{ij}})$.

[1]Standard Matlab library functions were used for performing k-means and SOM processing, all other processing was done by code written by me specifically for this project.

2.2.2 Training Level $\mathcal{B}$

Once level $\mathcal{C}$ has been trained, it can be used to generate training samples for level $\mathcal{B}$. To do this, the original training data is processed again and this time each node in $\mathcal{C}$ calculates the maximum likelihood estimate c_{ij} to pass to its parent node. This is done by using a modified version of equation (1). If we assume that the observation model does not depend on the background, then for each training record

$$c_{ij} = \text{argmax}_{c_{ij}} p(I_{ij}|c_{ij})p(c_{ij}|x_{b_{ij}}) \tag{2}$$

where $p(I_{ij}|c_{ij})$ is approximated by 1 divided by the Euclidean distance between I_{ij} and the code vector associated with c_{ij}[2], and $p(c_{ij}|x_{b_{ij}})$ is approximated by $T_{c_{ij}}$. Note that for each original training record, the node corresponding to b_i sees $\langle c_{i1}, c_{i2}, c_{i3}, c_{i4}\rangle$ twice. The first time it only collects it and sends it back to its children, and the second time it uses the updated $\langle c_{i1}, c_{i2}, c_{i3}, c_{i4}\rangle$ as its training record. Figure 3 shows a reconstruction of the handwritten digit in Figure 1 using the code vectors corresponding the the indices in $\mathcal{C}$ during training of $\mathcal{B}$. Notice that the reconstruction is not ideal, this will be discussed in Section 3.

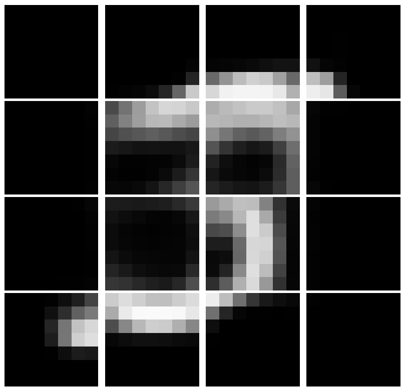

Figure 3: The handwritten digit in Figure 1 reconstructed from the code vectors at level $\mathcal{C}$.

Once the input to level $\mathcal{B}$ has been generated, level $\mathcal{B}$ is trained in the same way as level $\mathcal{C}$. The process loops through the training data again, and for each $b_i \in \mathcal{B}$ its corresponding node discretizes the space of input vectors of the form $\langle c_{i1}, c_{i2}, c_{i3}, c_{i4}\rangle$ into k_b code vectors using the same methods as in level $\mathcal{C}$. The indices of these code vectors provide the set of possible values for b_i. Also as in level $\mathcal{C}$, a table T_{b_i} of size $k_b^3 \times k_b$ populated by the same method as $T_{c_{ij}}$ is populated.

[2]I tried approximating $p(I_{ij}|c_{ij})$ using a multivariate normal, but Σ was often nearly singular.

2.2.3 Training Level $\mathcal{A}$

Levels $\mathcal{C}$ and $\mathcal{B}$ were trained in an unsupervised manner, however level $\mathcal{A}$ is trained in a supervised manner using the class labels of the handwritten digit training records. The input vectors to level $\mathcal{A}$ are generated in a similar way as the input vectors into level $\mathcal{B}$ are generated, except that Hamming distance is used at level $\mathcal{B}$ instead of Euclidean distance. Level $\mathcal{B}$ generates training samples for level $\mathcal{A}$ using appropriate variables in equation (2) and the input vectors provided to it from level $\mathcal{C}$.

Since there are 10 digits to be identified, training at this level consists of populating the table T_a of size $k_b^4 \times 10$. As each vector of the form $\gamma = \langle b_1, b_2, b_3, b_4 \rangle$ is received, the class label $\omega \in \{0, 1, 2, 3, 4, 5, 7, 8, 9\}$ is retrieved and $T_a[\gamma, \omega]$ is incremented by 1.

2.3 Inference

Generally, inference in such a network is done using loopy propagation [Pearl, 1988]. However, here, for simplicity, inference was done using a controlled looping mechanism that has much in common with how the training samples at various levels were generated. An unknown digit image is first partitioned by level $\mathcal{I}$ and then sent to level $\mathcal{C}$. Each $c_{ij} \in \mathcal{C}$ is then populated with the index of the code vector of its node that is closest to I_{ij} using Euclidean distance, and these values are then passed to level $\mathcal{B}$. The nodes in level $\mathcal{B}$ then collect the inputs from their children and pass a vector containing those inputs back to each child. Each node in $\mathcal{C}$ then uses equation (2) to pass up its revised c_{ij} value. Each $b_i \in \mathcal{B}$ is then populated with the index of the code vector of its node that is closest to the revised input vector from its children using Hamming distance, and these values are passed to level $\mathcal{A}$. The node at level $\mathcal{A}$ then collects the values $\langle b_1, b_2, b_3, b_4 \rangle$ and passes this vector back to each child. Each node in $\mathcal{B}$ then uses an equation equivalent to equation (2) to pass up its revised b_i value.

Finally, level $\mathcal{A}$ receives a vector $\langle b_1, b_2, b_3, b_4 \rangle$ and uses it to query table T_a. The normalized row retrieved from table T_a becomes the value of a. It then declares the index with the highest value in a to be the label of the unknown image.

3 Results

This implementation was not able to accurately classify unseen handwritten digits. The nearest neighbor and eigenface implementations both performed much better. The results are listed in Table 1. The running times for each method are listed in table Table 2. It is interesting to note that the SOM as a discretization method

Table 1: Percentage of Testing Data Correctly Classified (HB refers to hierarchical Bayesian implementation)

	10,000 train, 1000 test	1000 train, 1000 test
HB: k-means, $k_c = 7$, $k_b = 7$	28.2	36.0
HB: k-means, $k_c = 8$, $k_b = 8$	44.0	31.9
HB: k-means, $k_c = 9$, $k_b = 9$	42.5	32.6
HB: k-means, $k_c = 10$, $k_b = 10$	40.5	33.5
HB: SOM $k_c = 9$, $k_b = 9$	34.9	30.0
Nearest Neighbor, $k = 1$	92.0	82.8
Nearest Neighbor, $k = 3$	89.1	80.2
Eigenface method	91.1	81.6

Table 2: Running Time Comparison (HB refers to hierarchical Bayesian implementation)

	10,000 train, 1000 test
HB: k-means, $k_c = 7$, $k_b = 7$	9 min. 27 sec.
HB: k-means, $k_c = 8$, $k_b = 8$	10 min. 25 sec.
HB: k-means, $k_c = 9$, $k_b = 9$	12 min. 17 sec.
HB: k-means, $k_c = 10$, $k_b = 10$	14 min. 48 sec.
HB: SOM $k_c = 9$, $k_b = 9$	96 min. 43 sec.
Nearest Neighbor, $k = 1$	19 min. 15 sec.
Nearest Neighbor, $k = 3$	73 min. 46 sec.
Eigenface method	7 min. 36 sec.

performed worse than using k-means. For k-means the running time increased as k_c and k_b increased. This is most likely due to both k-means itself taking more time and the increase in size of $T_{c_{ij}}$, T_{b_i}, and T_a. The memory taken up by $T_{c_{ij}}$, T_{b_i}, and T_a could become an issue with larger implementations, and some method such as Parzen windows that does not store each value but instead approximates areas of value may need to be employed.

Although nearest neighbor did well on this task, in general such an approach does not scale well to larger images due to the curse of dimensionality. In the high-dimensional case, preprocessing must be done to extract the relevant features and reduce the dimensionality. An interesting outcome of the nearest neighbor experiments is that it did better with $k = 1$ than with $k = 3$. Further investigation showed that what was happening with $k = 3$ was that the closest neighbor was often correct but was outvoted by the next two closest.

The method of using eigenfaces also did well. This is somewhat surprising, since while centered faces are relatively similar to one another, handwritten digits of different numbers, e.g. 1 and 8, are quite different. The first three eigenvectors of the training data are shown in Figure 4. Notice that the eigenvectors, except for the first one, do not look nearly as similar to numbers as eigenfaces look similar to faces. Figure 5 shows the handwritten digit in Figure 1 reconstructed using eigenvectors. This approach might be more applicable with this dataset if the digits were first separated into classes and then the method was performed on each class.

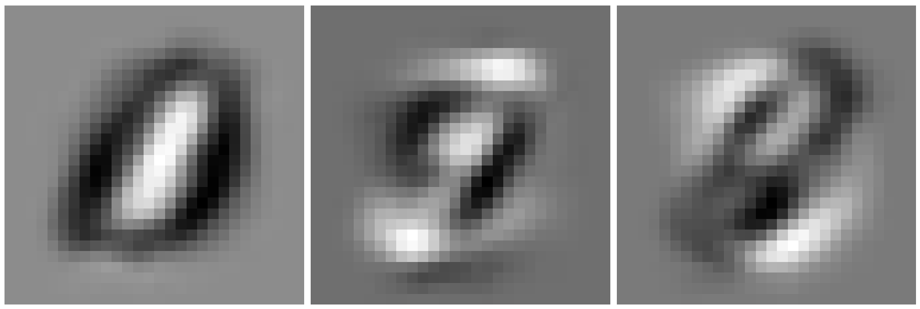

Figure 4: First 3 eigenvectors taken from the handwritten digit training data.

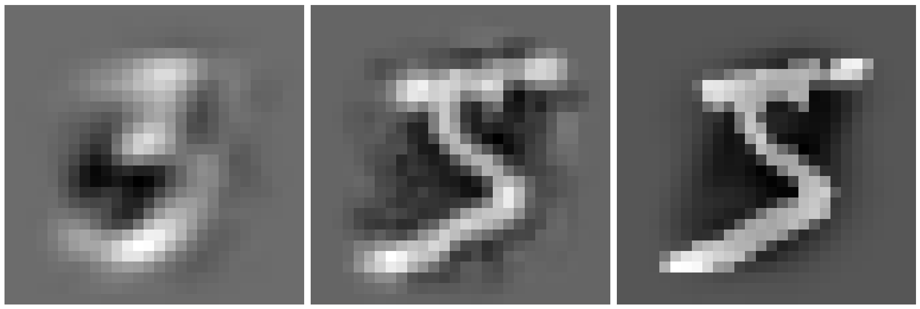

Figure 5: Reconstruction of the digit in Figure 1 using 10, 100, and 600 eigenvectors.

No explicit analysis of the variance was performed, and the value of 100 was chosen arbitrarily for the number of eigenvectors to use. At such a high number of eigenvectors this approach becomes similar to nearest neighbor, which may explain why their results were so close to each other. Since the images were relatively small, the trick outlined in [Turk and Pentland, 1991] was not necessary.

Thomas Dean [2005] implemented a more elaborate version of the Lee and

Mumford model. In his implementation he used the same number of levels but used more nodes at each level. Notice that in Figure 3 the middle part is poorly reconstructed. Having more nodes in this area would allow for a finer level of representation. Dean uses 49 nodes at this first processing layer, as opposed to the 16 used here.

Additionally, in Dean's implementation there was overlap in the area covered by some nodes, meaning that some nodes had more that one parent. He also used mixture-of-Gaussian classifiers at the lowest processing layer (corresponding to level $\mathcal{C}$) and EM in the intermediate layers, as as opposed to discretization via k-means or SOM's as was done here. This added complexity combined with the use of Kevin Murphy's Bayes Net Toolbox [Murphy, 2001] for inference allowed his implementation to achieve accuracy of up to 81% on this dataset. His implementation, however, took up to 20 hours to run on 10,000 training records, whereas this implementation generally finished in 20 minutes.

4 Conclusion

Although this implementation was not as successful as others at the task of classifying handwritten digits, it may still useful as a starting point for research. The hierarchical and the distributed nature of the network could be leveraged to allow the method to scale to larger images. The hierarchy allows the data to be viewed at multiple levels of abstraction and the hierarchy can be expanded in depth and width as the problem domain dictates. Additionally, the distributed nature of the message passing algorithm allows for parallel computation. And, when one considers that this is not a two class problem with an handful of features, but rather a ten class problem with $28 \times 28 = 784$ real-valued features, then 44% accuracy does not seem as bad.

Another interesting aspect of this implementation is that most of the training is unsupervised, it is only at the very top level that the class labels are used. This implies that the system is learning general features of the data that are not restricted to one learning task. In this way, this method is somewhat distinct from most learning methods that learn for the purpose of a particular classification problem. An interesting line of research, therefore, would be to create a methodology to add new code vectors at appropriate levels as the system encounters more input and as the system is tasked with new classification problems.

References

[Dean, 2005] Thomas Dean. Hierarchical expectation refinement for learning generative perception models. Technical Report CS-05-13, Brown University, Providence, RI, 2005.

[Geisler and Diehl, 2003] Wilson S. Geisler and Randy L. Diehl. A Bayesian approach to the evolution of perceptual and cognitive systems. *Cognitive Science*, 27(3):379–402, 2003.

[George and Hawkins, 2005] Dileep George and Jeff Hawkins. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Proc of the International Joint Conference on Neural Networks (IJCNN 2005)*. IEEE Computer Society, 2005.

[Knill and Pouget, 2004] D. C. Knill and A. Pouget. The Bayesian brain: the role of uncertainty in neural coding and computation. *Trends Neurosci*, 27(12):712–719, December 2004.

[LeCun and Cortes, 1998] Yann LeCun and Corinna Cortes. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist, 1998.

[LeCun *et al.*, 1998] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[Lee and Mumford, 2003] Tai Sing Lee and D. Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America*, **20**, 2003, 1434-1448., 2003.

[Murphy, 2001] Kevin Murphy. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33, 2001.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[Turk and Pentland, 1991] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neurosccience*, 3(1):71–86, 1991.