A Distributed Model for Image Recognition Using Pyramidal Bayesian Networks

Ethan L. Schreiber
Department of Computer Science
Brown University
Providence, RI 02912

Abstract

Pyramidal Bayesian Networks are graphical models that possess a biologically plausible structure for modeling the human visual cortex. Well known algorithms for exact inference on such networks are $\in O(2^n)$ which is intractable for any data set of interesting size. In this paper, we discuss the implementation of the hierarchical expectation refinement algorithm which breaks up the problem of learning on a large pyramid graph bayesian network into smaller tractable component learning problems. This allows our model to run on a massively distributed parallel network.

1. Introduction

There are many methods currently used for recognizing objects in digital images. These range from SVD [9] to PCA to convolution networks to support vector machines to nearest neighbor methods. [5] However, there are still many shortcomings to all of these methods. Namely, none of these methods are able to recognize diverse objects in images with the accuracy of a human counterpart.

Citing the results of neuro-physiological research on monkey's brains, Lee and Mumford suggest that in order for primates to recognize objects visually, messages are passed from the retina up the chain of the visual cortex and back down to the retina in a data propagation loop. They used this information to argue that inference based on particle filtering and belief propagation in Bayesian networks could be a good model of the visual recognition system of the human brain. They propose a "hierarchical Bayesian framework in the cortex." This is a feed-forward and feed-backwards Markov chain structure that is analogous to the hierarchical layers of the primate visual cortex. [6]

In "Scalable Inference in Hierarchical Generative Models" [4], Dean takes the ideas of Lee and Mumford and proposes a class of algorithms for performing inference on a pyramid graph Bayesian network. It is these algorithms which we have implemented that we discuss in this paper.

2 Pyramid graph Bayesian Network (PBN)

2.1 Neurological Background

The primate visual cortex consists of separate visual areas known as V1, V2, V3, V4 and V5 (also known as MT, the medial temporal). Pathways exist between all of these areas to allow for extensive message passing back and forth between visual areas. Studies in neuroscience have shown that these areas form a hierarchy in which visual features become more and more abstracted as the areas are traversed. For instance, V1 - the "bottom" layer - is the primary target of axons from the lateral geniculate nucleus (LGN), which receives input from retinal cells. Based on the retinal input provided by the LGN, V1 can abstract visual features such as lines in various orientations. This information is fed to V2, which is able to recognize more abstract features and feed it to higher visual areas. This correlates to the nature of our PBN. [2].

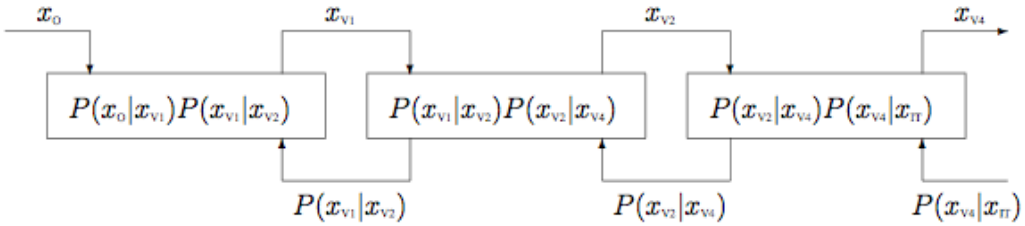

Figure 1: This is Dean's representation of Lee and Mumford's hierarchical Bayesian model. You can see how each region of the visual cortex (v1, v2, v4) is dependent directly and most importantly by its direct neighbors. [4, 6]

2.2 Structure

A Bayesian network is a set of random variables and a joint probability distribution over those variables. A pyramid graph Bayesian network is a standard Bayesian network with the property that the nodes are hierarchical belonging to a level in the pyramid. Only the lowest (base) level is observed. Each level is a rectangular grid. Nodes can have dependencies on axis aligned neighbors in the same level or adjacent nodes in the parent level. This type of network works particularly well with image recognition. The bottom level corresponds to the pixels of an image. As we go up the pyramid, the nodes represent larger regions of the image or in our case, larger receptive fields. [8, 3]

Imagine that we had a 100×100 pixel image and we wanted to build a 5 level PBN as follows:

level	Size	Receptive Field	overlap	number of nodes
1	100×100	-	0	10000
2	20×20	5×5	0	400
3	5×5	4×4	0	25
4	2×2	3×3	1	4
5	1×1	2×2	0	1

For a 100×100 pixel image which is the size of an image thumbnail, the network already has 14,030 nodes. Since exact inference is an exponential endeavor on such a network, we have a major problem. In order to deal with this problem, we describe a method for breaking up inference into smaller tractable problems.

3 Subnet Decomposition

After we have specified the pyramid Bayesian Network, we need to break it up into its components which we refer to as Subnets. Each subnet in our model consists of two levels, an observed level i and a hidden level $i + 1$. They are called hidden and observed because that is how they act as we propagate data from the input level to the output level in our distributed network. The observed node is observed with respect to the data it receives from its child. In reality, only the level 1 subnet's inputs are truly observed.

Each subnet has a small Bayesian network (or two in the case of a variable-order subnet) composed of the nodes in its hidden and observed levels. The connections between the nodes in each subnet are the same connections as the connections between the corresponding nodes in the PBN. It is important to realize that nodes in the PBN will appear in multiple locations within the distributed network. The hidden nodes of the subnets of level i will also be observed nodes in the subnets of level $i + 1$.

The following is a BNF specification of a subnet in our hierarchical network.

```
START :=  SUBNET {
            RANK{INTEGER}
            TYPE { SUBNET_TYPE }
            LEVEL { INTEGER }
            PROCESSOR { INTEGER }
```

```
  SUPERVISED { BOOLEAN }
  INPUT { [ NODES ] }
   * DOMAIN { [ NODES ] }
  *OUTPUT { [ NODES ] }
   *HIDDEN { [ NODES ] }
  *OBSERVED { [ NODES ] }
  *OBN_DAG { ADJ_SPECS }
  *OBN_SIZES { [ INTEGERS ] }
  *DBN_INTRA { ADJ_SPECS }
  *DBN_INTER { ADJ_SPECS }
  *DBN_SIZES { [ INTEGERS ] }
   LOCATIONS { LOC_SPECS }
   MESSAGES { MSG_SPECS }
}

LOC_SPECS :=  LOC_SPEC | LOC_SPEC LOC_SPECS
LOC_SPEC :=  LOCATION(SUBNET,NODE,CLASS)
MSG_SPECS :=  MSG_SPEC | MSG_SPEC MSG_SPECS
MSG_SPEC :=  CMD(SUBNET,SUBNET) { LOCATION(SUBNET,NODE,CLASS) }
ADJ_SPECS :=  ADJ_SPEC | ADJ_SPEC ADJ_SPECS
ADJ_SPEC :=  NEIGHBORS(NODE, [ NODES ])
SUBNET :=  INTEGER
CLASS :=  EVIDENCE | LABEL | LAMBDA | OUTPUT | PI
SUBNET_TYPE :=  BASE | ZERO_ORDER | FIRST_ORDER | VARIABLE_ORDER
NODES :=  NODE | NODE, NODES
NODE :=  INTEGER
INTEGERS :=  INTEGER | INTEGER, INTEGERS
INTEGER :=  1 | 2| 3| ...
BOOLEAN :=  0 | 1
CMD :=  RECV | SEND
```

Definition of terms	
SUBNET {	Signifies the start of a new subnet block.
RANK{**INTEGER**}	A unique number which can be used to identify the subnet in a distributed environment
TYPE { **SUBNET_TYPE** }	This specifies the type of the subnet which can be BASE, ZERO_ORDER, FIRST_ORDER or VARIABLE_ORDER
LEVEL { **INTEGER** }	The level in the PBN. 1 signifies an input subnet and k (where k is the height of the PBN) signifies an output subnet.
PROCESSOR { **INTEGER** }	This is used to specify the processor which this subnet's computation is to be executed on.
SUPERVISED { **BOOLEAN** }	Is this a supervised node? If so, we need to handle incoming label messages.
***INPUT** { **[NODES]** }	The local id of the input nodes in this subnet. These correspond to
***DOMAIN** { **[NODES]** }	The global id of the nodes in this subnet.
***OUTPUT** { **[NODES]** }	The output node of the subnet. This is typically only one node; the one which gives a classification to our image.
HIDDEN { **[NODES]** }	The list of hidden nodes for this subnet. A subnet receives PI messages corresponding to its hidden nodes. Every PBN node except for the ones in the highest level (so called output nodes) is a hidden node in exactly one subnet.
OBSERVED { **[NODES]** }	The list of observed nodes for this subnet. Every PBN node is an observed node in at least one subnet.

***OBN_DAG** { **ADJ_SPECS** }	An adjacency matrix for zero-order and first-order subnets. All adjacency matrices are stores as neighborhood lists. The form is a node followed by the list of its neighbors. This applies to DBN_INTRA and DBN_INTER as well.
***OBN_SIZES** { **[INTEGERS]**	The arity of the observation nodes.
***DBN_INTRA** { **ADJ_SPECS** }	An adjacency matrix for first-order and variable-order subnets. This matrix describe the intra-slice adjacencies.
***DBN_INTER** { **ADJ_SPECS** }	An adjacency matrix for first-order and variable-order subnets. This matrix describes the inter-slice adjacencies.
***DBN_SIZES** { **[INTEGERS]** }	The arity of the dynamic nodes.
LOCATIONS { **LOC_SPECS** }	The location of the nodes and the type of messages associated with that node
MESSAGES { **MSG_SPECS** }	The messages that are sent and received from this subnet.
LOC_SPEC	LOCATION(SUBNET,NODE,CLASS) Specifies a node location where SUBNET is the id of the subnet specified (for send messages, the subnet to send the message to and for receive messages, the subnet that is receiving the message), the NODE is the local id of the node the message is being sent to or received from and the CLASS is the class of the message: (EVIDENCE, LABEL, LAMBDA, OUTPUT or PI)
ADJ_SPEC	NEIGHBORS(NODE, [NODES]) Specifies an adjacency matrix using a neighbors list where NODE is the node we are referring to and NODES is a set of nodes that are adjacent to NODE.

An example subnet specification:

```
SUBNET {
  RANK { 5 }
  TYPE { ZERO_ORDER }
  LEVEL { 2 }
  PROCESSOR { 1 }
  SUPERVISED { 1 }
  OUTPUT { [1] }
  HIDDEN { [1] }
  OBSERVED { [2,3,4,5] }
  OBN_DAG {
    NEIGHBORS(1,[2,3,4,5])
    NEIGHBORS(2,[6])
    NEIGHBORS(3,[7])
    NEIGHBORS(4,[8])
    NEIGHBORS(5,[9])
  }
  OBN_SIZES { [5,5,5,5,5,1,1,1,1] }
  LOCATIONS {
    LOCATION(5,1,LABEL)
    LOCATION(5,2,LAMBDA)
    LOCATION(5,3,LAMBDA)
    LOCATION(5,4,LAMBDA)
    LOCATION(5,5,LAMBDA)
  }
  MESSAGES {
    RECV(5,6) { LOCATION(5,1,LABEL) }
    RECV(5,1) { LOCATION(5,2,LAMBDA) }
    RECV(5,2) { LOCATION(5,3,LAMBDA) }
    RECV(5,3) { LOCATION(5,4,LAMBDA) }
    RECV(5,4) { LOCATION(5,5,LAMBDA) }
```

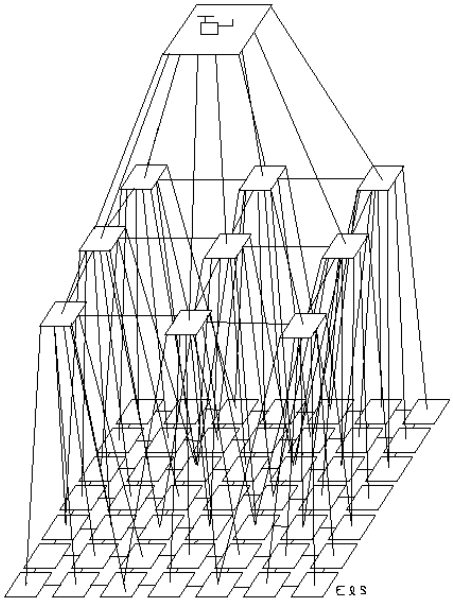

Figure 2: A drawing of a bayesian network of three levels. Level 1, the input level is 7*7. Level 2 is 3*3 and maps onto level 1 with an overlap of 1. Level 3, the output node is 1*1.

```
    SEND(5,1) { LOCATION(1,1,PI) }
    SEND(5,2) { LOCATION(2,1,PI) }
    SEND(5,3) { LOCATION(3,1,PI) }
    SEND(5,4) { LOCATION(4,1,PI) }
    SEND(5,6) { LOCATION(6,1,OUTPUT) }
  }
}
```

4 Subnet Types

We use three different types of Subnets: zero-order, first-order and variable-order. The difference between the types has to do with how they learn their internal parameters.

4.1 Zero Order

The zero-order subnet has a static internal Bayesian network. This means that it learns the properties of images one by one. There is no sequence of images it can learn from.

4.2 First Order

The first-order subnet has a dynamic Bayesian network. When we learn the parameters of our network, it is based not only on the current evidence but also of the evidence of the previous time slice. In order for this type of subnet to be effective, data is fed to the network in sequences.

4.3 Variable Order

The variable-order subnet has both a static and dynamic Bayesian network. The static network is called an observation network. It is used to model the observation data during the learning phase of our algorithm. This data is then passed

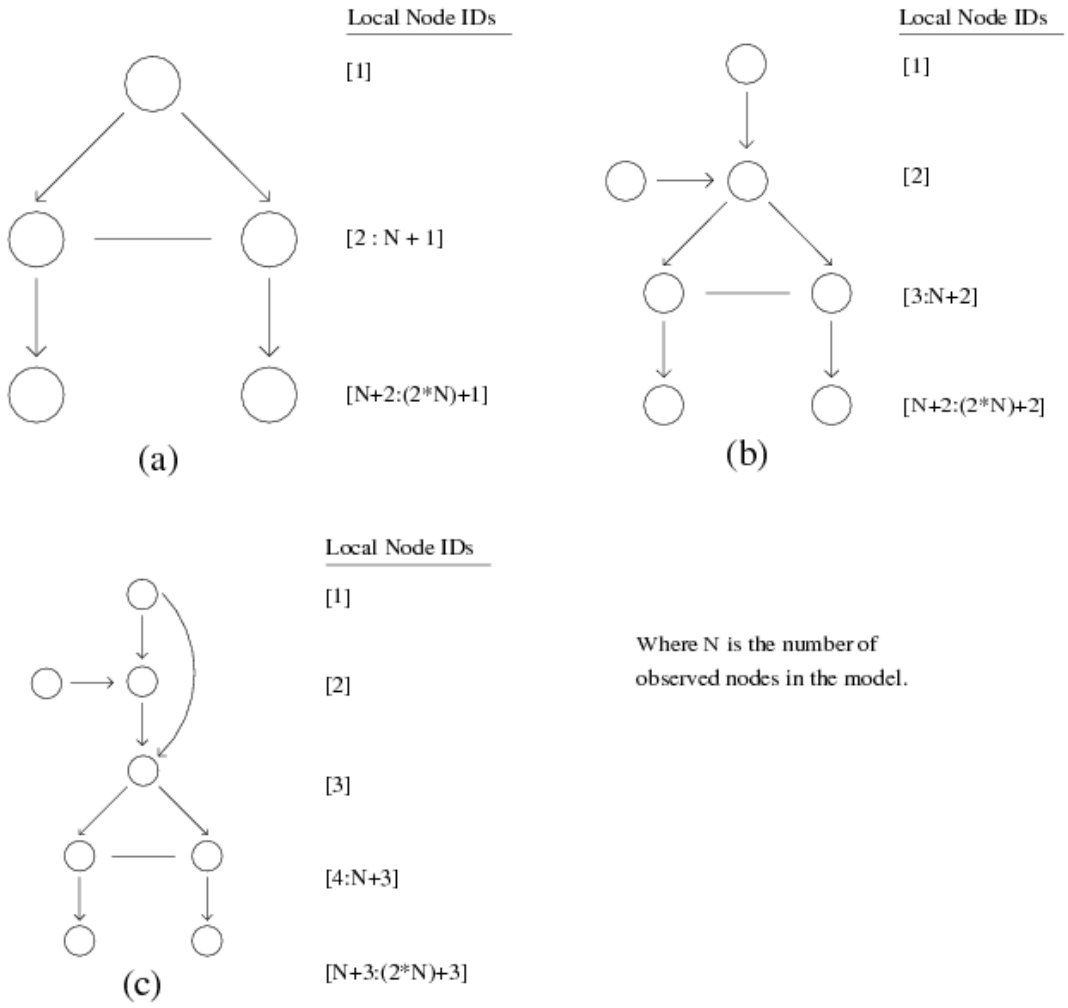

Figure 3: The Bayesian networks for the different subnet types. (a) is a static Bayes net for zero-order. (b) is a dynamic Bayes net for first-order. (c) is a dynamic Bayes net for variable-order.

into a Variable Length Markov Model. This model will generate a predictive suffix tree that we can use to generate our dynamic Bayesian network. This network is now dependent on the evidence of the current slice of evidence and a variable number of previous slices. [1]

5. Message Passing

Now that we have defined the basic structure of each subnet, the next step is for the subnets to communicate with each other. As seen in the subnet specification BNF, subnets can pass messages of 5 different classes:

1. LAMBDA messages are passed upward from child Subnet to parent subnet. They contain a set of probabilistic potentials over the possible values that a node can take on.
2. PI messages are passed downward from parent subnet to child subnet. They contain a set of probabilistic potentials over the possible values that a node can take on.
3. EVIDENCE messages contain an integer which corresponds to the class of the evidence. It will typically be a pixel intensity. They are passed to the bottom level subnets.

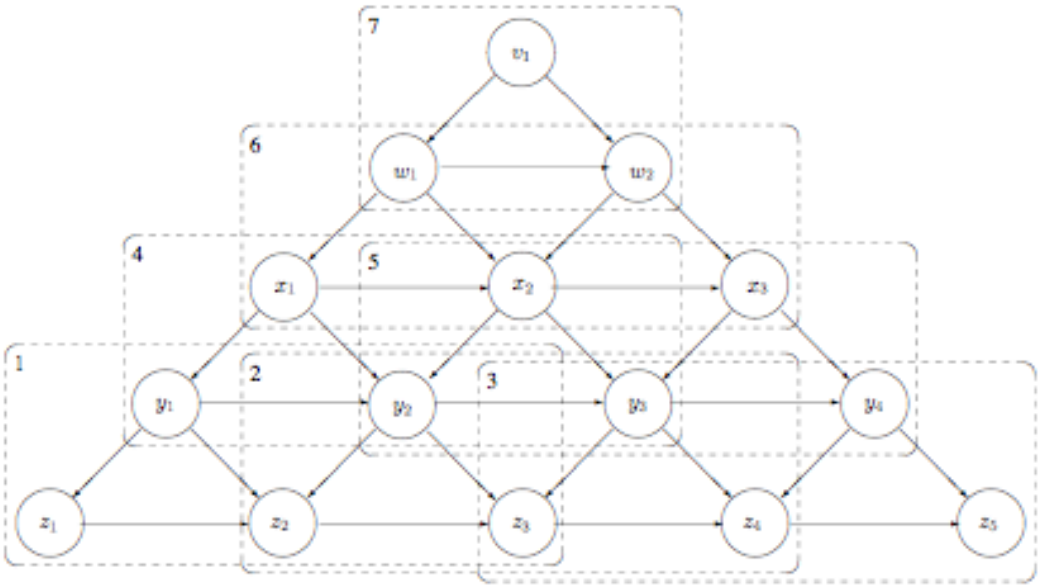

Figure 4: This is Dean's example of a pbn broken up into its subnets. [4]

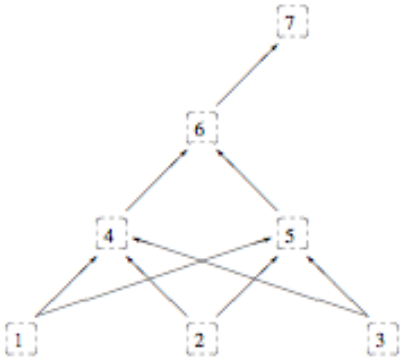

Figure 5: This is the resulting subnet graph. [4]

4. LABEL are passed to the output node during the supervised learning period. It is later passed when testing to see how the networked performed.

5. OUTPUT messages are sent from the Output subnet. They contain the best guess of the network as to what object it was looking at.

5.1 Expected Messages

Each subnet is specified with certain messages it expects to receive. On each iteration of evidence being entered into the model, each subnet waits until they receive all of their messages. Once they receive all of their messages, then there next action is dependent on the state of the network. When we first begin training our network, we are in a training phase. Before we can do any meaningful computation, we need to see a certain number of samples. There is a threshold set of the number of cycles of messages a subnet needs to receive before it can start passing messages further up the hierarchy. Before the threshold is met, the subnet

Our algorithm starts with Evidence messages being sent to the first-level subnets. During the learning phase of the algorithm, subnets cannot continue to pass messages up the chain until they have seen a certain number of messages

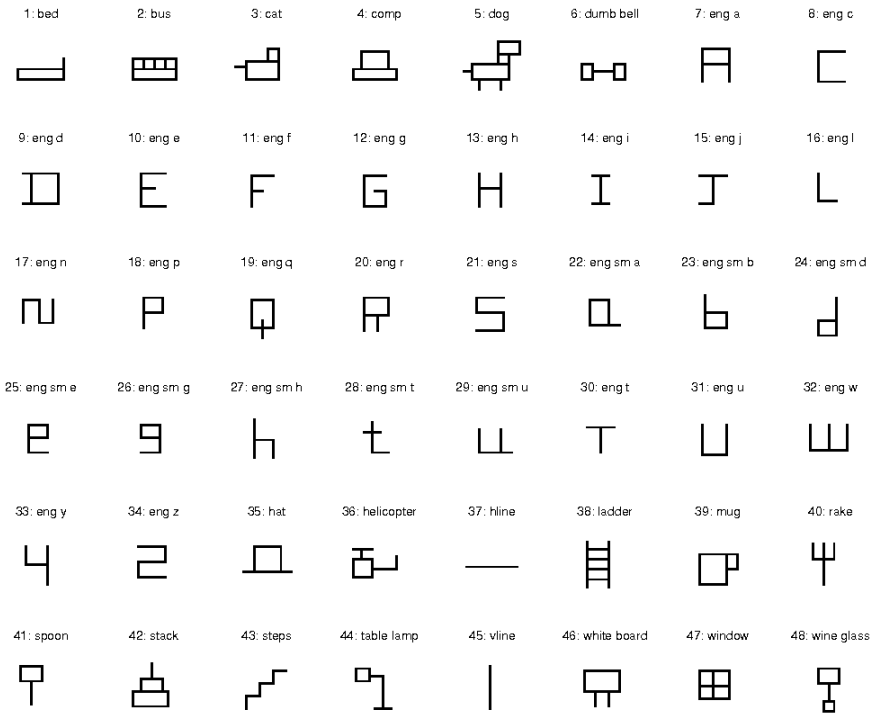

Figure 6: Dileep George's data set of line drawings.

already. This threshold is set globally. Until that threshold is met, we simply store the evidence passed in and pass uniform distributions back down in PI messages.

After we have seen enough messages to pass our threshold, the subnet calls a learn method to run the learning method that corresponds to the type of subnet as discussed above. After this, the Subnet then computes likelihoods over the specified nodes and passes LAMBDA messages up to its parents. The whole process then starts at the next level. Now that this level is trained, when its parents pass their PI messages back down to this level's subnets, this level can now marginalize actual PI distributions to pass down to its children as opposed to uniform distributions. The PI and LAMBDA messages are computed according to Pearl's polytree algorithm. [**?**]

5.2 Implementation

The subnet and message passing code has been implemented using C++ and Intel's Probabilistic Network Library. (http://www.intel.com/technology/computing/pnl/) The code is available upon request.

In our current implementation, messages are passed serially to a dummy Message Router as a proof of concept. All of the foundation is in place to convert this to a parallel architecture. We are collaborating with a group at Google who will take our current architecture and run it on their large scaled distributed network. All of the messages passed to a specific level and the computation that the subnet needs to perform after receiving the messages can be done in parallel. Since out pyramid networks are relatively shallow, this should speed up performance tremendously.

6. Dataset

The initial goal for our project is to be able to effectively recognize sequences of images from Dileep George's dataset of line drawings consisting of 48 total images of either English characters or simple renderings of objects. Each image

is a 32 x 32 pixel bitmap, with pixel values represented by one bit: 1 for white and 0 for black. The dataset is displayed in Figure 6. As our distributed network is completed, we hope to be able to deal with much more complex datasets.

7. Summary and Conclusions

We have described an architecture for building a hierarchical pyramid graph network inspired by the primate visual cortex for image recognition. The architecture that we currently have is a stepping stone to a distributed network which will allow us to experiment with computer algorithms that will attempt to recognize patterns much like primates do. Since modern pattern recognition algorithms in general do not have the power of primates, this is an exciting step indeed.

Note

Excerpts of the introduction and neurological background sections were taken from Theresa Vu and my final project for our CS0297 S10 Reading and Research Class for the Fall of 2005.

Acknowledgments

I would like to thank my research collaborator Theresa Vu for helping me in no small way throughout this project. From working through belief propagation algorithms to giving me a place to stay when we had to go to Google to present out work, you were always there.

I would also like to thank Professor Thomas Dean of Brown University for his years of work on this subject and tireless efforts explaining it to me. Your kindness and especially patience was very important to the completion of this masters degree.

References

[1] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order markov models. Technical Report CS-2004-07, Technion - Israel Institute of Technology, May 2004.

[2] Thomas Dean. A computational model of the cerebral cortex. *Proceedings of Twentieth National Conference on Artificial Intelligence*, AAAI-05:938–943, 2005.

[3] Thomas Dean. Hierarchical expectation refinement for learning generative perception models. *Technical Report CS-05-13*, pages 1–12, 2005.

[4] Thomas Dean. Scalable inference in hierarchical generative models. *Submitted to the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 1–9, 2005.

[5] Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[6] Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *Journal of Optical Society of America*, pages 1434–1448, 2003.

[7] Kevin P. Murphy. The bayes net toolbox for matlab. *Computing Science and Statistics*, 33, 2001.

[8] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. pages 467–475, 99.

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